A Learning & Inference Algorithm

We present the detailed algorithms for learning and decoding from Levenshtein Transformer as follows. For simplicity, we always omit the source information $\boldsymbol{x}$ in conditional sequence generation tasks such as machine translation which is handled by the cross-attention with an encoder on $\boldsymbol{x}$.

The learning algorithm is shown in Algorithm 1. $\mathcal{E}$ is the environment and $\mathcal{D}$ is denoted as the Levenshtein distance, and we can easily back-track the optimal insertion and deletion operations through dynamic programming. We only show the the case with single batch-size for convenience. We also present the inference algorithm in Algorithm 2. If the initial sequence $\boldsymbol{y}^0$ is empty (<s></s>), the proposed model will skip the first deletion and do sequence generation. Otherwise, the model starts with deletion operations and refine the input sequence.

Algorithm 1 Learning for Levenshtein Transformer

Initialize: Training set $\mathcal{T}$, expert policy π^*, model policy π_θ, random deletion policy π^{RND}, α, β
repeat
  Sample a training pair $(\boldsymbol{y}^0, \boldsymbol{y}^*) \sim \mathcal{Y}$
  if expert π^* is a teacher model **then**
    Set the teacher's output as the target $\boldsymbol{y}^* = \boldsymbol{y}^{\text{AR}}$
  end if
  Sample $u, v \sim \text{Uniform}[0, 1]$
  if $u < \beta$ **then**
    $\boldsymbol{y}_{\text{ins}} = \mathcal{E}(\boldsymbol{y}^0, \tilde{\boldsymbol{d}})$, where $\tilde{\boldsymbol{d}} = \arg\min_{\boldsymbol{d}} \mathcal{D}(\boldsymbol{y}^*, \mathcal{E}\left(\boldsymbol{y}^0, \boldsymbol{d}\right))$
  else
    $\boldsymbol{y}_{\text{ins}} = \mathcal{E}(\boldsymbol{y}^*, \tilde{\boldsymbol{d}})$, where $\tilde{\boldsymbol{d}} \sim \pi^{\text{RND}}(\cdot|\boldsymbol{y}^*)$
  end if
  $\boldsymbol{y}'_{\text{ins}} = \mathcal{E}(\boldsymbol{y}_{\text{ins}}, \boldsymbol{p}^*)$, where $\boldsymbol{p}^*, \boldsymbol{t}^* = \arg\min_{\boldsymbol{p},\boldsymbol{t}} \mathcal{D}(\boldsymbol{y}^*, \mathcal{E}\left(\boldsymbol{y}_{\text{ins}}, \{\boldsymbol{p}, \boldsymbol{t}\}\right))$
  if $v < \alpha$ **then**
    $\boldsymbol{y}_{\text{del}} = \boldsymbol{y}^0$
  else
    $\boldsymbol{y}_{\text{del}} = \mathcal{E}(\boldsymbol{y}'_{\text{ins}}, \hat{\boldsymbol{t}})$, where $\hat{\boldsymbol{t}} = \arg\max_{\boldsymbol{t}} \sum_{y_i \in \boldsymbol{y}'_{\text{ins}}, y_i = \texttt{<PLH>}} \log \pi^{\text{tok}}_\theta(t_i|i, \boldsymbol{y}'_{\text{ins}})$
  end if
  $\mathcal{L}^{\text{ins}}_\theta = -\left[\sum_{y_i \in \boldsymbol{y}_{\text{ins}}, p^*_i \in \boldsymbol{p}^*} \log \pi^{\text{plh}}_\theta(p^*_i|i, \boldsymbol{y}_{\text{ins}}) + \sum_{y_i \in \boldsymbol{y}'_{\text{ins}}, y_i = \texttt{<PLH>}, t^*_i \in \boldsymbol{t}^*} \log \pi^{\text{tok}}_\theta(t^*_i|i, \boldsymbol{y}'_{\text{ins}})\right]$
  $\mathcal{L}^{\text{del}}_\theta = -\sum_{y_i \in \boldsymbol{y}_{\text{del}}, d^*_i \in \boldsymbol{d}^*} \log \pi^{\text{del}}_\theta(d^*_i|i, \boldsymbol{y}_{\text{del}})$, where $\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} \mathcal{D}(\boldsymbol{y}^*, \mathcal{E}\left(\boldsymbol{y}_{\text{del}}, \boldsymbol{d}\right))$
  $\theta = \theta - \lambda \cdot \bigtriangledown_\theta \left[\mathcal{L}^{\text{ins}}_\theta + \mathcal{L}^{\text{del}}_\theta\right]$
until Maximum training steps reached

