EDITOR: An Edit-Based Transformer with Repositioning for Neural Machine Translation with Soft Lexical Constraints

Weijia Xu
University of Maryland
weijia@cs.umd.edu

Marine Carpuat
University of Maryland
marine@cs.umd.edu

Abstract

We introduce an **Edi**t-Based **T**ransf**O**rmer with **R**epositioning (EDITOR), which makes sequence generation flexible by seamlessly allowing users to specify preferences in output lexical choice. Building on recent models for non-autoregressive sequence generation (Gu et al., 2019), EDITOR generates new sequences by iteratively editing hypotheses. It relies on a novel reposition operation designed to disentangle lexical choice from word positioning decisions, while enabling efficient oracles for imitation learning and parallel edits at decoding time. Empirically, EDITOR uses soft lexical constraints more effectively than the Levenshtein Transformer (Gu et al., 2019) while speeding up decoding dramatically compared to constrained beam search (Post and Vilar, 2018). EDITOR also achieves comparable or better translation quality with faster decoding speed than the Levenshtein Transformer on standard Romanian-English, English-German, and English-Japanese machine translation tasks.

1 Introduction

Neural machine translation (MT) architectures (Bahdanau et al., 2015; Vaswani et al., 2017) make it difficult for users to specify preferences that could be incorporated more easily in statistical MT models (Koehn et al., 2007) and have been shown to be useful for interactive machine translation (Foster et al., 2002; Barrachina et al., 2009) and domain adaptation (Hokamp and Liu, 2017). Lexical constraints or preferences have previously been incorporated by re-training NMT models with constraints as inputs (Song et al., 2019; Dinu et al., 2019) or with constrained beam search that drastically slows down decoding (Hokamp and Liu, 2017; Post and Vilar, 2018).

In this work, we introduce a translation model that can seamlessly incorporate users' lexical choice preferences without increasing the time and computational cost at decoding time, while being trained on regular MT samples. We apply this model to MT tasks with soft lexical constraints. As illustrated in Figure 1, when decoding with soft lexical constraints, user preferences for lexical choice in the output language are provided as an additional input sequence of target words in any order. The goal is to let users encode terminology, domain, or stylistic preferences in target word usage, without strictly enforcing hard constraints that might hamper NMT's ability to generate fluent outputs.

Our model is an **Edi**t-Based **T**ransf**O**rmer with **R**epositioning (**EDITOR**), which builds on recent progress on non-autoregressive sequence generation (Lee et al., 2018; Ghazvininejad et al., 2019).[1] Specifically, the Levenshtein Transformer (Gu et al., 2019) showed that iteratively refining output sequences via insertions and deletions yields a fast and flexible generation process for MT and automatic post-editing tasks. EDITOR replaces the deletion operation with a novel reposition operation to disentangle lexical choice from reordering decisions. As a result, EDITOR exploits lexical constraints more effectively and efficiently than the Levenshtein Transformer, as a single reposition operation can subsume a sequence of deletions and insertions. To train EDITOR via imitation learning, the reposition operation is defined to preserve the ability to use the Levenshtein edit distance (Levenshtein, 1966) as an efficient oracle. We also introduce a dual-path roll-in policy, which lets the reposition and deletion models learn to refine their respective outputs more effectively.

Experiments on Romanian-English, English-German, and English-Japanese MT show that EDITOR achieves comparable or better translation quality with faster decoding speed than

[1] https://github.com/Izecson/fairseq-editor.

Transactions of the Association for Computational Linguistics, vol. 9, pp. 311–328, 2021. https://doi.org/10.1162/tacl_a_00368
Action Editor: Francois Yvon. Submission batch: 7/2020; Revision batch: 10/2020; Published 3/2021.

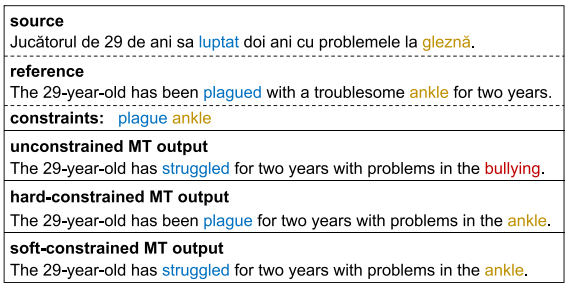


Figure 1: Romanian to English MT example. Unconstrained MT incorrectly translates ‘‘gleznă’’ to ‘‘bullying’’. Given constraint words ‘‘plague’’ and ‘‘ankle’’, soft-constrained MT correctly uses ‘‘ankle’’ and avoids disfluencies introduced by using ‘‘plague’’ as a hard constraint in its exact form.

the Levenshtein Transformer (Gu et al., 2019) on the standard MT tasks and exploits soft lexical constraints better: It achieves significantly better translation quality and matches more constraints with faster decoding speed than the Levenshtein Transformer. It also drastically speeds up decoding compared with lexically constrained decoding algorithms (Post and Vilar, 2018). Furthermore, results highlight the benefits of soft constraints over hard ones—EDITOR with soft constraints achieves translation quality on par or better than both EDITOR and Levenshtein Transformer with hard constraints (Susanto et al., 2020).

2 Background

Non-Autoregressive MT Although autoregressive models that decode from left-to-right are the *de facto* standard for many sequence generation tasks (Cho et al., 2014; Chorowski et al., 2015; Vinyals and Le, 2015), non-autoregressive models offer a promising alternative to speed up decoding by generating a sequence of tokens in parallel (Gu et al., 2018; van den Oord et al., 2018; Ma et al., 2019). However, their output quality suffers due to the large decoding space and strong independence assumptions between target tokens (Ma et al., 2019; Wang et al., 2019). These issues have been addressed via partially parallel decoding (Wang et al., 2018; Stern et al., 2018) or multi-pass decoding (Lee et al., 2018; Ghazvininejad et al., 2019; Gu et al., 2019). This work adopts multi-pass decoding, where the model generates the target sequences by iteratively editing the outputs from previous iterations. Edit operations such as substitution (Ghazvininejad et al., 2019) and insertion-deletion (Gu et al., 2019) have reduced the quality gap between non-autoregressive and autoregressive models. However, we argue that these operations limit the flexibility and efficiency of the resulting models for MT by entangling lexical choice and reordering decisions.

Reordering vs. Lexical Choice EDITOR’s insertion and reposition operations connect closely with the long-standing view of MT as a combination of a translation or lexical choice model, which selects appropriate translations for source units given their context, and reordering model,which encourages the generation of a target sequence order appropriate for the target language. This view is reflected in architectures ranging from the word-based IBM models (Brown et al., 1990), sentence-level models that generate a bag of target words that is reordered to construct a target sentence (Bangalore et al., 2007), or the Operation Sequence Model (Durrani et al., 2015; Stahlberg et al., 2018), which views translation as a sequence of translation and reordering operations over bilingual minimal units. By contrast, autoregressive NMT models (Bahdanau et al., 2015; Vaswani et al., 2017) do not explicitly separate lexical choice and reordering, and previous non-autoregressive models break up reordering into sequences of other operations. This work introduces the reposition operation, which makes it possible to move words around during the refinement process, as reordering models do. However, we will see that reposition differs from typical reordering to enable efficient oracles for training via imitation learning, and parallelization of edit operations at decoding time (Section 3).

