An Empirical Study of Iterative Refinements for Non-autoregressive Translation

Yisheng Xiao[♡], **Pei Guo**[♡], **Zechen Sun**[♡], **Juntao Li**[♡], **Kai Song**[◇], **Min Zhang**[♡]
[♡]Soochow University, China
[◇]ByteDance
{ysxiaoo,pguolst,zcsuns}@stu.suda.edu.cn
{ljt,minzhang}@suda.edu.cn
songkai.neu@gmail.com

Abstract

Iterative non-autoregressive (NAR) models share a spirit of mixed autoregressive (AR) and fully NAR models, seeking a balance between generation quality and inference efficiency. These models have recently demonstrated impressive performance in varied generation tasks, surpassing the autoregressive Transformer. However, they also face several challenges that impede further development. In this work, we target building more efficient and competitive iterative NAR models. Firstly, we produce two simple metrics to identify the potential problems existing in current refinement processes, and look back on the various iterative NAR models to find the key factors for realizing our purpose. Subsequently, based on the analyses of the limitations of previous inference algorithms, we propose a simple yet effective strategy to conduct efficient refinements without performance declines. Experiments on five widely used datasets show that our final models set the new state-of-the-art performance compared to all previous NAR models, even with fewer decoding steps, and outperform AR Transformer by around one BLEU on average. Our codes and models are available on Github[1].

1 Introduction

Transformer-based models (Vaswani et al., 2017) have achieved promising performance in various tasks, particularly after the emergence and progress of large language models recently (Touvron et al., 2023a; OpenAI, 2023; Touvron et al., 2023b). However, these models adopt an autoregressive (AR) decoding paradigm where tokens are generated one by one in a strict left-to-right order. Consequently, they suffer from low inference efficiency, which even worsens as model parameters increase (Zhao et al., 2023). Non-autoregressive (NAR) models provide an alternative text generation paradigm (Gu et al., 2018). Unlike AR models, NAR models can predict all the target tokens in parallel, significantly accelerating the inference process. However, this parallel decoding paradigm also leads to performance degradation due to independent predictions lacking target side dependency (Qian et al., 2021; Xiao et al., 2022; Huang et al., 2023).

To balance the generation quality and inference efficiency, researchers have proposed iterative NAR models that utilize multiple decoding steps to generate the final results and retain the non-autoregressive decoding paradigm in each step (Lee et al., 2018; Ghazvininejad et al., 2019; Chan et al., 2020). Results generated from the previous decoding steps can provide partial target side information, and then be refined better in the subsequent steps. Through iterative refinements, the performance of these models can achieve significant improvements, even surpassing their AR counterparts (Huang et al., 2022c; Xiao et al., 2023). However, utilizing multiple decoding steps also increases the decoding time overhead, leading to less efficient inference processes (Kasai et al., 2020b; Helcl et al., 2022), and directly reducing the number of decoding steps for relatively faster decoding always brings inferior performance. Besides, these models have also revealed some flaws in the corresponding research, including the gaps between training and inference (Ghazvininejad et al., 2020; Huang et al., 2022c) and the anisotropic problem (Guo et al., 2023a), which hinders the further developments of iterative NAR models and results in less competitive performance. Therefore, ***how to build more efficient and competitive iterative NAR models*** deserves further exploration.

In this paper, we closely examine various aspects related to the abilities of iterative NAR models, and conduct systematic studies and analytical experiments to address the above-mentioned question:

- We conduct in-depth explorations of previous iterative NAR models (§3). Specifically, we

[1] github.com/LitterBrother-Xiao/rethink-iterative-nar

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 29851–29865

verify and quantitatively analyze the potential problems existing in current refinement processes through two metrics (§3.1). Besides, we conduct analytical experiments based on various iterative NAR models and discover that different enhanced methods play different roles in building efficient and competitive models (§3.2). Then, we attempt to realize our purpose by combining previous superior methods, but notice performance declines with previous efficient strategies (§3.3).

- We trial better strategies for iterative NAR models to become efficient while maintaining competitive performances (§4). We first analyze the limitations of current refinement strategies (§4.1) and then propose a simple yet effective inference algorithm for iterative NAR models (§4.2). Combining it with previous competitive strategies can achieve superior performance with fewer decoding steps.

Experiments on 5 widely used datasets demonstrate the effectiveness of our final models. We yield significant performance improvements (around 0.8 BLEU score on average) over the previous best iterative NAR models and realize completely surpassing AR Transformer (over 1 BLEU score on average). Besides, our models only need 4 decoding steps to set new SOTA performance on all WMT datasets compared with those achieved by previous models with 10 decoding steps, which leads to more efficient inference processes.

2 Preliminaries & Motivation

Non-autoregressive Language Model Up to now, most generative models are autoregressive (AR) models which generate the target sequence one by one from a left-to-right order during inference. They adopt AR factorization during training to maximize the following likelihood: $\mathcal{L}_{\text{AR}} = \sum_{t=1}^{T} \log P(y_t|y_{<t}, X; \theta)$, where $y_{<t}$ denotes the previous generated target tokens, T denotes the target length, X is the source sentence, and θ denotes the model parameters. Unlike these AR models, non-autoregressive (NAR) language models generate the target sequence in parallel during inference, they adopt conditional independent factorization during training to maximize the following likelihood: $\mathcal{L}_{\text{NAR}} = \sum_{t=1}^{T} \log P(y_t|X; \theta)$.

CMLM Conditional Masked Language Model (CMLM) is a typical and widely-used iterative NAR model (Ghazvininejad et al., 2019), which adopts a Transformer-based encoder-decoder architecture with some specific modifications in the decoder blocks to support NAR generation manner. During each decoding step, CMLM predicts the masked tokens in parallel conditioned on the source sequence and the unmasked part in the target sequence as shown in Figure 1. Then, CMLM will refine the generated output by re-masking and re-predicting several specific tokens according to the Mask-Predict algorithm (Ghazvininejad et al., 2019). During training, CMLM adopts masked language modeling tasks to learn this paradigm. As shown in Figure 1, given a training pair (X, Y), CMLM selects partial tokens in Y to be masked, denoted as Y_{mask}, while the unmasked tokens as Y_{obs}, then CMLM aims to maximize: $\mathcal{L}_{\text{CMLM}} = \sum_{y_t \in Y_{mask}} \log P(y_t|Y_{obs}, X; \theta)$. Besides, CMLM adopts a special token (e.g., `[LENGTH]`) in its encoder (we omit this in Figure 1) to predict the target length conditional on the source representation following the previous works (Ghazvininejad et al., 2019; Huang et al., 2022c; Xiao et al., 2023).

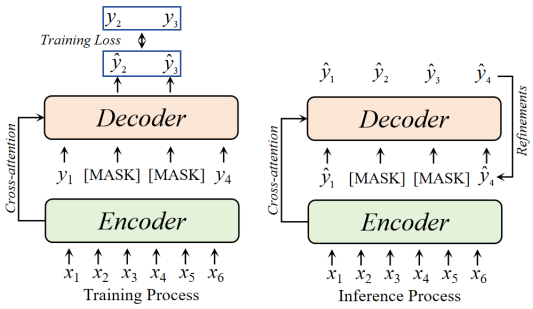

Figure 1: Presentation of the training and inference process of the conditional masked language model.