B Dataset and Preprocessing Details

Table 4 and 5 list the statistics (# of sentences, vocabulary) for all the datasets used in this work. We learn BPE vocabulary with $32,000$ joint operations for WMT En-De and Gigaword and $40,000$ joint operations for WMT Ro-En. For WAT En-Ja, we adopt the official $16,384$ BPE vocabularies learned separately on source and target side.

Table 4: Dataset statistics for sequence generation tasks (MT and TS).

Dataset		Train	Valid	Test	Vocabulary
Translation	WMT'16 Ro-En	608,319	1999	1999	34,983
	WMT'14 En-De	4,500,966	3000	3003	37,009
	WAT'17 En-Ja	2,000,000	1790	1812	17,952 / 17,801
Summarization	English Gigaword	3,803,957	189,651	1951	30,004

Algorithm 2 Decoding for Levenshtein Transformer

Initialize: Input $\boldsymbol{y} = \boldsymbol{y}^0$, step $t = 0$, maximum step $T_{\max}$, model policy π_θ.
repeat
  if $\boldsymbol{y} =$ <s></s> **then**
    Empty sequence, skip deletion: $\boldsymbol{y}' = \boldsymbol{y}$
  else
    Delete tokens: $\boldsymbol{y}' = \mathcal{E}(\boldsymbol{y}, \hat{\boldsymbol{d}})$, where $\hat{\boldsymbol{d}} = \arg\max_{\boldsymbol{d}} \sum_{y_i \in \boldsymbol{y}} \log \pi^{\text{del}}_\theta(d_i | i, \boldsymbol{y})$
  end if
  if (t > 0) & $(\boldsymbol{y}' = \tilde{\boldsymbol{y}})$ **then**
    Termination condition satisfied: direct loop
    break
  end if
  Assign deleted output for back-up $\tilde{\boldsymbol{y}} = \boldsymbol{y}'$
  Insert placeholders: $\boldsymbol{y}'' = \mathcal{E}(\boldsymbol{y}', \hat{\boldsymbol{p}})$, where $\hat{\boldsymbol{p}} = \arg\max_{\boldsymbol{p}} \sum_{y_i y_{i+1} \in \boldsymbol{y}'} \log \pi^{\text{plh}}_\theta(p_i | i, \boldsymbol{y}')$
  if $\boldsymbol{y}'' = \boldsymbol{y}' = \boldsymbol{y}$ **then**
    Termination condition satisfied: nothing to delete, nothing to insert.
    break
  end if
  if $\boldsymbol{y}'' = \boldsymbol{y}'$ **then**
    Nothing to insert, skip insertion: $\boldsymbol{y} = \boldsymbol{y}''$
  else
    Replace placeholders: $\boldsymbol{y} = \mathcal{E}(\boldsymbol{y}'', \hat{\boldsymbol{t}})$, where $\hat{\boldsymbol{t}} = \arg\max_{\boldsymbol{t}} \sum_{y_i \in \boldsymbol{y}'', y_i = \text{<PLH>}} \log \pi^{\text{tok}}_\theta(t_i | i, \boldsymbol{y}'')$
  end if
  Update steps: t = t + 1
until Reach the maximum length $t = T_{\max}$
return $\boldsymbol{y}$

Table 5: Dataset statistics for sequence refinement tasks (APE).

Dataset		MT-Train	APE-Train	Valid	Test	Vocabulary
Synthetic	WMT'16 Ro-En	300,000	308,319	1999	1999	34,983
	WMT'14 En-De	2,250,000	2,250,967	3000	3003	37,009
	WAT'17 En-Ja	1,000,000	1,000,000	1790	1812	17,952 / 17,801
Real	WMT'17 APE En-De	4,391,180	526,368 (fake) + 24,000 (real)	2000	2000	40,349

C Model and Training Details

C.1 Sequence Generation Tasks

Transformer models are used for autoregressive baselines as well as teacher models (for the expert policy). By default, we set $d_{\text{model}} = 512$, $d_{\text{hidden}} = 2048$, $n_{\text{heads}} = 8$, $n_{\text{layers}} = 6$, $\text{lr}_{\max} = 0.0005$, label-smooth $= 0.1$, warmup $= 10000$ and dropout $= 0.3$. Source and target side share embeddings in all the training pairs except for WAT En-Ja where BPE vocabularies of both side are learned separately and are almost non-overlapping.