MT with Soft Lexical Constraints NMT models lack flexible mechanisms to incorporate users preferences in their outputs. Lexical constraints have been incorporated in prior work via 1) constrained training where NMT models are trained on parallel samples augmented with constraint target phrases in both the source and target sequences (Song et al., 2019; Dinu et al., 2019), or 2) constrained decoding where beam search is modified to include constraint words or phrases in the output (Hokamp and Liu, 2017; Post and Vilar, 2018). These mechanisms can incorporate domain-specific knowledge and lexicons which is particularly helpful in low-resource cases (Arthur et al., 2016; Tang et al., 2016). Despite their success at domain adaptation for MT

(Hokamp and Liu, 2017) and caption generation (Anderson et al., 2017), they suffer from several issues: Constrained training requires building dedicated models for constrained language generation, while constrained decoding adds significant computational overhead and treats all constraints as hard constraints which may hurt fluency. In other tasks, various constraint types have been introduced by designing complex architectures tailored to specific content or style constraints (Abu Sheikha and Inkpen, 2011; Mei et al., 2016), or via segment-level ''side-constraints'' (Sennrich et al., 2016a; Ficler and Goldberg, 2017; Agrawal and Carpuat, 2019), which condition generation on users' stylistic preferences, but do not offer fine-grained control over their realization in the output sequence. We refer the reader to Yvon and Abdul Rauf, (2020) for a comprehensive review of the strengths and weaknesses of current techniques to incorporate terminology constraints in NMT.

Our work is closely related to Susanto et al. (2020)'s idea of applying the Levenshtein Transformer to MT with hard terminology constraints. We will see that their technique can directly be used by EDITOR as well (Section 3.3), but this does not offer empirical benefits over the default EDITOR model (Section 4.3).

3 Approach

3.1 The EDITOR Model

We cast both constrained and unconstrained language generation as an iterative sequence refinement problem modeled by a Markov Decision Process $(\mathcal{Y}, \mathcal{A}, \mathcal{E}, \mathcal{R}, \boldsymbol{y}^0)$, where a state $\boldsymbol{y}$ in the state space $\mathcal{Y}$ corresponds to a sequence of tokens $\boldsymbol{y} = (y_1, y_2, \ldots, y_L)$ from the vocabulary $\mathcal{V}$ up to length L, and $\boldsymbol{y}^0 \in \mathcal{Y}$ is the initial sequence For standard sequence generation tasks, $\boldsymbol{y}^0$ is the empty sequence $(\langle s \rangle, \langle /s \rangle)$. For lexically constrained generation tasks, $\boldsymbol{y}^0$ consists of the words to be used as constraints $(\langle s \rangle, c_1, \ldots, c_m, \langle /s \rangle)$.

At the k-th decoding iteration, the model takes as input $\boldsymbol{y}^{k-1}$, the output from the previous iteration, chooses an action $\boldsymbol{a}^k \in \mathcal{A}$ to refine the sequence into $\boldsymbol{y}^k = \mathcal{E}(\boldsymbol{y}^{k-1}, \boldsymbol{a}^k)$, and receives a reward $r^k = \mathcal{R}(\boldsymbol{y}^k)$. The policy π maps the input sequence $\boldsymbol{y}^{k-1}$ to a probability distribution $P(\mathcal{A})$ over the action space $\mathcal{A}$. Our model is based on the Transformer encoder-decoder (Vaswani et al. 2017) and we extract the decoder representations $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n)$ to make the policy predictions.

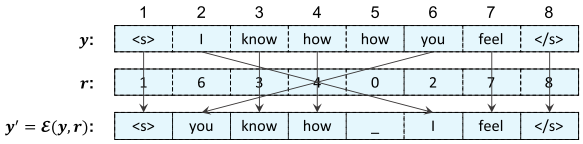


Figure 2: Applying the reposition operation $\boldsymbol{r}$ to input $\boldsymbol{y}$: $r_i > 0$ is the 1-based index of token y'_i in the input sequence; y_i is deleted if $r_i = 0$.

Each refinement action is based on two basic operations: reposition and insertion.

Reposition For each position i in the input sequence $\boldsymbol{y}_{1\ldots n}$, the reposition policy $\pi_{rps}(r \mid i, \boldsymbol{y})$ predicts an index $r \in [0, n]$: If $r > 0$, we place the r-th input token y_r at the i-th output position, otherwise we delete the token at that position (Figure 2). We constrain $\pi_{rps}(1 \mid 1, \boldsymbol{y}) = \pi_{rps}(n \mid n, \boldsymbol{y}) = 1$ to maintain sequence boundaries. Note that reposition differs from typical reordering because 1) it makes it possible to delete tokens, and 2) it places tokens at each position independently, which enables parallelization at decoding time. In principle, the same input token can thus be placed at multiple output positions. However, this happens rarely in practice as the policy predictor is trained to follow oracle demonstrations which cannot contain such repetitions by design.[2]

The reposition classifier gives a categorical distribution over the index of the input token to be placed at each output position:

$$\pi_{rps}(r \mid i, \boldsymbol{y}) = \text{softmax}(\boldsymbol{h}_i \cdot [\boldsymbol{b}, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_n]) \quad (1)$$

where $\boldsymbol{e}_j$ is the embedding of the j-th token in the input sequence, and $\boldsymbol{b} \in \mathbb{R}^{d_{model}}$ is used to predict whether to delete the token. The dot product in the softmax function captures the similarity between the hidden state $\boldsymbol{h}_i$ and each input embedding $\boldsymbol{e}_j$ or the deletion vector $\boldsymbol{b}$.

Insertion Following Gu et al. (2019), the insertion operation consists of two phases: (1) *placeholder insertion:* Given an input sequence $\boldsymbol{y}_{1\ldots n}$, the placeholder predictor $\pi_{plh}(p \mid i, \boldsymbol{y})$ predicts the number of placeholders $p \in [0, K_{max}]$ to be inserted between two neighboring tokens (y_i, y_{i+1});[3] (2) *token prediction:* Given the output of the placeholder predictor, the token predictor

[2]Empirically, fewer than 1% of tokens are repositioned to more than one output position.

[3]In our implementation, we set $K_{max} = 255$.

$\pi_{tok}(t \mid i, \boldsymbol{y})$ replaces each placeholder with an actual token.

The Placeholder Insertion Classifier gives a categorical distribution over the number of placeholders to be inserted between every two consecutive positions:

$$\pi_{plh}(p \mid i, \boldsymbol{y}) = \text{softmax}([\boldsymbol{h}_i\,;\,\boldsymbol{h}_{i+1}] \cdot \boldsymbol{W}^{plh}) \quad (2)$$

where $\boldsymbol{W}^{plh} \in \mathbb{R}^{(2d_{model}) \times (K_{max}+1)}$.

The Token Prediction Classifier predicts the identity of each token to fill in each placeholder:

$$\pi_{tok}(t \mid i, \boldsymbol{y}) = \text{softmax}(\boldsymbol{h}_i \cdot \boldsymbol{W}^{tok}) \quad (3)$$

where $\boldsymbol{W}^{tok} \in \mathbb{R}^{d_{model} \times |\mathcal{V}|}$.

Action Given an input sequence $\boldsymbol{y}_{1\ldots n}$, an action consists of repositioning tokens, inserting and replacing placeholders. Formally, we define an action as *a sequence* of reposition ($\boldsymbol{r}$), placeholder insertion ($\boldsymbol{p}$), and token prediction ($\boldsymbol{t}$) operations: $\boldsymbol{a} = (\boldsymbol{r}, \boldsymbol{p}, \boldsymbol{t})$. $\boldsymbol{r}$, $\boldsymbol{p}$, and $\boldsymbol{t}$ are applied in this order to adjust non-empty initial sequences via reposition before inserting new tokens. Each of $\boldsymbol{r}$, $\boldsymbol{p}$, and $\boldsymbol{t}$ consists of a set of basic operations that can be applied *in parallel*:

$$\begin{aligned}\boldsymbol{r} &= \{r_1, \ldots, r_n\}\\ \boldsymbol{p} &= \{p_1, \ldots, p_{m-1}\}\\ \boldsymbol{t} &= \{t_1, \ldots, t_l\}\end{aligned}$$

where $m = \sum_i^n \mathbb{I}(r_i > 0)$ and $l = \sum_i^{m-1} p_i$. We define the policy as

$$\pi(\boldsymbol{a}|\boldsymbol{y}) = \prod_{r_i \in \boldsymbol{r}} \pi_{rps}(r_i \mid i, \boldsymbol{y}) \cdot \prod_{p_i \in \boldsymbol{p}} \pi_{plh}(p_i \mid i, \boldsymbol{y'}) \cdot \prod_{t_i \in \boldsymbol{t}} \pi_{tok}(t_i \mid i, \boldsymbol{y''})$$

with intermediate outputs $\boldsymbol{y'} = \mathcal{E}(\boldsymbol{y}, \boldsymbol{r})$ and $\boldsymbol{y''} = \mathcal{E}(\boldsymbol{y'}, \boldsymbol{p})$.

