An Imitation Learning Curriculum for Text Editing with Non-Autoregressive Models

Sweta Agrawal
Department of Computer Science
University of Maryland
sweagraw@cs.umd.edu

Marine Carpuat
Department of Computer Science
University of Maryland
marine@cs.umd.edu

Abstract

We propose a framework for training non-autoregressive sequence-to-sequence models for editing tasks, where the original input sequence is iteratively edited to produce the output. We show that the imitation learning algorithms designed to train such models for machine translation introduces mismatches between training and inference that lead to undertraining and poor generalization in editing scenarios. We address this issue with two complementary strategies: 1) a roll-in policy that exposes the model to intermediate training sequences that it is more likely to encounter during inference, 2) a curriculum that presents easy-to-learn edit operations first, gradually increasing the difficulty of training samples as the model becomes competent. We show the efficacy of these strategies on two challenging English editing tasks: controllable text simplification and abstractive summarization. Our approach significantly improves output quality on both tasks and controls output complexity better on the simplification task.

1 Introduction

Neural sequence-to-sequence (seq2seq) models primarily developed and tested for machine translation (MT) Bahdanau et al. (2015); Vaswani et al. (2017); Gu et al. (2018) are increasingly used for other sequence transduction tasks. This paper focuses on *editing tasks*, such as post-editing of MT output (Simard et al., 2007), style transfer (Jin et al., 2020), or text simplification (Chandrasekar and Srinivas, 1997; Xu et al., 2015), where systems directly edit the input sequence, instead of generating the output from scratch as in MT. As illustrated in Table 1, in these tasks, there might be substantial overlap in content between inputs and outputs, and also diverse rewrites, ranging from local substitutions to more complex restructuring.

While dedicated architectures have been designed for these editing tasks, based on e.g., a multistep, tag-then-edit approach (Alva-Manchego et al., 2017; Malmi et al., 2019; Dong et al., 2019; Mallinson et al., 2020), they can also be addressed with non-autoregressive (NAR) seq2seq models which generate their output by iteratively editing intermediate sequences (Lee et al., 2018; Gu et al., 2019; Awasthi et al., 2019; Stern et al., 2019; Chan et al., 2020). NAR models hold the promise of providing a more generic solution, where the model does not need to be tailored to a given editing task.

Original: The Mauritshuis museum is *staging an exhibition focusing* on the 17th century self-portraits*, highlighting* the similarities and the differences between *modern-day snapshots and historic works of art.*

Simplified: The Mauritshuis museum is *now set to open an exhibit* on the 17th century self-portraits*. It shows* the similarities and differences between *modern photos and artworks.*

Table 1: Text simplification is an editing task, where the output sequence overlaps with the input, while incorporating multiple rewrite types to restructure and simplify content.

This work is centered on the hypothesis that training NAR models for editing tasks using the same strategy as for MT leads to a mismatch between train and test settings that limits their generalization ability and output quality. Specifically, the learning algorithms designed for MT are aligned with inference strategies that generate output from an empty initial sequence. By contrast, in sequence editing tasks, the inference step is initialized instead with the original input sequence. In addition, since editing samples might range from limited lexical substitutions to more thorough rewrites, training samples cover a wide range of edit distances. During training, the loss can thus be dominated by the more distant samples leading to undertrained models and poor generalization. By contrast, the

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 7550 - 7563

distance between input and output samples in MT is more uniform, since it always involves at least lexical translation of the input tokens.

To address these issues, we introduce a new training framework, EDITING CURRICULUM, which dynamically exposes the model to more relevant edit actions during training and exploits the full spectrum of available training samples more effectively. First, we design a new roll-in strategy, EDITING *roll-in*, that exposes the model to intermediate sequences that it is more likely to encounter during inference. Second, we introduce a training CURRICULUM to expose the model to training samples in order of increasing edit distance, thus gradually increasing the complexity of oracle edit operations that the model learns to imitate.

We show that our approach improves the quality of outputs on two challenging English text editing tasks: controllable text simplification (TS) and abstractive summarization. It also improves the degree of TS control by generating simplified outputs that match the target reading grade level better than the baselines. We conduct an extensive analysis which supports our hypothesis, and show that the sequences generated by our training policy improve exploration during training and are easier to learn from, leading to better generalization across samples with varying edit distances. Training with curriculum further improves output quality.

2 Background

Model NAR edit-based models (Chan et al., 2020; Gu et al., 2019; Stern et al., 2019; Xu and Carpuat, 2021) cast sequence editing as an iterative sequence refinement problem modeled by a Markov Decision Process $(\mathcal{Y}, \mathcal{A}, \mathcal{E}, \mathcal{R}, \boldsymbol{y}^0)$. A state $y = (y_1, y_2, ..., y_L) \in \mathcal{Y}$ is a sequence of tokens where each y_i represents a token from the vocabulary $\mathcal{V}$, L is the sequence length and $y^0 \in \mathcal{Y}$ is the initial sequence to be refined, using actions drawn from the set $\mathcal{A}$. The reward $\mathcal{R}$ is based on the distance $\mathcal{D}$ between the generated output and the reference sequence $y^* \in \mathcal{Y}$: $\mathcal{R}(y) = -\mathcal{D}(y, y^*)$. At each decoding iteration, the model takes an input y, chooses an action $a \in \mathcal{A}$ to refine the sequence using a policy π, resulting in state $\mathcal{E}(y, a)$.

Models differ based on the nature of edit actions used and support different operations such as insertion, deletion, reposition and substitution. We select the operations from the EDITOR model based on its competitive performance on constrained decoding tasks that require editing non-empty initial sequences (Xu and Carpuat, 2021). It is a Transformer model that uses two types of actions or edits on sequences, y:

1. The **reposition** operation, modeled by π_{rps}, predicts the new position of each token in the input sequence. For each input position, the reposition policy predicts a value r that corresponds to the index of the input token to be placed at the position and 0 if the input token is to be deleted.

2. The **insertion** operation has two components: *placeholder prediction*, π_{plh} that predicts the number of placeholders to be inserted and *token prediction*, π_{ins} that generates the actual output tokens for each placeholder.