Since the training objectives for Levenshtein Transformer contains randomness terms (Eq. (6) (7)), we instead use BLEU (for MT) or ROUGE-2 (for TS) to select the best checkpoint by validation scores. We do not average checkpoints in this work.

C.2 Sequence Refinement Tasks

For synthetic APE tasks, we keep the same training conditions for LevT as those for MT tasks (§C.1). As described earlier in §4.2, we build the baseline Transformer by concatenating the source and MT system's output as the input sequence for the encoder. Specially, we restart the positional embeddings

Table 6: The percentage of WMT En-De test sentence generation terminated at each iteration using LevT(T) with a maximum iteration of 10.

Iterations	1	2	3	4	5	6	7	8	9	10	2.43
%	12.3	48.1	28.5	8.5	2.0	0.4	0.1	0	0	0.1	AVG

for the MT output, add an additional language embedding for each token of the input sequence to show its language type. The detailed hyperpameters are the same as the standard Transformer.

As described in §4.2, we consider the following two different imperfect MT systems to provide the refinement inputs. Firstly, we consider the traditional statistical phrase-based machine translation system (PBMT). We follow the instruction to build the basic baseline model via moses[9]. As for the NMT-based model, we use a single layer attention-based model composed by LSTM. We build this model on fairseq-py[10] with the default configuration.

For the real APE task, we follow the procedures introduced in Junczys-Dowmunt and Grundkiewicz (2016). Synthetic corpus has two subsets: a 500K one and a 4M one. We over-sample real data by 10 times and merge it with the 500K synthetic data to train APE models. Besides, we also train a LevT MT model on the bigger (4M) synthetic corpus where we only use the source and target pairs.

C.3 Implementation

Both the proposed Levenshtein Transformer and the baseline Transformer are implemented using PyTorch[11]. The codes are released as part of the Fairseq-py [12].

C.4 Maximum Number of Iterations

We also presented in general how many sentences will be generated using the maximum iteration (for instance 10). As shown in Table 6, surprisingly, most predictions are gotten in 1-4 iterations, and the average number of iterations is 2.43. Only a tiny portion ($\sim 0.1\%$) require the maximum number of iterations demonstrating the efficiency of the proposed approach.

[9] http://www.statmt.org/moses/?n=Moses.Baseline

[10] https://github.com/pytorch/fairseq/blob/master/fairseq/models/lstm.py

[11] https://pytorch.org/

[12] https://github.com/pytorch/fairseq/tree/master/examples/nonautoregressive_translation

D More Decoding Examples

We present more examples from the proposed Levenshtein Transformer as follows.

