Incorporating a Local Translation Mechanism into Non-autoregressive Translation

Xiang Kong[*], **Zhisong Zhang**[*], **Eduard Hovy**
Language Technologies Institute, Carnegie Mellon University
{xiangk,zhisongz,hovy}@cs.cmu.edu

Abstract

In this work, we introduce a novel local autoregressive translation (LAT) mechanism into non-autoregressive translation (NAT) models so as to capture local dependencies among target outputs. Specifically, for each target decoding position, instead of only one token, we predict a short sequence of tokens in an autoregressive way. We further design an efficient merging algorithm to align and merge the output pieces into one final output sequence. We integrate LAT into the conditional masked language model (CMLM; Ghazvininejad et al., 2019) and similarly adopt iterative decoding. Empirical results on five translation tasks show that compared with CMLM, our method achieves comparable or better performance with fewer decoding iterations, bringing a 2.5x speedup. Further analysis indicates that our method reduces repeated translations and performs better at longer sentences. The code for our model is available at https://github.com/shawnkx/NAT-with-Local-AT.

1 Introduction

Traditional neural machine translation (NMT) models (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2014; Gehring et al., 2017; Vaswani et al., 2017) commonly make predictions in an incremental token-by-token way, which is called autoregressive translation (AT). Although this strategy can capture the full translation history, it has relatively high decoding latency. To make the decoding more efficient, non-autoregressive translation (NAT) (Gu et al., 2018) is introduced to generate multiple tokens at once instead of one-by-one. However, with the conditional independence property (Gu et al., 2018), NAT models do not directly consider the dependencies among output tokens, which may cause errors of repeated translation and incomplete translation (Wang et al., 2019). There have been various methods in previous work (Stern et al., 2019; Gu et al., 2019; Ma et al., 2018; Wei et al., 2019; Ma et al., 2019; Tu et al., 2020) to mitigate this problem, including iterative decoding (Lee et al., 2018; Ghazvininejad et al., 2019).

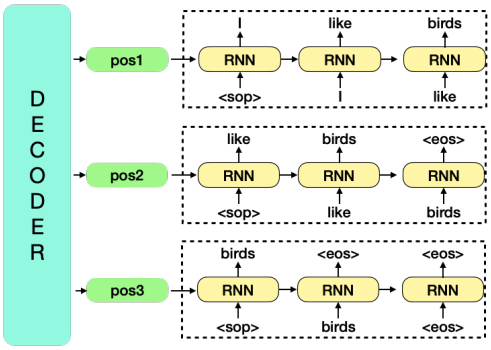

Figure 1: An example of the LAT mechanism. For each decoding position, a short sequence of tokens is generated in an autoregressive way. ⟨sop⟩ is the special start-of-piece symbol. 'pos*' denotes the hidden state from the decoder at that position.

In this work, we introduce a novel mechanism, i.e., local autoregressive translation (LAT), to take local target dependencies into consideration. For a decoding position, instead of generating one token, we predict a short sequence of tokens (which we call a translation piece) for the current and next few positions in an autoregressive way. A simple example is shown in Figure 1.

With this mechanism, there can be overlapping tokens between nearby translation pieces. We take advantage of these redundancies, and apply a simple algorithm to align and merge all these pieces to obtain the full translation output. Specifically, our algorithm builds the output by incrementally aligning and merging adjacent pieces, based on the hypothesis that each local piece is fluent and there are overlapping tokens between adjacent pieces as aligning points. Moreover, the final output se-

[*] Zhisong and Xiang contributed equally for this paper

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 1067–1073,
November 16–20, 2020.

quence is dynamically decided through the merging algorithm, which makes the decoding process more flexible.