Follow-up Methods of CMLM Based on CMLM, researchers have proposed many follow-up enhanced methods from different perspectives to improve the training and inference process, e.g., adopting better masking methods (Guo et al., 2020; Xiao et al., 2023) or enhanced modeling mechanism (Kasai et al., 2020a; Cheng and Zhang, 2022; Chen et al., 2024) during training, utilizing extra modules to help training or inference (Hao et al., 2021; Liang et al., 2022; Geng et al., 2021), introducing better inference mechanisms (Ghazvininejad et al., 2020; Huang et al., 2022c), and etc. More details about these variants are included in the Appendix A due to the length limitation. Although we have learned from their own papers that there has

been a corresponding improvement in the performance of these methods compared to CMLM, we should compare them under more consistent hardware and settings to analyze the effects of different enhanced methods from different perspectives. In this paper, we conduct comprehensive analysis and analytical experiments of CMLM and these follow-up methods, targeting building more efficient and competitive iterative NAR models.

3 In-depth Explorations of Previous Iterative NAR Models

In this section, we conduct in-depth explorations of current iterative NAR models. Specifically, we introduce several sub-problems and make detailed analyses. We aim to find the key factors for building efficient and competitive iterative NAR models.

Problems and Explorations. Firstly, previous works always adopt the final generated output after pre-defined decoding steps to evaluate iterative NAR models, but overlook the fine-grained analysis of intermediate states throughout the refinement process. Consequently, some potential problems (e.g., useless and negative decoding steps) during the refinement process can not be reflected based on the current evaluation process. Therefore, we introduce two metrics (DRR and ROR) to quantitatively analyze the potential problems mentioned above, we aim to answer: ***how to evaluate the intermediate states of the whole refinement process of different iterative NAR models*** (§3.1). Based on our proposed two metrics, we compare different iterative NAR models under a consistent re-implementation. We aim to find ***what are the key components for iterative NAR models to perform better*** (§3.2). Finally, we conduct extended experiments to answer ***can combining superior methods brings benefits*** (§3.3), and make a summary (§3.4).

Experimental Settings. We adopt the vanilla CMLM and several typical variants which contain different improving strategies from different respects as mentioned in Section 2 for exploration. We summarize them as different categories: adopting enhanced training skills (JM-NAT, AMOM, Multitask-NAT), using adaptive inference algorithms (Disco, Rewrite-NAT), and introducing self-correction mechanisms (SMART, CORR, CMLMC). To make more consistent comparisons, we re-implement all these models based on the same hardware and training hyper-parameters. For the evaluation dataset, we select the IWSLT'14 DE→EN dataset containing about $170k$ training sentence pairs, $7k$ valid pairs, and $7k$ test pairs. We train each model on the training set and then evaluate them on the test set. Following the previous work (Kasai et al., 2020a), we apply sequence-level knowledge distillation (Kim and Rush, 2016) for all backbone models. All experiments use the Fairseq library (Ott et al., 2019) with GTX 3090 GPU cards. We adopt the same training hyper-parameters following CMLM realization in Fairseq. During inference, we average the 5 best checkpoints chosen by validation BLEU as our final model. Finally, we evaluate the generation quality with BLEU score (Papineni et al., 2002). Besides, to eliminate the effects of randomness, we follow the previous works to use statistical significance tests (Koehn, 2004) to detect if the difference in BLEU score between the traditional CMLM and other enhanced iterative NAR models is significant.

3.1 *How to Evaluate the Intermediate States of the whole Refinement Process of Different Iterative NAR Models?*

The current common evaluation process of iterative NAR models, where we compare the generated output of the last decoding step with ground truth to compute a score (e.g., BLEU), can not directly reflect the potential problems as mentioned above. Therefore, we introduce Decline Risks of Refinements (DRR) and Ratio of Over-Refinements (ROR) to respectively measure the ratio of these potential problems, and evaluate the stability and reliability of the refinement process.

Decline Risks of Refinements. Decline Risks of Refinements (DRR) evaluates the stability of the refinement process of iterative NAR models. It measures the performance decline rate after one specific decoding step, i.e., the extent of the negative decoding step. Specifically, given a test set with N examples, a fixed decoding step T, we compute the ratio of each example during the whole refinement process where the performance declines compared with the previous iteration, formatted as:

$$\text{DRR} = \frac{1}{T-1}\sum_{t=1}^{T-1}\frac{|\textbf{Score}_i^t > \textbf{Score}_i^{t+1}|}{N}, \quad (1)$$

where Score_i^t denotes the performance of sample i in the tth step.

Ratio of Over-Refinements. Ratio of Over-Refinements (ROR) evaluates the reliability of the final generated output in iteration T. It measures the failure rate of the output from the last decoding step to be the best, i.e., the extent of the useless decoding steps. Specifically, given a test set with N examples, a fixed decoding step T, we compute the ratio of each example whose best performance is achieved in the intermediate steps of the refinement process, formatted as:

$$\text{ROR} = \frac{1}{T-1}\sum_{t=1}^{T-1}\frac{|\textbf{Score}_i^t > \textbf{Score}_i^T|}{N}, \quad (2)$$

where Score_i^t denotes the performance of sample i in the tth step, Score_i^T denotes the performance of sample i in the final iteration T.

3.2 *What are the Key Components for Iterative NAR Models to Perform Better?*

Exploration Process. We look for the key components for two aspects, i.e., ***competitive*** and ***efficient***. The former can be directly reflected in the final performances, and the latter is reflected in the efficiency of achieving relatively promising performances. We re-implement and evaluate the related enhanced CMLM methods. For these with adaptive inference algorithms (i.e., Disco and RewriteNAT), we set T as the number of adaptive decoding steps of each sentence pair in Equation 1 and Equation 2 during inference, and 10 for other methods following the previous works. Besides, for the models that support two inference algorithms (e.g., CMLMC can omit the self-correction process and transform to the Mask-Predict algorithm), we both report the results with the Mask-Predict algorithm and the corresponding enhanced inference strategy.

Main Findings. The results are presented in Table 1, we find that: (1) ***DRR and ROR are relatively lower while decoding with adaptive inference algorithms.*** These models aim to find more suitable methods to decide how many and which tokens to mask, and when to stop refinements during inference. They can achieve comparable performance with fewer decoding steps, indicating that adaptive inference algorithms bring benefits to building more efficient iterative NAR models. (2) ***Enhanced training skills bring benefits on performance, but there is no evident improvement on DRR and ROR.*** These models trained with enhanced training skills can improve performance compared with the vanilla CMLM, but DRR and ROR are still high, indicating that enhanced training skills are useful for building more competitive iterative NAR models, the performance improvements of these models come from the better ability to model token dependency during training rather than benefitting the refinement process. (3) ***Introducing the self-correction mechanism can improve performance, but DRR gets higher.*** These models with the self-correction mechanism achieve performance improvement. However, DRR increases, indicating that this mechanism may bring more unstable factors during the refinement process.