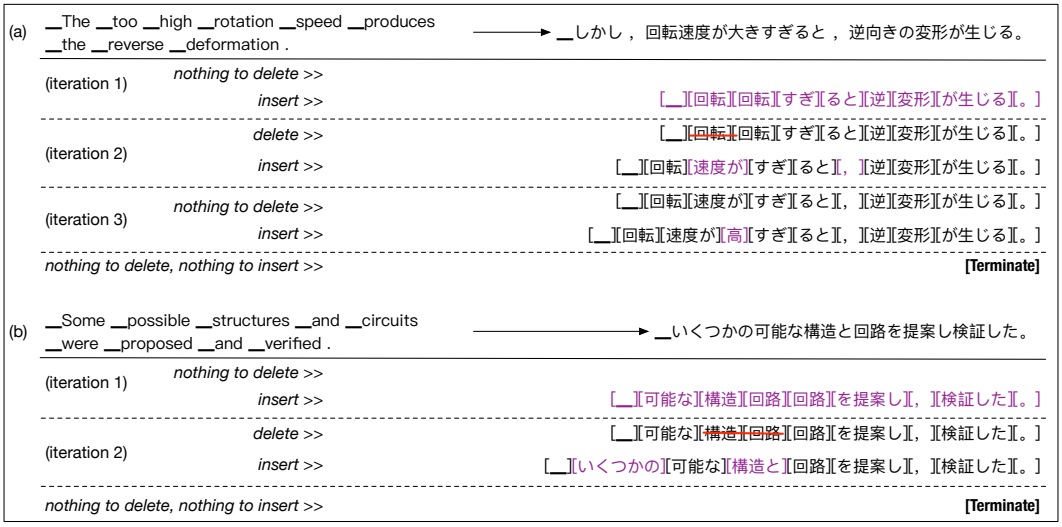

Figure 6: Translation examples for WAT'17 Small-NMT En-Ja with the Levenshtein Transformer.

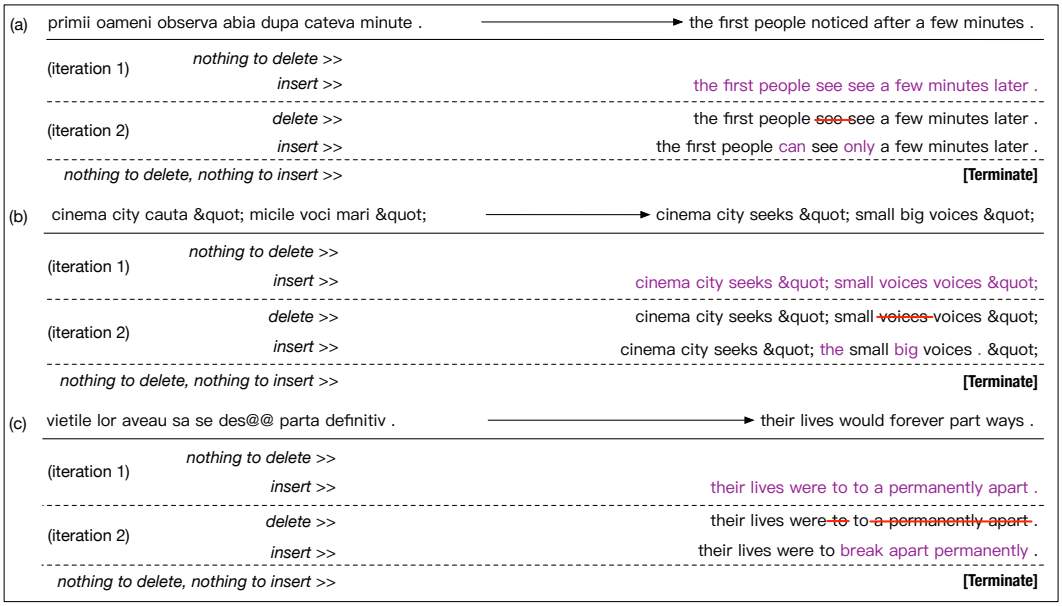

Figure 7: Translation examples for WMT'16 Ro-En with the Levenshtein Transformer.