We integrate our mechanism into the conditional masked language model (CMLM) (Ghazvininejad et al., 2019) and similarly adopt iterative decoding, where tokens with low confidence scores are masked for prediction in more iterations. With evaluations on five translation tasks, i.e., WMT'14 EN↔DE, WMT'16 EN↔RO and IWSLT'14 DE→EN, we show that our method could achieve similar or better performance compared with CMLM and AT models while gaining nearly 2.5 and 7 times speedups, respectively. Furthermore, our method is shown to effectively reduce repeated translations and perform better at longer sentences.

2 CMLM with LAT

2.1 Model

We integrate our LAT mechanism into CMLM, which predicts the full target sequence based on the source and partial target sequence. We adopt a lightweight LSTM-based sequential decoder as the local translator upon the CMLM decoder outputs. For a target position i, the CMLM decoder produces a hidden vector pos_i, based on which the local translator predicts a short sequence of tokens in an autoregressive way, i.e., $t_i^1, t_i^2, ..., t_i^K$. Here K is the number of location translation steps, which is set to 3 in our experiments to avoid affecting the speed much.

2.2 Decoding

During inference, a special token, ⟨sop⟩ (start of piece) is fed into the local translator to generate a short sequence based on the pos_i. After generating the local pieces for all target positions in parallel, we adopt a simple algorithm to merge them into a full output sequence. This merging algorithm is described in detail in Section 3. We also perform iterative decoding following the same Mask-Predict strategy (Ghazvininejad et al., 2019; Devlin et al., 2019). In each iteration, we take the output sequence from the last iteration and mask a subset of tokens with low confidence scores by a special ⟨mask⟩ symbol. Then the masked sequence is fed together with the source sequence to the decoder for the next decoding iteration.

Following Ghazvininejad et al. (2019), a special token LENGTH is added to the encoder, which is utilized to predict the initial target sequence length. Nevertheless, our algorithm can dynamically adjust the final output sequence and we find that our method is not sensitive to the choice of target length as long as it falls in a reasonable range.

2.3 Training

The training procedure is similar to that of Ghazvininejad et al. (2019). Given a pair of source and target sequences S and T, we first sample a masking size from a uniform distribution from [1, N], where N is the target length. Then this size of tokens are randomly picked from the target sequence and replaced with the ⟨mask⟩ symbol. We refer to the set of masked tokens as T_{mask}. Then for each target position, we adopt a teacher-forcing styled training scheme to collect the cross-entropy losses for predicting the corresponding ground-truth local sequences, the size of which is $K = 3$.

Assume that we are at position i, we simply setup the ground-truth local sequence $t_i^1, t_i^2, ..., t_i^K$ as $T_i, T_{i+1}, ..., T_{i+K-1}$, where T_i denotes the i-th token in the full target ground-truth sequence. We include all tokens in our final loss, whether they are in T_{mask} or not, but adopt different weights for the masked tokens that do not appear in the inputs. Therefore, our token prediction loss function is:

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{j=1}^{K} \mathbb{1}\left\{t_i^j \in T_{mask}\right\} \log(p(t_i^j)) - \sum_{i=1}^{N}\sum_{j=1}^{K} \mathbb{1}\left\{t_i^j \notin T_{mask}\right\} \alpha \log(p(t_i^j))$$

Here, we adopt a weight α for the tokens that are not masked in the target input, which is set as 0.1 so that the model could be trained more on the unseen tokens. Furthermore, we randomly delete certain positions (the number of deletion is randomly sampled from [1, 0.15*N]) from the target inputs to encourage the model to learn insertion-styled operations. The final loss is the addition of the token prediction and the target length prediction loss.

3 Merging Algorithm

In decoding, the model generates local translation pieces for all decoding positions. We adopt a simple algorithm that incrementally builds the output through a piece-by-piece merging process. Our hypothesis is that if the local autoregressive translator is well-trained, then 1) the token sequence inside each piece is fluent and well-translated, 2) there are

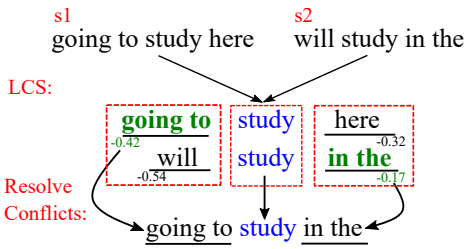

Figure 2: An example of merging two pieces of tokens.

overlaps between nearby pieces, acting as aligning points for merging.

