Towards Example-Based NMT with Multi-Levenshtein Transformers
Maxime Bouthors, Josep Crego, François Yvon

To cite this version:

HAL Id: hal-04332427
https://hal.science/hal-04332427
Submitted on 8 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - ShareAlike 4.0 International License
Towards Example-Based NMT with Multi-Levenshtein Transformers

Maxime Bouthors♡,♣, Josep Crego♣, and François Yvon♡

♡Sorbonne Université, CNRS, ISIR, F-75005 Paris, France
firstname.lastname@isir.upmc.fr
♣Systran, 5 rue Feydeau, F-75002 Paris, France
firstname.lastname@systrangroup.fr

Abstract

Retrieval-Augmented Machine Translation (RAMT) is attracting growing attention. This is because RAMT not only improves translation metrics, but is also assumed to implement some form of domain adaptation. In this contribution, we study another salient trait of RAMT, its ability to make translation decisions more transparent by allowing users to go back to examples that contributed to these decisions. For this, we propose a novel architecture aiming to increase this transparency. This model adapts a retrieval-augmented version of the Levenshtein Transformer and makes it amenable to simultaneously edit multiple fuzzy matches found in memory. We discuss how to perform training and inference in this model, based on multi-way alignment algorithms and imitation learning. Our experiments show that editing several examples positively impacts translation scores, notably increasing the number of target spans that are copied from existing instances.

1 Introduction

Neural Machine Translation (NMT) has become increasingly efficient and effective thanks to the development of ever larger encoder-decoder architectures relying on Transformer models (Vaswani et al., 2017). Furthermore, these architectures can readily integrate instances retrieved from a Translation Memory (TM) (Bulte and Tezcan, 2019; Xu et al., 2020; Hoang et al., 2022), thereby improving the overall consistency of new translations compared to past ones. In such context, the autoregressive and generative nature of the decoder can make the process (a) computationally inefficient when the new translation has very close matches in the TM; (b) practically ineffective, as there is no guarantee that the output translation, regenerated from scratch, will resemble that of similar texts.

An alternative that is attracting growing attention is to rely on computational models tailored to edit existing examples and adapt them to new source sentences, such as the Levenshtein Transformer (LevT) model of Gu et al. (2019). This model can effectively handle fuzzy matches retrieved from memories, performing minimal edits wherever necessary. As decoding in this model occurs non-autoregressive, it is likely to be computationally more efficient. More important for this work, the reuse of large portions of existing translation examples is expected to yield translations that (a) are more correct; (b) can be transparently traced back to the original instance(s), enabling the user to inspect the edit operations that were performed. To evaluate claim (a) we translate our test data (details in Section 5.1) using a basic implementation of a retrieval-augmented LevT with TM (TM-LevT). We separately compute the modified unigram and bigram precisions (Papineni et al., 2002) for tokens that are copied from the fuzzy match and tokens that are generated by the model.1 We observe that copies account for the largest share of output units and have better precision (see Table 1).

Based on this observation, our primary goal is to optimize further the number of tokens copied from the TM. To do so, we propose simultaneously editing multiple fuzzy matches retrieved from memory, using a computational architecture – Multi-LevT,

<table>
<thead>
<tr>
<th>unigram</th>
<th>precision</th>
<th>% units</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>87.5</td>
<td>64.9</td>
</tr>
<tr>
<td>gen</td>
<td>52.6</td>
<td>35.1</td>
</tr>
<tr>
<td>bigram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy-copy</td>
<td>81.4</td>
<td>55.0</td>
</tr>
<tr>
<td>copy-gen</td>
<td>40.1</td>
<td>8.9</td>
</tr>
<tr>
<td>gen-copy</td>
<td>39.5</td>
<td>10.7</td>
</tr>
<tr>
<td>gen-gen</td>
<td>34.2</td>
<td>25.4</td>
</tr>
</tbody>
</table>

Table 1: Modified precision of copy vs. generated unigrams and bigrams for TM-LevT. For bigrams, we consider four cases: bigrams made of two copy tokens, two generated tokens, and one token of each type.

1Copies and generations are directly inferred from the sequence of edit operations used to compute the output sentence.
or $\text{TML}^N$-LevT for short – which extends $\text{TML}$-LevT to handle several initial translations. The benefit is twofold: (a) an increase in translation accuracy; (b) more transparency in the translation process. Extending $\text{TML}$-LevT to $\text{TML}^N$-LevT however requires solving multiple algorithmic and computational challenges related to the need to compute Multiple String Alignments (MSAs) between the matches and the reference translation, which is a notoriously difficult problem; and designing appropriate training procedures for this alignment module.

Our main contributions are the following:

1. a new variant of the LevT model that explicitly maximizes target coverage (§4.2);
2. a new training regime to handle an extended set of editing operations (§3.3);
3. two novel multiway alignment (§4.2) and re-alignment (§6.2) algorithms;
4. experiments in 11 domains where we observe an increase of BLEU scores, COMET scores, and the proportion of copied tokens (§6).

Our code and experimental configurations are available on github.²

## 2 Preliminaries / Background

### 2.1 TM-based machine translation

Translation Memories, storing examples of past translations, is a primary component of professional Computer Assisted Translation (CAT) environments (Bowker, 2002). Given a translation request for source sentence $x$, TM-based translation is a two-step process: (a) retrieval of one or several instances $(\tilde{x}, \tilde{y})$ whose source side resembles $x$, (b) adaptation of retrieved example(s) to produce a translation. In this work, we mainly focus on step (b), and assume that the retrieval part is based on a fixed similarity measure $\Delta$ between $x$ and stored examples. In our experiments, we use:

$$\Delta(x, \tilde{x}) = 1 - \frac{\text{ED}(x, \tilde{x})}{\max(|x|, |\tilde{x}|)}, \quad (1)$$

with $\text{ED}(x, \tilde{x})$ the edit distance between $x$ and $\tilde{x}$ and $|x|$ the length of $x$. We only consider TM matches for which $\Delta$ exceeds a predefined threshold $\tau$ and filter out the remaining ones. The next step, adaptation, is performed by humans with CAT tools. Here, we instead explore ways to perform this step automatically, as in Example-Based MT (Nagao, 1984; Somers, 1999; Carl et al., 2004).

### 2.2 Adapting fuzzy matches with LevT

The Levenshtein transformer of Gu et al. (2019) is an encoder-decoder model which, given a source sentence, predicts edits that are applied to an initial translation in order to generate a revised output (Figure 1). The initial translation can either be empty or correspond to a match from a TM. Two editing operations – insertion and deletion – are considered. The former is composed of two steps: first, placeholder insertion, which predicts the position and number of new tokens; second, the predictions of tokens to fill these positions. Editing operations are applied iteratively in rounds of refinement steps until a final translation is obtained.

In LevT, these predictions rely on a joint encoding of the source and the current target and apply in parallel for all positions, which makes LevT a representative of non-autoregressive translation (NAT) models. As editing operations are not observed in the training data, LevT resorts to Imitation Learning, based on the generation of decoding configurations for which the optimal prediction is easy to compute. Details are in (Gu et al., 2019), see also (Xu and Carpuat, 2021), which extends it with a repositioning operation and uses it to decode with terminology constraints, as well as the studies of Niwa et al. (2022) and Xu et al. (2023) who also explore the use of LevT in conjunction with TMs.

![Figure 1: First decoding pass of TML-LevT, a variant of LevT augmented with Translation Memories.](https://github.com/Maxwell1447/fairseq/)

### 2.3 Processing multiple fuzzy matches

One of the core differences between $\text{TML}^N$-LevT and LevT is its ability to handle multiple matches. This
implies adapting the edit steps (in inference) and the roll-in policy (in imitation learning).

