AMOM: Adaptive Masking over Masking for Conditional Masked Language Model

Yisheng Xiao[1], Ruiyang Xu[1], Lijun Wu[2], Juntao Li[1*], Tao Qin[2], Tie-Yan Liu[2], Min Zhang[1]

[1]Institute of Computer Science and Technology, Soochow University
[2]Microsoft Research Asia
{ysxiaoo, ryxu1}@stu.suda.edu.cn, {ljt, minzhang}@suda.edu.cn, {lijuwu, taoqin, tyliu}@microsoft.com

Abstract

Transformer-based autoregressive (AR) methods have achieved appealing performance for varied sequence-to-sequence generation tasks, e.g., neural machine translation, summarization, and code generation, but suffer from low inference efficiency. To speed up the inference stage, many non-autoregressive (NAR) strategies have been proposed in the past few years. Among them, the conditional masked language model (CMLM) is one of the most versatile frameworks, as it can support many different sequence generation scenarios and achieve very competitive performance on these tasks. In this paper, we further introduce a simple yet effective adaptive masking over masking strategy to enhance the refinement capability of the decoder and make the encoder optimization easier. Experiments on **3** different tasks (neural machine translation, summarization, and code generation) with **15** datasets in total confirm that our proposed simple method achieves significant performance improvement over the strong CMLM model. Surprisingly, our proposed model yields state-of-the-art performance on neural machine translation (**34.62** BLEU on WMT16 EN→RO, **34.82** BLEU on WMT16 RO→EN, and **34.84** BLEU on IWSLT De→En) and even better performance than the **AR** Transformer on **7** benchmark datasets with at least **2.2**× speedup. Our code is available at GitHub[1].

Introduction

Transformer-based models (Vaswani et al. 2017) have been proven effective for various sequence to sequence generation tasks, such as machine translation (Wu et al. 2019; Liang et al. 2021), text summarization (Savelieva, Au-Yeung, and Ramani 2020; Elsaid et al. 2022), dialogue systems (Zhang et al. 2020; Ma et al. 2020), code generation (Wang et al. 2020), etc. Despite the excellent performance of Transformer-based models, they usually adopt the autoregressive (AR) decoding paradigm in which the decoding of a target sequence is decomposed into multi-step predictions in left-to-right order, i.e., the next prediction is conditioned on the previously generated part. Such an attribute increases the inference time cost linearly with the target sequence length, which is time-consuming for long sequences. To alleviate this problem, many recent works explore non-autoregressive (NAR) methods (Gu et al. 2018; Qian et al. 2021; Xiao et al. 2022) to predict a target sequence in parallel, which can dramatically increase inference speed. As the cost of increasing decoding speed, NAR models remove the internal dependency of the target sequence and perform each decoding prediction depending entirely upon the source/input sequence. Inevitably, the generation quality of NAR methods falls behind their AR counterparts without target-side information in decoding (Gu et al. 2018).

To achieve a better trade-off between inference speedup and generation quality, the conditional masked language model (CMLM) (Ghazvininejad et al. 2019) has been proposed and has already become one of the most competitive and widely-used NAR frameworks, which exploits an iterative mask-predict decoding strategy. In the training stage, CMLM leverages a masked language model objective to generate the masked subset of the target sequence in parallel conditioned on the source input and unmasked part in target sequence. During inference, CMLM first generates the whole target sequence in parallel (the first iteration) and then iteratively masks and predicts low-confidence tokens. Based on CMLM, many recent works have achieved performance improvements with advanced enhancement strategies from different perspectives, e.g., improving the inference strategy (Kasai et al. 2020a; Geng, Feng, and Qin 2021), benefiting from the AT counterpart (Hao et al. 2021), training with better criterion (Marjan et al. 2020; Du, Tu, and Jiang 2021), introducing self-correction mechanism (Huang, Perez, and Volkovs 2022) and pre-training (Li et al. 2022b).

In this paper, we further introduce a simple yet very effective strategy to enhance the refinement capability of CMLM without changing the model structure and the inference algorithm, named adaptive masking over masking (AMOM). Concretely, we present two adaptive masking operations for both the source and target sequence based on the conventional one-time masking in CMLM. The masking operation for the source sequence can make the encoder optimization easier by adaptively masking a proportion of tokens based on the masked target sequence. In contrast, the vanilla CMLM constructs multiple masked target sequences for each source sequence in model training, making the encoder difficult to converge (Guo, Xu, and Chen 2020). Another potential merit of the source-side masking is to improve the stability of the

*Corresponding Author

[1]https://github.com/amom-nar/AMOM

CMLM model against different decoder inputs by preventing the internal co-adaptation (akin to dropout (Hinton et al. 2012)). Moreover, cooperating it with the masking condition of the target sentence can better improve the ability rather than fixed masking. Notice that JM-NAT (Guo, Xu, and Chen 2020) also explores the source-side masking operation but has a clear difference from our strategy. It introduces a BERT-like masked language model task on the encoder side to enhance the encoder training, whereas our adaptive strategy does not introduce any extra task and can dynamically capture target-side information. The target-side adaptive masking operation is presented to enhance the refinement process of CMLM, motivated by the masking ratio changes of the target sequence in different inference iterations, which cannot be captured by the one-time masking. Simultaneously, unlike the adaptive target-side masking strategy in GLAT (Qian et al. 2021) to achieve curriculum learning, we design the masking strategy specially to encourage the model to perform steadily and conduct refinements effectively. We focus on the promotion of each iteration rather than only enhancing the first iteration in GLAT. More comparisons between our strategy and the counterparts used in GLAT can be found in the experiments part.

Though AMOM is simple, i.e., only two extra masking operations in model training, we find it is surprisingly effective on different sequence generation tasks, including neural machine translation, summarization, and code generation (**15** datasets in total). It achieves state-of-the-art performance on multiple datasets based on the vanilla CMLM, e.g., **34.62** BLEU score on WMT16 EN→RO, **34.82** BLEU on WMT16 RO→EN, and **34.84** BLEU on IWSLT De→En. AMOM even performs better than the strong autoregressive Transformer on **7** datasets with at least **2.2**× speedup.