We first illustrate the core operation of merging two consecutive pieces of tokens. Algorithm 1 describes the procedure and Figure 2 provides an example. Given two token pieces $s1$ and $s2$, we first use the Longest Common Subsequence (LCS) algorithm to find matched tokens (Line 1). If there is nothing that can be matched, then we simply do concatenation (Line 3), otherwise we solve the conflicts of the alternative spans by comparing their confidence scores (Line 9-14). Finally we can arrive at the merged output after resolving all conflicted spans.

In the above procedure, we need to specify the score of a span. Through preliminary experiments, we find a simple but effective scheme. From the translation model, each token gets a model score of its log probability. For the score of a span, we average the scores of all the tokens inside. If the span is empty, we utilize a pre-defined value, which is empirically set to $\log 0.25$. For aligned tokens, we choose the highest scores among them for later merging process (Line 16).

With this core merging operation, we apply a left-to-right scan to merge all the pieces in a piece-by-piece fashion. For each merging operation, we only take the last K tokens of $s1$ and the first K tokens of $s2$, while other tokens are directly copied. This ensures that the merging will only be local, to mitigate the risk of wrongly aligned tokens. Here, K is again the local translation step size.

Our merging algorithm can be directly applied at the end of each iteration in the iterative decoding. However, since the output length of the merging algorithm is not always the same as the number of input pieces, we further adopt a length adjustment procedure for intermediate iterations. Briefly speaking, we adjust the output length to the predicted length by adding or deleting certain amounts of special $\langle$mask$\rangle$ symbols. Please refer to the Appendix for more details.

Algorithm 1: Merging two pieces.

Input: Two pieces of tokens: $s1$, $s2$.
Output: A merged sequence s'.

```
// Call Longest Common Subsequence
1  MatchedPairs = LCS(s1, s2);
2  if MatchedPairs.size() == 0 then
3      return s1+s2 ;          // Simple concat
4  else
5      s' = [] ;                    // Initialize
6      p1, p2 = -1, -1 ;       // Previous idxes
       // Add sentinel indexes.
7      MatchedPairs += [(∞, ∞)];
8      foreach i1, i2 in MatchedPairs do
9          span1 = s1[p1+1:i1];
10         span2 = s2[p2+1:i2];
           // Solve conflicts by scores.
11         if score(span1) ≥ score(span2) then
12             s' += span1;
13         else
14             s' += span2;
           // Align the matched ones.
15         if i1 ≠ ∞ then
16             s' += [align(s1[i1], s2[i2])];
17         p1, p2 = i1, i2;
18     return s';
```

Although our merging algorithm is actually autoregressive, it does not include any neural network computations and thus can run efficiently. In addition to efficiency, our method also makes the decoding more flexible, since the final output is dynamically created through the merging algorithm.

4 Experiments

4.1 Experimental Setup

We evaluate our proposed method on five translation tasks, i.e., WMT'14 EN↔DE, WMT'16 EN↔RO and IWSLT'14 DE→EN. Following previous works (Hinton et al., 2015; Kim and Rush, 2016; Gu et al., 2018; Zhou et al., 2020), we train a vanilla base transformer (Vaswani et al., 2017) on each dataset and use its translations as the training data. The BLEU score (Papineni et al., 2002) is used to evaluate the translation quality. Latency, the average decoding time (ms) per sentence with batch size 1, is employed to measure the inference speed. All models' decoding speed is measured on a single NVIDIA TITAN RTX GPU.