At each decoding iteration, the model applies an action a that consists of a reposition and an insertion operation. This refinement process is repeated until two consecutive decoding iterations return the same output (Gu et al., 2019), or a preset maximum number of them is reached (Lee et al., 2018; Ghazvininejad et al., 2019).

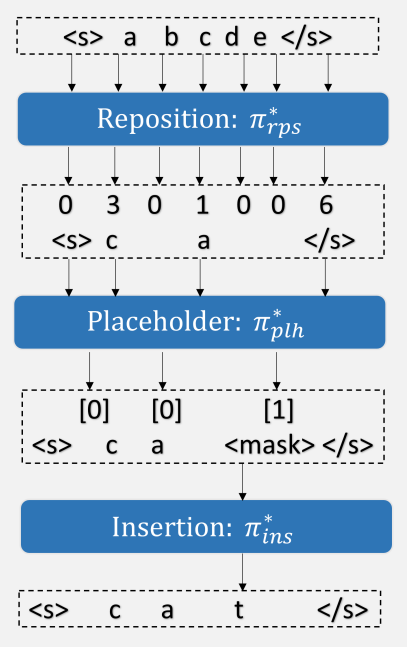


Figure 1: One refinement iteration for the input sequence: "a b c d e" using the operations generated by the Levenshtein Edit Distance Algorithm.

Training NAR models are typically trained via imitation learning that uses a roll-in policy and a roll-out policy. The roll-in policy is used to generate the sequences that the model learns to refine

	Operations	Roll-In	Roll-in Policies
Gu et al. (2019)	Insertion, Deletion	Mixed	$y' = \{\mathcal{E}(y^*, \tilde{d}), \tilde{d} \sim \pi_{rnd}\}$ $y_{ins} = \{y' \text{ if } u < \alpha \text{ else } \mathcal{E}(y^s, d^*), d^* \sim \pi^*_{del}\}$ $y_{del} = \{y^s \text{ if } u < \beta \text{ else } \mathcal{E}(\mathcal{E}(y_{ins}, p^*), \tilde{t}), p^* \sim \pi^*_{plh}, \tilde{t} \sim \pi_{ins}\}$
Stern et al. (2019)	Insertion	Expert	$y_{ins} = \{\mathcal{E}(y^*, \tilde{d}), \tilde{d} \sim \pi_{rnd}\}$
Ghazvininejad et al. (2019)	Substitution	Expert	$y_{sub} = \{\mathcal{E}(y^*, \tilde{m}), \tilde{m} \sim \pi_{mask}\}$
Saharia et al. (2020)	Substitution	(Offline) Learned	$y_{sub} = \{\mathcal{E}(y, \tilde{m}), \tilde{m} \sim \pi_{mask}\}$
Qian et al. (2020)	Substitution	Expert	$y_{sub} = \{\mathcal{E}(y, \tilde{m}), \tilde{m} \sim \pi_{mask}\}$
Xu and Carpuat (2021)	Insertion, Reposition (including deletions)	Learned	$y' = \{\mathcal{E}(\mathcal{E}(y^*, \tilde{d}), \tilde{p}), \tilde{d} \sim \pi_{rnd}, \tilde{p} \sim \pi_{per}\}$ $y_{ins} = \{y' \text{ if } u < \alpha \text{ else } \mathcal{E}(y, r), r \sim \pi_{rps}\}$ $y_{rps} = \{y' \text{ if } u < \beta \text{ else } \mathcal{E}(\mathcal{E}(y, p^*), \tilde{t}), p^* \sim \pi^*_{plh}, \tilde{t} \sim \pi_{ins}\}$

Table 2: Training Policies and Edit Operations performed by different NAR models: y^s: original input sequence, y^*: output sequence, y: model generated variant of reference sequence, π_{rnd}/ π_{masks} drops/masks random words from y^* according to a distribution (e.g. uniform, bernoulli, etc.), π_p generates a permutation, $u :\sim Uniform[0, 1]$, $\pi_{ins}, \pi_{plh}, \pi_{del}, \pi_{rps}$ are insertion, placeholder prediction, deletion and reposition policies.

from. A roll-out policy is then used to estimate the cost-to-go from the generated roll-in sequences to the desired output sequences. The cost-to-go is calculated by comparing the model actions to oracle demonstrations. We summarize the policies of various NAR models proposed for MT in Table 2.

For EDITOR, the roll-in sequences for the reposition (or insertion) module are stochastic mixtures (parameterized by α or β) of the output of the insertion (or reposition) module or a noised version of the output sequence. The oracle is the Levenshtein edit distance (Gu et al., 2019). The noisy sequence is generated by applying random word dropping (Gu et al., 2019) and random word shuffle (Lample et al., 2018) with a probability of 0.5 and maximum shuffle distance of 3. Figure 1 shows an example instantiation of the edit actions generated by the Levenshtein Edit Distance to transform the original input sequence ("a b c d e") to the output sequence ("c a t"). In this example, the oracle action is to delete the tokens ["b", "d", "e"], reposition "a" and "c" and insert "t" at the appropriate position. The reposition and the insertion modules are trained in a supervised fashion to predict these oracle operations during training.

3 Our Approach: Editing Curriculum

To tailor training to editing tasks, we propose to modify the roll-in policy to better match the intermediate sequences encountered at inference, and introduce a curriculum to increase the difficulty of oracle actions learned throughout training.

Editing Roll-in Sequences generated using the roll-in policy control the search space explored during training. Those sequences should therefore be representative of the intermediate sequences generated at inference time (Ross and Bagnell, 2010). While typically, the roll-in policy is a stochastic mixture of the model and the expert demonstrations as described above, the noise incurred early on due to the large difference between the expert demonstration and the learner's policy actions may hurt overall performance (Brantley et al., 2019; He et al., 2012; Leblond et al., 2018). As we will see (§5), this is what happens on editing tasks when training the model to imitate experts using learned roll-in sequences. At the same time, rolling in with expert demonstrations raises its own issues, as it can limit the exploration of the search space.