Methodology

Our proposed adaptive masking over masking (AMOM) strategy is a simple yet effective add-on for the conditional masked language model (CMLM) (Ghazvininejad et al. 2019) training, which comprises two adaptive masking operations for the encoder and decoder, respectively, to enhance the encoder training and the refinement capability of CMLM. Specifically, we adopt the same encoder-decoder architecture as the CMLM.

Conditional Masked Language Model

A conditional masked language model feeds a source sequence X to the encoder and a target sequence in which part of the tokens are masked by replacing them with the `[mask]` token to the decoder. The training objective of CMLM is to learn to predict the masked tokens Y_{mask} in parallel given X and the unmasked tokens Y_{obs} in the rest part of the target sequence, based on the assumption that all target tokens in Y_{mask} are independent of each other, i.e., the prediction of each Y_{mask} token is merely conditioned on X and Y_{obs}. To eliminate the particularity of Y_{mask}, CMLM samples a different number of tokens each time as Y_{mask} from the uniformly distributed number between one to the target length during training, rather than a fixed proportion of the target sequence. The training objective of CMLM is to maximize:

$$\mathcal{L}_{\text{CMLM}} = \sum_{y_t \in Y_{mask}} \log P(y_t | Y_{obs}, X; \theta), \tag{1}$$

where θ denotes the trainable parameters of CMLM. Unlike AR methods that can automatically decide the decoding end by generating a special `[EOS]` (end of a sentence) token, typical NAR methods require learning to predict the target length in advance. CMLM adds a special token `[LENGTH]` (akin to the `[cls]` token in BERT) into its encoder to predict the target length. During inference, given the input X and the predicted target length, CMLM executes k iterations of mask-predict operation (Ghazvininejad et al. 2019) to create the final target sequence. At the first iteration, the CMLM predicts the entire Y in parallel fully depending on X. In the next $k-1$ iterations, CMLM repeatedly masks a specific number of low-confidence tokens generated from the last iteration and regenerates them in parallel.

Adaptive X Masking

Basically, CMLM leverages an encoder-decoder structure to achieve sequence to sequence generation, which requires the mutual cooperation between encoder and decoder. However, during model training, each X will be paired with multiple Y_{mask} due to the uniform masking strategy of CMLM, making the encoder optimization much harder than the decoder. Guo, Xu, and Chen also empirically prove that the convergence speed of the encoder is significant lower than the decoder. Another drawback of conditioning different Y_{mask} on the same X is the internal co-adaptation of X, i.e., each prediction of Y_{mask} relies on the whole input sequence, making the decoder less focused on the changes of decoder inputs.

To enhance the encoder training and address the above-mentioned flaws, we propose a simple yet effective adaptive masking for input X. Unlike previous research, our proposed adaptive X masking is included in the sequence to sequence generation task, and the number of masked tokens is coordinated with the number of masked Y tokens. More concretely, given a training pair (X, Y) in CMLM, where Y will be divided into Y_{obs} and Y_{mask}, the masking ratio α of Y can be calculated as $\frac{N_{mask}}{N_{obs}+N_{mask}}$. N_{obs} and N_{mask} denote the number of tokens in Y_{obs} and Y_{mask}, respectively. Then, we introduce a mapping function $\varphi(\cdot)$ to decide the masking ratio of X based on the masking ratio in Y, i.e., we will randomly mask $\varphi(\alpha) * L_X$ tokens in the source sequence, where L_X denotes the length of the source sequence. Then the training loss of CMLM with adaptive X masking can be computed as:

$$\mathcal{L}_{\text{cmlm}} = -\sum_{y_t \in Y_{mask}} \log P(y_t | Y_{obs}, \hat{X}; \theta), \tag{2}$$

where $\hat{X}$ refers to the input sequence with $\varphi(\alpha) * L_X$ tokens being masked. We introduce different variations of φ in Table 5 and compare their performance.

Adaptive Y Masking

As mentioned above, the superior performance of CMLM-based methods comes from the iterative refinement process,

i.e., the previously generated target sequence draft is repeatedly polished by regenerating a specific number of low-confidence tokens in the subsequent iterations. In seeing the self-correction nature of the refinement process, many recent works introduce a correction objective in CMLM training to enhance its refinement capability e.g., SMART (Ghazvininejad et al. 2020), CMLMC (Huang, Perez, and Volkovs 2022). Unlike these works that introduce extra training objectives and optimize the inference process of CMLM, we present an ultra-simple yet effective adaptive masking operation for Y in model training without any change to the CMLM inference[2]. Our strategy is motivated by the quality improvement of predicted tokens along with the refinement iterations, i.e., the proportion of low-confidence tokens (for regeneration in each iteration) from Y_{mask} will gradually decrease along with the refinement iterations, resulting in a varied masking ratio between Y_{mask} and Y_{obs} in the refinement process.

To capture the masking ratio changes in CMLM inference, we add another masking operation (adaptive Y masking) upon the one-time masking in the vanilla CMLM model. Specifically, for each training pair (X, Y), Y is divided into Y_{obs} and Y_{mask}. CMLM generates the masked tokens based on Y_{obs} and X, where the generated result is denoted as $\hat{Y}_{mask}$ to distinguish with Y_{mask}. Then, we compute the correctness ratio of predicted tokens in $\hat{Y}_{mask}$ by comparing with target tokens in Y_{mask}, formulated as $\beta = \frac{|\hat{Y}_{mask}=Y_{mask}|}{N_{mask}}$. Similar to adaptive X masking, we introduce another mapping function $\psi(\cdot)$ to decide the masking proportion of $\hat{Y}_{mask}$ and Y_{obs} tokens. Different types of mapping function $\psi(\cdot)$ are experimented in Analysis, and more details are given in Appendix. We assign a masking probability of $1 - \psi(\beta)$ to each token in $\hat{Y}_{mask}$ and a masking probability of $\psi(\beta)$ to each token in Y_{obs}. As a result, the newly masked tokens in the second time denote Y'_{mask}, and the rest tokens will serve as a new Y'_{obs}, for the next iteration. The training loss of the new subset Y'_{mask} is computed the same as the first-time masking in CMLM, formulated as:

$$\mathcal{L}_{\text{aday}} = - \sum_{y_t \in Y'_{mask}} \log P(y_t | Y'_{obs}, \hat{X}'; \theta), \tag{3}$$

where $\hat{X}'$ refers to the input sequence with an adaptive masking ratio of Y'_{mask} being masked.