Methods	Iteration	BLEU	DRR (%)	ROR (%)
Enhanced Training Skills				
CMLM	10	33.55	13.4	19.1
JM-NAT	10	32.60	14.4	17.5
Multitask-NAT	10	33.60	16.5	18.4
Disco	10	33.22	14.6	13.1
RewriteNAT †	10	33.88	12.1	14.4
CORR †	10	33.65	13.3	14.1
CMLMC †	10	34.02	13.1	13.8
AMOM †	10	34.68	16.3	17.9
Adaptive Inference Algorithms				
Disco	Adv.	33.32	11.8	6.9
RewriteNAT †	Adv.	33.91	7.9	1.1
Self-correction Mechanism				
SMART	10	33.17	14.5	16.6
CORR †	10	33.76	15.0	15.3
CMLMC †	10	34.40	15.2	14.9
Combining Superior Methods				
AMOMC †	10	35.08	16.8	16.7
w/ Locator †	Adv.	34.68	5.9	6.0

Table 1: DRR and ROR of different models. Adv. denotes adaptive decoding steps, which are always less than 10. † denotes that the BLEU improvements over CMLM are statistically significant with $p < 0.05$.

3.3 *Can Combining Superior Methods Brings Benefits?*

Exploration Process. We can learn from the explorations in Section 3.2 that different enhanced methods are independently beneficial to making the models more efficient and competitive. Naturally, we wonder: can combining superior methods bring benefits? We further explore the following questions: (1) Since the adaptive inference algorithms can bring promising performance with fewer decoding steps, can they further improve the performance with more steps? (2) Since adopting enhanced training skills and the self-correction mechanism can

boost performance but not stabilize the refinement process, can we incorporate the adaptive inference algorithms into these models to make them more efficient? Specifically, for question 1, we force these models (Disco and RewriteNAT) to continue the refinement process until reaching the maximal T decoding step. For question 2, we first combine the previous superior methods of enhanced training skills and adaptive inference algorithms (AMOM and CMLMC, denoted as AMOMC), and then we further apply the Locator module proposed in RewriteNAT into AMOMC.

Main Findings. The results are shown in Table 1, we can find that: (1) Concerning the models with adaptive inference algorithms, the performance even declines once we adopt more decoding steps for them, e.g., the performance declines from 33.32 to 33.22 for Disco, from 33.91 to 33.88 for RewriteNAT. Besides, DRR and ROR get much higher with more decoding steps, indicating that models with adaptive inference algorithms do not need many decoding steps to achieve the best performance during inference. (2) Further utilizing the Locator module for AMOMC can make the refinement process more efficient since it can achieve comparable performance with fewer decoding steps and get lower DRR and ROR, but it also leads to performance declines compared with the original AMOMC.

3.4 Summary

Now, we summarize our above explorations. We first propose two simple metrics to analyze the potential problems existing in current refinement methods. We encourage the researchers to pay more attention to the intermediate stages of the refinement process. Next, we conduct comparative experiments to look for the key components for building more efficient and competitive iterative NAR models, and then further combine superior methods to realize our purpose. However, we find that the current efficient strategy leads to performance declines. This motivates us to explore better strategies for building efficient iterative NAR models while maintaining competitive performance.

4 Trials for Better Efficient Strategies

In this section, we explore better strategies for building more efficient iterative NAR models while keeping them maintain competitive performance. We conduct a detailed analysis of original refinement methods and then propose a simple yet effective strategy to realize our purpose.

Problems and Explorations. Firstly, the Mask-Predict algorithm exhibits higher DRR and ROR than adaptive inference algorithms as shown in Table 1. Therefore, we aim to analyze: ***what makes the Mask-Predict algorithm fail to do efficient refinements*** (§4.1). Besides, noticing that although current adaptive inference algorithms are advantageous for reducing the decoding steps, they also lead to performance declines. Therefore, we analyze the corresponding reasons and further investigate: ***are there better efficient strategies for iterative NAR models*** (§4.2). Finally, we summarize the aforementioned questions and point out future directions for iterative NAR models (§4.3).

Experimental Settings. During the analysis of the failure of the Mask-Predict algorithm, we adopt the CMLM checkpoint achieved from the above exploration process. To explore more effective inference algorithms, we adopt more datasets, i.e., WMT'16 English↔Roman (En↔Ro) and WMT'14 English↔German (En↔De) language pairs which are widely used in previous NAR works, to evaluate our proposed methods. The training data sizes are respectively about 0.6M and 4.5M, and the test data are from the corresponding newest data containing around 2,000 and 3,000 samples. Besides, the training and evaluation settings are the same as those mentioned in Section 3.

4.1 *What Makes the Mask-Predict Algorithm Fail to Do Efficient Refinements?*

We attribute the success of the adaptive inference algorithm to the reasonable strategy to determine *"which token should be masked in the next decoding step?"* Comparatively, the Mask-Predict algorithm relies on predicted confidence to select masked tokens in the subsequent decoding step. However, we have identified two shortcomings with this confidence-based refinement process:

1) ***The independent confidence updating strategy for each token is sub-optimal.*** In the Mask-Predict algorithm, the prediction confidence is updated only for masked tokens during each decoding step, i.e., those for unmasked tokens remain unchanged after the last decoding step when they were predicted. This denotes that the prediction confidences of masked and unmasked tokens are derived from different decoding steps and under

different masking conditions. Consequently, this inconsistency poses challenges in determining which tokens to be masked in the subsequent decoding step. This shortcoming is also supported by the comparison presented in Table 1. The models that update the confidence scores of all tokens in the same decoding step can alleviate this problem, e.g., Disco, RewriteNAT, and CMLMC all achieve lower DRR and ROR even without adopting adaptive inference algorithms during inference.

2) ***The prediction confidence of CMLM is not strongly related to the generation quality.*** As discussed in Section 2, CMLM selects the prediction probability as the confidence to choose newly masked tokens. This approach assumes that tokens with higher prediction probabilities are more reliable. However, previous works have highlighted several issues. Ding et al. observe that some specific tokens, such as high-frequency words and conjunctions, consistently exhibit high confidence, leading to repetitive output and neglect of low-frequency but important words. Additionally, Liang et al. notes that the function words dominate the high probability region of the output distribution, making it challenging to generate informative tokens using the Mask-Predict algorithm with CMLM. We also perform a simple experiment to exhibit the irrelevance between the prediction confidence and final generation output.

Set	Win (%)	Lose (%)
Valid	54.61	45.39
Test	54.10	45.90

Table 2: **Win** denotes the model predicts the ground truth token as the final results, **Lose** denotes the vice.

Exploration Process. Since the traditional Mask-Predict algorithm always selects tokens with the highest prediction probability as output during each decoding step, we verify whether the probability of ground truth tokens ranks first. Specifically, we first randomly mask several tokens in the target sequence and send them into CMLM to obtain the prediction probability, then we find the probability place of ground truth tokens, e.g., given the test sentence *"Thank you."* We first replace the token *"you"* with the `[MASK]` token, then we send the sequence *"Thank* `[MASK]` *."* into CMLM, and verify whether the prediction probability of token *"you"* ranks first. If not, the highest prediction confidence does not equal the correct token.