3.2 Dual-Path Imitation Learning

We train EDITOR using imitation learning (Daumé III et al., 2009; Ross et al., 2011; Ross and Bagnell, 2014) to efficiently explore the space of valid action sequences that can reach a reference translation. The key idea is to construct a *roll-in* policy π^{in} to generate sequences to be refined and a *roll-out* policy π^{out} to estimate cost-to-go for all possible actions given each input sequence. The model is trained to choose actions that minimize the cost-to-go estimates. We use a search-based oracle policy π^* as the roll-out policy and train the model to imitate the optimal actions chosen by the oracle.

Formally, $\boldsymbol{d}_{\pi^{in}_{rps}}$ and $\boldsymbol{d}_{\pi^{in}_{ins}}$ denote the distributions of sequences induced by running the roll-in policies π^{in}_{rps} and π^{in}_{ins} respectively. We update the model policy $\pi = \pi_{rps} \cdot \pi_{plh} \cdot \pi_{tok}$ to minimize the expected cost $\mathcal{C}(\pi\,;\,\boldsymbol{y}, \pi^*)$ by comparing the model policy against the cost-to-go estimates under the oracle policy π^* given input sequences $\boldsymbol{y}$:

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{y}_{rps} \sim \boldsymbol{d}_{\pi^{in}_{rps}}} \left[\mathcal{C}(\pi_{rps}\,;\,\boldsymbol{y}_{rps}, \pi^*)\right] +\\ &\mathbb{E}_{\boldsymbol{y}_{ins} \sim \boldsymbol{d}_{\pi^{in}_{ins}}} \left[\mathcal{C}(\pi_{plh}, \pi_{tok}\,;\,\boldsymbol{y}_{ins}, \pi^*)\right]\end{aligned} \quad (4)$$

The cost function compares the model vs. oracle actions. As prior work suggests that cost functions close to the cross-entropy loss are better suited to deep neural models than the squared error (Leblond et al., 2018; Cheng et al., 2018), we define the cost function as the KL divergence between the action distributions given by the model policy and by the oracle (Welleck et al., 2019):

$$\begin{aligned}&\mathcal{C}(\pi\,;\,\boldsymbol{y}, \pi^*)\\ =&D_{\text{KL}}\left[\,\pi^*(\boldsymbol{a} \mid \boldsymbol{y}, \boldsymbol{y}^*) || \,\pi(\boldsymbol{a} \mid \boldsymbol{y})\right]\\ =&\mathbb{E}_{\boldsymbol{a} \sim \pi^*(\boldsymbol{a} \mid \boldsymbol{y}, \boldsymbol{y}^*)}\left[-\log \pi(\boldsymbol{a} \mid \boldsymbol{y})\right] + const.\end{aligned} \quad (5)$$

where the oracle has additional access to the reference sequence $\boldsymbol{y}^*$. By minimizing the cost function, the model learns to imitate the oracle policy without access to the reference sequence.

Next, we describe how the reposition operation is incorporated in the roll-in policy (Section 3.2.1) and the oracle roll-out policy (Section 3.2.2).

3.2.1 Dual-Path Roll-in Policy

As shown in Figure 3, the roll-in policies π^{in}_{ins} and π^{in}_{rps} for the reposition and insertion policy predictors are stochastic mixtures of the noised reference sequences and the output sequences sampled from their corresponding dual policy predictors. Figure 4 shows an example for creating the roll-in sequences: We first create the initial sequence $\boldsymbol{y}^0$ by applying random word dropping (Gu et al., 2019) and random word shuffle (Lample et al., 2018) with probability of 0.5 and maximum

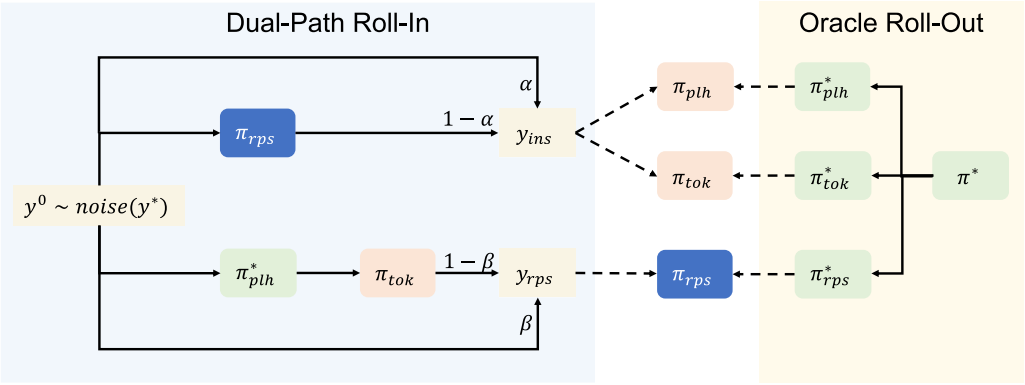


Figure 3: Our dual-path imitation learning process uses both the reposition and insertion policies during roll-in so that they can be trained to refine each other's outputs: Given an initial sequence $\boldsymbol{y}^0$, created by noising the reference $\boldsymbol{y}^*$, the roll-in policy stochastically generates intermediate sequences $\boldsymbol{y}_{ins}$ and $\boldsymbol{y}_{rps}$ via reposition and insertion respectively. The policy predictors are trained to minimize the costs of reaching $\boldsymbol{y}^*$ from $\boldsymbol{y}_{ins}$ and $\boldsymbol{y}_{rps}$ estimated by the oracle policy π^*.

shuffle distance of 3 to the reference sequence $\boldsymbol{y}^*$, and produce the roll-in sequences for each policy predictor as follows:

1. **Reposition:** The roll-in policy π^{in}_{rps} is a stochastic mixture of the initial sequence $\boldsymbol{y}^0$ and the output sequence by applying one iteration of the oracle placeholder insertion policy $\boldsymbol{p}^* \sim \pi^*$ and the model's token prediction policy $\tilde{\boldsymbol{t}} \sim \pi_{tok}$ to $\boldsymbol{y}^0$:

$$\boldsymbol{d}_{\pi^{in}_{rps}} = \begin{cases} \boldsymbol{y}^0, & \text{if } u < \beta \\ \mathcal{E}(\mathcal{E}(\boldsymbol{y}^0, \boldsymbol{p}^*), \tilde{\boldsymbol{t}}), & \text{otherwise} \end{cases} \quad (6)$$

   where the mixture factor $\beta \in [0, 1]$ and random variable $u \sim \text{Uniform}(0, 1)$.

2. **Insertion:** The roll-in policy π^{in}_{ins} is a stochastic mixture of the initial sequence $\boldsymbol{y}^0$ and the output sequence by applying one iteration of the model's reposition policy $\tilde{\boldsymbol{r}} \sim \pi_{rps}$ to $\boldsymbol{y}^0$:

$$\boldsymbol{d}_{\pi^{in}_{ins}} = \begin{cases} \boldsymbol{y}^0, & \text{if } u < \alpha \\ \mathcal{E}(\boldsymbol{y}^0, \tilde{\boldsymbol{r}}), & \text{otherwise} \end{cases} \quad (7)$$

   where the mixture factor $\alpha \in [0, 1]$ and random variable $u \sim \text{Uniform}(0, 1)$.