**Inference in TM\textsuperscript{N}-LevT** Decoding follows the same key ideas as for LevT (see Figure 1) but enables co-editing an arbitrary number \( N \) of sentences. Our implementation (1) applies deletion, then placeholder insertion simultaneously on each retrieved example; (2) combines position-wise all examples into one single candidate sentence; (3) performs additional steps as in LevT: this includes first completing token prediction, then performing Iterative Refinement operations that edit the sentence to correct mistakes and improve it (§3.2).

**Training in TM\textsuperscript{N}-LevT** TM\textsuperscript{N}-LevT is trained with imitation learning and needs to learn the edit steps described above for both the first pass (1–2) and the iterative refinement steps (3). This means that we teach the model to perform the sequence of correct edit operations needed to iteratively generate the reference output, based on the step-by-step reproduction of what an expert ‘teacher’ would do. For this, we need to compute the optimal operation associated with each configuration (or state) (§4). The roll-in and roll-out policies specify how the model is trained (§3.3).

### 3 Multi-Levenshtein Transformer

#### 3.1 Global architecture

TM\textsuperscript{N}-LevT has two modes of operations: (a) the combination of multiple TM matches into one single sequence through alignment, (b) the iterative refinement of the resulting sequence. In step (a), we use the Transformer encoder-decoder architecture, extended with additional embedding and linear layers (see Figure 2) to accommodate multiple matches. In each of the \( N \) retrieved instances \( y = (y_1, \ldots, y_N) \), \( y_{n,i} \) (the \( i \)th token in the \( n \)th instance) is encoded as \( E_{y_{n,i}} + P_i + S_n \), where \( E \in \mathbb{R}^{|V| \times d_{model}} \), \( P \in \mathbb{R}^{L_{max} \times d_{model}} \) and \( S \in \mathbb{R}^{(N+1) \times d_{model}} \) are respectively the token, position and sequence embeddings. The sequence embedding identifies TM matches, and the positional encodings are reset for each \( y_n \). The extra row in \( S \) is used to identify the results of the combination and will yield a different representation for these single sequences.\(^3\) Once embedded, TM matches are concatenated and passed through multiple Transformer blocks, until reaching the last layer, which outputs \( (h_1, \ldots, h_{|y|}) \) for a single input match or \( (h_{1,1}, \ldots, h_{1,|y|_1}, \ldots, h_{N,1}, \ldots, h_{N,|y|_N}) \) in the case of multiple ones. The learned policy \( \pi_0 \) computes its decisions from these hidden states. We use four classifiers, one for each sub-policy:

1. **deletion**: predicts keep or delete for each token \( y_{n,i}^{del} \) with a projection matrix \( A \in \mathbb{R}^{2 \times d_{model}} \):
   \[
   \pi_0^{del}(d|n,i,y_1^{del}, \ldots, y_N^{del}; x) = \text{softmax}(h_{n,i}A^T)
   \]

2. **insertion**: predicts the number of placeholder insertions between \( y_{n,i}^{plh} \) and \( y_{n,i+1}^{plh} \) with a projection matrix \( B \in \mathbb{R}^{(K_{max}+1) \times 2d_{model}} \):
   \[
   \pi_0^{plh}(p|n,i,y_1^{plh}, \ldots, y_N^{plh}; x) = \text{softmax}(h_{n,i} [h_{n,i},h_{n,i+1}] B^T),
   \]
   with \( K_{max} \) the max number of insertions.

3. **combination**: predicts if token \( y_{n,i}^{cmb} \) in sequence \( n \) must be kept in the combination, with a projection matrix \( C \in \mathbb{R}^{2 \times d_{model}} \):
   \[
   \pi_0^{cmb}(c|n,i,y_1^{cmb}, \ldots, y_N^{cmb}; x) = \text{softmax}(h_{n,i}C^T).
   \]

4. **prediction**: predicts a token in vocabulary \( V \) at each placeholder position, with a projection matrix \( D \in \mathbb{R}^{|V| \times d_{model}} \):
   \[
   \pi_0^{tok}(t|n,i,y_{tok}; x) = \text{softmax}(h_{j}D^T)
   \]

Except for step 3, these classifiers are similar to those used in the original LevT.
3.2 Decoding

Decoding is an iterative process: in a first pass, the \( N \) fuzzy matches are combined to compute a candidate translation; then, as in LevT, an additional series of iterative refinement rounds (Gu et al., 2019) is applied until convergence or timeout. Figure 3 illustrates the first pass, where \( N = 2 \) matches are first edited in parallel, then combined into one output.

To predict deletions (resp. insertions and token predictions), we apply the argmax operator to \( \pi_{\text{del}}^\theta \) (resp. \( \pi_{\text{plh}}^\theta \), \( \pi_{\text{tok}}^\theta \)). For combinations, we need to aggregate separate decisions \( \pi_{\text{del}}^\theta \) (one per token and match) into one sequence. For this, at each position, we pick the most likely token.

During iterative refinement, we bias the model towards generating longer sentences since LevT outputs tend to be too short (Gu et al., 2019). As in LevT, we add a penalty to the probability of inserting 0 placeholder in \( \pi_{\text{plh}}^\theta \) (Stern et al., 2019). This only applies in the refinement steps to avoid creating more misalignments (see §6.2).

3.3 Imitation learning

We train TM\(^N\)-LevT with Imitation Learning (Daumé et al., 2009; Ross et al., 2011), teaching the system to perform the right edit operation for each decoding state. As these operations are unobserved in the training data, the standard approach is to simulate decoding states via a roll-in policy: for each of these, the optimal decision is computed via an expert policy \( \pi_* \), composed of intermediate experts \( \pi_{\text{del}}^\epsilon \), \( \pi_{\text{plh}}^\epsilon \), \( \pi_{\text{tok}}^\epsilon \). The notion of optimality is discussed in §4. Samples of pairs (state, decision) are then used to train the system policy \( \pi_\theta \).

---

1. **Additional triplets**\( ^\dagger \): \( \pi_{\text{rnd-del-N}} \) turns \( y_* \) into \( N \) random substrings, which simulates the edition of \( N \) artificial examples.

2. **Token selection**\( ^\dagger \): our expert policy never aligns two distinct tokens at a given position (§4.3). We simulate such cases that may occur at inference, as follows: with probability \( \gamma \), each \( \langle \text{PLH} \rangle \) is replaced with a random token from fuzzy matches (Figure 5).

The expert always completes its translation in one decoding pass. Policies used in iterative refinement are thus trained with the following simulated states, based on roll-in and roll-out policies used in LevT and its variants (Gu et al., 2019; Xu et al., 2023; Zheng et al., 2023):

3. **Add missing words** (uses \( \pi_{\text{rnd-del-1}} \)): with probability \( \alpha \), \( y_{\text{post-plh}} = y_* \). With probability \( 1 - \alpha \), generate a subsequence \( y_{\text{post-plh}} \) with length sampled uniformly in \([0, |y_*|] \).

4. **Correct mistakes** (uses \( \pi_{\text{tok}}^\epsilon \)): using the output of token prediction \( y_{\text{post-del}} \), teach the model to erase the wrongly predicted tokens.

5. **Remove extra tokens**\( ^\dagger \): \( \pi_{\text{ins}}^\epsilon \), \( \pi_{\text{plh}}^\epsilon \): insert placeholders in \( y_{\text{post-tok}} \) and predict tokens, yielding \( y_{\text{post-del-extra}} \), which trains the model to delete wrong tokens. These sequences differ from case (4) in the way \( \langle \text{PLH} \rangle \) are inserted.