Main Findings. We conduct experiments on the validation and test set of IWSLT'14 DE→ dataset and show the results in Table 2. We find that only around 54 percent of tokens meet our expectations, i.e., these ground truth tokens have the highest prediction probabilities. This shows that the model's own prediction probabilities are not strongly related to the correct tokens. We attribute this failure to the conditional independent factorization during training, which causes CMLM to fail to capture the target-side dependency well (Gu and Kong, 2021).

4.2 *Are There Better Efficient Strategies for Iterative NAR Models?*

The explorations in Section 4.1 explain why adaptive inference algorithms are more effective than the traditional Mask-Predict algorithm. However, noticing that adopting the Locater leads to performance declines as shown in Table 1 (34.68 v.s. 35.08). We further analyze the corresponding reason. Since the Locator module assigns zero-one discrete scores for predicted tokens, i.e., these tokens scored as zero will be masked again, and one will not be masked in the next decoding step. We point out that this scoring mechanism is too absolute, i.e., there is no difference for unreliable tokens that are scored as zero, and once all the tokens are scored as one, there are no subsequent actions to further improve the generation quality. To explore the potential of a more effective scoring module for iterative NAR models, we intended to replace the zero-one discrete score with a zero-one continuous distribution, in which we can design the refinement process more flexibly and constantly.

Exploration Process. We aim to find a simple yet effective mechanism to score each token within a sentence, and then we can depend on these scores to determine which tokens should be masked in the subsequent decoding step. Motivated by the previous practice that a pre-trained AR model can successfully serve as an effective scorer on the sentence-lever to evaluate the fluency of sentences, we can extend it as a token-level scorer, named ARSCORER in the remaining space of this paper. Specifically, we utilize the generated tokens from each decoding step as inputs for a pre-trained AR model. The AR model conducts its prediction on this input sequence in an autoregressive manner. Subsequently, we obtain the corresponding prediction distribution and use the probability associated with the input token index as the final score. The

	Model	Iter.	WMT'14 EN→DE	WMT'14 DE→EN	WMT'16 EN→RO	WMT'16 RO→EN	Speedup
AR	Transformer (Vaswani et al., 2017)	N	27.30	31.29	-	-	-
	Transformer*	N	28.41	32.28	34.23	34.28	1.0x
	Transformer (SCORER 12-1)*	N	28.91	32.65	33.75	34.34	2.5x
Fully NAR	GLAT (Qian et al., 2021)	1	25.21	29.84	31.19	32.04	15.3x
	Fully NAT (Gu and Kong, 2021)	1	27.49	31.39	33.79	34.16	16.5x
	latent-GLAT (Bao et al., 2022)	1	26.64	29.93	-	-	11.3x
	DA-Transformer (Huang et al., 2022b)	1	27.49	31.37	-	-	13.9x
	RenewNAT (Guo et al., 2023b)	1	26.65	30.65	33.02	33.74	11.2x
	FA-DAT (Ma et al., 2023)	1	27.49	31.37	-	-	14.0x
	PCFG-NAT (Gui et al., 2024)	1	27.02	31.29	32.72	33.07	12.6x
Iterative NAR	Levenshtein (Gu et al., 2019)	Adv.	27.73	-	33.02	-	4.0x
	CMLM (Ghazvininejad et al., 2019)	10	27.03	30.53	33.08	33.31	1.7x
	DisCo (Kasai et al., 2020a)	Adv.	27.34	-	33.25	33.22	3.5x
	SMART (Ghazvininejad et al., 2020)	10	27.65	31.27	33.85	33.53	1.7x
	JM-NAT (Guo et al., 2020)	10	27.69	32.24	33.52	33.72	5.7x
	RewriteNAR (Geng et al., 2021)	Adv.	27.83	31.52	33.63	34.09	3.9x
	MvCR-NAT (Xie et al., 2021)	10	27.39	31.18	33.38	33.56	3.8x
	CORR (Huang et al., 2022c)	10	28.19	31.31	34.31	34.08	-
	CMLMC (Huang et al., 2022c)	10	28.37	31.41	34.57	34.13	-
	AMOM (Xiao et al., 2023)	10	27.57	31.67	34.62	34.82	1.7x
	EECR (Chen et al., 2024)	10	28.04	31.65	34.33	34.32	3.8x
	DCMCL (Liao et al., 2024)	10	28.14	31.47	33.76	33.64	-
*Ours**	AMOMC	4	28.39	32.83	34.73	35.11	4.1x
		10	28.81	33.22	34.99	35.20	1.9x
	AMOMC w/ ARSCORER	4	**29.08**	**33.22**	**35.06**	**35.78**	3.1x
		10	**29.18**	**33.41**	**35.30**	**36.02**	1.4x

Table 3: Results on 4 WMT machine translation tasks. * denotes the results of our implementations. † denotes that the BLEU improvements over AMOMC are statistically significant with $p < 0.05$. **Bold** denotes the best performance, underline denotes the previous best AR or NAR performance.

scores range from zero to one after undergoing the normalized softmax operation. Comparatively, adopting ARSCORER offers several advantages over the Mask-Predict algorithm, which have also been mentioned in the previous section: (1) The AR model can assess the validity of each token in the whole sentence and update the corresponding prediction probability of each token after each decoding step of NAR model. (2) Previous studies have shown that models trained with autoregressive factorization excel in capturing target side dependencies compared to NAR models (Huang et al., 2022a). Besides, these AR models do not suffer from the multi-modality problem. Thus, adopting extra ARSCORER to provide the prediction score is more robust and effective. We should recognize that adopting the AR model to achieve the prediction score leads to extra inference time. Therefore, we adopt the structure of deep encoder and shallow decoder as mentioned in Kasai et al. (2020b) for ARSCORER to reduce inference efficiency.

Main Findings. The results on various WMT datasets are shown in Table 3, we can find that: (1) Combining superior methods (AMOMC) achieves significant performance improvements, outperforming all baseline models around 0.8 BLEU score. (2) Further adopting ARSCORER can quickly achieve competitive performance, i.e., it can achieve new state-of-the-art performance with only 4 decoding steps while getting 3.1 times speedup compared to the vanilla AR Transformer. (3) Adopting ARSCORER increases the inference time compared to AMOMC, but it makes up for this deficiency by achieving superior performance with relatively fewer decoding steps, which still indicates that ARSCORER can bring benefits for building efficient iterative NAR models.