We follow most of the hyperparameters for the CMLM (Ghazvininejad et al., 2019) in the base configuration, i.e., 6 layers for encoder and decoder, 8 attention heads, 512 embedding dimensions and 2048 hidden dimensions. The LAT is an

#	Model	Iterations	WMT'14 EN-DE	WMT'14 DE-EN	WMT'16 EN-RO	WMT'16 RO-EN	IWSLT'14 DE-EN	latency (ms)
1	AT	N	27.46	31.87	34.39	33.98	34.18	486
2	CMLM	1	18.05	21.83	27.32	28.20	28.14	27
3	**LAT**		**25.20**	**29.91**	**30.74**	**31.24**	**31.92**	**31**
4	CMLM	4	25.94	29.90	32.53	33.23	32.87	72
5	**LAT**		**27.35**	**32.04**	**32.87**	**33.26**	**34.08**	**73**
6	CMLM	10	27.03	30.53	33.08	33.31	33.40	166

Table 1: The comparisons (on BLEU score and decoding latency) of CMLM, LAT and AT models.

Model	Iteration	ngram repeat rate (%) 1	2	3	4
CMLM	1	20.85	3.78	1.06	0.37
LAT		**4.89**	**0.42**	**0.05**	**0.00**
CMLM	4	3.97	0.14	0.03	0.02
LAT		**3.32**	**0.08**	**0.00**	**0.00**
CMLM	10	3.56	0.08	0.02	0.02
AT	N	3.27	0.05	0.00	0.00
Reference	-	2.49	0.03	0.00	0.00

Table 2: N-gram repeat rates of various models on WMT'14 EN-DE test set.

	# local translation steps (K) 2	3	4	5	6
BLEU	32.9	33.8	34.4	34.5	34.2
latency (ms)	69	72	74	77	79

Table 3: The performance of LAT models with respect to the number of local translation steps (K) on IWSLT'14 DE-EN test set.

LSTM-based neural network of size 512. Finally, we average 5 best checkpoints according to the validation loss as our final model. Please refer to the Appendix for more details of the settings.

4.2 Main results

The main results are shown in Table 1. Compared with CMLM at the same number of decoding iterations (row 2 vs. 3 and row 4 vs. 5), LAT performs much better while keeping similar speed, especially when the iteration number is 1. Note that since our method is not sensitive to predicted length, we only take one length candidate from our length predictor instead of 5 as in CMLM. Furthermore, LAT with 4 iterations could achieve similar or better results than CMLM with 10 iterations (row 5 vs. 6) but have a nearly 2.5x decoding speedup.

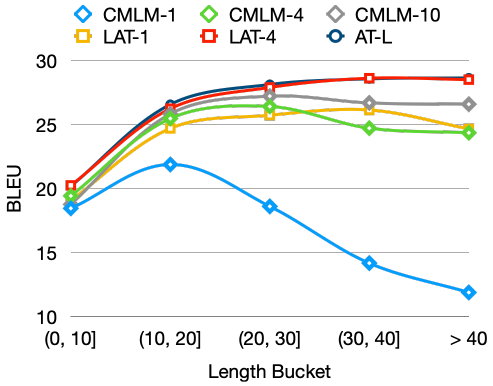

Figure 3: The BLEU scores of various systems with respect to the reference sentence lengths on WMT'14 EN-DE testset.

4.3 Analysis

On local translation step. We also explore the effects of the number of local translation steps (K) on the IWSLT'14 DE-EN dataset. The results are shown in Table 3. Generally, with more local translation steps, there can be certain improvements on BLEU but with an extra cost at inference time.

On repeated translation. We compute the n-gram repeat rate (nrr, what percentage of n-grams are repeated by certain nearby n-grams) of different systems on WMT'14 EN-DE test set and the result is shown in Table 2. The nrr of CMLM with one iteration is much higher than other systems, showing that it suffers from a severe repeated translation problem. On the other hand, LAT can mitigate this problem thanks to the merging algorithm.