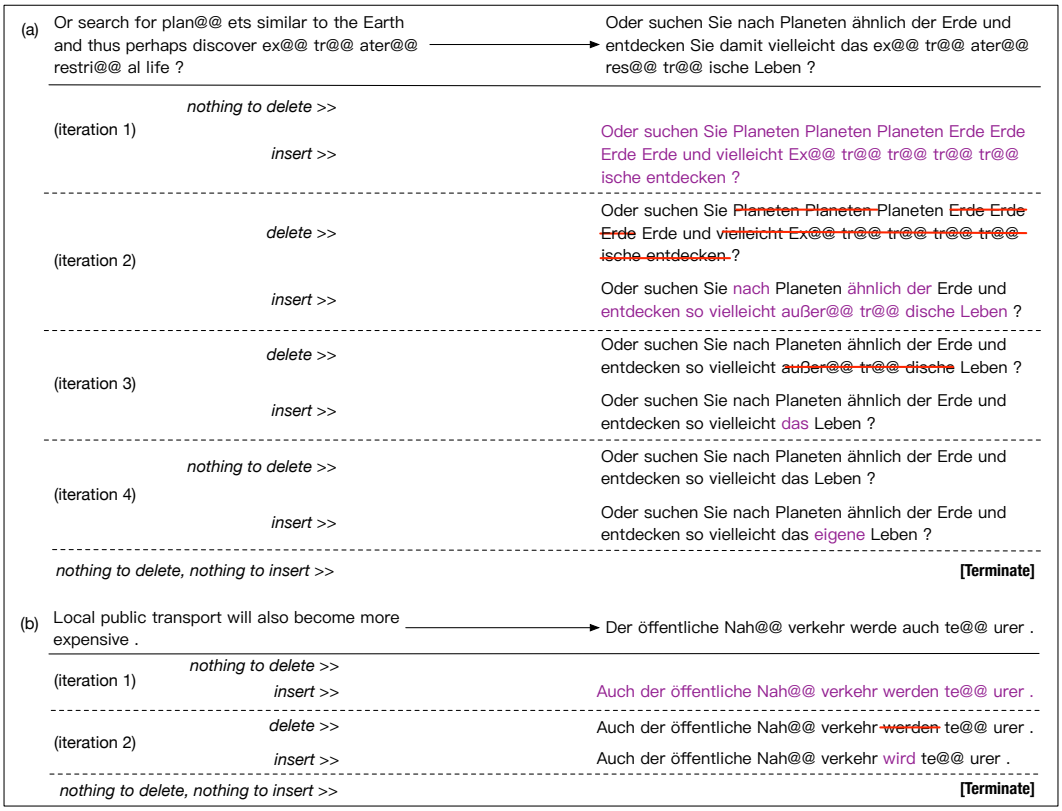

Figure 8: Translation examples for WMT'14 En-De with the Levenshtein Transformer.

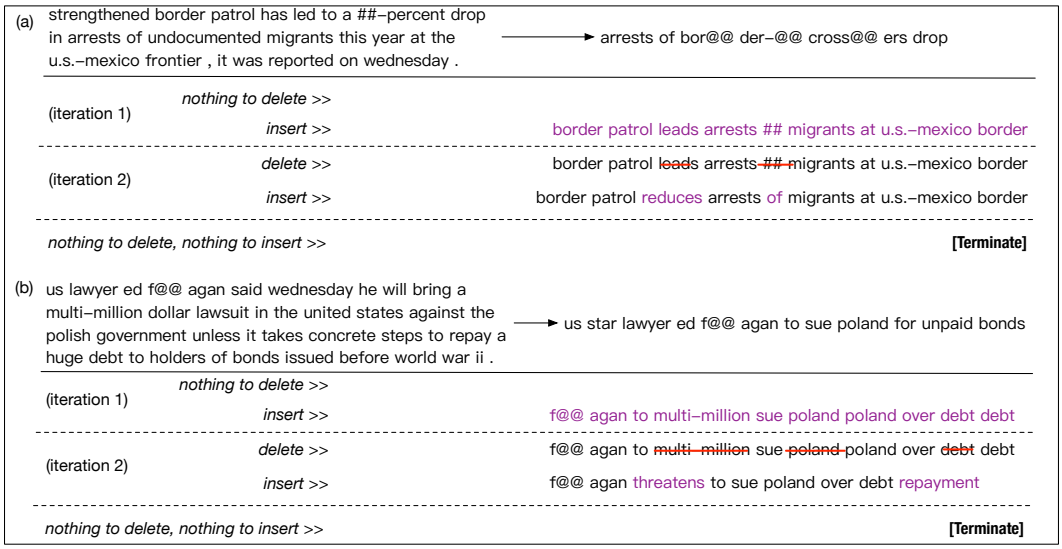

Figure 9: Translation examples for English Gigaword with the Levenshtein Transformer.