Further Analysis. We further conduct detailed analytical experiments of our proposed methods. Firstly, we compare the backbones models without and with ARSCORER based on our proposed two metrics, DRR and ROR, as mentioned in Section 3.1. Results on IWSLT'14 DE→EN and WMT'16 RO→EN datasets are presented in Table 4. We can find that: (1) The models with ARSCORER can achieve lower DRR and ROR compared with the corresponding baselines. (2) DRR and ROR are higher on the WMT'16 RO→EN

Methods	Iter.	BLEU	DRR (%)	ROR (%)
IWSLT'14 DE→EN				
CMLM	10	33.55	13.4	19.1
w/ ARSCORER	10	34.05	10.0	13.4
AMOMC	10	35.08	16.8	16.7
w/ ARSCORER	10	35.61	9.8	13.6
WMT'16 RO→EN				
CMLM	10	33.87	21.0	36.1
w/ ARSCORER	10	34.51	13.5	19.4
AMOMC	10	35.20	19.4	28.7
w/ ARSCORER	10	36.02	14.0	19.1

Table 4: Results of DRR and ROR with ARSCORER.

dataset across all models, indicating that this dataset is relatively difficult to learn. Besides, we compare the results using different AR models to serve as ARSCORER. As we adopt the model with the structure of deep encoder and shallow (denoted as DESD ARSCORER) to reduce inference efficiency in our main result, we also adopt the model with common layers (i.e., 6 encoders and 6 decoders, denoted as vanilla ARSCORER) here. Results are presented in Table 5. We can find that (1) Adopting vanilla ARSCORER leads to lower decoding speed due to the high cost of passing the AR model to achieve the output probabilities. (2) The DESD ARSCORER can reduce the decoding latency due to the shallow decoder structure but still brings extra expense compared to the vanilla NAR model without ARSCORER with the same decoding steps. (3) The DESD ARSCORER can effectively reduce the decoding steps while achieving significant performance improvements, which can make up for the flaw of the extra expense of the AR model. Besides, considering that BLEU is sensitive to the predicted length, we also adopt Comet (Rei et al., 2020) to evaluate our methods, results are also shown in Table 5, we can find a similar phenomenon, i.e., adopting vanilla and DESD ARSCORER can both achieve better Comet score.

4.3 Summary

In this section, we aim to explore the potential for better efficient strategies. We begin by examining the limitations of the Mask-Predict algorithm in facilitating consistent and efficient refinements. Through thorough analysis and corresponding experimentation, we attribute these limitations to the independent confidence updating strategies and the unrelated prediction confidence to generation output. Consequently, we endeavor to identify a superior strategy to address these issues. Fortunately,

Methods	Iter.	BLEU	Comet	Latency
WMT'14 EN→DE				
w/o ARSCORER	4	28.39	0.584	130.8
	10	28.81	0.585	282.1
w/ vanilla ARSCORER	4	28.82	0.587	222.0
	10	29.17	0.587	522.7
w/ DESD ARSCORER	4	29.08	0.587	181.6
	10	29.18	0.588	402.1
WMT'16 RO→EN				
w/o ARSCORER	4	35.11	0.765	87.9
	10	35.20	0.768	211.7
w/ vanilla ARSCORER	4	35.26	0.768	147.6
	10	35.65	0.775	341.7
w/ DESD ARSCORER	4	35.78	0.772	116.0
	10	36.02	0.772	253.9

Table 5: Results of different scorer models. Iter. denotes the decoding step. Latency denotes the total seconds.

by adopting the pre-trained AR models to serve as a scorer, iterative NAR models can conduct steady and effective refinements, thereby achieving superior performance with even fewer decoding steps, and getting closer to the efficient iterative NAR models. It is worth noting that there are other viable options for scoring, such as adopting a pre-trained language model or even current well-known large language models, we leave this as future work.

5 Conclusion and Future Outlook

In this paper, we conduct extensive experiments and detailed analysis based on various iterative NAR models to explore ***how to build more efficient and competitive iterative NAR models***. By combining competitive strategies and the newly proposed ARSCORER, our final models set the new state-of-the-art results on several widely used datasets even with fewer decoding steps, which leads to completely outperforming their AR counterparts.

In the future, we will extend our explorations to more scenarios since CMLM-based iterative NAR models have been successfully applied in speech and video-related fields (Higuchi et al., 2021). Besides, there is also a need to explore methods for conducting efficient denoising steps for diffusion models (Sohl-Dickstein et al., 2015) since they suffer greatly from low efficiency with numerous denoising steps (Tang et al., 2023; Gong et al., 2023). Lastly, recent advancements in LLMs (Touvron et al., 2023b) hold promise in serving as better scorers for iterative NAR models.

Limitations

Firstly, since CMLM-based iterative NAR models have been applied to various language generation tasks, we only conduct our explorations on machine translation task. Besides, although CMLM-based methods are one of the most widely-used and well-known iterative NAR models, there exist other categories of iterative NAR models, such as editing-based models (Stern et al., 2019; Gu et al., 2019), denoising based models (Lee et al., 2018; Savinov et al., 2021), we only consider CMLM-based methods in this paper. Besides, our proposed efficient strategy, ARSCORER, relies on a pre-trained AR model to serve as a scorer for each token, it brings some extra costs to achieve this AR model and the corresponding prediction confidence.

Acknowledge

We want to thank all the anonymous reviewers for their valuable comments. The first three authors (Yisheng Xiao, Pei Guo, Zechen Sun) contribute equally to this work. Juntao Li is the corresponding author. This work was supported by the National Science Foundation of China (NSFC No. 62206194), the Natural Science Foundation of Jiangsu Province, China (Grant No. BK20220488), the Young Elite Scientists Sponsorship Program by CAST (2023QNRC001), and the Priority Academic Program Development of Jiangsu Higher Education Institutions. We also acknowledge MetaStone Tech. Co. for providing us with the software, optimization on high-performance computing, and computational resources required by this work.

References

Yu Bao, Hao Zhou, Shujian Huang, Dongqi Wang, Lihua Qian, Xinyu Dai, Jiajun Chen, and Lei Li. 2022. *latent*-glat: Glancing at latent variables for parallel text generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 8398–8409.

William Chan, Chitwan Saharia, Geoffrey Hinton, Mohammad Norouzi, and Navdeep Jaitly. 2020. Imputer: Sequence modelling via imputation and dynamic programming. In *ICML*, pages 1403–1413. PMLR.

Xinran Chen, Sufeng Duan, and Gongshen Liu. 2024. Improving non-autoregressive machine translation with error exposure and consistency regularization. *arXiv preprint arXiv:2402.09725*.

Hao Cheng and Zhihua Zhang. 2022. Con-nat: Contrastive non-autoregressive neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6219–6231.

Liang Ding, Longyue Wang, Xuebo Liu, Derek F Wong, Dacheng Tao, and zhaopeng Tu. 2021. Rejuvenating low-frequency words: Making the most of parallel data in non-autoregressive translation. In *ACL-IJCNLP*, pages 3431–3441.

Xinwei Geng, Xiaocheng Feng, and Bing Qin. 2021. Learning to rewrite for non-autoregressive neural machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3297–3308.

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*, pages 6112–6121.

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

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. 2023. Diffuseq-v2: Bridging discrete and continuous text spaces for accelerated seq2seq diffusion models. *arXiv preprint arXiv:2310.05793*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

Jiatao Gu and Xiang Kong. 2021. Fully non-autoregressive neural machine translation: Tricks of the trade. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 120–133.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, volume 32, pages 11181–11191.

Shangtong Gui, Chenze Shao, Zhengrui Ma, Yunji Chen, Yang Feng, et al. 2024. Non-autoregressive machine translation with probabilistic context-free grammar. *Advances in Neural Information Processing Systems*, 36.

Junliang Guo, Linli Xu, and Enhong Chen. 2020. Jointly masked sequence-to-sequence model for non-autoregressive neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 376–385.

Pei Guo, Yisheng Xiao, Juntao Li, Yixin Ji, and Min Zhang. 2023a. Isotropy-enhanced conditional masked language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8278–8289.