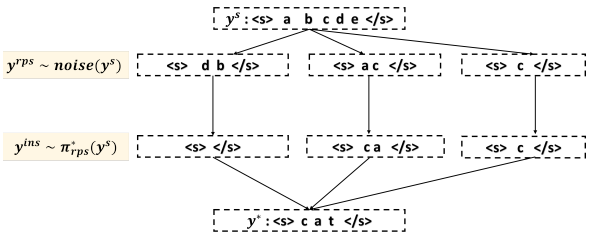


Figure 2: Example roll-in sequences for the reposition and the insertion modules: The same initial input sequence (y^s) can enable the model to learn to generate the reference output (y^*) using different edit operations from its noised version.

Motivated by these observations, we propose a new policy, Editing, that allows exploration by injecting noise to the input sequence to generate new intermediate sequences for training. This lets the model learn to fix errors without deviating from learning the task at hand. Figure 2 shows an example of intermediate sequences generated by our proposed roll-in policy. Different intermediate sequences encourage the model to learn differ-

Algorithm 1: Our proposed framework: EDITING CURRICULUM

Input: Dataset, $D = \{y, y^*\}_{i=1}^{M}$, difficulty scoring function, d, and competence function, c.

Compute the difficulty, $d(s_i)$, for each $s_i = \{y_i, y_i^*\} \in D$.

Compute the cumulative density function (CDF) of the difficulty scores. This results in one difficulty CDF score per sample, $\tilde{d}(s_i) \in [0, 1]$

Initialize π_{rps} and π_{ins}.

for *training step* $t = 1...T$ **do**

  Compute the competence value, $c(t)$.

  Create training dataset from by selecting all samples, B_t using $s_i \in D$, such that $\tilde{d}(s_i) \leq c(t)$.

  for *i in* $1..|B_t|$ **do**

    Generate roll-in sequences:

    $y_{rps} = noise(y^s)$

    $y_{ins} = \mathcal{E}(y_{rps}, r^*), r^* \sim \pi^*_{rps}$

    Train π_{rps} and π_{ins} on y_{rps} and y_{ins} minimizing cost-to-go to y^*.

Return **best** π_{rps} and π_{ins} evaluated on validation set.

ent reposition and insertion edit operations starting from the same input sequence, hence enabling exploration. We modify the roll-in policies to be aligned with the editing inference process, where the reposition operation is followed by insertion on the original input sequence:

- The roll-in sequence for training the reposition module, π_{rps}, is generated by applying noise to the original source sequence y^s, i.e. $y_{rps} = noise(y^s) = \{\mathcal{E}(\mathcal{E}(y^s, \tilde{d}), \tilde{p}), \tilde{d} \sim \pi_{rnd}, \tilde{p} \sim \pi_{per}\}$. Unlike EDITOR, the random word dropping ($\tilde{d} \sim \pi_{rnd}$) and the word shuffling ($\tilde{p} \sim \pi_{per}$) are applied to the original input sequence instead of the output sequence. This aligns the training with the inference scenario where the model edits an original input sequence instead of generating an output from scratch.
- The roll-in sequence for training the insertion module, π_{ins} is an intermediate sequence generated by applying the expert reposition policy to y_{rps}, i.e. $y_{ins} = \{\mathcal{E}(y_{rps}, r^*), r^* \sim \pi^*_{rps}\}$. The expert reposition policy corresponds to the deletion and reposition actions derived by using the levenshtein edit distance algorithm between the noisy input sequence, $noise(y^s)$ and the target sequence, y^*.

Curriculum controlled roll-out To prevent undertraining when samples with large edit distances overwhelm the loss, we use a curriculum to expose the model to easy-to-learn actions first, then gradually increase the difficulty of the edit-operations performed as the learner becomes more competent. Prior work on curriculum learning (CL) does not agree on standard measures of sample difficulty for seq2seq tasks (Kumar et al., 2019; Yao et al., 2021; Zhang et al., 2018; Zhou et al., 2020) or apply CL for the different problem of shifting the training of a Transformer model from AR to NAR regimes (Guo et al., 2020; Liu et al.). By contrast, in our settings, the Levenshtein distance provides a measure of difficulty that directly aligns with the model design and the training oracle.

Resulting Algorithm Given a training dataset $\mathcal{D} = \{y^s, y^*\}_{i=1}^{M}$ consisting of $\mathcal{M}$ samples, the difficulty score $d(s_i)$ for each sample $s_i = \{y_i^s, y_i^*\} \in \mathcal{D}$ is measured by the Levenshtein Distance between the input and the output sequence. The cumulative density function (CDF) of the difficulty scores results in one difficulty CDF score per sample, $\tilde{d}(s_i)$. At each training step t, we estimate the progress made by the learner by computing the competence of the model $c(t) \in (0, 1]$ as follows:

$$c_{\text{sqrt}}(t) = \min\left(1, \sqrt{t\frac{1 - c_0^2}{\lambda_t} + c_0^2}\right)$$

where, λ_t defines the length of the curriculum[1]; $c_0 = 0.1$ as in Platanios et al. (2019).

Based on this competence value $c(t)$, the model is then trained on all the samples whose difficulty as measured by the Levenshtein distance between the input and the output sequence is lower than that

[1]We set the curriculum length to 5K for our experiments.

competence value, i.e. $\tilde{d}(s_i) \leq c(t)$. The resulting algorithm is also shown in Algorithm 1.

4 Experimental Settings

We evaluate our approach on Controllable Simplification and Abstractive Summarization, two challenging sequence editing tasks that are motivated by real world information access needs. They are challenging because they require learning to perform a wide range of rewrites (from local substitution to sentence restructuring).

4.1 Controllable Simplification

Task Definition Given a complex text and a target grade level, the goal is to generate a simplified output that is appropriate for the desired grade level. The type of operations performed across different grade levels span sentence splitting, paraphrasing, deletion, content elaboration and substitution.

Data We use English Newsela samples as extracted by Agrawal and Carpuat (2019) with 470k/2k/19k for training, development and test sets respectively. Grade side-constraints are defined using a distinct special token for each grade level (from 2 to 12) and are introduced as side constraints for both the input and the output grade levels Scarton and Specia (2018).