While Gu et al. (2019) define roll-in using only the model's insertion policy, we call our approach dual-path because roll-in creates two distinct intermediate sequences using the model's reposition or insertion policy. This makes it possible for the reposition and insertion policy predictors to learn to refine one another's outputs

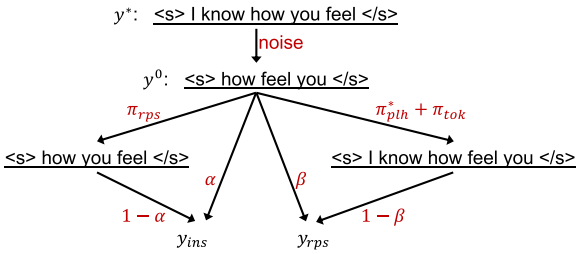


Figure 4: The roll-in sequence for the insertion predictor is a stochastic mixture of the noised reference $\boldsymbol{y}^0$ and the output by applying the model's reposition policy π_{rps} to $\boldsymbol{y}^0$. The roll-in sequence for the reposition predictor is a stochastic mixture of the noised reference $\boldsymbol{y}^0$ and the output by applying the oracle placeholder insertion policy π^*_{plh} and the model's token prediction policy π_{tok} to $\boldsymbol{y}^0$.

during roll-out, mimicking the iterative refinement process used at inference time.[4]

3.2.2 Oracle Roll-Out Policy

Policy Given an input sequence $\boldsymbol{y}$ and a reference sequence $\boldsymbol{y}^*$, the oracle algorithm finds the optimal action to transform $\boldsymbol{y}$ into $\boldsymbol{y}^*$ with the minimum number of basic edit operations:

$$\text{Oracle}(\boldsymbol{y}, \boldsymbol{y}^*) = \arg\min_{\boldsymbol{a}} \text{NumOps}(\boldsymbol{y}, \boldsymbol{y}^* \mid \boldsymbol{a}) \quad (8)$$

The associated oracle policy is defined as:

$$\pi^*(\boldsymbol{a} \mid \boldsymbol{y}, \boldsymbol{y}^*) = \begin{cases} 1, & \text{if } \boldsymbol{a} = \text{Oracle}(\boldsymbol{y}, \boldsymbol{y}^*) \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

[4]Different from the inference process, we generate the roll-in sequences by applying the model's reposition or insertion policy for only one iteration.

Algorithm The reposition and insertion operations used in EDITOR are designed so that the Levenshtein edit distance algorithm (Levenshtein, 1966) can be used as the oracle. The reposition operation (Section 3.1) can be split into two distinct types of operations: (1) deletion and (2) replacing a word with any other word appearing in the input sequence, which is a constrained version of the Levenshtein substitution operation. As a result, we can use dynamic programming to find the optimal action sequence in $O(|\boldsymbol{y}||\boldsymbol{y}^*|)$ time. By contrast, the Levenshtein Transformer restricts the oracle and model to insertion and deletion operations only. While in principle substitutions can be performed indirectly by deletion and reinsertion, our results show the benefits of using the reposition variant of the substitution operation.

3.3 Inference

During inference, we start from the initial sequence $\boldsymbol{y}^0$. For standard sequence generation tasks, $\boldsymbol{y}^0$ is an empty sequence, whereas for lexically constrained generation $\boldsymbol{y}^0$ is a sequence of lexical constraints. Inference then proceeds in the exact same way for constrained and unconstrained tasks. The initial sequence is refined iteratively by applying a sequence of actions $(\boldsymbol{a}^1, \boldsymbol{a}^2, \ldots) = (\boldsymbol{r}^1, \boldsymbol{p}^1, \boldsymbol{t}^1\,;\, \boldsymbol{r}^2, \boldsymbol{p}^2, \boldsymbol{t}^2\,;\, \ldots)$. We greedily select the best action at each iteration given the model policy in Equations (1) to (3). We stop refining if 1) the output sequences from two consecutive iterations are the same (Gu et al., 2019), or 2) the maximum number of decoding steps is reached (Lee et al., 2018; Ghazvininejad et al., 2019).[5]

Incorporating Soft Constraints Although EDITOR is trained without lexical constraints, it can be used seamlessly for MT with constraints without any change to the decoding process except using the constraint sequence as the initial sequence.

Incorporating Hard Constraints We adopt the decoding technique introduced by Susanto et al. (2020) to enforce hard constraints at decoding time by prohibiting deletion operations on constraint tokens or insertions within a multi-token constraints.

[5]Following Stern et al. (2019), we also experiment with adding penalty for inserting ''empty'' placeholders during inference by subtracting a penalty score $\gamma = [0, 3]$ from the logits of zero in Equation (2) to avoid overly short outputs. However, preliminary experiments show that zero penalty score achieves the best performance.

	Train	Valid	Test	Provenance
Ro-En	599k	1911	1999	WMT16
En-De	3,961k	3000	3003	WMT14
En-Ja	2,000k	1790	1812	WAT2017

Table 1: MT Tasks. Data statistics (# sentence pairs) and provenance per language pair.

4 Experiments

We evaluate the EDITOR model on standard (Section 4.2) and lexically constrained machine translation (Sections 4.3–4.4).

4.1 Experimental Settings

Dataset Following Gu et al. (2019), we experiment on three language pairs spanning different language families and data conditions (Table 1): Romanian-English (Ro-En) from WMT16 (Bojar et al., 2016), English-German (En-De) from WMT14 (Bojar et al., 2014), and English-Japanese (En-Ja) from WAT2017 Small-NMT Task (Nakazawa et al., 2017). We also evaluate EDITOR on the two En-De test sets with terminology constraints released by Dinu et al. (2019). The test sets are subsets of the WMT17 En-De test set (Bojar et al., 2017) with terminology constraints extracted from Wiktionary and IATE.[6] For each test set, they only select the sentence pairs in which the exact target terms are used in the reference. The resulting Wiktionary and IATE test sets contain 727 and 414 sentences respectively. We follow the same preprocessing steps in Gu et al. (2019): We apply normalization, tokenization, true-casing, and BPE (Sennrich et al., 2016b) with 37k and 40k operations for En-De and Ro-En. For En-Ja, we use the provided subword vocabularies (16,384 BPE per language from SentencePiece [Kudo and Richardson, 2018]).

Experimental Conditions We train and evaluate the following models in controlled conditions to thoroughly evaluate EDITOR:

- **Auto-Regressive Transformers (AR)** built using Sockeye (Hieber et al., 2017) and

[6]Available at `https://www.wiktionary.org/` and `https://iate.europa.eu`.

fairseq (Ott et al., 2019). We report AR baselines with both toolkits to enable fair comparisons when using our fairseq-based implementation of EDITOR and Sockeye-based implementation of lexically constrained decoding algorithms (Post and Vilar, 2018).

- **Non Auto-Regressive Transformers (NAR)** In addition to **EDITOR**, we train a Levenshtein Transformer (**LevT**) with approximately the same number of parameters. Both are implemented using fairseq.

Model and Training Configurations All models adopt the *base* Transformer architecture (Vaswani et al., 2017) with $d_{\text{model}} = 512$, $d_{\text{hidden}} = 2048$, $n_{\text{heads}} = 8$, $n_{\text{layers}} = 6$, and $p_{\text{dropout}} = 0.3$. For En-De and Ro-En, the source and target embeddings are tied with the output layer weights (Press and Wolf, 2017; Nguyen and Chiang, 2018). We add dropout to embeddings (0.1) and label smoothing (0.1). AR models are trained with the Adam optimizer (Kingma and Ba, 2015) with a batch size of 4096 tokens. We checkpoint models every 1000 updates. The initial learning rate is 0.0002, and it is reduced by 30% after 4 checkpoints without validation perplexity improvement. Training stops after 20 checkpoints without improvement. All NAR models are trained using Adam (Kingma and Ba, 2015) with initial learning rate of 0.0005 and a batch size of 64,800 tokens for maximum 300,000 steps.[7] We select the best checkpoint based on validation BLEU (Papineni et al., 2002). All models are trained on 8 NVIDIA V100 Tensor Core GPUs.