6. **Predict token**\( ^\dagger \): (uses \( \pi_{\text{masks}}^\epsilon \)): each token in \( y_* \) is replaced by \( \langle \text{PLH} \rangle \) with probability \( \varepsilon \). As token prediction applies for both decoding steps, these states also improve the first pass.

The expert decisions (e.g. inserting deleted tokens like in state (3)) or deleting wrongly predicted
Figure 4: Roll-in policies used in training. Blue cells contain sets of target sentences (e.g. \( y^{\text{del}} \)), optionally associated with the expert prediction (e.g. \( \hat{y}^{\text{del}} \)). Model’s predictions are in Thistle and circumflexed (e.g. \( \hat{y} \)). Pairs of model / expert predictions are summed in the loss: (\( \text{del}, \hat{\text{del}} \)), (\( \text{plh}, \hat{\text{plh}} \)), (\( \text{cmb}, \hat{\text{cmb}} \)), (\( \text{post} \cdot \text{del}, \hat{\text{post}} \cdot \hat{\text{del}} \)), (\( \text{post} \cdot \hat{\text{plh}}, \text{post} \cdot \hat{\text{plh}} \)), (\( \text{post} \cdot \hat{\text{del}} \cdot \hat{\text{extra}}, \hat{\text{post}} \cdot \hat{\text{plh}} \cdot \hat{\text{extra}} \)). "post" denotes policies applied in refinement steps.

Figure 5: Noising \( y^{\text{cmb}} \) with \( \pi^{\text{el}} \) using tokens from \( y^{\text{init}} = (y_1, \cdots, y_N) \).

tokens in state (4) associated with most states are obvious, except for the initial state and state (5), which require an optimal alignment computation.

4 Optimal Alignment

Training the combination operation introduced above requires specifying the expert decision for each state. While LevT derives its expert policy \( \pi_e \) from the computation of edit distances, we introduce another formulation based on the computation of maximal covers. For \( N=1 \), these formulations can be made equivalent\(^5\) (Gusfield, 1997).

4.1 N-way alignments

We formulate the problem of optimal editing as an N-way alignment problem (see figure 6) that we define as follows. Given \( N \) examples \( (y_1, \cdots, y_N) \) and the target sentence \( y_n \), a N-way alignment of \( (y_1, \cdots, y_N) \) w.r.t. \( y_n \) is represented as a bipartite graph \((V, V_s, E)\), where \( V \) is further partitioned into \( N \) mutually disjoint subsets \( V_1 \cdots V_N \). Vertices in each \( V_n \) (resp. \( V_s \)) correspond to tokens in \( y_n \) (resp. \( y_s \)). Edges \( (n, i, j) \in E \) connect node \( i \) in \( V_n \) to node \( j \) in \( V_s \). An N-way alignment satisfies properties (i)-(ii):

(i) Edges connect identical (matching) tokens: 
\[(n, i, j) \in E \Rightarrow y_{n,i} = y_{s,j} \.\]

(ii) Edges that are incident to the same subset \( V_n \) do not cross: 
\[(n, i, j), (n, i', j') \in E \Rightarrow (i' - i)(j' - j) > 0 \.\]

An optimal N-way alignment \( E_s \) maximizes the coverage of tokens in \( y_s \), then the total number of edges, where \( y_{s,j} \) is covered if there exists at least one edge \((n, i, j) \in E \). Denoting \( \mathbb{E} \) the set of alignments maximizing target coverage:

\[ E = \arg \max_{E} \{ |y_{s,j} : \exists (n, i), (n, i, j) \in E \} \.\]

\[ E_s = \arg \max_{E \in \mathbb{E}} |E| \.\]

4.2 Solving optimal alignment

Computing the optimal N-way alignment is NP-hard (see Appendix D). This problem can be solved using Dynamic Programming (DP) techniques similar to Multiple Sequence Alignment (MSA) (Carrillo and Lipman, 1988) with a complexity \( O(N \cdot y_s \cdot \prod_{n=1}^{N} |y_n|) \). We instead implemented the following two-step heuristic approach:

1. separately compute alignment graphs between each \( y_n \) and \( y_s \), then extract \( k \)-best 1-way alignments \( \{E_{n,1} \cdots E_{n,k}\} \). This requires time \( O(k \cdot |y_n| \cdot |y_s|) \) using DP (Gusfield, 1997);

\[^{5}\text{When the cost of replaces is higher than insertion + deletion. This is the case in the original LevT code.}\]
2. search for the optimal recombination of these graphs, selecting 1-way alignments \((E_1,k_1)\ldots(E_N,k_N)\) to form \(E_* = \bigcup_n E_{n,k_n}\). Assuming \(N\) and \(k\) are small, we perform an exhaustive search in \(O(k^N)\).

### 4.3 From alignments to edits

From an alignment \((V,V_*E)\), we derive the optimal edits needed to compute \(y_*\), and the associated intermediary sequences. Edges in \(E\) indicate the tokens that are preserved throughout this process:

1. **deletion**: \(\forall n, \forall i, \forall n,i,j \exists (n,i,j) \in E\) for some \(j\); otherwise it is deleted. The resulting sequences are \(\{y_{plh}^n\}_{n=1\ldots N}\).

2. **insertion**: Placeholders are inserted between successive tokens in all \(y_{plh}^n\), resulting in the set \(\{y_{cmb}^n\}_{n=1\ldots N}\), under the constraints that (a) all \(y_{cmb}^n\) have the same length as \(y_*\) and (b) non-placeholder tokens \(y_{cmb}^n\) are equal to the reference token \(y_{*,i}\).

3. **combination**: Sequences \(\{y_{cmb}^n\}_{n=1\ldots N}\) are combined into \(y_{tok}\) such that for each position \(i\), \(y_{cmb}^n_{r,i} \neq <PLH> \Rightarrow y_{tok}^i = y_{cmb}^n_{r,i}\).

If \(\forall n, y_{cmb}^n = <PLH>\), then \(y_{tok}^i = <PLH>\).

4. **prediction**: The remaining \(<PLH>\) symbols in \(y_{tok}\) are replaced by the corresponding target token in \(y_*\) at the same position.

The expert policy \(\pi_*\) edits examples \(y_1,\ldots,y_N\) into \(y_*\) based on the optimal alignment \((V,V_*E)\).

#### 5 Experiments

5.1 Data and metrics

We focus on translation from English to French and consider multiple domains. This allows us to consider a wide range of scenarios, with a varying density of matching examples: our datasets include ECB, EMEA, Europarl, GNOME, JRC-Acquis, KDE4, PHP, Ubuntu, where high-quality matches are often available, but also News-Commentary, TED2013, and Wikipedia, where matches are more scarce (see Table 6, §B).

For each training sample \((x,y)\), we retrieve up to 3 in-domain matches. We filter matches \(x_n\) to keep only those with \(\Delta(x,x_n) > 0.4\). We then manually split each of the 11 datasets into train, valid, test-0.4, test-0.6, where the valid and test sets contain 1,000 lines each. test-0.4 (resp. test-0.6) contains samples whose best match is in the range \([0.4,0.6]\) (resp. \([0.6,1]\)). As these two test sets are only defined based on the best match score, it may happen that some test instances will only retrieve 1 or 2 close matches (statistics are in Table 6).

For the pre-training experiments (§6.2), we use a subsample of 2M random sentences from WMT’14. For all data, we use Moses tokenizer and 32k BPEs trained on WMT’14 with SentencePiece (Kudo, 2018). We report BLEU scores (Papineni et al., 2002) and ChrF scores (Popovi´c, 2015) as computed by SacreBLEU (Post, 2018) and COMET scores (Rei et al., 2020).