On sentence length. We explore how various systems perform on sentences with various lengths. The WMT'14 EN-DE test set is split into 5 length buckets by target length. Figure 3 show that LAT performs better than CMLM on longer sentences, which indicates the effectiveness of our methods at capturing certain target dependencies.

5 Related Work

Gu et al. (2018) begin to explore non-autoregressive translation, the aim of which is to generate sequences in parallel. In order to mitigate multimodality issue, recent work mainly tries to narrow the gap between NAT and AT. Libovickỳ and Helcl (2018) design a NAT model using CTC loss. Lee et al. (2018) uses iteration decoding to refine translation. The conditional masked language model (CMLM) (Ghazvininejad et al., 2019) predicts partial target tokens based on the source text and partially masked target sentence. Ma et al. (2019) employs normalizing flows as the the latent variable to produce sequences. Sun et al. (2019) designs an efficient approximation for CRF for NAT. Besides that, there are some works trying to improving the decoding speed of the autoregressive models. For example, Wang et al. (2018) propose a semi-autoregressive translation model, which adopts locally non-autoregressive, but autoregressive decoding. And works mentioned in Hayashi et al. (2019) use techniques such as knowledge distillation, block-sparse regularization to improve the decoding speed of autoregressive models.

6 Conclusion

In this work, we incorporate a novel local autoregressive translation mechanism (LAT) into non-autoregressive translation, predicting multiple short sequences of tokens in parallel. With a simple and efficient merging algorithm, we integrate LAT into the conditional masked language model (CMLM Ghazvininejad et al., 2019) and similarly adopt iterative decoding. We show that our method could achieve similar results to CMLM with less decoding iterations, which brings a 2.5x speedup. Moreover, analysis shows that LAT can reduce repeated translations and perform better at longer sentences.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. 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 4171–4186.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

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.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, Canada, April 30-May 3, 2018, Conference Track Proceedings*.

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

Hiroaki Hayashi, Yusuke Oda, Alexandra Birch, Ioannis Konstas, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Katsuhito Sudoh. 2019. Findings of the third workshop on neural generation and translation. *EMNLP-IJCNLP 2019*, page 1.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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.

Jindřich Libovickỳ and Jindřich Helcl. 2018. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021.

Shuming Ma, Xu Sun, Yizhong Wang, and Junyang Lin. 2018. Bag-of-words as target for neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 332–338, Melbourne, Australia. Association for Computational Linguistics.

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 4273–4283.

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 on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. *arXiv preprint arXiv:1902.03249*.

Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin, and Zhihong Deng. 2019. Fast structured decoding for sequence models. In *Advances in Neural Information Processing Systems*, pages 3011–3020.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Lifu Tu, Richard Yuanzhe Pang, Sam Wiseman, and Kevin Gimpel. 2020. Engine: Energy-based inference networks for non-autoregressive machine translation. *arXiv preprint arXiv:2005.00850*.

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*, pages 5998–6008.

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.

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*, volume 33, pages 5377–5384.

Bingzhen Wei, Mingxuan Wang, Hao Zhou, Junyang Lin, and Xu Sun. 2019. Imitation learning for non-autoregressive neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1304–1312.

Chunting Zhou, Jiatao Gu, and Graham Neubig. 2020. Understanding knowledge distillation in non-autoregressive machine translation. *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, Conference Track Proceedings*.

Appendices

A Preprocessing

We follow the standard pre-processing procedure in prior works (Vaswani et al., 2017; Lee et al., 2018). All datasets are segmented into subwords through byte pair encoding (BPE) (Sennrich et al., 2016). The BPE code is learnt from the combination of source and target data for WMT datasets. For IWSLT, the bpe code is learned from the source and target data separately. Table 4 lists some details.

