Candidate Soups: Fusing Candidate Results Improves Translation Quality for Non-Autoregressive Translation

Huanran Zheng[1], **Wei Zhu**[1], **Pengfei Wang**[1], **Xiaoling Wang**[1]
[1]East China Normal University, Shanghai, China
`{hrzheng,wzhu,pfwang}@stu.ecnu.edu.cn`
`{xlwang}@cs.ecnu.edu.cn`

Abstract

Non-autoregressive translation (NAT) model achieves a much faster inference speed than the autoregressive translation (AT) model because it can simultaneously predict all tokens during inference. However, its translation quality suffers from degradation compared to AT. And existing NAT methods only focus on improving the NAT model's performance but do not fully utilize it. In this paper, we propose a simple but effective method called "Candidate Soups," which can obtain high-quality translations while maintaining the inference speed of NAT models. Unlike previous approaches that pick the individual result and discard the remainders, Candidate Soups (CDS) can fully use the valuable information in the different candidate translations through model uncertainty. Extensive experiments on two benchmarks (WMT'14 EN–DE and WMT'16 EN–RO) demonstrate the effectiveness and generality of our proposed method, which can significantly improve the translation quality of various base models. More notably, our best variant outperforms the AT model on three translation tasks with **7.6**× speedup.[1]

1 Introduction

Autoregressive translation (AT) models based on Transformer (Vaswani et al., 2017; So et al., 2019; Sun et al., 2022; Zhu et al., 2021a), where each generation step depends on the previously generated tokens, achieve state-of-the-art (SOTA) performance on most datasets for machine translation tasks. AT model can better model the process of translation generation but leads to a massive limitation of its inference speed.

Therefore, the non-autoregressive translation (NAT) (Gu et al., 2018) model is proposed, which is 15.6 times faster than AT model. NAT assumes that the generated tokens are conditionally independent

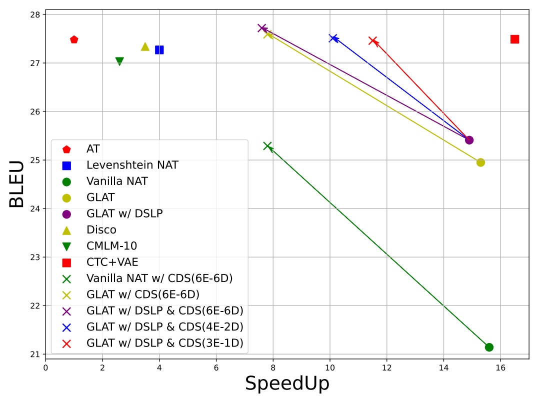


Figure 1: Efficiency (Speedup) and Translation quality (BLEU) of NAT models in the WMT'14 EN-DE translation dataset. A cross "×" represents our Candidate Soups (CDS) variants. Its base model is shown in the shape "●", and its correspondence is represented by an arrow. CDS (mE-nD) refers to the AT model for rescoring that has m encoder layers and n decoder layers.

given the source sentence, so the translation can be generated in parallel, significantly improving its inference speed compared to AT. However, due to the strong independence assumption, the ability of the NAT model modeling sequence generation is weakened. So NAT model usually has multimodality problem (Gu et al., 2018) in the inference process, resulting in its performance worse than AT models.

Several methods have been proposed to alleviate the multimodality problem and improve the performance of the NAT model, such as the iteration-based NAT model (Ghazvininejad et al., 2019; Gu et al., 2019; Kasai et al., 2020) and the semi-autoregressive translation model (Wang et al., 2018; Ran et al., 2020).

Most of the previous methods are modified from the model's perspective, either modifying the structure of the model (Shu et al., 2020; Huang et al., 2021; Zhu et al., 2021b) or modifying the training method of the model (Du et al., 2021; Qian et al., 2021). Different from the previous methods, in this paper, we propose a simple but effective method:

[1]Our code is released at `https://github.com/boom-R123/Candidate_Soups`.

Src	Die Beschaffung des erforderlichen Personalausweises kostet oft über hundert Dollar.
Candidate 1	It often costs over a hundred dollars to obtain the require identity card .
Candidate 2	It often cost over a hundred dollars to obtain the required identity card .
NPD	It often cost over a hundred dollars to obtain the required identity card .
CDS	It often costs over a hundred dollars to obtain the required identity card .

Figure 2: Comparison between NPD and Candidate Soups (CDS). Red fonts represent mistranslated tokens. Compared with NPD, Candidate Soups does not preserve the erroneous parts in the candidate results.

Candidate Soups, which can significantly improve the translation quality without any modification to the model. Moreover, Candidate Soups is a general approach that can be used by any NAT model that can generate multiple candidate results, such as Vanilla NAT (Gu et al., 2018), GLAT (Qian et al., 2021), etc.