Knowledge Distillation We apply sequence-level knowledge distillation from autoregressive teacher models as widely used in non-autoregressive generation (Gu et al., 2018; Lee et al., 2018; Gu et al., 2019). Specifically, when training the non-autoregressive models, we replace the reference sequences $\boldsymbol{y}^*$ in the training data with translation outputs from the AR teacher model (Sockeye, with beam $= 4$).[8] We also report the results when applying knowledge distillation to autoregressive models.

Evaluation We evaluate translation quality via case-sensitive tokenized **BLEU** (as in Gu et al. (2019))[9] and **RIBES** (Isozaki et al., 2010), which is more sensitive to word order differences. Before computing the scores, we tokenize the German and English outputs using Moses and Japanese outputs using KyTea.[10] For lexically constrained decoding, we report the constraint preservation rate (**CPR**) in the translation outputs.

We quantify decoding speed using **latency** per sentence. It is computed as the average time (in ms) required to translate the test set using batch size of one (excluding the model loading time) divided by the number of sentences in the test set.

4.2 MT Tasks

Because our experiments involve two different toolkits, we first compare the same Transformer AR models built with Sockeye and with fairseq: The AR models achieve comparable decoding speed and translation quality regardless of toolkit —the Sockeye model obtains higher BLEU than the fairseq model on Ro-En and En-De but lower on En-Ja (Table 2). Further comparisons will therefore center on the Sockeye AR model to better compare EDITOR with the lexically constrained decoding algorithm (Post and Vilar, 2018).

Table 2 also shows that knowledge distillation has a small and inconsistent impact on AR models (Sockeye): It yields higher BLEU on Ro-En, close BLEU on En-De, and lower BLEU on En-Ja.[11] Thus, we use the AR models trained without distillation in further experiments.

Next, we compare the NAR models against the AR (Sockeye) baseline. As expected, both EDITOR and LevT achieve close translation quality to their AR teachers with 2–4 times speedup. BLEU differences are small ($\Delta < 1.1$), as in prior work (Gu et al., 2019). The RIBES trends are more surprising: Both NAR models significantly outperform the AR models (Sockeye) on RIBES, except for En-Ja, where EDITOR and the AR models significantly outperforms LevT. This illustrates the strength of EDITOR in word reordering.

Finally, results confirm the benefits of EDITOR's reposition operation over LevT: Decoding with EDITOR is 6–7% faster than LevT

[7] Our preliminary experiments and prior work show that NAR models require larger training batches than AR models.

[8] This teacher model was selected for a fairer comparison on MT with lexical constraints.

[9] https://github.com/pytorch/fairseq/blob/master/fairseq/clib/libbleu/libbleu.cpp.

[10] http://www.phontron.com/kytea/.

[11] Kasai et al. (2020) found that AR models can benefit from knowledge distillation but with a Transformer large model as a teacher, while we use the Transformer base model.

		Distill	Beam	Params	BLEU ↑	RIBES ↑	Latency (ms) ↓
Ro-En	AR (fairseq)		4	64.5M	32.0	83.8	357.14
	AR (sockeye)		4	64.5M	32.3	83.6	369.82
	AR (sockeye)		10	64.5M	32.5	83.8	394.52
	AR (sockeye)	✓	10	64.5M	32.9	84.2	371.75
	NAR: LevT	✓	–	90.9M	**31.6**	**84.0**	98.81
	NAR: EDITOR	✓	–	90.9M	**31.9**	**84.0**	**93.20**
En-De	AR (fairseq)		4	64.9M	27.1	80.4	363.64
	AR (sockeye)		4	64.9M	27.3	80.2	308.64
	AR (sockeye)		10	64.9M	27.4	80.3	332.73
	AR (sockeye)	✓	10	64.9M	27.6	80.5	363.52
	NAR: LevT	✓	–	91.1M	**26.9**	**81.0**	113.12
	NAR: EDITOR	✓	–	91.1M	**26.9**	**80.9**	**105.37**
En-Ja	AR (fairseq)		4	62.4M	44.9	85.7	292.40
	AR (sockeye)		4	62.4M	43.4	85.1	286.83
	AR (sockeye)		10	62.4M	43.5	85.3	311.38
	AR (sockeye)	✓	10	62.4M	42.7	85.1	295.32
	NAR: LevT	✓	–	106.1M	**42.4**	84.5	143.88
	NAR: EDITOR	✓	–	106.1M	**42.3**	**85.1**	**96.62**

Table 2: Machine Translation Results. For each metric, we underline the top scores among all models and boldface the top scores among NAR models based on the paired bootstrap test with $p < 0.05$ (Clark et al., 2011). EDITOR decodes 6–7% faster than LevT on Ro-En and En-De, and 33% faster on En-Ja, while achieving comparable or higher BLEU and RIBES.

on Ro-En and En-De, and 33% faster on En-Ja —a more distant language pair which requires more reordering but no inflection changes on reordered words—with no statistically significant difference in BLEU nor RIBES, except for En-Ja, where EDITOR significantly outperforms LevT on RIBES. Overall, EDITOR is shown to be a good alternative to LevT on standard machine translation tasks and can also be used to replace the AR models in settings where decoding speed matters more than small differences in translation quality.

4.3 MT with Lexical Constraints

We now turn to the main evaluation of EDITOR on machine translation with lexical constraints.

Experimental Conditions We conduct a controlled comparison of the following approaches:

- NAR models: **EDITOR** and **LevT** view the lexical constraints as **soft constraints**, provided via the initial target sequence. We also explore the decoding technique introduced in Susanto et al. (2020) to support **hard constraints**.
- AR models: They use the provided target words as hard constraints enforced at decoding time by an efficient form of constrained beam search: dynamic beam allocation (**DBA**) (Post and Vilar, 2018).[12]

Crucially, all models, including EDITOR, are the exact same models evaluated on the standard MT tasks above, and do not need to be trained specifically to incorporate constraints.

We define lexical constraints as Post and Vilar (2018): For each source sentence, we randomly select one to four words from the reference as lexical constraints. We then randomly shuffle the constraints and apply BPE to the constraint sequence. Different from the terminology test sets in Dinu et al. (2019), which contain only several hundred sentences with mostly nominal constraints, our constructed test sets are larger and include lexical constraints of all types.

[12]Although the beam pruning option in Post and Vilar (2018) is not used here (since it is not supported in Sockeye anymore), other Sockeye updates improve efficiency. Constrained decoding with DBA is 1.8–2.7 times slower than unconstrained decoding here, while DBA is 3 times slower when beam $= 10$ in Post and Vilar (2018).