Evaluation Metrics We automatically evaluate truecased detokenized system outputs using: **SARI** (Xu et al., 2016), which measures the lexical simplicity based on the n-grams kept, added, and deleted by the system relative to the input and the output sequence. It computes the F1 score for the n-grams that are added (**add-F1**). The model's deletion capability is measured by the F1 score for n-grams that are kept (**keep-F1**) and precision for the n-grams that are deleted (**del-P**) [2]; **Pearson's correlation coefficient (PCC)** between the complexity of the system and reference outputs as measured by Automatic Readability Index (ARI) (Senter and Smith, 1967) and **ARI-Accuracy** (Heilman et al., 2008) representing the percentage of sentences where the system output grade level is within 1 grade of the reference text according to the ARI.

4.2 Abstractive Summarization

Task Given a short paragraph (one or two sentences on average), the goal is to generate a concise summary that captures the salient ideas of the source text. It contains heavy deletions with moderate amounts of substitutions and frequent shifts caused by re-orderings.

Data We use the dataset from Toutanova et al. (2016), which contains 6K short input texts, with upto 5 summaries each. We use the same split as provided by the authors with 4937/448/786 unique input texts in the training, development and test sets respectively. The human experts were allowed to insert new words and reorder parts of the sentence when generating the summary, which makes this dataset particularly suited for abstractive summarization models.

Evaluation Metrics We automatically evaluate truecased detokenized system outputs using: **Rouge-L**[3](Lin, 2004). Even though it is not a summarization metric, we also report **SARI** to track the nature and type of edit operations performed. Given multiple references for each input text, we define the corpus level score as the arithmetic mean of automated metrics at the instance level, which is further averaged across the multiple references.

4.3 Model configurations

Data Preprocessing We pre-process all data using Moses tools for normalization, and truecasing. We apply subword segmentation with a joint input-output byte pair encoding model with $32,000$ operations. We use ARI to compute the input grade level at the inference time.

Architecture We adopt the base Transformer architecture (Vaswani et al., 2017) with d_{model} = 512, d_{hidden} = 2048, n_{heads} = 8, n_{layers} = 6, and $p_{dropout}$ = 0.1 for all our models. We add dropout to embeddings (0.1) and label smoothing (0.1). The base EDITOR model is trained using Adam with initial learning rate of 0.0005 and a batch size of $16,000$ tokens. The model is further fine-tuned on the editing task with a learning rate of 0.0001. We train all our models on two GeForce GTX 1080Ti GPUs. The average training time for a single seed of AR model is ∼8-9 hrs and for the EDITOR model is ∼20-22 hrs. Fine-tuning EDITOR takes additional 5-6 hrs. Training stops after 8 checkpoints without improvement of validation perplexity. All models are implemented using the Fairseq toolkit.

[2] https://github.com/cocoxu/simplification

[3] https://github.com/pltrdy/rouge

Models We compare our proposed approaches against the following models trained from scratch in controlled conditions: 1) **AR** is a auto-regressive (AR) transformer model (Scarton and Specia, 2018). 2) We train EDITOR with the dual-path roll-in policy as in Xu and Carpuat (2021), refered to as **From Reference**. We fine-tune EDITOR with the following policy variants: 3) **From Input** replaces the reference with the input for generating the initial sequence as in Agrawal et al. (2021). 4) **Editing** is our proposed roll-in policy. 5) **Editing Curriculum**, EDITCL, refers to our approach as described in §3. During inference, we start from the input sequence (y^s), which is refined iteratively by applying a sequence of actions, as described in §2 until 1) the output sequences from two consecutive iterations are the same, or 2) the maximum number of decoding steps ($N = 10$) is reached. The edit distance between two sequences is measured by the Levenshtein edit distance (Levenshtein et al., 1966).

5 Findings

Controllable Simplification As can be seen in Table 3, our overall training framework, EDITCL improves over the prior training strategy for EDITOR— From Reference — significantly for all metrics (SARI: +3.8, PCC: +0.091, ARI-Acc: +10.1%), and over the AR baseline. Ablations show that this is a combined effect of multiple factors. Dual-path roll-in, From Input improves over From Reference as expected (SARI: +1.9, PCC: +0.077, ARI-Acc: +8.0%), as the roll-in sequences encountered during training are similar to those encountered during inference. Using expert roll-in (EDITING) performs better than using learned roll-in (dual-path roll-in) across the board, with gains of up to 3 SARI points over From Reference. Training with CL (EDITCL) improves over the best roll-in strategy[4], improving the precision of deletions (+1.6) and leading to a significant improvement in SARI score (+0.7) over EDITING with no significant change in grade-specific metrics.

We also report training and inference statistics. For training, we report the number of training updates to convergence, i.e. when the model achieves the best validation perplexity on the development dataset. For inference, we report the average number of actions taken by the model to generate the refined output counts. Each iteration encompasses a reposition operation followed by an insertion applied to the all the tokens in the input sequence in parallel. CL reduces the average number of actions needed to generate outputs compared to EDITING, while taking only $\sim$ 2K more updates during training than From Input. These results show that our roll-in policy, EDITING and the curriculum play a complementary role in improving training for editing.

Abstractive Summarization On the Abstractive Summarization task (Table 4), EDITCL achieves the best performance across the board compared to alternative training strategies for EDITOR with gain of upto $\sim$ 4 SARI, and $\sim$ 3 ROUGE points. Our proposed approach improves the precision of the deletion operation (DEL-P, +7). It also preserves the tokens from the source sequence that are present in the reference suggested by the improvement in KEEP-F1(+3.9) over the EDITOR (From Reference) model.

For completeness, we also compare our approach with systems trained in prior work: (1) ILP (Clarke and Lapata, 2008), an integer linear programing approach for deletion-based compression, (2) T3 (Cohn and Lapata, 2008), a tree transducer-based model for abstractive compression, (3) SEQ2SEQ (Filippova et al., 2015), a neural network model for deletion-based compression, (4) NAMAS (Rush et al., 2015), a neural model for abstractive compression and summarization and (5) FELIX (Mallinson et al., 2020), a non-autoregressive approach to text editing. We use the outputs provided by Toutanova et al. (2016) for [1-4] and Mallinson et al. (2020) for [5]. We endeavored to make the comparison as fair as possible[5], but it is not possible to have a fully controlled comparison. In particular, FELIX is trained on uncased data and generates uncased outputs, while we train and evaluate our models with truecasing.