Pei Guo, Yisheng Xiao, Juntao Li, and Min Zhang. 2023b. Renewnat: renewing potential translation for non-autoregressive transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12854–12862.

Yongchang Hao, Shilin He, Wenxiang Jiao, Zhaopeng Tu, Michael Lyu, and Xing Wang. 2021. Multi-task learning with shared encoder for non-autoregressive machine translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3989–3996.

Jindřich Helcl, Barry Haddow, and Alexandra Birch. 2022. Non-autoregressive machine translation: It's not as fast as it seems. *arXiv preprint arXiv:2205.01966*.

Yosuke Higuchi, Hirofumi Inaguma, Shinji Watanabe, Tetsuji Ogawa, and Tetsunori Kobayashi. 2021. Improved mask-ctc for non-autoregressive end-to-end asr. In *ICASSP 2021*, pages 8363–8367. IEEE.

Fei Huang, Pei Ke, and Minlie Huang. 2023. [tacl] directed acyclic transformer pre-training for high-quality non-autoregressive text generation. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. 2022a. On the learning of non-autoregressive transformers. In *International Conference on Machine Learning*, pages 9356–9376. PMLR.

Fei Huang, Hao Zhou, Yang Liu, Hang Li, and Minlie Huang. 2022b. Directed acyclic transformer for non-autoregressive machine translation. In *International Conference on Machine Learning*, pages 9410–9428. PMLR.

Xiao Shi Huang, Felipe Perez, and Maksims Volkovs. 2022c. Improving non-autoregressive translation models without distillation. In *International Conference on Learning Representations*.

Jungo Kasai, James Cross, Marjan Ghazvininejad, and Jiatao Gu. 2020a. Parallel machine translation with disentangled context transformer. *arXiv preprint arXiv:2001.05136*.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah Smith. 2020b. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. In *ICLR*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *EMNLP*, pages 1317–1327.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 388–395.

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.

Xiaobo Liang, Zecheng Tang, Juntao Li, and Min Zhang. 2023. Open-ended long text generation via masked language modeling. In *ACL*.

Xiaobo Liang, Lijun Wu, Juntao Li, and Min Zhang. 2022. Janus: Joint autoregressive and non-autoregressive training with auxiliary loss for sequence generation. In *EMNLP*, pages 1067–1073.

Yusheng Liao, Yanfeng Wang, and Yu Wang. 2024. Leveraging diverse modeling contexts with collaborating learning for neural machine translation. *arXiv preprint arXiv:2402.18428*.

Zhengrui Ma, Chenze Shao, Shangtong Gui, Min Zhang, and Yang Feng. 2023. Fuzzy alignments in directed acyclic graph for non-autoregressive machine translation. *arXiv preprint arXiv:2303.06662*.

OpenAI. 2023. Gpt-4 technical report.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Lihua Qian, Hao Zhou, Yu Bao, Mingxuan Wang, Lin Qiu, Weinan Zhang, Yong Yu, and Lei Li. 2021. Glancing transformer for non-autoregressive neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 1993–2003.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. Comet: A neural framework for mt evaluation. *arXiv preprint arXiv:2009.09025*.

Nikolay Savinov, Junyoung Chung, Mikolaj Binkowski, Erich Elsen, and Aaron van den Oord. 2021. Step-unrolled denoising autoencoders for text generation. *arXiv preprint arXiv:2112.06749*.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, pages 2256–2265. PMLR.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *ICML*, pages 5976–5985. PMLR.

Zecheng Tang, Pinzheng Wang, Keyan Zhou, Juntao Li, Ziqiang Cao, and Min Zhang. 2023. Can diffusion model achieve better performance in text generation? bridging the gap between training and inference! *arXiv preprint arXiv:2305.04465*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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.

Yisheng Xiao, Lijun Wu, Junliang Guo, Juntao Li, Min Zhang, Tao Qin, and Tie-yan Liu. 2022. A survey on non-autoregressive generation for neural machine translation and beyond. *arXiv preprint arXiv:2204.09269*.

Yisheng Xiao, Ruiyang Xu, Lijun Wu, Juntao Li, Tao Qin, Tie-Yan Liu, and Min Zhang. 2023. Amom: Adaptive masking over masking for conditional masked language model. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):13789–13797.

Pan Xie, Zexian Li, and Xiaohui Hu. 2021. Mvsr-nat: Multi-view subset regularization for non-autoregressive machine translation. *arXiv preprint arXiv:2108.08447*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

A Details for Follow-up Methods

We supplement the details for follow-up methods of CMLM we adopted for explorations as mentioned in Section 2.

JM-NAT Guo et al. introduce a jointly masked sequence-to-sequence model. Unlike the traditional CMLM which only masks the target sequence during training, JM-NAT also masks the source sequence to help train the encoder more rigorously. Besides, in order to alleviate the problem of translating duplicate words, they propose to train the decoder based on the consecutive masking of the decoder input with an ngram loss function rather than the original uniform masking.

Disco Kasai et al. propose an attention-masking based model, Disentangled Context (DisCo) transformer. During training, Disco is learned to predict each target token given an arbitrary subset of the other reference tokens, which is more efficient than just predicting masked tokens in the original CMLM. During inference, unlike the previous Mask-Predict algorithm which just updates masked tokens in each decoding step (i.e., predicting Y_{mask} based on Y_{obs}), Disco introduces an easy-first policy where each token will be predicted in each step dependent on relatively easier tokens (i.e., predicting each Y_i based on $Y_{<i}$, where $Y_{<i}$ denotes tokens whose prediction confidence is higher than Y_i in the previous iteration). Disco stops decoding when no new tokens are generated in one specific decoding step. This easy-first policy can largely improve the inference latency.

Multitask-NAT Hao et al. introduces Multitask-NAT which utilizes a shared encoder and separated decoders for both AR and NAR modeling during training. They assume that AR training can bring benefits for NAR training and aim to adopt multi-task learning to transfer the AR knowledge to NAR models through encoder sharing.

RewriteNAT Geng et al. propose RewriteNAT, a new framework that contains a Locator and Revisor module that locate the incorrect words within previously generated translations and then revise them, respectively. Specifically, the Locator module can transform the problem of determining which tokens to be masked in the next decoding step into a binary classification problem instead of depending on the self-predicted confidence, i.e., the Locator will predict a special symbol (`[MASK]` or `[KEEP]`) for each token. Once the token is predicted as `[MASK]`, it will be masked again, and vice versa. RewriteNAT can finish the generation process once the Locator module predicts all the target tokens as `[KEEP]`.