		Distill	Beam	BLEU ↑	RIBES ↑	CPR ↑	Latency (ms) ↓
Ro-En	AR + DBA (sockeye)		4	31.0	79.5	99.7	436.26
	AR + DBA (sockeye)		10	34.6	84.5	99.5	696.68
	NAR: LevT	✓	–	31.6	83.4	80.3	121.80
	+ hard constraints	✓	–	27.7	78.4	99.9	140.79
	NAR: EDITOR	✓	–	**33.1**	**85.0**	**86.8**	**108.98**
	+ hard constraints	✓	–	28.8	81.2	95.0	136.78
En-De	AR + DBA (sockeye)		4	26.1	74.7	99.7	434.41
	AR + DBA (sockeye)		10	30.5	81.9	99.5	896.60
	NAR: LevT	✓	–	27.1	80.0	75.6	127.00
	+ hard constraints	✓	–	24.9	74.1	100.0	134.10
	NAR: EDITOR	✓	–	**28.2**	**81.6**	**88.4**	**121.65**
	+ hard constraints	✓	–	25.8	77.2	96.8	134.10
En-Ja	AR + DBA (sockeye)		4	44.3	81.6	100.0	418.71
	AR + DBA (sockeye)		10	48.0	85.9	100.0	736.92
	NAR: LevT	✓	–	42.8	84.0	74.3	161.17
	+ hard constraints	✓	–	39.7	77.4	99.9	159.27
	NAR: EDITOR	✓	–	**45.3**	**85.7**	**91.3**	**109.50**
	+ hard constraints	✓	–	43.7	82.6	96.4	132.71

Table 3: Machine Translation with lexical constraints (averages over 5 runs). For each metric, we underline the top scores among all models and boldface the top scores among NAR models based on the independent student's t-test with $p < 0.05$. EDITOR exploits constraints better than LevT. It also achieves comparable RIBES to the best AR model with 6–7 times decoding speedup.

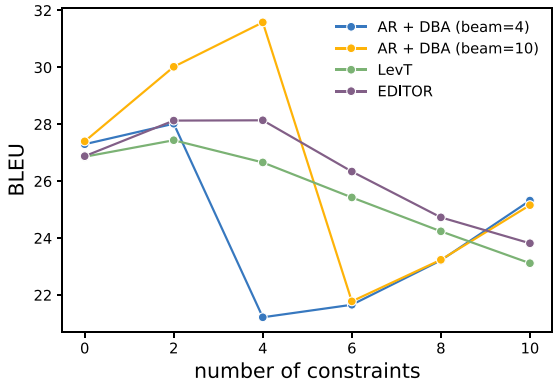
Main Results Table 3 shows that EDITOR exploits the soft constraints to strike a better balance between translation quality and decoding speed than other models. Compared to LevT, EDITOR preserves 7–17% more constraints and achieves significantly higher translation quality (+1.1–2.5 on BLEU and +1.6–1.8 on RIBES) and faster decoding speed. Compared to the AR model with beam = 4, EDITOR yields significantly higher BLEU (+1.0–2.2) and RIBES (+4.1–6.9) with 3–4 times decoding speedup. After increasing the beam to 10, EDITOR obtains lower BLEU but comparable RIBES with 6–7 times decoding speedup.[13] Note that AR models treat provided words as hard constraints and therefore achieve over 99% CPR by design, while NAR models treat them as soft constraints.

Results confirm that enforcing hard constraints increases CPR but degrades translation quality compared to the same model using soft constraints: For LevT, it degrades BLEU by 2.2–3.9 and RIBES by 5.0–6.6. For EDITOR, it degrades BLEU by 1.6–4.3 and RIBES by 3.1–4.4 (Table 3). By contrast, EDITOR with soft constraints strikes a better balance between translation quality and constraint preservation.

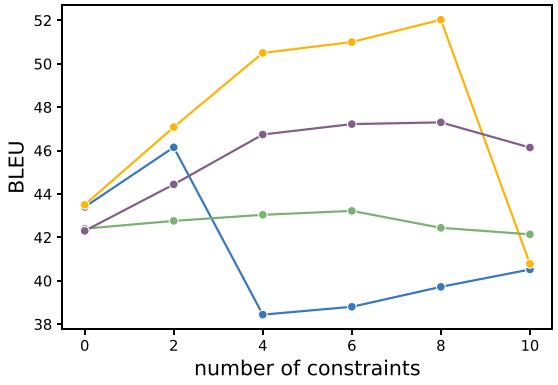
The strengths of EDITOR hold when varying the number of constraints (Figure 5). For all tasks and models, adding constraints helps BLEU up to a certain point, ranging from 4 to 10 words. When excluding the slower AR model (beam = 10), EDITOR consistently reaches the highest BLEU score with 2–10 constraints: EDITOR outperforms LevT and the AR model with beam = 4. Consistent with Post and Vilar (2018), as the number of constraints increases, the AR model needs larger beams to reach good performance. When the number of constraints increases to 10, EDITOR yields higher BLEU than the AR model on En-Ja and Ro-En, even after incurring the cost of increasing the AR beam to 10.

Are EDITOR improvements limited to preserving constraints better? We verify that this is not the case by computing the target word F1 binned by frequency (Neubig et al., 2019). Figure 6 shows that EDITOR improves over LevT across all test frequency classes and closes

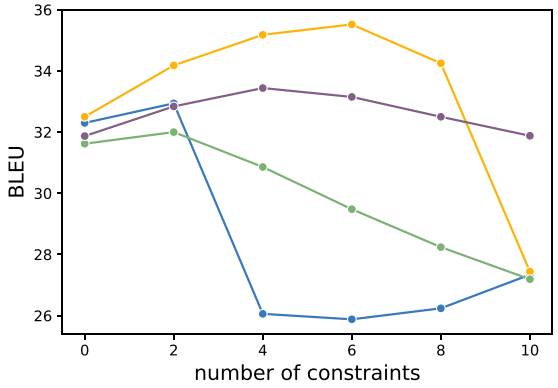
[13] Post and Vilar (2018) show that the optimal beam size for DBA is 20. Our experiment on En-De shows that increasing the beam size from 10 to 20 improves BLEU by 0.7 at the cost of doubling the decoding time.



(a) En-De

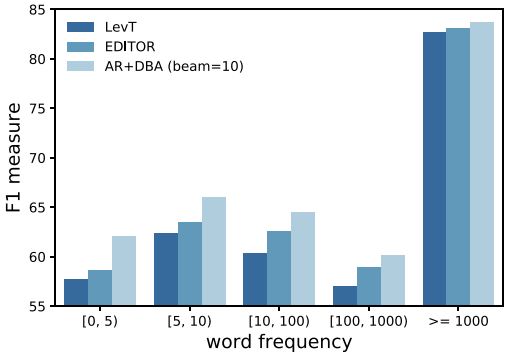


(b) En-Ja

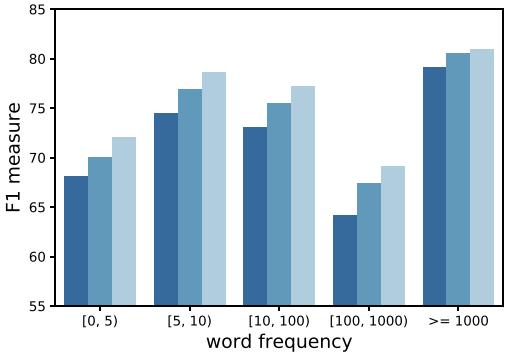


(c) Ro-En

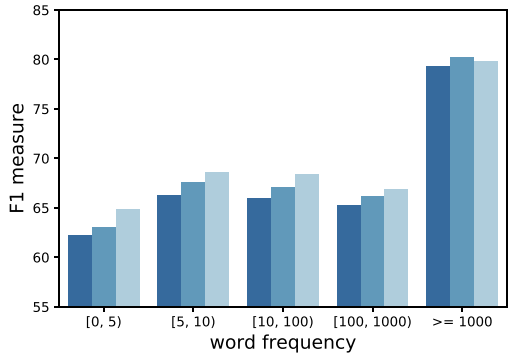
Figure 5: EDITOR improves BLEU over LevT for 2–10 constraints (counted pre-BPE) and beats the best AR model on 2/3 tasks with 10 constraints.