When evaluated using our pipeline, our training strategy applied to generic NAR models achieve scores that are on par with, or better than, those of dedicated summarization models (Table 5). However, this evaluation penalizes FELIX as it is trained to address the simpler problem of sum-

[4]As the order of the training samples as governed by our curriculum strategy will be same for From Input, EDITING, we only report results over the best roll-in strategy.

[5]We detokenized and manually checked the outputs from Mallinson et al. (2020) and corrected for de-tokenization errors such as "1. 23" to "1.23" and "wanda 's" to "wanda's".

Model	SARI				ARI-based		Training Updates	Inference action/sample
	keep-F1	add-F1	del-P	combined	PCC	% ARI-Acc		
AR	66.2 ±0.3	4.4 ±0.3	43.4 ±1.4	38.0 ±0.5	0.716 ±0.004	34.5 ±0.4	-	-
dual-path roll-in								
FROM REFERENCE	66.1 ±0.2	2.2 ±0.2	45.5 ±1.2	37.9 ±0.4	0.656 ±0.003	29.7 ±0.2	50K	1.175
FROM INPUT	66.5 ±0.1	3.6 ±0.2	49.3 ±0.5	39.8 ±0.2	0.733 ±0.003	37.7 ±0.4	10K	2.669
EDITING	66.1 ±0.2	**5.2** ±0.1	51.7 ±0.2	41.0 ±0.1	0.745 ±0.005	39.7 ±0.2	6K	2.161
EDITCL	66.8 ±0.2	4.9 ±0.2	**53.3** ±0.4	**41.7** ±0.3	**0.747** ±0.004	**39.8** ±0.3	12K	1.802

Table 3: Results on the Newsela-Grade test dataset for Controllable Simplification: our proposed framework, EDITCL, achieves the best performance on SARI and ARI-based metrics across the board.

Model	SARI				Rouge-L		
	keep-F1	add-F1	del-P	combined	P	R	F1
AR	20.0	1.7	58.5	26.8	35.6	30.1	32.1
dual-path roll-in							
FROM REFERENCE	49.5	3.7	58.8	37.3	54.2	70.1	60.8
FROM INPUT	45.5	3.6	61.4	36.8	52.8	63.4	57.2
EDITING	54.7	4.1	62.9	40.6	55.9	**74.6**	63.6
EDITCL	54.4	4.4	65.5	41.4	56.1	74.0	**63.8**

Table 4: Results on the Summarization dataset: EDITCL improves ROUGE-F1 and SARI over EDITOR.

marization on uncased text. On lower-cased outputs, our best model falls behind FELIX by 1.7 ROUGE points. However, FELIX has about twice as many parameters as our model and benefits from BERT pre-training (Devlin et al., 2019). As a result, this comparison confirms the promise of our approach overall.

Model	Rouge-L		
	P	R	F1
ILP (Clarke and Lapata, 2008)	**60.6**	63.2	60.6
T3 (Cohn and Lapata, 2008)	48.3	20.0	26.8
NAMAS (Rush et al., 2015)	48.8	55.2	51.5
SEQ2SEQ (Filippova et al., 2015)	57.6	51.5	53.1
FELIX (Mallinson et al., 2020)	53.7	58.1	55.5
EDITCL	56.1	**74.0**	**63.8**
FELIX (LC)	65.3	71.5	**67.8**
EDITCL (LC)	57.7	77.2	66.1

Table 5: Comparison to prior work on Summarization dataset: Our approach outperforms all the baselines in ROUGE-L (F1). LC:lower-cased.

6 Analysis

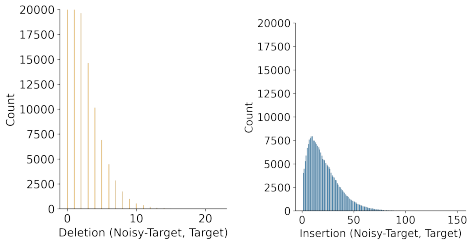
We conduct further experiments to better understand the factors that help our training strategies improve editing quality.

6.1 Impact of EDITING roll-in

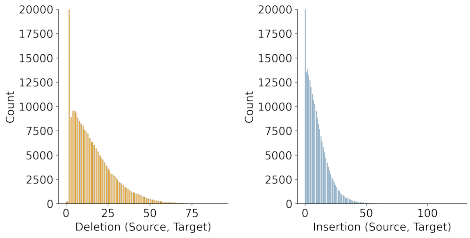
First, we seek to measure whether our approach has the intended effect of bridging the gap between training and test for editing tasks. Figure 3 shows the distribution of oracle insertion and deletions observed when (a) training with EDITOR ś default roll-in policy; (b) refining an original input sequence and (c) exposed to the model with our EDITING roll-in policy for Controllable TS. The plots show that with the default learning policy of the Editor model, the model doesn't learn to perform complex deletion operation at inference time. By contrast, our proposed roll-in exposes the model to the distribution that has higher overlap with the inference distribution as as well as additional intermediate sequences that encourages exploration during training.

6.2 Impact of Curriculum Controlled roll-out

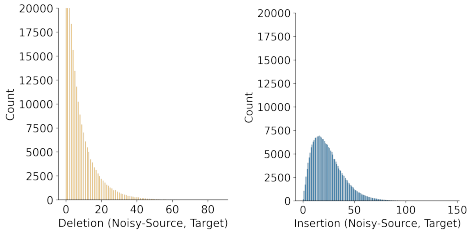
Training Dynamics To verify that curriculum learning helps our model better exploit its training data, we train EDITOR on $x\% \in [0, 100]$ of the data, and compare using random samples with samples ranked by increasing edit distance. Figure 4 shows the number of updates to convergence on the development dataset for controllable simplification with/without CL. Training converges early (70 iterations only) on 13% of the easiest



(a) EDITOR ś roll-in (Training)



(b) Inference Distribution



(c) EDITING roll-in (Training)

Figure 3: Distribution of Oracle Edit Operations (*Insertions*/*Deletions*) observed on Controllable TS. Our proposed roll-in policyś distribution of edit operations is closer to the inference distribution, while enabling exploration during training.

samples with oracle edit distance between the input and the output sequence $<= 2$. This supports the hypothesis that despite adding noise, our approach yields easier examples to train on. The order in which samples are presented matters, as adding batches with larger edit distance ($> 63\%$ data) without maintaining the order of the samples converges early. By contrast, the curriculum pacing function adds samples in order of increasing difficulty, allowing the model sufficient training time to learn from new samples while improving overall performance across metrics.