SMART Ghazvininejad et al. introduce Semi-Autoregressive Training (SMART) to help the training process better match the Mask-Predict algorithm with multiple decoding steps. Specifically, since the model can not see the ground truth tokens during inference, it only takes the model prediction in the previous decoding steps as partially observed

tokens to make predictions. This leads to inconsistency compared with training methods. Thus SMART first constructs a mixed training example and then encourages the model to recover from the model prediction errors during training,

CMLMC Huang et al. propose Conditional Masked Language Model with Correction (CMLMC) which incorporates a self-correction mechanism into traditional CMLM and several modifications on the decoder structure such as exposing the positional encodings and incorporating causal attention layers to differentiate adjacent tokens. CORR is the corresponding variant which only adopts the self-correction mechanism without the structure modifications in CMLMC. Specifically, except for adopting masking methods in target sequence during training, CMLMC also replaces the partially unmasked tokens with model predictions based on a fully masked target sequence. Then CMLMC learns to predict the masked tokens and correct the replaced tokens simultaneously during training. During inference, this self-correction mechanism helps the model to correct the unreliable tokens in the unmasked subset. During training, CMLMC first adopts the same masking operation in CMLM training as mentioned in Section 2 in the current paper to split the target sequence Y into Y_{obs} and Y_{mask}, and then learns to predict Y_{mask} based on Y_{obs} and the source sequence X, as $L_{CMLM} = -\sum_{y_t \in Y_{mask}} \log P(y_t|Y_{obs}, X; \theta)$. Besides, CMLMC adopts an extra correction task, specifically, CMLMC selects partial tokens in Y_{obs} and replaces them as unreliable tokens, denotes as Y_{rep}, and the remaining tokens in Y_{obs} is Y_{unrep}. These unreliable tokens are achieved from the model predictions based on a fully masked target sequence. Then, CMLMC learns to correct these unreliable tokens Y_{rep}, as $L_{CORR} = -\sum_{y_t \in Y_{rep}} \log P(y_t|Y_{unrep}, X; \theta)$. Overall, CMLMC simultaneously learns to predict the masked tokens and correct the unreliable tokens in a training step, as $L_{CMLMC} = L_{CMLM} + L_{CORR}$.

AMOM Xiao et al. propose an Adaptive Masking Over Masking (AMOM) strategy based on CMLM which contains two different adaptive masking mechanisms which work on the inputs of encoder and decoder respectively. Specifically, based on the ratio of the target sequence, AMOM also masks the specific number of tokens in the source sequence to make the encoder optimization easier. Besides, AMOM conducts an extra masking step where the masking ratio of the target sequence in this step is adaptive to the correction ratio of the model prediction. This two-step masking strategy can help the model capture the masking ratio changes in various decoding steps during inference. Specifically, based on the original masking operation in CMLM, AMOM also masks several tokens in source sequence X conditional on the masking condition of Y_{mask} (called adaptive X masking in AMOM paper), denoted as $\hat{X}$, but it does not need to predict them during training, thus AMOM first learns to predict Y_{mask} based on Y_{obs} and X_{obs}, as $L_{Ada-X} = -\sum_{y_t \in Y_{mask}} \log P(y_t|Y_{obs}, \hat{X}; \theta)$. Then, AMOM will adopt an extra masking operation to split the new Y (the new Y is achieved by combining Y_{obs} and the model predictions for Y_{mask} in the first training step) into new masked and unmasked parts (called adaptive Y masking in AMOM paper), denoted as Y'_{mask} and Y'_{obs}, then AMOM will predict Y'_{mask} based on Y'_{obs} and $\hat{X}'$, where $\hat{X}'$ is the newly masked X for the as $L_{Ada-Y} = -\sum_{y_t \in Y'_{mask}} \log P(y_t|Y'_{obs}, \hat{X}'; \theta)$. Overall, AMOM adopts two-step training schedules, as $L_{AMOM} = L_{Ada-X} + L_{Ada-Y}$.

AMOMC AMOMC combines the training schedules in CMLMC and AMOM. Specifically given X and Y, AMOMC first splits Y into Y_{mask} and Y_{obs} similar to CMLM, then it will mask several tokens in X as mentioned in AMOM, then the first task is to learn to predict Y_{mask} based on Y_{obs} and $\hat{X}$. Then AMOMC constructs the samples to learn the correction task as mentioned in CMLMC, i.e., AMOMC selects partial tokens in Y_{obs} and replaces them as unreliable tokens, denotes as Y_{rep}, and the remaining tokens in Y_{obs} is denoted as Y_{unrep}, then AMOMC learns to correct Y_{rep} based on Y_{unrep} and $\hat{X}$. Besides, after AMOMC predicts the tokens in Y_{mask} in the first training step, we obtain a new Y, denoted as Y', then AMOMC adopts the extra masking step as mentioned in AMOM to split Y' into Y'_{obs} and Y'_{mask}, then AMOMC learns to predict Y'_{mask} based on Y'_{obs} and $\hat{X}$', Overall, AMOMC adopts two steps to predict the corresponding masked tokens in Y, and one step to correct the unreliable tokens, as $L_{AMOM} = L_{Ada-X} + L_{Ada-Y} + L_{CORR}$.

Models	Parameters	IWSLT'14 DE→EN	WMT'14 EN↔DE	WMT'16 EN↔RO
CMLM	learning rate	5e-4	7e-4	5e-4
	warmup_step	4k	10k	10k
	dropout	0.3	0.2	0.3
	update_step	300k	300k	300k
	GPU	1xGTX 3090	4xGTX 3090	4xGTX 3090
AMOMC	learning rate	5e-4	7e-4	5e-4
	warmup_step	30k	40k	15k
	dropout	0.3	0.2	0.3
	update_step	175k	150k	120k
	GPU	1xGTX 3090	4xGTX 3090	4xGTX 3090

Table 6: Training hyper-parameters for CMLM and AMOMC.

B Training Hyper-parameters

During our experiments, we set training hyper-parameters for CMLM in the same way as CMLM realization in the Fariseq library, and for AMOMC, we follow those adopted in CMLMC (Huang et al., 2022c). Now, we present these training hyper-parameters in Table 6.

C More Explorations between Iterative NAR and AR Models

Researchers commonly assess the effectiveness of iterative NAR models by comparing their performance (generation quality and inference efficiency) with their AR counterparts. This section delves into further exploration of experimental settings for a comprehensive comparison.

Problems and Explorations. Firstly, the structure of deep encoder and shallow decoder (DESD) has been proven effective for AR models to increase inference speed and improve generation quality but does not work well for iterative NAR models (Kasai et al., 2020b). We assume that their experimental setting, which only adopts the decoder with one layer, is too strict for iterative NAR models, resulting in their relatively biased conclusion that iterative NAR models fail to benefit from DESD. Therefore, we experiment with more layer allocation schemes based on the structure of DESD to explore: ***do iterative NAR models really fail with deep encoder and shallow decoder?*** (§C.1) Besides, most previous works on iterative NAR models adopt the same decoding steps (4 and 10) for all the testing instances with different lengths (Ghazvininejad et al., 2019; Huang et al., 2022c). Comparatively, as AR models only predict the next token in each decoding step, they need decoding steps proportional to the length of the target sentence to achieve the final results. Thus, they adopt more decoding steps for longer sentences. Since the general knowledge for iterative NAR models is that longer sentences need more decoding steps, adopting the fixed decoding steps for all the target sentences naturally causes inconsistent comparisons with AR models. Consequently, we explore: ***can iterative NAR models achieve better performance beyond fixed decoding steps?*** (§C.2) Finally, we summarize the above two questions and conclude briefly (§C.3).