(a) En-De



(b) En-Ja



(c) Ro-En

Figure 6: Target word F1 score binned by word test set frequency: EDITOR improves over LevT the most for words of low or medium frequency. AR achieves higher F1 than EDITOR for words of low or medium frequency at the cost of much longer decoding time.

the gap between NAR and AR models: The largest improvements are obtained for low and medium frequency words—on En-De and En-Ja, the largest improvements are on words with frequency between 5 and 1000, while on Ro-En, EDITOR improves more on words with frequency between 5 and 100. EDITOR also improves F1 on rare words (frequency in $[0, 5]$), but not as much as for more frequent words.

We now conduct further analysis to better understand the factors that contribute to EDITOR's advantages over LevT.

Impact of Reposition We compare the average number of basic edit operations (Section 3.1) of different types used by EDITOR and LevT on each test sentence (averaged over the 5 runs): Reposition (excluding deletion for controlled comparison with LevT), deletion, and insertion performed by LevT and EDITOR at decoding

	Repos.	Del.	Ins.	Total	Iter.
Ro-En					
LevT	0.00	4.61	33.05	37.67	2.01
EDITOR	8.13	2.50	28.68	39.31	1.81
En-De					
LevT	0.00	7.13	45.45	52.58	2.14
EDITOR	5.85	4.01	28.75	38.61	2.07
En-Ja					
LevT	0.00	5.24	32.83	38.07	2.93
EDITOR	4.73	1.69	21.64	28.06	1.76

Table 4: Average number of repositions (excluding deletions), deletions, insertions, and decoding iterations to translate each sentence with soft lexical constraints (averaged over 5 runs). Thanks to reposition operations, EDITOR uses 40–70% fewer deletions, 10–40% fewer insertions, and 3–40% fewer decoding iterations overall.

	BLEU↑	RIBES↑	CPR↑	Lat. ↓
Ro-En				
EDITOR	33.1	85.0	86.8	108.98
-dual-path	32.2	84.4	74.8	119.61
LevT	31.6	83.4	80.3	121.80
En-De				
EDITOR	28.2	81.6	88.4	121.65
-dual-path	27.2	80.4	78.7	130.85
LevT	27.1	80.0	75.6	127.00
En-Ja				
EDITOR	45.3	85.7	91.3	109.50
-dual-path	44.0	83.9	80.0	154.10
LevT	42.8	84.0	74.3	161.17

Table 5: Ablating the dual-path roll-in policy hurts EDITOR on soft-constrained MT, but still outperforms LevT, confirming that reposition and dual-path imitation learning both benefit EDITOR.

	Wiktionary		IATE	
	Term%↑	BLEU↑	Term%↑	BLEU↑
Prior Results				
Base Trans.	76.9	26.0	76.3	25.8
Post18	99.5	25.8	82.0	25.3
Dinu19	93.4	26.3	94.5	26.0
Base LevT	81.1	30.2	80.3	29.0
Susanto20	100.0	31.2	100.0	30.1
Our Results				
LevT	84.3	28.2	83.9	27.9
+ soft constraints	90.5	28.5	92.5	28.3
+ hard constraints	100.0	28.8	100.0	28.9
EDITOR	83.5	28.8	83.0	27.9
+ soft constraints	96.8	29.3	97.1	28.8
+ hard constraints	99.8	29.3	100.0	28.9

Table 6: Term usage percentage (*Term%*) and BLEU scores of En-De models on terminology test sets (Dinu et al., 2019) provided with correct terminology entries (exact matches on both source and target sides). EDITOR with soft constraints achieves higher BLEU than LevT with soft constraints, and on par or higher BLEU than LevT with hard constraints.

time. Table 4 shows that LevT deletes tokens 2–3 times more often than EDITOR, which explains its lower CPR than EDITOR. LevT also inserts tokens 1.2–1.6 times more often than EDITOR and performs 1.4 times more edit operations on En-De and En-Ja. On Ro-En, LevT performs −4% fewer edit operations in total than EDITOR but is overall slower than EDITOR, since multiple operations can be done in parallel at each action step. Overall, EDITOR takes 3–40% fewer decoding iterations than LevT. These results suggest that reposition successfully reduces redundancy in edit operations and makes decoding more efficient by replacing sequences of insertions and deletions with a single repositioning step.

Furthermore, Figure 7 illustrates how reposition increases flexibility in exploiting lexical constraints, even when they are provided in the wrong order. While LevT generates an incorrect output by using constraints in the provided order, EDITOR's reposition operation helps generate a more fluent and adequate translation.

Impact of Dual-Path Roll-In Ablation experiments (Table 5) show that EDITOR benefits greatly from dual-path roll-in. Replacing dual-path roll-in with the simpler roll-in policy used in Gu et al. (2019), the model's translation quality drops significantly (by 0.9–1.3 on BLEU and 0.6–1.9 on RIBES) with fewer constraints preserved and slower decoding. It still achieves better translation quality than LevT thanks to the reposition operation: specifically, it yields significantly higher BLEU and RIBES on Ro-En, comparable BLEU and significantly higher RIBES on En-De, and comparable RIBES and significantly higher BLEU on En-Ja than LevT.

Source:		Cred că Stephen Thompson are încredere în noi .
Reference:		I think Stephen Thompson has faith in us .
Constraints:		faith Stephen think
LevT:		
	y^0:	faith Stephen think
Action a^1:	$y' = \mathcal{E}(y^0, d^1)$:	faith Stephen think
	$y'' = \mathcal{E}(y', p^1)$:	[plh] [plh] faith [plh] Stephen [plh] [plh] [plh] [plh] think [plh]
	$y^1 = \mathcal{E}(y'', t^1)$:	I think faith that Stephen Thom@@ p@@ son can think .
	no further actions:	**[Terminate]**
EDITOR:		
	y^0:	faith Stephen think
Action a^1:	$y' = \mathcal{E}(y^0, r^1)$:	think Stephen faith
	$y'' = \mathcal{E}(y', p^1)$:	[plh] think Stephen [plh] [plh] [plh] [plh] faith [plh] [plh] [plh]
	$y^1 = \mathcal{E}(y'', t^1)$:	I think Stephen Thom@@ p@@ son has faith in us .
	no further actions:	**[Terminate]**

Figure 7: Ro-En translation with soft lexical constraints: while LevT uses the constraints in the provided order, EDITOR's reposition operation helps generate a more fluent and adequate translation.

4.4 MT with Terminology Constraints

We evaluate EDITOR on the terminology test sets released by Dinu et al. (2019) to test its ability to incorporate terminology constraints and to further compare it with prior work (Dinu et al., 2019; Post and Vilar, 2018; Susanto et al., 2020).

Compared to Post and Vilar (2018) and Dinu et al., (2019), EDITOR with soft constraints achieves higher absolute BLEU, and higher BLEU improvements over its counterpart without constraints (Table 6). Consistent with previous findings by Susanto et al. (2020), incorporating soft constraints in LevT improves BLEU by +0.3 on Wiktionary and by +0.4 on IATE. Enforcing hard constraints as in Susanto et al. (2020) increases the term usage by +8–10% and improves BLEU by +0.3–0.6 over LevT using soft constraints.[14] For EDITOR, adding soft constraints improves BLEU by +0.5 on Wiktionary and +0.9 on IATE, with very high term usages (96.8% and 97.1% respectively). EDITOR thus correctly uses the provided terms almost all the time when they are provided as soft constraints, so there is little benefit to enforcing hard constraints instead: They help close the small gap to reach 100% term usage and do not improve BLEU. Overall, EDITOR achieves on par or higher BLEU than LevT with hard constraints.

[14]We use our implementations of Susanto et al.'s (2020) technique for a more controlled comparison. The LevT baseline in Susanto et al. (2020) achieves higher BLEU than ours on the small Wiktionary and IATE test sets, while it underperforms our LevT on the full WMT14 test set (26.5 vs. 26.9).

Results also suggest that EDITOR can handle phrasal constraints even though it relies on token-level edit operations, since it achieves above 99% term usage on the terminology test sets where 26–27% of the constraints are multi-token.