We also report the learning curves when training EDITOR on the Newsela dataset in Figure 5. Training with curriculum reduces the overall loss consistently on the development dataset, leading to better generalization.

Ranking Criteria We compare the edit-distance (EDITCL) with other curriculum criteria in Table 6 where the order of examples is a) random, b) controlled by the length ratio between source and target sequence (Length Ratio), c) governed by the difference between the source and target grade levels (Grade Difference). Our proposed criterion outperforms both task-specific (`Grade Difference`) and task-agnostic criteria (`Length Ratio`) on the Newsela Grade development set across all the metrics. `Length Ratio` achieves better correlation with Edit distance than `Grade Difference` which is also reflected by its performance (SARI: +0.3, PCC: 0.032, ARI: 0.7) on the Controllable Simplification task. This might reflect the fact that higher grade differences do not necessarily require more edits to be performed, for instance when the sentence to be simplified is already relatively simple. These mismatches do not occur when the edit distance itself is used as the sample difficulty criterion.

Criteria	SARI	PCC	% ARI-Acc	Corr.
Random	40.7	0.749	38.6	-
Length Ratio	41.0	0.762	39.0	0.26
Grade Difference	40.7	0.730	38.3	0.19
EDITCL	42.0	0.758	39.6	1.00
- EDITING roll-in	40.1	0.734	37.8	-
- CL	41.2	0.742	39.3	-

Table 6: On Newsela-grade dev dataset: Using Edit distance as the difficulty criteria improves over both task-specific (Grade Difference) and task-agnostic (Length ratio) criteria. Our proposed EDITING roll-in and curriculum-controlled roll-out provides complementary advantages to the model training.

Complementarity of roll-in and roll-out design We report the performance of the `From Input` model, when trained with curriculum only without the EDITING policy, i.e. `EDITCL- EDITING` in the same Table 6. Both `EDITING` roll-in and curriculum controlled roll-out provides complementary advantages to the model training as removing either results in the drop in performance across all the metrics for controllable TS. However, we observe larger drop in the scores when we do not apply the EDITING policy which shows that our proposed roll-in policy is necessary to reap the benefits of curriculum learning.

7 Related Work

NAR models They have been used to enable parallel generation of output tokens for Machine translation. (Stern et al., 2019; Chan et al., 2020; Xu and Carpuat, 2021). Mallinson et al. (2020) design a custom multi-step non-autoregressive edit-based model for sequence editing where each source to-

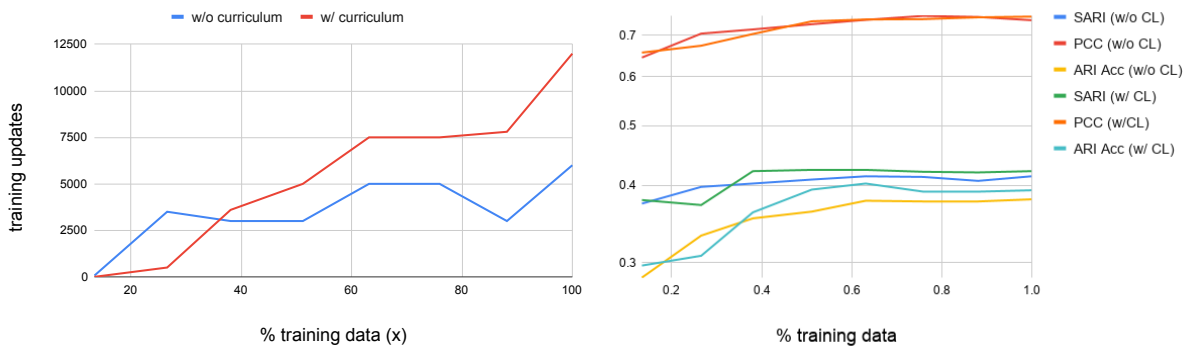


Figure 4: The sample order during training matters as training without curriculum on the same amount of data ($>= 40\%$) converges early (plot on the left) and to lower performance across all metrics (plot on the right) relative to training with curriculum using the same data.

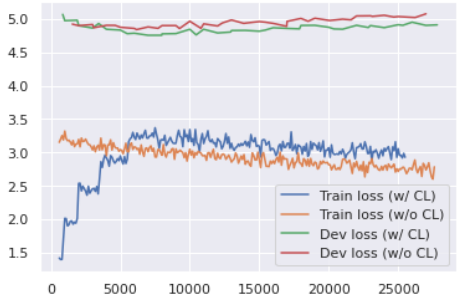


Figure 5: Training with curriculum reduces the loss on the development dataset leading to better generalization on Controllable TS.

ken is first tagged to represent the type of edit operation to be performed and then a secondary model is used to in-fill new tokens. The tagging and editing models are trained independently. By contrast, we propose approaches to adapt NAR models designed for MT for these tasks and train an end-to-end model to generate an edited sequence.

Curriculum Learning for Sequence Refinement While curriculum learning has been applied to many tasks such as MT (Haffari, 2009; Platanios et al., 2019; Kumar et al., 2019), sentiment analysis (Sido and Konopík, 2019), natural language understanding (Xu et al., 2020), reading comprehension (Tay et al., 2019), their application to sequence refinement tasks has not been explored yet. Various strategies have been proposed to control the sample difficulty like n-gram frequency (Haffari, 2009; Platanios et al., 2019), token rarity, and sentence length (Liu et al., 2020). Chang et al. (2021) use Levenshtein edit distance as a sample difficulty criteria to order the samples for the task of data-to-text generation where the training model uses an AR seq2seq model. Instead, we focus on edit distance as a sample difficulty criteria that is directly tied to the training oracle and model design.