Experimental Settings. We adopt two popular iterative NAR models (CMLM and CORR) as mentioned in Section 2) and their AR counterpart (the vanilla Transformer) to conduct detailed analytical experiments. The training and inference settings are the same as mentioned in Section 3. We report the decoding time during inference, i.e., we select S_1 measured by L_1^{GPU} to compare the inference efficiency following the previous work (Kasai et al., 2020b; Helcl et al., 2022)), S_1 denotes the translation latency by running the model with one sentence at a time on a single GPU.

C.1 *Do Iterative NAR Models Really Fail with Deep Encoder and Shallow Decoder?*

Exploration Process. We adopt the structure of DESD on two backbone iterative NAR models and the vanilla AR Transformer to make comparisons. We conduct experiments with different layer allocation schemes, each marked as x-y. x and y denote the number of encoder and decoder layers, respectively. Specifically, we design the schemes based on two rules: (1) We keep the total layers of encoder and decoder as 12 following the common setting (6-6), then assign different layer allocations for them, e.g., 7-5, 8-4, 9-3, etc. (2) We keep the layers of encoder as 12 (12-1) following the setting in Kasai et al. (2020b), and adopt more decoder layers, e.g., 12-2 and 12-3.

Main Findings. Figure 2 presents the corresponding results, we can find that: (1) The number of decoder layers of iterative NAR models significantly affects the inference speed. Comparatively, the number of encoder layers has little effect. (2) Iterative NAR models can also perform well with the structure of DESD (8-4, 9-3, 12-2, 12-3), and accelerate the inference process (more than 1.5x Speedup). (3) Decoder layers are essential for iterative NAR models; at least two decoder layers are needed for iterative NAR models, and more decoder layers achieve better performance. Only one decoder layer harms performance seriously,

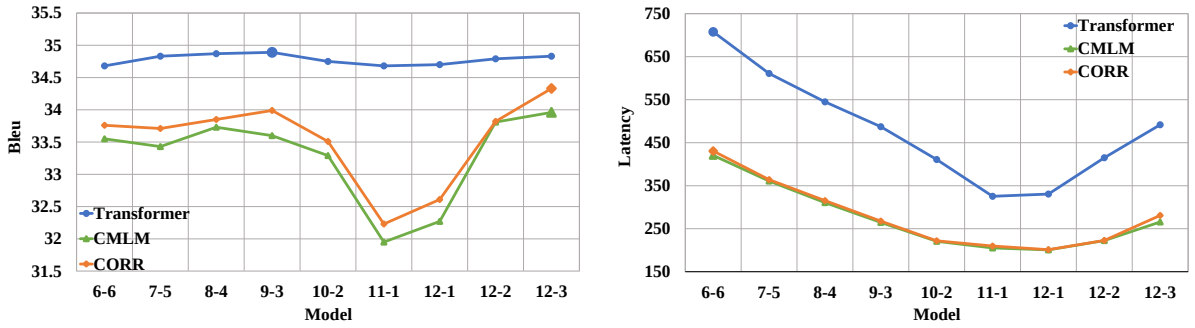

Figure 2: Results with different models and layer allocations.

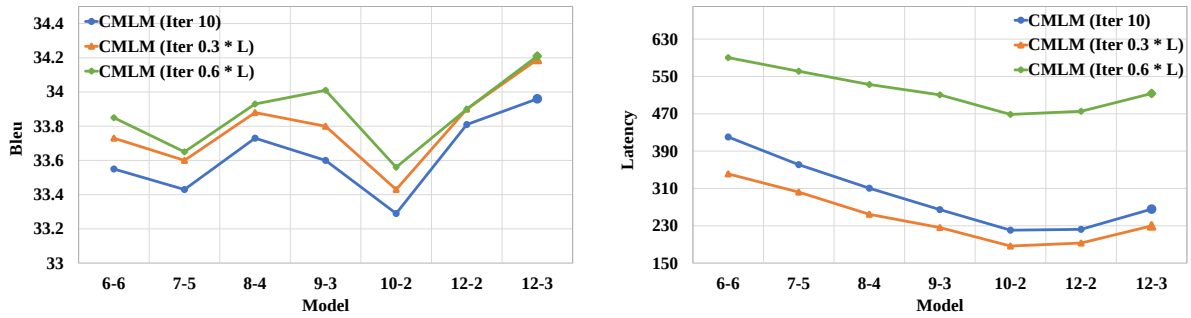

Figure 3: Results with different decoding steps (Iter) during inference. L denotes the length of the source sentence.

which is consistent with the findings in Kasai et al. (2020b). Generally, the model with 12-3 layer allocation achieves the best trade-off between generation quality and inference efficiency.

C.2 *Can Iterative NAR Models Achieve Better Performance Beyond Fixed Decoding Steps?*

Exploration Process We aim to design an adaptive algorithm to decide the decoding steps for each testing instance. To keep consistent with AR models, we can design a mapping function according to the sequence length. We select the relatively simple and understandable linear mapping function, i.e., given the sequence length L, the decoding step is $\alpha * L$, α is 1 in AR models. We can further set it smaller (e.g., 0.3 and 0.6) to realize relatively fast decoding for iterative NAR models.

Main Findings. Figure 3 presents the corresponding results, we can find that: (1) While adopting 0.3 * L decoding steps can improve the performance on BLEU score slightly (about 0.2), the inference latency of different models consistently reduces compared with adopting fixed 10 decoding steps. (2) Adopting 0.6 * L decoding steps can further improve the performance of all models but hurt the speed, indicating that blindly increasing the decoding steps is not advisable. (3) Adopting feasible adaptive decoding steps for different test instances can benefit both generation quality and inference efficiency. We recognize that the simple linear mapping function with $\alpha = 0.3$ may not be optimal, but it has been verified that improvements can be achieved beyond fixed decoding steps. Furthermore, we also compare the improvements in generation quality based on different source lengths. Specifically, we divide the source sentences into five intervals by the corresponding length and then compare the performance improvements through increasing iterations. We plot the performance improvements from fixed 10 to $0.3 * L$ decoding steps in Figure 4, demonstrating that long sequences need more decoding steps.

C.3 Summary

In this section, we explore more settings during comparison between iterative NAR and AR models. Different from the findings in Kasai et al. (2020b), our experiments demonstrate that the structure of DESD can also work well for iterative NAR models by simply adopting one more decoder layer and more performance improvement can be achieved with more layer allocation schemes. Besides, adopting the fixed decoding steps leads to an inconsistent setting since AR models adopt decoding steps

adaptive to the sequence length. As a result, we design a simple adaptive mapping function to decide the decoding steps for different testing instances, and achieve both performance and inference speed improvements. In general, we point out that commonly used and recognized settings still need exploration for iterative NAR models, and we should consider more consistent settings when comparing iterative NAR and AR models.

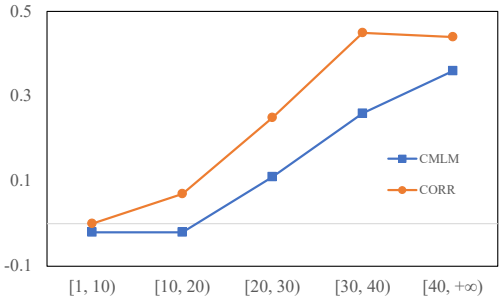

Figure 4: The performance improvements with more iterations based on different source lengths.