The conventional recipe for maximizing translation quality through candidate results is noisy parallel decoding (NPD) (Gu et al., 2018), which regards each candidate translation as an independent individual and ultimately only selects one of them as the final result and discards others. Therefore NPD can not utilize the valuable information in all the candidate translations. For example, there are a total of two candidate translations. The first candidate translation has the wrongly translated word in the second half, and the second candidate translation has the wrongly translated word in the first half. Using the NPD algorithm, in this case, can not get the correct translations (Figure 2).

However, Candidate Soups will effectively use the valuable information of all the candidate translations to fuse the different candidate results and obtain a higher-quality translation (Figure 2). Specifically, Candidate Soups first finds the common subsequence among all candidate results. Based on the uncertainty of the model, we consider the common subsequence to be the most confident part of the model's predictions, so we make it part of the final translation and use it to align the candidate results. For the remaining parts, we select the part with the highest average log-probability among all candidate results to add to the final translation. Candidate Soups can be regarded as an implicit model ensemble method, which generates multiple different results by introducing uncertainty in the inference process, and further enhances the translation quality by making full use of the information of multiple results.

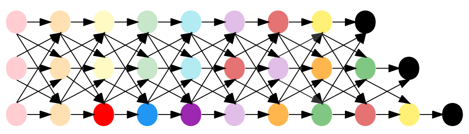
We conduct extensive experiments in two datasets commonly used in machine translation, WMT'14 EN–DE and WMT'16 EN–RO. The results demonstrate that our proposed method can significantly improve the base models' translation quality on different tasks while maintaining the fast inference speed of the NAT model. Remarkably, our best variant achieves better performance than the AT teacher model on three translation tasks with **7.6**$\times$ speedup. Figure 1 demonstrates the quality-speed trade-off compared with AT and recent NAT models. And relevant background knowledge is introduced in Appendix A.

2 Related Work

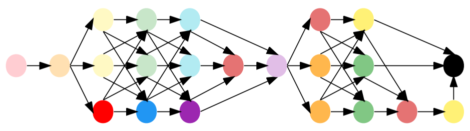
Since the NAT model was proposed, it has attracted the attention of many researchers due to its superior inference speed. However, its translation quality suffers from degradation compared to AT model. Therefore various methods have been proposed to bridge the performance gap between NAT and AT model.

Some researchers constrain the distribution of NAT model outputs by introducing various latent variables (Gu et al., 2018; Shu et al., 2020; Ran et al., 2021). Through latent variables, the diversity of the NAT model output space can be significantly reduced so that the model can better handle the dependencies between output words and alleviate the multimodality problem. Such methods can usually maintain the efficient inference speed of the NAT model, but the improvement in translation quality is relatively small.

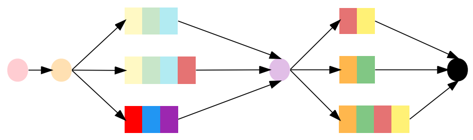
Some other researchers have proposed iterative decoding methods (Ghazvininejad et al., 2019; Gu et al., 2019; Kasai et al., 2020), which continuously optimize the model's output by introducing more information in the iterative process. For example, Ghazvininejad et al. (2019) mask the partial token of the previous output result and then use it as the input of the decoder for the next round of iteration. Although such models can achieve high performance, multiple iterations can also significantly affect the inference speed of the NAT model.



(a) Initial Lattice



(b) Simplified Lattice



(c) Final Lattice

Figure 3: Definition and simplification of translation search problem. Assume there are three candidate translations. Nodes with different colors represent different tokens. (a) Initial search space. (b) Simplified search space using the common subsequence. (c) Further simplified search space after node fusion.

Recently, Qian et al. (2021) borrowed ideas from curriculum learning and proposed a novel way to train NAT models, which let the model starts from learning the generation of sequence fragments and gradually moving to whole sequences. Huang et al. (2021) proposes to predict the result at each decoder layer and input it to the next layer together with the output of the current decoder layer to modify the result of the subsequent prediction.

The above methods are all improvements from the model perspective, and their purpose is to allow the model to generate higher quality translations. Unlike previous work, Candidate Soups wants to explore how to make the most of an existing model, and it can be applied to all NAT models that can generate multiple candidate results.

3 Candidate Soups

This section describes the details of the proposed method in the paper. We first show the problem definition and the general idea of Candidate Soups in Section 3.1, then introduce implementation details of the Candidate Soups in Section 3.2, followed by the example in Section 3.3.