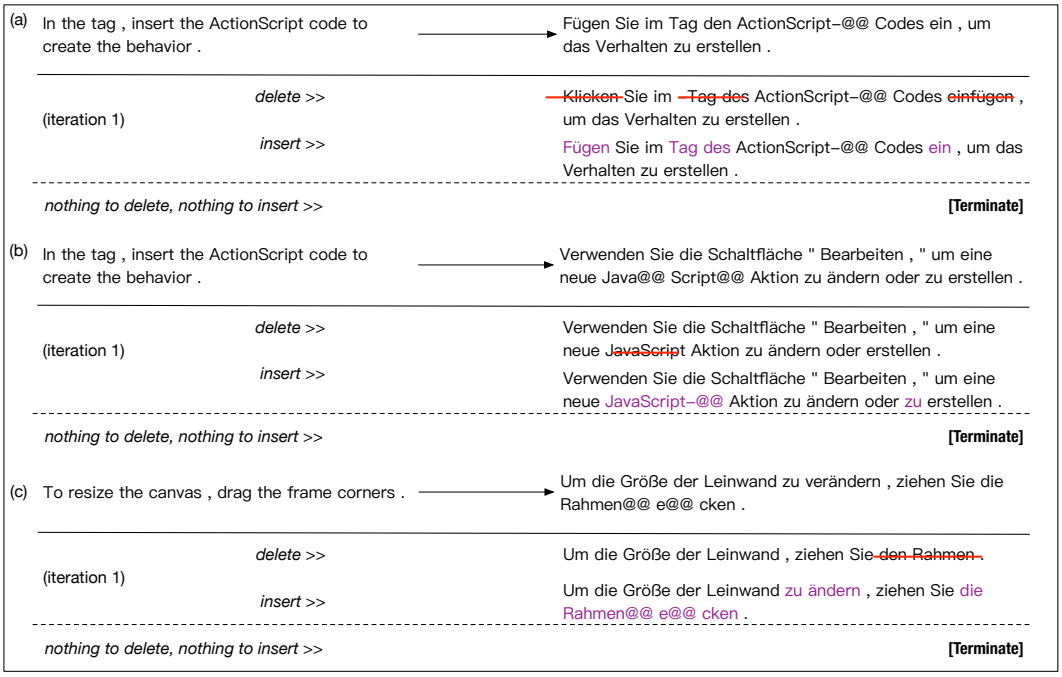

Figure 10: Post-editing examples for WMT'17-APE En-De with the Levenshtein Transformer.

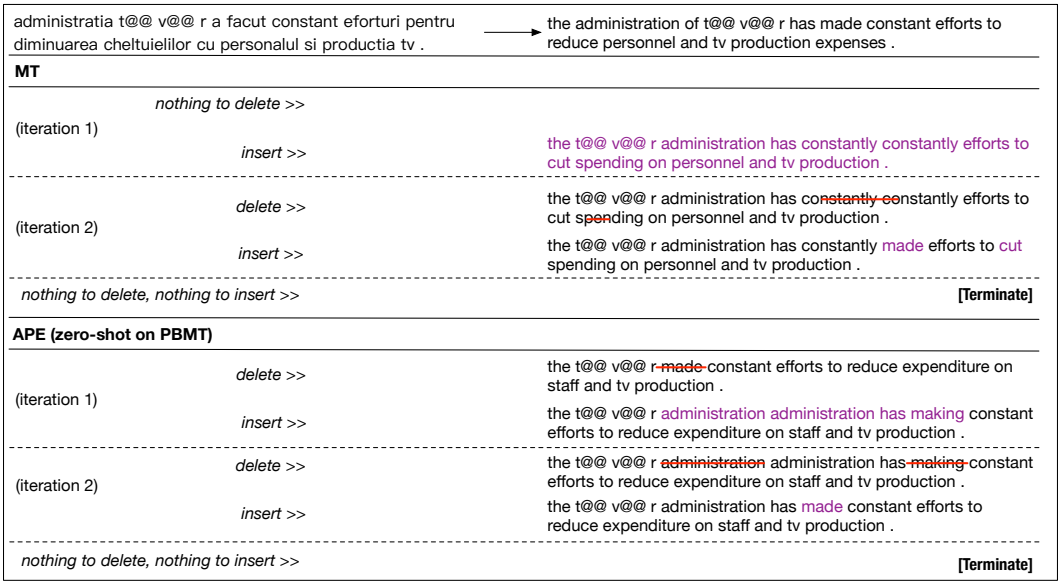

Figure 11: An example for machine translation and zero-shot post-editing over a PBMT system's output on WMT'16 Ro-En with the Levenshtein Transformer (LevT) trained for MT. It is clear to find that, the pre-trained LevT can directly adapt to the PBMT's output and have a different refinement results compared to translate from scratch.