5.2 Architecture and settings

Our code\(^6\) extends Fairseq\(^7\) implementation of LevT in many ways. It uses Transformer models (Vaswani et al., 2017) (parameters in Appendix A). Roll-in policy parameters (§3.3) are empirically set as: \(\alpha=0.3, \beta=0.2, \gamma=0.2, \delta=0.2, \varepsilon=0.4\). The AR baseline uses OpenNMT (Klein et al., 2017) and uses the same data as TM-LevT (Appendix A).

### 6 Results

6.1 The benefits of multiple matches

We compare two models in Table 2: one trained with one TM match, the other with three. Each

---

\(^6\)https://github.com/Maxwell11447/fairseq

\(^7\)https://github.com/facebookresearch/fairseq
Table 2: BLEU/ChrF and COMET scores on the full test set. All BLEU/ChrF differences are significant ($p = 0.05$).

<table>
<thead>
<tr>
<th>Model \N</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>4,719</td>
<td>2,369</td>
<td>14,912</td>
<td>22,000</td>
</tr>
<tr>
<td>TM$^1$-LevT</td>
<td>48.8/63.6</td>
<td>48.7/63.0</td>
<td>55.0/68.4</td>
<td>52.0/66.8</td>
</tr>
<tr>
<td>19.6</td>
<td>26.2</td>
<td>41.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM$^3$-LevT</td>
<td>46.6/64.1</td>
<td>50.0/65.8</td>
<td>56.0/69.3</td>
<td>53.0/67.5</td>
</tr>
<tr>
<td>14.0</td>
<td>16.0</td>
<td>38.2</td>
<td>30.8</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Proportion of unigrams and bigram from a given origin (copy vs. generation) for various models.

We report the performance of systems trained using $N=1, 2, 3$ for each domain and test set in Table 4 (BLEU) and 12 (COMET). We see comparable average BLEU scores for $N=1$ and $N=3$, with large variations across domains, from which we conclude that: (a) using 3 examples has a smaller return when the best match is poor, meaning that bad matches are less likely to help (test-0.4 vs. test-0.6); (b) using 3 examples seems advantageous for narrow domains, where training actually exploits several close matches (see also Appendix E). We finally note that COMET scores$^9$ for TM$^3$-LevT are always slightly lower than for TM$^1$-LevT, which prompted us to develop several extensions.

6.2 Improving TM$^N$-LevT

Realignment In preliminary experiments, we observed that small placeholder prediction errors in the first decoding pass could turn into catastrophic misalignments (Figure 7). To mitigate such cases, we introduce an additional realignment step during inference, where some predicted placeholders are added/removed if this improves the global alignment. Realignment is formulated as an optimization problem aimed to perform a tradeoff between the score $-\log \pi_{plh}$ of placeholder insertion and an alignment cost (see Appendix C).

We assess realignment for $N=3$ (Tables 4 and 12) and observe small, yet consistent average gains (+0.2 BLEU, +1.5 COMET) for both test sets.

Pre-training Another improvement uses pre-training with synthetic data. For each source/target pair $(x, y)$ in the pre-training corpus, we simulate $N$ fuzzy matches by extracting from $y$ $N$ substrings $y_n$ of length $\approx |y| \cdot r$, with $r \in [0, 1]$. Each $y_n$ is then augmented as follows:

1. We randomly insert placeholders to increase the length by a random factor between 1 and $1 + f$, $f = 0.5$ in our experiments.
2. We use the CamemBERT language model (Martin et al., 2020) to fill the masked tokens.

These artificial instances simulate diverse fuzzy matches and are used to pre-train a model, using the same architecture and setup as in §5.2. Pre-training yields markedly higher scores than the baseline (+1.3 BLEU, +6.4 COMET for test-0.4 and +0.9 BLEU, +4.6 COMET for test-0.6). Training curves also suggest that pre-trained models are faster to converge. Combining with realignment yields additional gains for TM$^3$-LevT, which outperforms TM$^1$-LevT in all domains and both metrics.
We evaluate the impact of the various elements in the mixture roll-in policy via an ablation study (Table 13). Except for $\pi^{sel}$, every new element in the roll-in policy increases performance. As for $\pi^{sel}$, our system seems to be slightly better with than without. An explanation is that, in case of misalignment, the model is biased towards selecting the first, most similar example sentence. As an ablation, instead of aligning by globally maximizing coverage (§ 4.2), we also compute alignments that maximize coverage independently as in figure 6a.

A complete run of $TM^N$-LevT is in Appendix F.

### Table 4: Per domain BLEU scores for $TM$-LevT, $TM^N$-LevT and variants. Bold (resp. italic) for scores significantly higher (resp. lower) than $TM^N$-LevT ($p = 0.05$). $p$-values from SacreBLEU paired bootstrap resampling ($n = 1000$). The Autoregressive (AR) system is our implementation of (Bulte and Tezcan, 2019).

<table>
<thead>
<tr>
<th>Domain</th>
<th>$test-0.4$</th>
<th>$test-0.6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>63.0</td>
<td>63.6</td>
</tr>
<tr>
<td>EME</td>
<td>43.6</td>
<td>46.3</td>
</tr>
<tr>
<td>Epp</td>
<td>69.3</td>
<td>75.1</td>
</tr>
<tr>
<td>GNO</td>
<td>71.4</td>
<td>75.1</td>
</tr>
<tr>
<td>JRC</td>
<td>62.8</td>
<td>65.7</td>
</tr>
<tr>
<td>KDE</td>
<td>28.8</td>
<td>31.8</td>
</tr>
<tr>
<td>News</td>
<td>42.1</td>
<td>45.2</td>
</tr>
<tr>
<td>PHP</td>
<td>40.2</td>
<td>43.3</td>
</tr>
<tr>
<td>TED</td>
<td>42.2</td>
<td>45.2</td>
</tr>
<tr>
<td>Ubu Wiki</td>
<td>59.1</td>
<td>59.8</td>
</tr>
<tr>
<td>all</td>
<td>55.8</td>
<td>58.3</td>
</tr>
</tbody>
</table>

### Table 5: Ablation study. We build models with variable roll-in policies: $sel$: no random selection noise ($\gamma=0$); $del_x$: no extra deletion; $rd-del$: no random deletion ($\beta=0$); $mask$: no random mask ($\delta=0$); $dum-plh$: never start with $y_{post-del}=y_*$ ($\alpha=0$); $indep-align$: alignments are independent. Full results in Appendix F.

<table>
<thead>
<tr>
<th>Roll-in Policy</th>
<th>$test-0.4$</th>
<th>$test-0.6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TM^N$-LevT</td>
<td>46.5</td>
<td>60.1</td>
</tr>
<tr>
<td>$sel$</td>
<td>46.2</td>
<td>60.0</td>
</tr>
<tr>
<td>$del_x$</td>
<td>44.8</td>
<td>58.6</td>
</tr>
<tr>
<td>$rd-del$</td>
<td>38.6</td>
<td>51.9</td>
</tr>
<tr>
<td>$mask$</td>
<td>46.0</td>
<td>59.0</td>
</tr>
<tr>
<td>$dum-plh$</td>
<td>41.0</td>
<td>50.9</td>
</tr>
<tr>
<td>$indep-align$</td>
<td>42.6</td>
<td>56.4</td>
</tr>
</tbody>
</table>

7 Related Work

As for other Machine Learning applications, such as text generation (Guu et al., 2018), efforts to integrate a retrieval component in neural-based MT have intensified in recent years. One motivation is to increase the transparency of ML models by providing users with tangible traces of their internal computations in the form of retrieved examples (Rudin, 2019). For MT, this is achieved by integrating fuzzy matches retrieved from memory as an additional conditioning context. This can be performed simply by concatenating the retrieved target instance to the source text (Bulte and Tezcan, 2019), an approach that straightforwardly accommodates several TM matches (Xu et al., 2020), or the simultaneous exploitation of their source and target sides (Pham et al., 2020). More complex schemes to combine retrieved examples with the source sentence are in (Gu et al., 2018; Xia et al.,...
The recent work of Cheng et al. (2022) handles multiple complementary TM examples retrieved in a contrastive manner that aims to enhance source coverage. Cai et al. (2021) also handle multiple matches and introduce two novelties: (a) retrieval is performed in the target language and (b) similarity scores are trainable, which allows to evaluate retrieved instances based on their usefulness in translation. Most of these attempts rely on an auto-regressive (AR) decoder, meaning that the impact of TM match(es) on the final output is only indirect.