Roll-in policies There has been a plethora of work in the Imitation learning landscape on algorithms that strike a balance between learned and expert roll-in policies (Ross et al., 2011; Venkatraman et al., 2015; Chang et al., 2015). However, large differences in expert and learner's policy action can hurt performance (Brantley et al., 2019; He et al., 2012; Leblond et al., 2018). In our work, we propose to roll-in with noised states instead, so that the model can be exposed to mimic expert demonstrations from states that the model is more likely to encounter during inference.

8 Conclusion

This paper introduced two complementary strategies to address undertraining and poor generalization when adapting NAR models to editing tasks: 1) a new roll-in policy that generates intermediate sequences that the model is likely to encounter during inference and 2) a curriculum to control the difficulty of the roll-out policy which estimates the cost-to-go from the roll-in sequences to the desired output sequences, throughout training. Together, these strategies improve output quality consistently on controllable simplification and abstractive summarization. These results open space for further research to evaluate the potential of this approach for other editing tasks (e.g., post editing, style transfer), and to further tailor imitation learning policies and curriculum design to these tasks.

Acknowledgments

We thank Eleftheria Briakou, Khanh Nguyen, Kianté Brantley, the members of the CLIP lab at UMD, and the anonymous ARR reviewers for their helpful and constructive comments.

References

Sweta Agrawal and Marine Carpuat. 2019. Controlling text complexity in neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1549–1564.

Sweta Agrawal, Weijia Xu, and Marine Carpuat. 2021. A non-autoregressive edit-based approach to controllable text simplification. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3757–3769, Online. Association for Computational Linguistics.

Fernando Alva-Manchego, Joachim Bingel, Gustavo Paetzold, Carolina Scarton, and Lucia Specia. 2017. Learning how to simplify from explicit labeling of complex-simplified text pairs. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 295–305, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2019. Parallel iterative edit models for local sequence transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4251–4261.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations (ICLR)*.

Kiante Brantley, Kyunghyun Cho, Hal Daumé, and Sean Welleck. 2019. Non-monotonic sequential text generation. In *Proceedings of the 2019 Workshop on Widening NLP*, pages 57–59, Florence, Italy. Association for Computational Linguistics.

William Chan, Chitwan Saharia, Geoffrey Hinton, Mohammad Norouzi, and Navdeep Jaitly. 2020. Imputer: Sequence modelling via imputation and dynamic programming. In *International Conference on Machine Learning*, pages 1403–1413. PMLR.

Raman Chandrasekar and Bangalore Srinivas. 1997. Automatic induction of rules for text simplification. *Knowledge-Based Systems*, 10(3):183–190.

Ernie Chang, Hui-Syuan Yeh, and Vera Demberg. 2021. Does the order of training samples matter? improving neural data-to-text generation with curriculum learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 727–733, Online. Association for Computational Linguistics.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. 2015. Learning to search better than your teacher. In *International Conference on Machine Learning*, pages 2058–2066. PMLR.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 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, Manchester, UK. Coling 2008 Organizing Committee.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, 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*, pages 3393–3402, Florence, Italy. Association for Computational Linguistics.

Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Łukasz 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.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6114–6123.

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

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11179–11189. Curran Associates, Inc.

Junliang Guo, Xu Tan, Linli Xu, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2020. Fine-tuning by curriculum learning for non-autoregressive neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7839–7846.

Gholamreza Haffari. 2009. *Machine learning approaches for dealing with limited bilingual training data in statistical machine translation*. Ph.D. thesis, Simon Fraser University.

He He, Jason Eisner, and Hal Daume. 2012. Imitation learning by coaching. *Advances in Neural Information Processing Systems*, 25:3149–3157.

Michael Heilman, Kevyn Collins-Thompson, and Maxine Eskenazi. 2008. An analysis of statistical models and features for reading difficulty prediction. In *Proceedings of the third workshop on innovative use of NLP for building educational applications*, pages 71–79. Association for Computational Linguistics.

Di Jin, Zhijing Jin, Zhiting Hu, Olga Vechtomova, and Rada Mihalcea. 2020. Deep learning for text style transfer: A survey. *arXiv preprint arXiv:2011.00416*.

Gaurav Kumar, George Foster, Colin Cherry, and Maxim Krikun. 2019. Reinforcement learning based curriculum optimization for neural machine translation. 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 2054–2061, Minneapolis, Minnesota. Association for Computational Linguistics.

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations*.

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. 2018. Searnn: Training rnns with global-local losses. In *International Conference on Learning Representations*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182.

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

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Jinglin Liu, Yi Ren, Chen Zhang Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Task-level curriculum learning for non-autoregressive neural machine translation.

Xuebo Liu, Houtim Lai, Derek F Wong, and Lidia S Chao. 2020. Norm-based curriculum learning for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 427–436.

Jonathan Mallinson, Aliaksei Severyn, Eric Malmi, and Guillermo Garrido. 2020. FELIX: Flexible text editing through tagging and insertion. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1244–1255, Online. Association for Computational Linguistics.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5057–5068.

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabás Póczos, and Tom M Mitchell. 2019. Competence-based curriculum learning for neural machine translation. In *NAACL-HLT (1)*.

Lihua Qian, Hao Zhou, Yu Bao, Mingxuan Wang, Lin Qiu, Weinan Zhang, Yong Yu, and Lei Li. 2020. Glancing transformer for non-autoregressive neural machine translation. *arXiv e-prints*, pages arXiv–2008.

Stephane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy. PMLR.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings.

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.

Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-autoregressive machine translation with latent alignments. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1098–1108, Online. Association for Computational Linguistics.

Carolina Scarton and Lucia Specia. 2018. Learning Simplifications for Specific Target Audiences. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 712–718.

R. J. Senter and Edgar A Smith. 1967. Automated readability index. Technical report, CINCINNATI UNIV OH.

Jakub Sido and Miloslav Konopík. 2019. Curriculum learning in sentiment analysis. In *International Conference on Speech and Computer*, pages 444–450. Springer.

Michel Simard, Cyril Goutte, and Pierre Isabelle. 2007. Statistical phrase-based post-editing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 508–515.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985. PMLR.