Dataset	Vocab. Size	Data size
IWSLT	10k	150k
WMT14 EN↔DE	32k	4.5M
WMT16 EN↔RO	40k	600k

Table 4: Pre-processing details of various translation benchmarks. Vocab. size denotes vocabulary size.

B Optimization

We sample weights from $\mathcal{N}(0, 0.02)$, initialize biases to zero, and set layer normalization parameters to $\beta = 0$, $\gamma = 1$. For regularization, we use 0.3 dropout, 0.01 L2 weight decay, and smoothed cross-entropy loss with $\epsilon = 0.1$. We train batches of 128k tokens using Adam (Kingma and Ba, 2015) with $\beta = (0.9, 0.999)$ and $\epsilon = 10^{-6}$. The learning rate warms up to a peak of 5×10^{-4} within 10,000

Model	Iterations	WMT'14		WMT'16		IWSLT'14
		EN-DE	DE-EN	EN-RO	RO-EN	DE-EN
AT	N	26.13	31.06	34.74	35.76	33.59
CMLM	1	18.47	22.83	26.92	28.77	24.57
LAT		22.14	29.20	32.16	32.07	28.34
CMLM	4	24.73	29.18	33.06	34.31	29.06
LAT		26.03	31.66	33.49	34.77	34.05
CMLM	10	25.25	29.83	33.66	34.65	33.23

Table 5: The comparisons (on BLEU score and decoding latency) of CMLM, LAT and AT models on development sets.

steps, and then decays with the inverse square-root schedule. We train our models for 300k steps with batch size 128k (Ghazvininejad et al., 2019) for WMT datasets. For the IWSLT dataset, we train our models for 50k steps with batch size 32k.

C Model Parameter Size

The averaged size of parameters for all models are shown in Table 6. These three kinds of models have similar number of parameters. LAT models have the most number of parameters due to the LSTM-based local translator.

Model	Parameter size
AT	60M
CMLM	62M
LAT	64M

Table 6: Number of Parameters of different models.

D Validation Performance

The performance of different models on translation tasks' validation sets is reported in the Table 5. We could find the similar trend to the performance on the test set.

E Length Adjustment for Intermediate Iterations

Since our merging algorithm produces the output dynamically, the output length is usually not the same as the number of input pieces. In iterative decoding, we find it helpful to adjust the output sequence's length to the input length in intermediate iterations. This is achieved by adding or deleting the special $\langle mask \rangle$ symbols. Notice that for the final iteration, we do not apply any adjustments and keep the merged output sequence as it is.

For the length adjustment in the intermediate iterations, our goal is to adjust the output length of the merger (L_{out}) to be close to the input target length (L_{in}). If these two lengths are already equal or their relative difference is within a certain range (which is empirically set to 5%), we will do nothing. Otherwise, there can be two cases: 1) when L_{in} is larger than L_{out}, we further insert $L_{in} - L_{out}$ $\langle mask \rangle$ tokens into the sequence; 2) otherwise, we try to delete $L_{out} - L_{in}$ $\langle mask \rangle$ tokens. Notice that the addition or deletion operations happen after the masking procedure for the next iteration.

Here, we describe the addition case in detail. Suppose we need to further insert M masks into the output sequence, we decide the insertion places according to the position gaps. We adopt a simple position scheme for all the tokens. For each original token t_i^j (the j-th token in the i-th piece) in the input translation pieces, we set $i + j$ as its position. For each token in the output sequence after merging, since it can originate from multiple input tokens through aligning, we take the averaged value of all its source input tokens' positions. We calculate the position gap between each pair of nearby unmasked tokens in the output sequence and maintain a priority queue for all these gaps. Then we insert M masks once at a time. For each time, we select the current maximal gap, insert a $\langle mask \rangle$ to that position, and subtract that gap by 1. The case for deletion would be similar but in the opposite direction: select the minimal gap, delete one $\langle mask \rangle$ if there are any, and increase that gap by 1. We will delete nothing if there are no masked tokens in the selected gap.