AMOM Training and Inference

We simply adopt two adaptive masking strategies based on the original CMLM training process. The training objective of our proposed adaptive masking over masking (AMOM) is the simple combination of $\mathcal{L}_{\text{cmlm}}$ and $\mathcal{L}_{\text{aday}}$ mentioned in Equation 2 and 3, formulated as:

$$\mathcal{L}_{\text{AMOM}} = \mathcal{L}_{\text{cmlm}} + \mathcal{L}_{\text{aday}}, \tag{4}$$

As for inference, we utilize the same decoding strategy with CMLM. As mentioned above, we utilize a special token `[LENGTH]` in the encoder to predict the target length in advance. Inevitably, there is a deviation between the predicted length and the ground-truth length. Thus, we also consider selecting the translation with the highest probability with different target lengths to obtain better results. Given the target length L_Y and the total number of refinement iterations T, the model performs generation based on the fully masked decoder input (i.e., empty Y_{obs}) at the first iteration. In the next $T - 1$ iterations, a specific number of low-confidence tokens will be masked and re-generated. The number of masked tokens in each iteration can be computed as $n = \frac{T-t}{T} * L_Y$, where t denotes the current iteration number. Given the number of masked tokens, the model will select them based on the output probability of each token, where tokens with the lowest probability will be masked, and their scores will be updated in the next iteration.

[2]More comparisons are given in Appendix.

Experiments

To evaluate our AMOM method and show its universal impact on various sequence generation tasks, we conduct experiments on natural machine translation, summarization, and code generation tasks.

Datasets

For machine translation, we conduct experiments both on IWSLT and WMT datasets, which are widely used for NMT tasks. The datasets from IWSLT competitions contain 4 language pairs (170k pairs), see details in Table 2. For WMT datasets, we choose two language pairs which are widely used in non-autoregressive machine translation task, WMT16 English→Roman (0.6M pairs) and WMT14 English→German (4.5M pairs) tasks. Following previous works on non-autoregressive machine translation, we apply sequence-level knowledge distillation (Kim and Rush 2016; Zhou, Gu, and Neubig 2019) for all datasets. For WMT datasets, we use the same distilled data as the same as CMLM (Ghazvininejad et al. 2019). Then, we amalgamate the raw and distilled data as our final training data, following (Ding et al. 2020). For all IWSLT datasets, we train the teacher model with Transformer$_{small}$, and use the generated results as the distilled data. Then, we train our AMOM on distilled data. For summarization task, we use the XSUM dataset (Narayan, Cohen, and Lapata 2018) which contains 204,045/11,332/11,334 online articles and single sentence summary pairs from the British Broadcasting Corporation for training/validation/test. We preprocess the dataset, following (Lewis et al. 2020). For code generation task, we use Py150 dataset (Raychev, Bielik, and Vechev 2016) and use GitHub-Java dataset (Allamanis and Sutton 2013). We use the Python official library tokenizer[3] and Javalang[4] to split the datasets into lines of codes. Then we use a sliding context window to adopt 10-lines of code tokens as the source sentences and the next 4-lines as the target sentences. We follow (Wang et al. 2020) to process the dataset to transform some special tokens as `[str]` token (without bpe).

Settings

All experiments are done using the Fairseq library (Ott et al. 2019). Following previous settings (Ghazvininejad

[3]https://docs.python.org/3/library/tokenize.html

[4]https://github.com/c2nes/javalang

	Model	Iterations	WMT16		WMT14		Speedup
			EN→RO	RO→EN	EN→DE	DE→EN	
AR	Transformer (Vaswani et al. 2017)*	N	34.23	34.28	28.41	32.28	1.0x
Full NAT	NAT-FT (Gu et al. 2018)	1	27.29	29.06	17.69	21.47	15.6×
	AXE (Marjan et al. 2020)	1	31.54	30.75	23.53	-	15.3x
	OAXE (Du, Tu, and Jiang 2021)	1	33.3	32.4	26.1	-	15.3x
	GLAT (Qian et al. 2021)	1	32.87	33.51	26.55	31.02	15.3x
	FullyNAT (Gu and Kong 2021)	1	33.71	34.16	27.20	31.39	16.8x
	DSLP (Huang et al. 2022a)	1	34.17	34.60	27.02	31.61	14.8x
	DAT (Huang et al. 2022b)	1	-	-	27.49	31.37	13.9x
Iterative	Refine-NAT (Lee, Mansimov, and Cho 2018)	10	27.11	30.19	21.61	25.48	1.5x
	LevenshteinNAR (Gu, Wang, and Zhao 2019)	>7	33.02	-	27.73	-	4.0x
	DisCo (Kasai et al. 2020a)	3.1	33.25	33.22	27.34	-	3.5x
CMLM-Based	CMLM (Ghazvininejad et al. 2019)*	10	33.46	33.83	27.21	31.03	2.3x
	SMART (Ghazvininejad et al. 2020)	10	33.85	33.53	27.65	31.27	1.7x
	JM-NAT (Guo, Xu, and Chen 2020)	10	33.52	33.72	27.69	**32.24**	-
	RDP (Ding et al. 2020)	10	33.7	-	27.8	-	1.5x
	LFR (Ding et al. 2021)	10	-	33.9	27.8	-	1.5x
	MvSR-NAT (Xie, Li, and Hu 2021)	10	33.38	33.56	27.39	31.18	3.8x
	CORR (Huang, Perez, and Volkovs 2022)	10	34.31	34.08	28.19	31.31	-
	CMLMC (Huang, Perez, and Volkovs 2022)	10	34.57	34.13	**28.37**	31.41	-
Ours	AMOM	10	**34.62**	**34.82**	27.57	31.67	2.3x

Table 1: Results on 4 WMT machine translation tasks. "*" denotes the results of our implementations.