The use of TM memory match with a NAT decoder is studied in (Niwa et al., 2022; Xu et al., 2023; Zheng et al., 2023), which adapt LevT for this specific setting, using one single retrieved instance to initialize the edit-based decoder. Other evolutions of LevT, notably in the context of constraint decoding, are in (Susanto et al., 2020; Xu and Carpuat, 2021), while a more general account of NAT systems is in (Xiao et al., 2023).

Zhang et al. (2018) explore a different set of techniques to improve translation using retrieved segments instead of full sentences. Extending KNN-based language models (He et al., 2021a) to the conditional case, Khandelwal et al. (2021) proposes k-nearest neighbor MT by searching for target tokens that have similar contextualized representations at each decoding step, an approach further elaborated by Zheng et al. (2021); Meng et al. (2022) and extended to chunks by Martins et al. (2022).

8 Conclusion and Outlook

In this work, we have extended the Levenshtein Transformer with a new combination operation, making it able to simultaneously edit multiple fuzzy matches and merge them into an initial translation that is then refined. Owing to multiple algorithmic contributions and improved training schemes, we have been able to (a) increase the number of output tokens that are copied from retrieved examples; (b) obtain performance improvements compared to using one single match. We have also argued that retrieval-based NMT was a simple way to make the process more transparent for end users.

Next, we would like to work on the retrieval side of the model: first, to increase the diversity of fuzzy matches e.g. thanks to contrastive retrieval, but also to study ways to train the retrieval mechanism and extend this approach to search monolingual (target side) corpora. Another line of work will combine our techniques with other approaches to TM-based NMT, such as keeping track of the initial translation(s) on the encoder side.

9 Limitations

As this work was primarily designed a feasibility study, we have left aside several issues related to performance, which may explain the remaining gap with published results on similar datasets. First, we have restricted the encoder to only encode the source sentence, even though enriching the input side with the initial target(s) has often been found to increase performance (Bulte and Tezcan, 2019), also for NAT systems (Xu et al., 2023). It is also likely that increasing the number of training epochs would yield higher absolute scores (see Appendix F).

These choices were made for the sake of efficiency, as our training already had to fare with the extra computing costs incurred by the alignment procedure required to learn the expert policy. Note that in comparison, the extra cost of the realignment procedure is much smaller, as it is only paid during inference and can be parallelized on GPUs. We would also like to outline that our systems do not match the performance of an equivalent AR decoder, a gap that remains for many NAT systems (Xiao et al., 2023). Finally, we have only reported here results for one language pair – favoring here domain diversity over language diversity – and would need to confirm the observed improvements on other language pairs and conditions.

10 Acknowledgements

This work was performed using HPC resources from GENCI-IDRIS (Grant 2022-AD011013583) and Lab-IA from Saclay-IA.

The authors wish to thank Dr. Jitao Xu and Dr. Caio Corro for their guidance and help.

References


We use a Transformer architecture with embeddings of dimension 512; feed-forward layers of size 2048; number of heads 8; number of encoder and decoder layers: 6; batch size: 3000 tokens; shared-embeddings; dropout: 0.3; number of GPUs: 6. The maximal number of additional placeholders is $K_{max} = 64$.

During training, we use Adam optimizer with $(\beta_1, \beta_2) = (0.9, 0.98)$; inverse sqrt scheduler; learning rate: $5e^{-4}$; label smoothing: 0.1; warmup updates: 10,000; float precision: 16. We fixed the number of iterations at 60k. For decoding, we use iterative refinement with an empty placeholder penalty of 3, and a max number of iterations of 10 (Gu et al., 2019).

For the $n$-way alignment (§4.1), we use $k=10$.

The hyper-parameters of the realigner (§C) were tuned on a subset of 1k samples extracted from the ECB training set.

Metrics are used with default settings: SacreBLEU signature is nrefs:1|case:mixed|eff:no |tok:13a|smooth:exp|version:2.1.0; the ChrF signature is nrefs:1|case:mixed|eff:yes |nc:6|nw:0|space:no|version:2.1.0; as for COMET we use the default model of version 1.1.3: Unbabel/wmt22-comet-da.

B Data Analysis

Table 6 contains statistics about all 11 domains. They notably highlight the relationship between the average number of retrieved sentences during training and the ability of $\text{TM}^3$-LevT to perform better than $\text{TM}^3$-LevT in Table 4. The domains with retrieval rates lesser than 1 (Epp, News, TED, Ubu) have quite a broad content, meaning that training instances have fewer close matches, which also means that for these domains, $\text{TM}^3$-LevT hardly sees two or three examples that it needs to use in inference.

C Realignment

The realignment process is an extra inference step aiming to improve the result of the placeholder insertion stage. To motivate our approach, let us consider the following sentences before placeholder insertion:

$$y_0^{plh} : < A B C > \times$$
$$y_1^{plh} : < B C > \times \times$$
$$y_2^{plh} : < A D C D >,$$

where letters represent tokens, $\times$ denotes padding, $<$ and $>$ respectively stand for $<$EOS$>$ and $<$EOP$>$.

The output of this stage is a prediction for all pairs of consecutive tokens. This prediction takes the form of a tensor $\log \pi_i^{plh}$ of dimensions $N \times (L-1) \times (K_{max}+1)$, corresponding respectively to the number of retrieved sentences $N$, the maximum sentence length $L$, and the maximum number of additional placeholders $K_{max}$.

Let $P$ (a $N \times (L-1)$ tensor) denote the arg max, e.g. $P = 0 0 1 0 0$

0 1 0 0 0

Inserting the prescribed number of placeholders (figured by _) then yields the following $y_{cmb}$:

$$y_0^{cmb} : < A B C _ _ >$$
$$y_1^{cmb} : < B C _ _ > \times \times$$
$$y_2^{cmb} : < A _ D C D >$$

This result is far from perfect, as it fails to align the repeated occurrences of $C$. For instance, a preferable alignment requiring 3 changes (1 change consists in a modification of $\pm 1$ in $P$) could be:

$$y_0^{cmb'} : < A B C _ > \times$$
$$y_1^{cmb'} : < _ B C _ > \times$$
$$y_2^{cmb'} : < A D C D > \times$$

The general goal of realignment is to improve such alignments by performing a small number of changes in $P$. We formalize this problem as a search for a good tradeoff between (a) the individual placeholder prediction scores, aggregated in $\mathcal{L}_L$ (likelihood loss) and (b) $\mathcal{L}_A$ an alignment loss. Under its simplest form, this problem is again an optimal multisequence alignment problem, for which exact dynamic programming solutions are computationally intractable in our setting.

We instead develop a continuous relaxation that can be solved effectively with SGD and is also easy to parallelize on GPUs. We, therefore, relax the integer condition for $P$ and assume that $P_{i,j}$ can take continuous real values in $[0, K_{max}]$, then solve the continuous optimization problem before turning $P_{i,j}$ values back into integers.