Yi Tay, Shuohang Wang, Anh Tuan Luu, Jie Fu, Minh C Phan, Xingdi Yuan, Jinfeng Rao, Siu Cheung Hui, and Aston Zhang. 2019. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4922–4931.

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, Austin, Texas. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. 2015. Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang. 2020. Curriculum learning for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6095–6104.

Wei Xu, Chris Callison-Burch, and Courtney Napoles. 2015. Problems in current text simplification research: New data can help. *Transactions of the Association for Computational Linguistics*, 3:283–297.

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

Weijia Xu and Marine Carpuat. 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.

Ziyu Yao, Frank F. Xu, Pengcheng Yin, Huan Sun, and Graham Neubig. 2021. Learning structural edits via incremental tree transformations. In *International Conference on Learning Representations*.

Xuan Zhang, Gaurav Kumar, Huda Khayrallah, Kenton Murray, Jeremy Gwinnup, Marianna J Martindale, Paul McNamee, Kevin Duh, and Marine Carpuat. 2018. An empirical exploration of curriculum learning for neural machine translation.

Yikai Zhou, Baosong Yang, Derek F. Wong, Yu Wan, and Lidia S. Chao. 2020. Uncertainty-aware curriculum learning for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6934–6944, Online. Association for Computational Linguistics.

A Results on Development set

Model	SARI				ARI-based		Inference action/sample
	keep-F1	add-F1	del-P	combined	PCC	ARI-Acc	
AR	0.653	0.043	0.456	0.384	0.711	0.349	-
dual-path roll-in							
FROM REFERENCE	0.648	0.021	0.454	0.374	0.645	0.285	1.188
FROM INPUT	0.660	0.035	0.510	0.402	0.727	0.368	2.545
EDITING	0.657	0.049	0.530	0.412	0.742	0.393	2.071
EDITCL	0.662	0.043	0.556	0.420	0.758	0.397	1.771

Table 7: Results on the Newsela-Grade development dataset for Controllable Simplification: our proposed framework, EDITCL, achieves the best performance on SARI and ARI-based metrics across the board.

B Impact of Noise

Figure 6 shows that adding noise to the training samples smoothes the distribution across training instances by creating intermediate sequences that have relatively lower (or higher) overall edit distance with the reference sequence compared to the original input sequence.

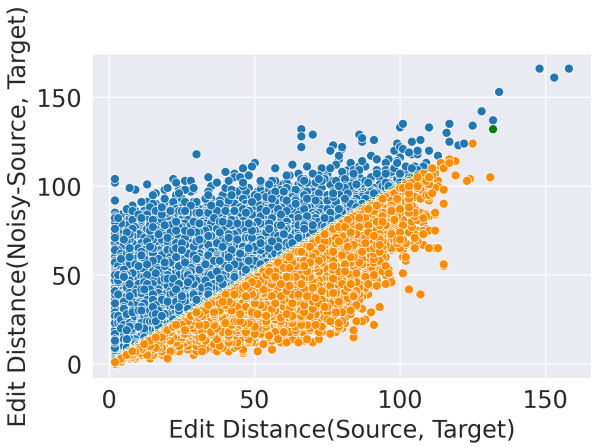
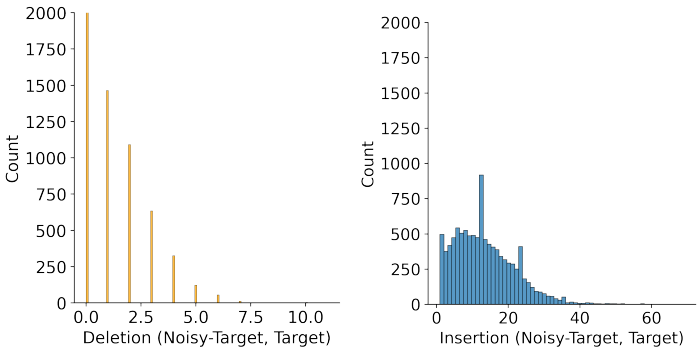
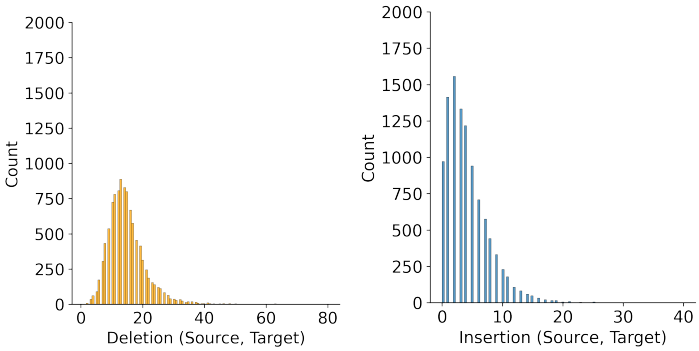


Figure 6: Adding noise to the source increases (*higher*) or decreases (*lower*) the edit distance uniformly across samples for Controllable TS.

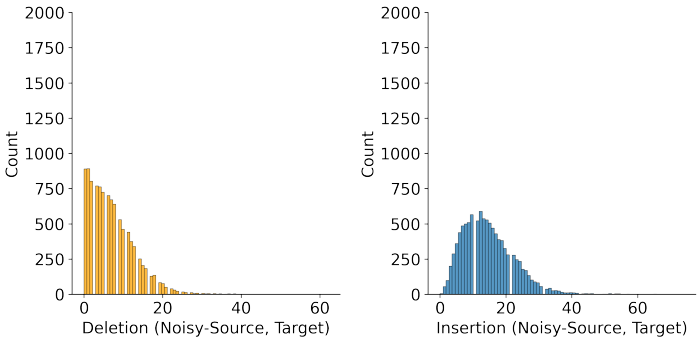
C Oracle Edit Distribution for Summarization



(a) EDITOR ś roll-in (Training)



(b) Inference Distribution



(c) EDITING roll-in (Training)

Figure 7: Distribution of Oracle Edit Operations (*Insertions*/*Deletions*) observed on Abstractive Summarization. Our proposed roll-in policyś distribution of edit operations is closer to the inference distribution, while enabling exploration via generated intermediate sequences during training.