Model	En↔De	En↔Fr	En↔Zh	En↔Es	Avg	Speedup
Transformer	28.71/34.68	36.2/37.0	25.7/18.2	37.8/39.5	32.22	1.0x
CMLM	27.77/33.87	35.2/35.0	26.0/17.9	37.1/39.0	31.48	2.2x
AMOM	28.41/**34.84**	35.6/36.3	**26.1/18.4**	**38.0/39.8**	32.18	2.2x

Table 2: Results on 8 IWSLT datasets. Numbers before and after "/" denote BLEU scores from and to English directions.

et al. 2019), we use the standard Transformer$_{base}$ configuration on WMT datasets and standard Transformer$_{small}$ configuration on IWSLT datasets for both auto-regressive and non-autoregressive experiments. During AMOM training, we follow the hyper-parameters in CMLMC (Huang, Perez, and Volkovs 2022) for WMT14 En↔De and follow the hyper-parameters of CMLM realization in Fairseq[5] for the other datasets. During inference, we average the 5 best checkpoints chosen by validation BLEU scores as our final model and set the length beam as 3/5 for IWSLT/WMT datasets. For XSUM, we choose Transformer$_{base}$ with embedding dimension 768 and follow the training schedule applied in NMT. During our training, we make a specific modification of the hyper-parameters referring to (Lewis et al. 2020). During inference we follow the process in (Qi et al. 2021), where the same consecutive tokens will be merged to avoid repeated n-gram tokens. For code generation tasks, we choose Transformer$_{base}$ with embedding size 512 and follow the original training schedule. We make a specific modification of the hyper-parameters referring to (Liu et al. 2022). For all datasets, we set the limits ratio of adaptive X from 10%-30% and adaptive Y from 20%-80%, and select a linear mapping function to decide the masking ratios. More details about training are presented in Appendix.

Main Results

Natural Machine Translation. Following previous works, we evaluate the performance with BLEU (Papineni et al. 2002) for WMT datasets and IWSLT En↔De dataset, and for the other IWSLT datasets, we use SacreBLEU [6] (Post 2018; Liang et al. 2021). Speedup is measured by L_1^{GPU} following the previous work (Kasai et al. 2020b; Gu and Kong 2021; Helcl, Haddow, and Birch 2022). Table 2 presents the results on 8 IWSLT datasets, we compare our AMOM with original CMLM and strong Transformer (AR) baseline. First, a significant improvement can be found over the original CMLM on all datasets, with about 0.7 BLEU on average. More excitingly, compared with the strong Transformer (AR) baseline, our AMOM has achieved better performance on five datasets, and only a tiny gap (0.04 BLEU) still exists on average. We show our results in Table 1 for WMT datasets, we compare our approach with various iterative NAR models, including two popular fully NAR models.

[5]https://github.com/facebookresearch/fairseq/tree/main/examples/nonautoregressive_translation

[6]https://github.com/mjpost/sacrebleu

Model	ROUGE-1	ROUGE-2	ROUGE-L
Transformer	30.66	10.80	24.48
Without pretrain			
vanilla NAT	24.04	3.88	20.32
InsertNAR	17.65	5.18	16.05
Levenshitein	25.33	7.40	21.48
Disco	26.85	6.86	21.72
POSPD	27.39	7.26	22.15
CMLM*	25.80	6.31	20.45
AMOM*	**31.59**	**9.30**	**24.98**
With pretrain			
BANG	34.71	11.71	29.16
MIST	34.63	11.29	28.70
ELMER	**37.30**	**13.17**	**29.92**

Table 3: Results on XSUM for the text summarization task. "*" denotes the results of our implementations.

Model	Python			JAVA		
	Iter.	BLEU	ES	Iter.	BLEU	ES
CMLM	4	49.61	69.58	4	60.54	76.68
	10	53.44	70.42	10	62.82	77.24
AMOM	4	50.57	70.22	4	62.86	76.61
	10	**56.50**	**71.38**	10	**65.43**	77.17

Table 4: Results on Py150 and Github-Java dataset.

We re-run the experiments of CMLM with the same settings in AMOM to avoid inconsistency. After applying our simple yet effective methods to the traditional CMLM framework, we achieved state-of-the-art (SOTA) BLEU score on WMT16 En→Ro (34.62) and Ro→En (34.82) with 10 iterations. For the WMT14 En↔De dataset, AMOM also outperforms most of the baselines on De→En (31.67). On the En→De dataset, AMOM only gains 0.36 BLEU improvement compared with CMLM and a comparable score compared with strong CMLM-Based baselines. This might be because our adaptive X strategy hurts the performance in the first iteration to some extent. Note that AMOM is complementary to other effective tricks applied in CMLM, and stronger results can be expected by combining our adaptive masking strategies with their methods.

Summarization. See Table 3, the performance is evaluated by ROUGE F1 score (Lin and Hovy 2002). Specifically, we report the unigram ROUGE-1 and bigram ROUGE-2 overlap to assess the informativeness, and the longest common subsequence ROUGE-L score to assess the fluency. We compare our AMOM with the original CMLM and several NAR baseline models, including vanilla NAT (Gu et al. 2018), InsertNAR (Stern et al. 2019), Levenshitein (Gu, Wang, and Zhao 2019), Disco (Kasai et al. 2020a), POSPD (Yang et al. 2021), CMLM (Ghazvininejad et al. 2019), BANG (Qi et al. 2021), MIST (Jiang et al. 2021), ELMER (Li et al. 2022a). Results show that AMOM outperforms all other NAR models without pre-training. Since pre-training always benefits summarization task a lot, models with pre-training achieve significant performance improvements. Notice that AMOM can also be applied to the pre-training and finetune stage, we believe it also works to improve the performance.

Code Generation. The performance is evaluated by BLEU and ES (Wang et al. 2020), which measure character-level edit similarity and n-gram level precision between the target codes and generated codes, respectively. We also report the results of different iterations in Table 4. Our AMOM outperforms the original CMLM with different iterations and gains better improvements during refinements.