The likelihood loss aims to keep the $P_{i,j}$ values close to the model predictions. Denoting $(\mu, \sigma)$ respectively the mean and variance of the model
predictions, our initial version of this loss is

$$\mathcal{L}_L(P) = \sum_{i,j} \frac{(P_{i,j} - \mu_{i,j})^2}{2\sigma^2}.$$  

In practice, we found that using a weighted average $\bar{\mu}$ and clamping the variance $\sigma^2$ both yield better realignments, yielding:

$$\mathcal{L}_L(P) = \sum_{i,j} \frac{(P_{i,j} - \bar{\mu}_{i,j})^2}{2\sigma^2}.$$  

To define the alignment loss, we introduce a position matrix $X$ of dimension $N \times L$ in $\mathbb{R}^+$. Where $X_{n,i}$ corresponds to the (continuous)position of token $y_{n,i}$ after inserting a real number of placeholders. $X$ is defined as:

$$X_{n,i}(P) = i + \sum_{j<i} P_{n,j}$$  

with $i$ the number of tokens occurring before $X_{n,i}$ and $\sum_{j<i} P_{n,j}$ the cumulated sum of placeholders. Using $X$, we derive the distance tensor $D$ of dimension $N \times L \times N \times L$ in $\mathbb{R}^+$ as:

$$D_{n,i,m,j}(P) = |X_{n,i} - X_{m,j}|$$  

Finally, let $G$ be an $N \times L \times N \times L$ alignment graph tensor, where $G_{n,i,m,j} = 1$ if and only if $y_{n,i} = y_{m,j}$ and $n \neq m$ and $D_{n,i,m,j} < D_{\max}$. $G$ connects identical tokens in different sentences when their distance after placeholder insertion is at most $D_{\max}$. This last condition avoids perturbations from remote tokens that coincidentally appear to be identical.

Each token $y_{n,i}$ is associated with an individual loss:

$$d_{n,i}(P) = \begin{cases} 
\min_{m,j} \{D_{n,i,m,j}(P) : G_{n,i,m,j} = 1\} & \text{if } \exists (m,j) \text{ s.t. } G_{n,i,m,j} = 1 \\
0 & \text{otherwise}
\end{cases}$$  

The alignment loss aggregates these values over sentences and positions as:

$$\mathcal{L}_A(P) = \sum_{n=0}^{N-1} \sum_{i=0}^{L-1} d_{n,i}(P)$$  

A final ingredient in our realignment model is related to the final discretization step. To avoid rounding errors, we further constrain the optimization process to deliver near-integer solutions. For this, we also include a integer constraint loss defined as:

$$\mathcal{L}_{\text{int}}(P) = \mu_t \sum_{i,j} \sin^2(\pi P_{i,j})$$  

where $\mu_t$ controls the scale of $\mathcal{L}_{\text{int}}(P)$. As $x \rightarrow \sin^2(\pi x)$ reaches its minimum 0 for integer values, minimizing $\mathcal{L}_{\text{int}}(P)$ has the effect of enforcing a near-integer constraint to our solutions. Overall, we minimize in $P$:

$$\mathcal{L} = \mathcal{L}_L(P) + \mathcal{L}_A(P) + \mathcal{L}_{\text{int}}(P),$$  

slowly increasing the scale of $\mu_t$ according to the following schedule

$$\mu_t = \begin{cases} 
0 & \text{if } t < t_0 \\
\mu_T & \text{if } t > T \\
\mu_T \left(\frac{t-t_0}{T-t_0}\right)^2 & \text{otherwise}
\end{cases}$$  

with $t_0, T$ the timestamps for respectively the activation of the integer constraint loss, and the activation of the clamping. This optimization is performed with gradient descent directly on GPUs, with a small additional cost to the inference procedure.

### D NP-hardness of Coverage Maximization in N-way Alignment

Given the set of possible N-way alignments, the problem of finding the one that maximizes the target coverage is NP-hard. To prove it, we can reduce the NP-hard set cover problem (Garey and Johnson, 1979) to the N-way alignment coverage maximization problem.
• Cover set decision problem (A):

Let $X = \{x_1, \cdots, x_N\}$ and $C_0 \subset 2^X$. Is there $c^* = (c_1, \cdots, c_K) \in C_0^K$ s.t. $|\bigcup_k c_k^*| = |X|$?

• N-way alignment coverage maximization decision problem (B):

Let $X = \{x_1, \cdots, x_N\}$ and $C = (C_1, \cdots, C_K) \subset (2^X)^K$. For $p \in \mathbb{N}$, is there $c \in \prod_{k=1}^K C_k$ s.t. $|\bigcup_k c_k| \geq p$?

A solution of (B) can be certified in polynomial time: we simply compute the cardinal of a union. Any instance of (A) can be transformed in polynomial time and space into a special instance of (B) where all $C_k = C_0$ and $p = |X|$.

E Results for fr-en

Table 7 reports the BLEU scores for the reverse direction (fr→en), using exactly the same configuration as in Table 2. Note that since we used the same data split (retrieving examples based on the similarity in English), and since the retrieval procedure is asymmetrical, 4,749 test samples happen to have no match. That would correspond to an extra column labeled "0", which is not represented here.

<table>
<thead>
<tr>
<th>Model</th>
<th>size</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{TM}^1$-LevT</td>
<td>2.753</td>
<td>1.675</td>
<td>12.823</td>
<td>17.251</td>
<td></td>
</tr>
<tr>
<td>$\text{TM}^3$-LevT</td>
<td>57.2</td>
<td>57.6</td>
<td>61.5</td>
<td>60.2</td>
<td></td>
</tr>
<tr>
<td>$\text{TM}^3$-LevT +pt +ra</td>
<td>58.2</td>
<td>59.4</td>
<td>64.0</td>
<td>62.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: BLEU scores on the full test set. $\text{TM}^3$-LevT is improved with pre-training and realignment. All BLEU differences are significant ($p = 0.05$). $p$-values from SacreBLEU paired bootstrap resampling ($n = 1000$).

The reverse direction follows a similar pattern, providing further evidence of the method’s effectiveness.

F Complementary Analyses

Diversity and difficulty Results in Table 4 show that some datasets do not seem to benefit from multiple examples. This is notably the case for Europarl, News-Commentary, TED2013, and Ubuntu. We claim that this was due to the lack of retrieved examples at training (as stated in §B), of diversity, and the noise in fuzzy matches. To further investigate this issue, we report two scores in Table 8. The first is the increase of bag-of-word coverage of the target gained by using $N=3$ instead of $N=1$; the second is the increase of noise in the examples, computed as the proportion of tokens in the examples that do not occur in the target. We observe that, in fact, low diversity is often associated with poor scores for $\text{TM}^3$-LevT, and higher diversity with better performance.