5 Conclusion

We introduce EDITOR, a non-autoregressive transformer model that iteratively edits hypotheses using a novel reposition operation. Reposition combined with a new dual-path imitation learning strategy helps EDITOR generate output sequences that flexibly incorporate user's lexical choice preferences. Extensive experiments show that EDITOR exploits soft lexical constraints more effectively than the Levenshtein Transformer (Gu et al., 2019) while speeding up decoding dramatically compared to constrained beam search (Post and Vilar, 2018). Results also confirm the benefits of using soft constraints over hard ones in terms of translation quality. EDITOR also achieves comparable or better translation quality with faster decoding speed than the Levenshtein Transformer on three standard MT tasks. These promising results open several avenues for future work, including using EDITOR for other generation tasks than MT and investigating its ability to incorporate more diverse constraint types into the decoding process.

Acknowledgments

We thank Sweta Agrawal, Kianté Brantley, Eleftheria Briakou, Hal Daumé III, Aquia

Richburg, François Yvon, the TACL reviewers, and the CLIP lab at UMD for their helpful and constructive comments. This research is supported in part by an Amazon Web Services Machine Learning Research Award and by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via contract #FA8650-17-C-9117. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

Fadi Abu Sheikha and Diana Inkpen. 2011. Generation of formal and informal sentences. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 187–193, Nancy, France. Association for Computational Linguistics.

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, Hong Kong, China. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D19-1166

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2017. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945, Copenhagen, Denmark. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D17-1098, **PMID:** 30027537, **PMCID:** PMC6220700

Philip Arthur, Graham Neubig, and Satoshi Nakamura. 2016. Incorporating discrete translation lexicons into neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1557–1567, Austin, Texas. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D16-1162

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3th International Conference on Learning Representations*.

Srinivas Bangalore, Patrick Haffner, and Stephan Kanthak. 2007. Statistical machine translation through global lexical selection and sentence reconstruction. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 152–159, Prague, Czech Republic. Association for Computational Linguistics.

Sergio Barrachina, Oliver Bender, Francisco Casacuberta, Jorge Civera, Elsa Cubel, Shahram Khadivi, Antonio Lagarda, Hermann Ney, Jesús Tomás, Enrique Vidal, and Juan-Miguel Vilar. 2009. Statistical approaches to computer-assisted translation. *Computational Linguistics*, 35(1):3–28. **DOI:** https://doi.org/10.1162/coli.2008.07-055-R2-06-29

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/v1/14-3302

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. 2017. Findings of the 2017 conference on machine translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*,

pages 169–214, Copenhagen, Denmark. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/W17-4717

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/W16-2301

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. 2018. Fast policy learning through imitation and reinforcement. In *Proceedings of the 2018 Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 845–855, Monterey, CA, USA.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Jan K. Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, Montreal, Canada.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 176–181, Portland, Oregon, USA. Association for Computational Linguistics.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75(3):297–325. **DOI:** https://doi.org/10.1007/s10994-009-5106-x

Georgiana Dinu, Prashant Mathur, Marcello Federico, and Yaser Al-Onaizan. 2019. Training neural machine translation to apply terminology constraints. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3063–3068, Florence, Italy. Association for Computational Linguistics.

Nadir Durrani, Helmut Schmid, Alexander Fraser, Philipp Koehn, and Hinrich Schütze. 2015. The operation sequence model—Combining n-gram-based and phrase-based statistical machine translation. *Computational Linguistics*, 41(2):157–186. **DOI:** https://doi.org/10.1162/COLI_a_00218

Jessica Ficler and Yoav Goldberg. 2017. Controlling linguistic style aspects in neural language generation. In *Proceedings of the Workshop on Stylistic Variation*, pages 94–104, Copenhagen, Denmark. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/W17-4912

George Foster, Philippe Langlais, and Guy Lapalme. 2002. User-friendly text prediction for translators. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 148–155. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/W17-4912

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 6112–6121, Hong Kong, China. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D19-1633

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK 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 *Advances in Neural Information Processing Systems 32*, pages 11181–11191. Curran Associates, Inc.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A toolkit for neural machine translation. *CoRR*, abs/1712.05690.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada. Association for Computational Linguistics.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. 2010. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952, Cambridge, MA. Association for Computational Linguistics.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *arXiv preprint arXiv:2006.10369*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3th International Conference on Learning Representations*. San Diego, CA, USA.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D18-2012, **PMID:** 29382465

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018. Unsupervised machine translation using monolingual corpora only. In *Proceedings of the 6th 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, Brussels, Belgium. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D18-1149

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. FlowSeq: Non-autoregressive conditional

sequence generation with generative flow. 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 4282–4292, Hong Kong, China. Association for Computational Linguistics.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. What to talk about and how? selective generation using LSTMs with coarse-to-fine alignment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 720–730, San Diego, California. Association for Computational Linguistics.

Toshiaki Nakazawa, Shohei Higashiyama, Chenchen Ding, Hideya Mino, Isao Goto, Hideto Kazawa, Yusuke Oda, Graham Neubig, and Sadao Kurohashi. 2017. Overview of the 4th workshop on Asian translation. In *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pages 1–54, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, and Xinyi Wang. 2019. compare-mt: A tool for holistic comparison of language generation systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 35–41, Minneapolis, Minnesota. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/N19-4007

Toan Q. Nguyen and David Chiang. 2018. Improving lexical choice in neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 334–343, Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/N18-1031, **PMID:** 29283496

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis Cobo , Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. 2018. Parallel WaveNet: Fast high-fidelity speech synthesis. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3918–3926, Stockholmsmässan, Stockholm Sweden. PMLR.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. Fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/N19-4009

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics. **DOI:** https://doi.org/10.3115/1073083.1073135

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324, New Orleans, Louisiana. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/N18-1119

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Computational*, pages 157–163.

Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/E17-2025

Stéphane Ross and J. Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *CoRR*, abs/1406.5979.

Stephane 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*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635. PMLR.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Controlling politeness in neural machine translation via side constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 35–40, San Diego, California. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/N16-1005

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/P16-1162

Kai Song, Yue Zhang, Heng Yu, Weihua Luo, Kun Wang, and Min Zhang. 2019. Code-switching for enhancing NMT with pre-specified 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 449–459, Minneapolis, Minnesota. Association for Computational Linguistics.

Felix Stahlberg, Danielle Saunders, and Bill Byrne. 2018. An operation sequence model for explainable neural machine translation. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 175–186, Brussels, Belgium. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/W18-5420

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5976–5985, Long Beach, California, USA. PMLR.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, pages 10086–10095, Montreal, Canada. Curran Associates, Inc..

Raymond Hendy Susanto, Shamil Chollampatt, and Liling Tan. 2020. Lexically constrained neural machine translation with Levenshtein transformer. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3536–3543, Online. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/2020.acl-main.325

Yaohua Tang, Fandong Meng, Zhengdong Lu, Hang Li, and Philip L. H. Yu. 2016. Neural machine translation with external phrase memory. *arXiv preprint arXiv:1606.01792*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, Long Beach, CA, USA. Curran Associates, Inc.

Oriol Vinyals and Quoc Le. 2015. A neural conversational model. In *ICML Deep Learning Workshop*. Lille, France.

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference*

on Empirical Methods in Natural Language Processing, pages 479–488, Brussels, Belgium. Association for Computational Linguistics. **DOI:** https://doi.org/10.18653/v1/D18-1044

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019. Non-autoregressive machine translation with auxiliary regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5377–5384. **DOI:** https://doi.org/10.1609/aaai.v33i01.33015377

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pages 6716–6726.

François Yvon and Sadaf Abdul Rauf. 2020. Utilisation de ressources lexicales et terminologiques en traduction neuronale. Research Report 2020-001, LIMSI-CNRS.