Analysis

The Mapping Function of Adaptive X Masking. In this subsection, we exhibit exhaustive experiments to explore encoder masking strategies and how to affect the model performance. In particular, we analyse the effects of different mapping functions, these strategies can utilize decoder masking ratio α_{dec} to obtain encoder masking ratio α_{enc}:

- φ_{linear}: $\alpha_{enc} = (b-a)\alpha_{dec} + a$;
- φ_{convex}: $\alpha_{enc} = (b-a)\alpha_{dec}^2 + b$;
- $\varphi_{concvae}$: $\alpha_{enc} = (a-b)\alpha_{dec}^2 + 2(b-a)\alpha_{dec} + b$;
- φ_{ladder}: $\alpha_{enc} = a - \lceil \frac{\alpha_{dec}}{a-b+0.1} \rceil$,

where a and b are two hyper-parameters controlling the masking limits, and the specific curves corresponding to the above mapping function are presented in Appendix. The results are shown in Table 5, and it is worth noting that the above experiments are based on the CMLM model and IWSLT14 De→En dataset for clear contrast. Early experiments show that encoder masking can boost the model performance, and at $\alpha_{enc} = 0.2$, the encoder masked model performs best when using the fixed masking strategy, results are shown in Appendix. That is why we design the mapping function to limit the masking raio around 0.2. Firstly, we take linear mapping functions as our priority. Fortunately, linear mapping has been proved by comprehensive experiments that it is indeed one of the most effective implementations to boost the performance. Besides, the results are consistent with our intuition that the more tokens in Y are masked, the few tokens in X should be masked to keep the masking ratio balanced. We also have briefly tried a few alternative implementations beyond linear mappings, but without achieving further performance improvement.

The Effect of Adaptive X Masking. We also compare our adaptive X masking strategy with several related works to further show its effectiveness. Since JM-NAT (Guo, Xu, and Chen 2020) also introduces masking operation in X, we also conduct experiments to compare AMOM and their bert-like masking. Also, they introduce an auxiliary MLM training objective to improve the encoder, we further verify if this can combine with AMOM, see Table 6. Notice that we keep the decoder-side the same as vanilla CMLM (without adaptive Y masking in AMOM and n-gram loss in JM-NAT) to make a fair comparison of encoder-side. Results show that this MLM training objective can also improve AMOM slightly, but seems less related to our assumption and purpose. Besides, we can find adaptive X outperforms the bert-like masking for CMLM. Also, we find that the adaptive X

Strategy	α_{enc}	**BLEU**
Linear	$\varphi_{linear}(\alpha_{dec}, 0.25, 0.15)$	34.20
	$\varphi_{linear}(\alpha_{dec}, 0.3, 0.1)$	**34.48**
	$\varphi_{linear}(\alpha_{dec}, 0.35, 0.15)$	34.30
	$\varphi_{linear}(\alpha_{dec}, 0.4, 0.1)$	34.40
	$\varphi_{linear}(\alpha_{dec}, 0.1, 0.3)$	33.64
	$\varphi_{linear}(\alpha_{dec}, 0.1, 0.4)$	33.76
Convex	$\psi_{convex}(\alpha_{dec}, 0.3, 0.1)$	33.55
Concave	$\psi_{concave}(\alpha_{dec}, 0.3, 0.1)$	33.96
Ladder	$\psi_{ladder}(\alpha_{dec}, 0.3, 0.1)$	34.17

Table 5: The BLEU scores of adaptive X masking strategy.

Method	BLEU	Method	BLEU
CMLM	33.87	CMLM	33.87
+ adax	**34.48**	+ mix cutoff	33.96
+ adax+mlmloss	**34.57**	+ span cutoff	33.93
+ jm-nat	34.13	+ random replace	34.13
+ jm-nat+mlmloss	34.21	+ random delete	33.95

Table 6: Comparison between adaptive X masking and related methods.

Strategy	α_{dec}	**BLEU**
Linear	$\psi_{linear}(\beta, 0.1, 0.9)$	34.65
	$\psi_{linear}(\beta, 0.2, 0.8)$	**34.84**
	$\psi_{linear}(\beta, 0.3, 0.7)$	34.79
	$\psi_{linear}(\beta, 0.2, 0.5)$	34.62
	$\psi_{linear}(\beta, 0.5, 0.8)$	34.77
	$\psi_{linear}(\beta, 0.8, 0.2)$	34.61
Convex	$\psi_{convex}(\beta, 0.2, 0.8)$	34.80
Concave	$\psi_{concave}(\beta, 0.2, 0.8)$	34.59
Ladder	$\psi_{ladder}(\beta, 0.2, 0.8)$	34.75

Table 7: The BLEU scores of adaptive Y masking strategy.

Masking Strategy	**BLEU**	**Masking Strategy**	**BLEU**
Adaptive (Ours)	**34.84**	Uniform	34.53
+ same ratio	34.65	Glancing	34.68
+ 3 step	34.50	Glat	33.72
+ exposure bias	34.79	Glat + fix-x (0.1)	33.64
+ confidence-based	33.85	Glat + ada-x	33.35

Table 8: Comparison of adaptive Y masking with different constraints and related methods.

masking operation is similar to a data augmentation strategy (such as cutoff (Shen et al. 2020)), and specially designed to improve the refinements ability of CMLM. To better analyze them, we also compare adaptive X masking with several common data augmentation strategies (including cutoff). Since fixed masking is similar to token cutoff, we conduct experiments with span cutoff and mix cutoff. We also compare with some other strategies (such as random delete, random replace). Results show that adaptive X masking outperforms all other operations on X, while various traditional strategies can boost vanilla CMLM to some extent.

The Mapping Function of Adaptive Y Masking. We also experiment with different masking strategies when applied to the decoder side in a two-step training scheme. We try same adaptive mapping function and denoted as ψ_{linear}, ψ_{convex}, $\psi_{concvae}$, and ψ_{ladder} to obtain masking ratio α_{dec}. Specifically, we can calculate α_{dec} based on randomly sampled variable β which is correctness ratio predicted by first step training as mentioned above : $\alpha_{dec} = \psi_{linear}(\beta, a, b) = (b-a)\beta + a$. Unlike the encoder masking mapping function, we choose a large masking ratio range because there exist various conditions of masking ratios and tokens confidence during inference. The schedule curves are also shown in Appendix. Table 7 lists the results of several adaptive decoder masking strategies. Notice that we achieve all results here with a linear mapping $\varphi_{linear}(\alpha_{dec}, 0.3, 0.1)$ for source-side masking. The simple linear mapping function achieves the best performance, and the large masking ratio range seems better. Besides, a high correctness ratio always indicates high token confidence, and then fewer tokens in $\hat{Y}_{mask}$ will be masked in the next iteration. Our adaptive Y masking strategy matches the inference strategy of the original CMLM.