<table>
<thead>
<tr>
<th>Model</th>
<th>test-0.4</th>
<th>test-0.6</th>
<th>test-0.4</th>
<th>test-0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>+8.2</td>
<td>+8.9</td>
<td>+3.7</td>
<td>+4.4</td>
</tr>
<tr>
<td>EME</td>
<td>+8.9</td>
<td>+8.5</td>
<td>+4.0</td>
<td>+5.0</td>
</tr>
<tr>
<td>Epp</td>
<td>+10.5</td>
<td>+13.7</td>
<td>+2.2</td>
<td>+4.0</td>
</tr>
<tr>
<td>GNO</td>
<td>+7.1</td>
<td>+6.2</td>
<td>+7.6</td>
<td>+9.3</td>
</tr>
<tr>
<td>JRC</td>
<td>+7.2</td>
<td>+6.8</td>
<td>+4.8</td>
<td>+5.2</td>
</tr>
<tr>
<td>KDE</td>
<td>+8.0</td>
<td>+7.5</td>
<td>+6.9</td>
<td>+7.8</td>
</tr>
<tr>
<td>News</td>
<td>+7.2</td>
<td>+12.5</td>
<td>+2.3</td>
<td>+5.0</td>
</tr>
<tr>
<td>PHP</td>
<td>+7.2</td>
<td>+7.6</td>
<td>+4.2</td>
<td>+5.4</td>
</tr>
<tr>
<td>TED</td>
<td>+9.4</td>
<td>+11.4</td>
<td>+2.9</td>
<td>+4.6</td>
</tr>
<tr>
<td>Ubu</td>
<td>+5.5</td>
<td>+6.0</td>
<td>+6.7</td>
<td>+8.6</td>
</tr>
<tr>
<td>Wiki</td>
<td>+8.0</td>
<td>+8.0</td>
<td>+2.5</td>
<td>+3.6</td>
</tr>
<tr>
<td>all</td>
<td>+8.1</td>
<td>+8.9</td>
<td>+4.4</td>
<td>+5.7</td>
</tr>
</tbody>
</table>

Table 8: Coverage and noise scores increase. "Difficulty" is highlighted in bold ($< 8.0$ for cover; $> 6.0$ for noise).

Long run All results in the main text were obtained with models trained for 60k iterations, which was enough to compare the various models while saving computation resources. For completeness, we also performed one longer training for 300k iterations for $\text{TM}^3$-LevT (see Table 9), which resulted in an improvement of around +2 BLEU for each test set. This is without realignment nor pretraining.

<table>
<thead>
<tr>
<th>Model</th>
<th>test-0.4</th>
<th>test-0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{TM}^3$-LevT</td>
<td>46.5</td>
<td>60.0</td>
</tr>
<tr>
<td>+ realign</td>
<td>46.7</td>
<td>60.2</td>
</tr>
<tr>
<td>$\text{TM}^3$-LevT long</td>
<td>48.7</td>
<td>61.9</td>
</tr>
<tr>
<td>+ realign</td>
<td>48.9</td>
<td>62.0</td>
</tr>
</tbody>
</table>

Table 9: BLEU score of $\text{TM}^3$-LevT: 60k iterations; and $\text{TM}^3$-LevT long: 300k iterations.

The Benefits of realignment Table 10 shows that realignment also decreases the average number of refinement steps to converge. These results suggest that the edition is made easier with realignment.

In Table 11, we present detailed results of the unigram modified precision of LevT, $\text{TM}^3$-LevT and
Table 10: Average number of extra refinement rounds.

<table>
<thead>
<tr>
<th>model</th>
<th>test-0.4</th>
<th>test-0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM$^3$-LevT</td>
<td>3.55</td>
<td>2.07</td>
</tr>
<tr>
<td>+realign</td>
<td>3.37</td>
<td>1.93</td>
</tr>
</tbody>
</table>

TM$^3$-LevT+realign. Using more examples indeed increases copy (+4.4), even though it diminishes copy precision (-1.7). Again we observe the positive effect of realignment, which amplifies the tendency of our model to copy input tokens.

<table>
<thead>
<tr>
<th>model</th>
<th>precision</th>
<th>% units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM$^3$-LevT</td>
<td>copy</td>
<td>87.5</td>
</tr>
<tr>
<td></td>
<td>gen</td>
<td>52.6</td>
</tr>
<tr>
<td>TM$^3$-LevT+realign</td>
<td>copy</td>
<td>85.8</td>
</tr>
<tr>
<td></td>
<td>gen</td>
<td>54.7</td>
</tr>
</tbody>
</table>

Table 11: Modified precision of copy vs. generated unigrams of LevT vs. TM$^3$-LevT.

**COMET scores** We compute COMET scores (Rei et al., 2020) separately for each domain with default wmt20-comet-da similarly to Table 4 (see Table 12). We observe that the basic version of TM$^3$-LevT underperforms TM$^4$-LevT; we also see a great variation in the scores. A possible explanation can be a fluency decline when using multiple examples, which is not represented by the precision scores computed by BLEU. The improved version, using realignment and pre-training, confirms that adding more matches is overall beneficial for MT quality.

**Per-domain ablation study** Table 13 details the results of our ablation study separately for each domain.

**Illustration** A full inference run is in Table 14, illustrating the benefits of considering multiple examples and realignment. Even though the realignment does not change here the final output, it reduces the number of atomic edits needed to generate it, making the inference more robust.
### Table 12: Per domain COMET scores (x 100) for TM^4-LevT and variants. Bold for scores better than TM^4-LevT.

<table>
<thead>
<tr>
<th></th>
<th>ECB</th>
<th>EME</th>
<th>Epp</th>
<th>GNO</th>
<th>JRC</th>
<th>KDE</th>
<th>News</th>
<th>PHP</th>
<th>TED</th>
<th>Ubu</th>
<th>Wiki</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test-0.4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM^4-LevT</td>
<td>33.0</td>
<td>43.1</td>
<td>39.9</td>
<td>56.3</td>
<td>70.7</td>
<td>37.4</td>
<td>-2.0</td>
<td>-39.6</td>
<td>-0.8</td>
<td>41.6</td>
<td>-9.9</td>
<td></td>
</tr>
<tr>
<td>TM^3-LevT</td>
<td>27.2</td>
<td>42.0</td>
<td>31.1</td>
<td>48.0</td>
<td>64.4</td>
<td>32.6</td>
<td>-10.8</td>
<td>-42.7</td>
<td>-8.7</td>
<td>39.3</td>
<td>-15.7</td>
<td>18.4</td>
</tr>
<tr>
<td>TM^2-LevT</td>
<td>27.3</td>
<td>42.1</td>
<td>26.8</td>
<td>51.5</td>
<td>64.2</td>
<td>33.5</td>
<td>-10.1</td>
<td>-39.9</td>
<td>-14.8</td>
<td>38.6</td>
<td>-16.3</td>
<td>18.4</td>
</tr>
<tr>
<td>+pre-train</td>
<td>30.6</td>
<td>44.7</td>
<td>38.7</td>
<td><strong>57.9</strong></td>
<td>67.8</td>
<td><strong>37.9</strong></td>
<td>-6.2</td>
<td><strong>35.3</strong></td>
<td>2.7</td>
<td>41.5</td>
<td>-5.0</td>
<td><strong>25.0</strong></td>
</tr>
<tr>
<td>+realign</td>
<td>31.0</td>
<td>45.1</td>
<td>32.0</td>
<td>53.2</td>
<td>66.4</td>
<td>34.9</td>
<td>-10.9</td>
<td>-39.0</td>
<td>-11.9</td>
<td>41.1</td>
<td>-10.8</td>
<td>21.0</td>
</tr>
<tr>
<td>+both</td>
<td>33.7</td>
<td>46.4</td>
<td>42.4</td>
<td><strong>59.9</strong></td>
<td>69.9</td>
<td><strong>40.1</strong></td>
<td>-1.0</td>
<td>-33.0</td>
<td>5.1</td>
<td><strong>43.5</strong></td>
<td>-3.7</td>
<td><strong>27.5</strong></td>
</tr>
</tbody>
</table>

| **test-0.6**   |     |     |     |     |     |     |      |     |     |     |      |      |
| TM^4-LevT      | 51.7| 53.9| 56.9| 65.3| 85.4| 37.0| -3.0 | -17.0| 48.5| 57.6| 64.5 |      |
| TM^3-LevT      | 51.2| **54.2**| 56.3| 64.2| 82.5| 34.6| -9.0 | -15.9| 46.1| 55.5| 61.6 | 43.7 |
| TM^2-LevT      | 50.9| **55.8**| 54.1| 65.1| 81.1| 33.6| -9.8 | -16.2| 41.1| 57.4| 61.7 | 43.1 |
| +pre-train     | **54.6**| **56.5**| **58.0**| **68.2**| 84.7| **41.5**| -4.3 | -11.8| **48.9**| 62.7 | **65.6**| **47.7**|
| +realign       | **53.0**| **56.4**| **56.3**| 65.4| 83.6| 34.3| -7.2 | -16.1| 42.8| **58.3**| 63.7 | 44.6 |
| +both          | **55.7**| **57.4**| **58.8**| **70.5**| **85.8**| **43.2**| -4.4 | -10.6| **49.4**| **62.3**| **67.7**| **48.7**|

### Table 13: Ablation study. We report BLEU scores for various settings. -sel: no random selection noise (γ=0); -delx: no extra deletion loss; -rd-del: no random deletion (β=0); -mask: no random mask (δ=0); -dum-plh: null probability to start with y_{post\cdot del}=y_\ast (α=0); -indep-align: the alignments are performed independently.