The Effect of Adaptive Y Masking. To better understand the two-step training scheme and how to guide model training, we analyze the effect of different masking and training settings, and notice that we all keep the uniform masking strategy in the first step as the original CMLM. First, we use uniform sampling to replace adaptive ψ sampling in the second masking step. Then we also keep the masking ratio $\alpha_{dec} = \beta$ to verify whether the masking ratio is critical for model training. Besides, we use an adaptive strategy to train three steps which simulate the multi-step inference scenarios. We also test the impact of whether recover ground truth tokens or keep the predicted token in the second training step. Since the masking tokens are chosen by prediction confidence during inference, we also apply confidence-based masking during training to further verify our adaptive Y masking. Moreover, we also compare our adaptive Y masking with the glancing masking strategy proposed in GLAT to improve the one-pass decoding. The results are shown in Table 8. We can observe that adaptive masking outperforms uniform masking in the second-step training, and the uniform masking seems to bring little improvements compared with adaptive X masking (34.48). This also indicates that although AMOM may expand training expenses, adaptive Y masking is truly valuable, and the performance improvements do not come from more updates. Moreover, results also reflect that two-step refinements are enough for model training without the necessity for more steps. Besides, using model prediction instead of ground truth can effectively reduce the problem of exposure bias, and introducing a confidence-based masking strategy does not bring improvements. Compared with GLAT, adopting the glancing masking as the second step masking strategy also performs better than uniform masking but is inferior to our adaptive

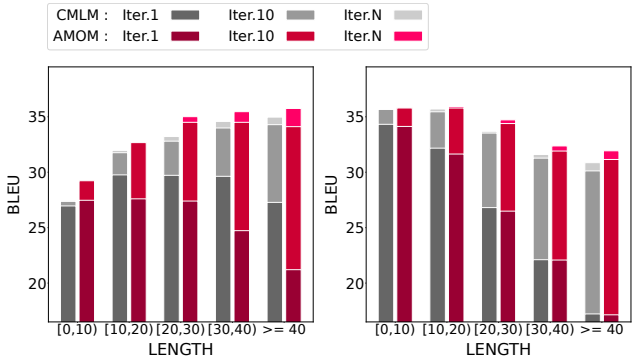

Figure 1: Comparison between different source language sentence length and decoding iterations.

Y masking. Besides, if we directly adopt glancing masking and one-step training the same as GLAT (Glat), the performance declines, and further combining it with encoder masking even harms the performance. This indicates that our methods play a different role compared with GLAT.

More Iterations for Long Sequence. For long source input sentences, it is almost impossible to obtain a fluent and relatively correct result for non-autoregressive machine translation models. It often requires multiple iterations to refine the translation results. Therefore, the ability to refine is a crucial evaluation criterion for a model. First, we compare the BLEU scores of AMOM and CMLM in different iterations steps, as shown in Appendix. We can see that the AMOM outperforms the CMLM model when the iterations step increases, which proves that an adaptive masking strategy can enhance refinement ability. In addition, we make a comparison of results with different source sentence length N and different decoding iterations T on two two datasets (IWSLT DE→EN and WMT EN→RO). We split each dataset into five segments according to sentence length and run inference three times according to different steps $N \in [1, 10, N]$. In Figure 1, we present the improvements of more decoding steps with different colours. Results show that AMOM exhibit significant gain than vanilla CMLM with more steps, e.g., although the performance of AMOM in Iter.1 is inferior than CMLM, it all outperforms CMLM in Iter.10, especially for long sentences. We can also find that long sentences often require more decoding steps, and AMOM perform better.

Related Work

Iterative-based Non-autoregressive Sequence Generation. Non-autoregressive models have attracted an increasing attention in recent years due to their efficient decoding, but the improvements in decoding speed come at the expense of generation quality. Thus, iterative-based non-autoregressive (NAR) models (Lee, Mansimov, and Cho 2018; Gu, Wang, and Zhao 2019; Saharia et al. 2020; Geng, Feng, and Qin 2021; Lu, Meng, and Peng 2022) are proposed to achieve a better trade-off between the inference speedup and generation quality. Lee, Mansimov, and Cho first propose the iterative model which aims refine the noised target sequence. Later, insertion and deletion operations are introduced in each decoding iteration to create the final translation. Among these iterative NAR methods, the conditional masked language model (CMLM) (Ghazvininejad et al. 2019) is widely-used owing to its promising performance when using the mask-predict strategy. In particular, CMLM leverages the masked language model objective to guide model training and iteratively masks and predicts tokens during inference. Many recently works have achieved performance improvements based on CMLM (Guo, Xu, and Chen 2020; Huang, Perez, and Volkovs 2022). Recently, Savinov et al. proposed step-unrolled denoising autoencoder which adopts denoising operation in each iteration.

Masked Language Model. The masked language model (MLM) first introduced by BERT (Devlin et al. 2018) has become the essential component of various popular pre-training methods (Song et al. 2019; Liu et al. 2019; Dong et al. 2019; Joshi et al. 2020; Li et al. 2022b; Xu, Van Durme, and Murray 2021). Its standard paradigm is to select some tokens in the source sequence by different strategies and then replace them with a `[mask]` token, and then the model is trained to predict the masked tokens. Since the masking strategy is significantly essential for these model, different masking strategies are served as different learning methods. As BERT is served as a single Transformer encoder and a monolingual framework, there are limitations in various applications, such as machine translation. Then much progress has been made to extend the applications of masked language modeling strategy (Guo et al. 2020; Zhu et al. 2020; Li et al. 2022b). The CMLM-based non-autoregressive models can also benefit from it by introducing a uniform masking strategy in training and a mask-predict decoding strategy during inference (Ghazvininejad et al. 2019). However, only few improvements on masking strategies are explored for CMLM. In this work, we further design a simple yet effective adaptive masking over masking method on both the encoder and decoder sides to enhance the CMLM training for better refinement capability during inference.