<table>
<thead>
<tr>
<th></th>
<th>ECB</th>
<th>EME</th>
<th>Epp</th>
<th>GNO</th>
<th>JRC</th>
<th>KDE</th>
<th>News</th>
<th>PHP</th>
<th>TED</th>
<th>Ubu</th>
<th>Wiki</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test-0.4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM^4-LevT</td>
<td>53.9</td>
<td><strong>55.6</strong></td>
<td><strong>34.2</strong></td>
<td>60.7</td>
<td><strong>66.0</strong></td>
<td><strong>53.5</strong></td>
<td>20.4</td>
<td>33.0</td>
<td><strong>28.6</strong></td>
<td>47.5</td>
<td><strong>32.8</strong></td>
<td><strong>46.5</strong></td>
</tr>
<tr>
<td>-sel</td>
<td><strong>54.5</strong></td>
<td><strong>55.6</strong></td>
<td>32.7</td>
<td><strong>61.2</strong></td>
<td>65.9</td>
<td>52.2</td>
<td>19.5</td>
<td><strong>33.3</strong></td>
<td><strong>27.7</strong></td>
<td><strong>48.1</strong></td>
<td>31.3</td>
<td>46.2</td>
</tr>
<tr>
<td>-delx</td>
<td>52.2</td>
<td>53.4</td>
<td>31.8</td>
<td>58.7</td>
<td>64.0</td>
<td>52.0</td>
<td>19.6</td>
<td>31.2</td>
<td>27.5</td>
<td>46.8</td>
<td>31.5</td>
<td>44.8</td>
</tr>
<tr>
<td>-rd-del</td>
<td>49.7</td>
<td>47.6</td>
<td>22.2</td>
<td>48.2</td>
<td>56.7</td>
<td>38.8</td>
<td>13.2</td>
<td>29.5</td>
<td>16.9</td>
<td>32.2</td>
<td>21.4</td>
<td>38.6</td>
</tr>
<tr>
<td>-mask</td>
<td>53.4</td>
<td>54.7</td>
<td>33.7</td>
<td>58.9</td>
<td>65.3</td>
<td>52.4</td>
<td>20.3</td>
<td>33.1</td>
<td>27.5</td>
<td>47.1</td>
<td>32.2</td>
<td>46.0</td>
</tr>
<tr>
<td>-dum-plh</td>
<td>50.8</td>
<td>43.3</td>
<td>32.7</td>
<td>45.6</td>
<td>61.2</td>
<td>42.4</td>
<td>21.0</td>
<td>30.9</td>
<td>24.3</td>
<td>35.6</td>
<td>28.2</td>
<td>41.0</td>
</tr>
<tr>
<td>-indep-align</td>
<td>51.5</td>
<td>52.4</td>
<td>29.7</td>
<td>53.8</td>
<td>60.7</td>
<td>47.2</td>
<td>16.9</td>
<td>30.4</td>
<td>21.8</td>
<td>39.0</td>
<td>28.4</td>
<td>42.6</td>
</tr>
</tbody>
</table>

| **test-0.6**   |     |     |     |     |     |     |      |     |     |     |      |      |
| TM^4-LevT      | **64.2**| **68.0**| **49.4**| **73.0**| **76.4**| **60.1**| 21.2 | **39.6**| 52.2 | **60.1**| **61.6**| **60.1**|
| -sel           | 63.8| 67.5 | **50.0**| 71.2 | 76.3 | 60.0 | **21.5**| 39.1| **54.2**| **59.7**| **62.2**| 60.0 |
| -delx          | 62.1| 66.7 | 47.7 | 70.6 | 75.0 | 58.8 | 20.1 | 37.9| 53.5 | 58.4 | 60.9 | 58.6 |
| -rd-del        | 58.4| 59.6 | 39.7 | 59.0 | 67.6 | 47.7 | 16.9 | 35.0| 39.8 | 44.1 | 47.4 | 51.9 |
| -mask          | 63.1| 65.3 | 49.1 | 69.4| 74.2 | 58.2 | 21.8 | 38.6| 50.9 | 58.7 | 59.7 | 59.0 |
| -dum-plh       | 57.3| 55.5 | 44.9 | 51.8| 68.5| 44.5 | 20.2 | 35.7| 42.6 | 38.7 | 50.7 | 50.9 |
| -indep-align   | 60.6| 64.0 | 46.6 | 64.3 | 71.9 | 55.5 | 18.6 | 35.6| 44.6 | 51.6 | 56.0 | 56.4 |
Le swf \_setfont ( ) remplace la police courante par la valeur donnée par l'identifiant fontid.

Table 14: Examples of full inference of several models on a test sample from test-0.4-PHP (sample n°571). Copied parts are in red.

<table>
<thead>
<tr>
<th>LevT</th>
<th>TM^2-LevT</th>
<th>TM^2-LevT + realign</th>
</tr>
</thead>
<tbody>
<tr>
<td>y^obs: swf _+1 font ( ) remplace la police actuelle à la valeur donnée par le paramètre fontid.</td>
<td>y^obs: Le swf _setfont () remplace la police de police courante donnée par la valeur de paramètre fontid.</td>
<td>y^obs: Le swf _setfont () remplace la police courante à la valeur donnée par le paramètre fontid.</td>
</tr>
<tr>
<td>y^hyp: swf _setfont () remplace la police de police courante donnée par la valeur de paramètre fontid.</td>
<td>y^hyp: swf _setfont () remplace la police actuelle à la valeur donnée par le paramètre fontid.</td>
<td>y^hyp: Le swf _setfont () remplace la police courante à la valeur donnée par le paramètre fontid.</td>
</tr>
<tr>
<td>y^obs: swf _+1 font ( ) remplace la police actuelle à la valeur donnée par le paramètre fontid.</td>
<td>y^obs: Le swf _setfont () remplace la police courante donnée par la valeur de paramètre fontid.</td>
<td>y^obs: Le swf _setfont () remplace la police courante à la valeur donnée par le paramètre fontid.</td>
</tr>
<tr>
<td>y^hyp: swf _setfont () remplace la police actuelle à la valeur donnée par le paramètre fontid.</td>
<td>y^hyp: Le swf _setfont () remplace la police courante à la valeur donnée par le paramètre fontid.</td>
<td>y^hyp: Le swf _setfont () remplace la police courante à la valeur donnée par le paramètre fontid.</td>
</tr>
<tr>
<td>hyp: Le swf _setfont () remplace la police de police courante donnée par la valeur de paramètre fontid.</td>
<td>hyp: swf _setfont () remplace la police actuelle à la valeur donnée par le paramètre fontid.</td>
<td>hyp: Le swf _setfont () remplace la police courante à la valeur donnée par le paramètre fontid.</td>
</tr>
</tbody>
</table>