Conclusion

In this paper, we present an adaptive masking over masking (AMOM) strategy to enhance the conditional masked language model (CMLM) for non-autoregressive sequence generation. Our AMOM only contains two masking operations in model training without modifying the model structure or changing the inference schedule. Extensive experiments on different sequence generation tasks indicate our proposed AMOM can yield significant performance improvement over the original CMLM model and even outperform the strong autoregressive (Transformer) counterpart on 7 NMT benchmark datasets and achieves SOTA performance on WMT16 EN→RO, **34.82** BLEU on WMT16 RO→EN, and **34.84** BLEU on IWSLT De→En. Due to the limitation of computational resources, we only test our AMOM for the CMLM model. In the near future, we will design more elegant AMOM strategies and explore their effectiveness on different NAR frameworks. We also will extend our AMOM to other types of masked language models, both in the pre-training and fine-tuning stages.

Acknowledgments

Ruiyang Xu contributes equally with Yisheng Xiao. Juntao Li is the corresponding author. This work is supported by the National Science Foundation of China (NSFC No. 62206194), the Natural Science Foundation of Jiangsu Province, China (No. BK20220488), and the Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions. This work is also supported by Beijing Academy of Artificial Intelligence (BAAI).

References

Allamanis, M.; and Sutton, C. 2013. Mining source code repositories at massive scale using language modeling. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, 207–216. IEEE.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ding, L.; Wang, L.; Liu, X.; Wong, D. F.; Tao, D.; and Tu, Z. 2020. Understanding and Improving Lexical Choice in Non-Autoregressive Translation. In *ICLR*.

Ding, L.; Wang, L.; Liu, X.; Wong, D. F.; Tao, D.; and Tu, z. 2021. Rejuvenating Low-Frequency Words: Making the Most of Parallel Data in Non-Autoregressive Translation. In *ACL-IJCNLP*, 3431–3441.

Dong, L.; Yang, N.; Wang, W.; Wei, F.; Liu, X.; Wang, Y.; Gao, J.; Zhou, M.; and Hon, H.-W. 2019. Unified language model pre-training for natural language understanding and generation. *NeurIPS*, 32.

Du, C.; Tu, Z.; and Jiang, J. 2021. Order-agnostic cross entropy for non-autoregressive machine translation. In *ICML*, 2849–2859. PMLR.

Elsaid, A.; Mohammed, A.; Fattouh, L.; and Sakre, M. 2022. A Comprehensive Review of Arabic Text summarization. *IEEE Access*.

Geng, X.; Feng, X.; and Qin, B. 2021. Learning to Rewrite for Non-Autoregressive Neural Machine Translation. In *EMNLP*, 3297–3308.

Ghazvininejad, M.; Levy, O.; Liu, Y.; and Zettlemoyer, L. 2019. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In *EMNLP-IJCNLP*, 6112–6121.

Ghazvininejad, M.; Levy, O.; Zettlemoyer; and Luke. 2020. Semi-autoregressive training improves mask-predict decoding. *arXiv preprint arXiv:2001.08785*.

Gu, J.; Bradbury, J.; Xiong, C.; Li, V. O.; and Socher, R. 2018. Non-Autoregressive Neural Machine Translation. In *ICLR*.

Gu, J.; and Kong, X. 2021. Fully Non-autoregressive Neural Machine Translation: Tricks of the Trade. In *Findings of ACL-IJCNLP*, 120–133.

Gu, J.; Wang, C.; and Zhao, J. 2019. Levenshtein Transformer. *NeurIPS*, 32: 11181–11191.

Guo, J.; Xu, L.; and Chen, E. 2020. Jointly masked sequence-to-sequence model for non-autoregressive neural machine translation. In *ACL*, 376–385.

Guo, J.; Zhang, Z.; Xu, L.; Wei, H.-R.; Chen, B.; and Chen, E. 2020. Incorporating BERT into Parallel Sequence Decoding with Adapters. In *NeurIPS*.

Hao, Y.; He, S.; Jiao, W.; Tu, Z.; Lyu, M.; and Wang, X. 2021. Multi-Task Learning with Shared Encoder for Non-Autoregressive Machine Translation. In *NAACL-HLT*, 3989–3996.

Helcl, J.; Haddow, B.; and Birch, A. 2022. Non-Autoregressive Machine Translation: It's Not as Fast as it Seems. *arXiv preprint arXiv:2205.01966*.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Huang, C.; Zhou, H.; Zaïane, O. R.; Mou, L.; and Li, L. 2022a. Non-autoregressive translation with layer-wise prediction and deep supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10776–10784.

Huang, F.; Zhou, H.; Liu, Y.; Li, H.; and Huang, M. 2022b. Directed Acyclic Transformer for Non-Autoregressive Machine Translation. *arXiv preprint arXiv:2205.07459*.

Huang, X. S.; Perez, F.; and Volkovs, M. 2022. Improving Non-Autoregressive Translation Models Without Distillation. In *ICLR*.

Jiang, T.; Huang, S.; Zhang, Z.; Wang, D.; Zhuang, F.; Wei, F.; Huang, H.; Zhang, L.; and Zhang, Q. 2021. Improving Non-autoregressive Generation with Mixup Training. *arXiv preprint arXiv:2110.11115*.

Joshi, M.; Chen, D.; Liu, Y.; Weld, D. S.; Zettlemoyer, L.; and Levy, O. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8: 64–77.

Kasai, J.; Cross, J.; Ghazvininejad, M.; and Gu, J. 2020a. Parallel machine translation with disentangled context transformer. *arXiv preprint arXiv:2001.05136*.

Kasai, J.; Pappas, N.; Peng, H.; Cross, J.; and Smith, N. 2020b. Deep Encoder, Shallow Decoder: Reevaluating Non-autoregressive Machine Translation. In *ICLR*.

Kim, Y.; and Rush, A. M. 2016. Sequence-Level Knowledge Distillation. In *EMNLP*, 1317–1327.

Lee, J.; Mansimov, E.; and Cho, K. 2018. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *EMNLP*, 1173–1182.

Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; and Zettlemoyer, L. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *ACL*, 7871–7880.

Li, J.; Tang, T.; Zhao, W. X.; Nie, J.-Y.; and Wen, J.-R. 2022a. ELMER: A Non-Autoregressive Pre-trained Language Model for Efficient and Effective Text Generation. *arXiv preprint arXiv:2210.13304*.

Li, P.; Li, L.; Zhang, M.; Wu, M.; and Liu, Q. 2022b. Universal Conditional Masked Language Pre-training for Neural Machine Translation. *ACL*.

Liang, X.; Wu, L.; Li, J.; Wang, Y.; Meng, Q.; Qin, T.; Chen, W.; Zhang, M.; Liu, T.-Y.; et al. 2021. R-drop: regularized dropout for neural networks. *NeurIPS*, 34.

Lin, C.-Y.; and Hovy, E. 2002. Manual and automatic evaluation of summaries. In *Proceedings of the ACL-02 Workshop on Automatic Summarization*, 45–51.

Liu, F.; Fu, Z.; Li, G.; Jin, Z.; Liu, H.; and Hao, Y. 2022. Non-autoregressive Model for Full-line Code Completion. *arXiv preprint arXiv:2204.09877*.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lu, S.; Meng, T.; and Peng, N. 2022. InsNet: An Efficient, Flexible, and Performant Insertion-based Text Generation Model. In *Advances in Neural Information Processing Systems*.

Ma, Y.; Nguyen, K. L.; Xing, F. Z.; and Cambria, E. 2020. A survey on empathetic dialogue systems. *Information Fusion*, 64: 50–70.

Marjan, G.; Karpukhin, V.; Zettlemoyer, L.; and Levy, O. 2020. Aligned cross entropy for non-autoregressive machine translation. In *ICML*, 3515–3523. PMLR.

Narayan, S.; Cohen, S.; and Lapata, M. 2018. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. In *EMNLP*, 1797–1807.

Ott, M.; Edunov, S.; Baevski, A.; Fan, A.; Gross, S.; Ng, N.; Grangier, D.; and Auli, M. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *NAACL-HLT*.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 311–318.

Post, M. 2018. A call for clarity in reporting BLEU scores. *arXiv preprint arXiv:1804.08771*.

Qi, W.; Gong, Y.; Jiao, J.; Yan, Y.; Chen, W.; Liu, D.; Tang, K.; Li, H.; Chen, J.; Zhang, R.; et al. 2021. Bang: Bridging autoregressive and non-autoregressive generation with large scale pretraining. In *ICML*, 8630–8639. PMLR.

Qian, L.; Zhou, H.; Bao, Y.; Wang, M.; Qiu, L.; Zhang, W.; Yu, Y.; and Li, L. 2021. Glancing Transformer for Non-Autoregressive Neural Machine Translation. In *ACL-IJCNLP*, 1993–2003.

Raychev, V.; Bielik, P.; and Vechev, M. 2016. Probabilistic model for code with decision trees. *ACM SIGPLAN Notices*, 51(10): 731–747.

Saharia, C.; Chan, W.; Saxena, S.; and Norouzi, M. 2020. Non-Autoregressive Machine Translation with Latent Alignments. In *EMNLP*, 1098–1108.

Savelieva, A.; Au-Yeung, B.; and Ramani, V. 2020. Abstractive summarization of spoken and written instructions with BERT. *arXiv preprint arXiv:2008.09676*.

Savinov, N.; Chung, J.; Binkowski, M.; Elsen, E.; and van den Oord, A. 2021. Step-unrolled Denoising Autoencoders for Text Generation. In *International Conference on Learning Representations*.

Shen, D.; Zheng, M.; Shen, Y.; Qu, Y.; and Chen, W. 2020. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818*.

Song, K.; Tan, X.; Qin, T.; Lu, J.; and Liu, T.-Y. 2019. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*.

Stern, M.; Chan, W.; Kiros, J.; and Uszkoreit, J. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *ICML*, 5976–5985. PMLR.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*, 5998–6008.

Wang, W.; Shen, S.; Li, G.; and Jin, Z. 2020. Towards Full-line Code Completion with Neural Language Models. *arXiv preprint arXiv:2009.08603*.

Wu, L.; Wang, Y.; Xia, Y.; Tian, F.; Gao, F.; Qin, T.; Lai, J.; and Liu, T.-Y. 2019. Depth Growing for Neural Machine Translation. In *ACL*, 5558–5563.

Xiao, Y.; Wu, L.; Guo, J.; Li, J.; Zhang, M.; Qin, T.; and Liu, T.-y. 2022. A Survey on Non-Autoregressive Generation for Neural Machine Translation and Beyond. *arXiv preprint arXiv:2204.09269*.

Xie, P.; Li, Z.; and Hu, X. 2021. MvSR-NAT: Multi-view Subset Regularization for Non-Autoregressive Machine Translation. *arXiv preprint arXiv:2108.08447*.

Xu, H.; Van Durme, B.; and Murray, K. 2021. BERT, mBERT, or BiBERT? A Study on Contextualized Embeddings for Neural Machine Translation. *arXiv preprint arXiv:2109.04588*.

Yang, K.; Lei, W.; Liu, D.; Qi, W.; and Lv, J. 2021. POS-Constrained Parallel Decoding for Non-autoregressive Generation. In *ACL*, 5990–6000.

Zhang, Y.; Sun, S.; Galley, M.; Chen, Y.-C.; Brockett, C.; Gao, X.; Gao, J.; Liu, J.; and Dolan, W. B. 2020. DIALOGPT: Large-Scale Generative Pre-training for Conversational Response Generation. In *ACL,System Demonstrations*, 270–278.

Zhou, C.; Gu, J.; and Neubig, G. 2019. Understanding Knowledge Distillation in Non-autoregressive Machine Translation. In *ICLR*.

Zhu, J.; Xia, Y.; Wu, L.; He, D.; Qin, T.; Zhou, W.; Li, H.; and Liu, T.-Y. 2020. Incorporating bert into neural machine translation. *arXiv preprint arXiv:2002.06823*.