3.1 Problem Definition

By introducing uncertainty into the NAT model, we can get a list of candidate results $\boldsymbol{R} = [R_0, \ldots, R_k]$, and each candidate result may have correctly and incorrectly translated parts that do not completely overlap. Thus, our goal is to find the optimal combination in $\boldsymbol{R}$, which has the highest average log-probability re-scored by an AT model. Because the word order in the original translations must be kept, we first use $\boldsymbol{R}$ to build a Lattice (Figure 3a). Each node represents a token, and each edge represents the change of average log-probability caused by adding the next token. So each path from the beginning node **[BOS]** to the end node **[EOS]** represents a possible translation. Therefore, our goal is to find the best path which has the highest average log-probability in this Lattice.

However, because the initial Lattice contains too many paths, we cannot calculate the values of all edges. Furthermore, due to the dislocation caused by the different lengths of the candidate results, most of the paths in the initial Lattice have word order errors, such as edges between the same tokens (Figure 3a). So we simplified and aligned the original Lattice. Specifically, after introducing the length uncertainty, we consider tokens that appear in all candidate results with the same word order as certain components (Gal and Ghahramani, 2016).

Algorithm 1: Candidate Soups

```
Input: A list of candidate results
  $\boldsymbol{R} = [R_0, \ldots, R_k]$ and the corresponding
  log-probability score sequence list
  $\boldsymbol{S} = [S_0, \ldots, S_k]$
Result = []
$\boldsymbol{R}$, $\boldsymbol{S}$ = Remove_duplicates($\boldsymbol{R}$, $\boldsymbol{S}$)
Initialize I = [$i_0$ = 0, ..., $i_k$=0]
Get candidate results length list L = [$l_0, \ldots, l_k$]
while $I < L$ do
    if all item in $\boldsymbol{R}$[I] is equal then
        Result.append($R_0[i_0]$)
        I = I + 1
    else
        $\boldsymbol{H}$ = $\boldsymbol{R}$[I]
        I$^*$ = I + 1
        // H is used to record tokens
           going from I to I*
        while $I^* < L$ do
            Add $\boldsymbol{R}$[I$^*$] to $\boldsymbol{H}$
            if Exist $\hat{I}$ meet all item in $\boldsymbol{H}$[$\hat{I}$] is
              equal then
                I$^*$ = $\hat{I}$
                break
            I$^*$ = I$^*$ + 1
        T = [$S_j[i_j$-1:$i_j^*$+1].mean() for j from 0 to
          k]
        t = T.argmax()
        Result.append($R_t[i_t$:$i_t^*]$)
        I = I$^*$
return Result
```

	Src	Die republikanischen Behörden beeilten sich , diese Praxis auf andere Staaten auszudehnen .
t=0	Candidate 1	The Republican authorities were quick extend to other States .
	Candidate 2	The Republican authorities were quick to extend this practice States .
	Candidate 3	The Republican and the authority extend this practice to other States .
t=1	Candidate 1	The Republican authorities were quick extend to other States .
	Candidate 2	The Republican authorities were quick to extend this practice States .
	Candidate 3	The Republican and the authority extend this practice to other States .
t=2	Candidate 1	The Republican authorities were quick extend to other States .
	Candidate 2	The Republican authorities were quick to extend this practice States .
	Candidate 3	The Republican and the authority extend this practice to other States .
t=3	Candidate 1	The Republican authorities were quick extend to other States .
	Candidate 2	The Republican authorities were quick to extend this practice States .
	Candidate 3	The Republican and the authority extend this practice to other States .
t=4	Candidate 1	The Republican authorities were quick extend to other States .
	Candidate 2	The Republican authorities were quick to extend this practice States .
	Candidate 3	The Republican and the authority extend this practice to other States .
t=5	Candidate 1	The Republican authorities were quick extend to other States .
	Candidate 2	The Republican authorities were quick to extend this practice States .
	Candidate 3	The Republican and the authority extend this practice to other States .
	Final Result	The Republican authorities were quick to extend this practice to other States .

Figure 4: An example from the WMT'14 DE-EN validation set illustrates how Candidate Soups generates high-quality final translations from candidate results. The highlighted tokens represent the tokens added to the final translation, and the red tokens represent the discarded tokens.

Thus, we find the common subsequence of all candidate results and directly use it as part of the final translation (Figure 3b). This way, the alignment of candidate translations can be achieved, and the original complex Lattice can be simplified into the connection of multiple simple Lattices.

For the remaining simple Lattice, the cost of calculating each edge value is still unbearable. Therefore, we fuse the nodes belonging to the same candidate result in each Lattice into a single node (Figure 3c) and ignore the influence of previous Lattice results on subsequent Lattice. In this way, we can quickly calculate each edge value by using the AT model to re-score each candidate translation. Then we only need to calculate the best path in each simple Lattice and obtain the final translation through a simple greedy algorithm.

3.2 Implementation

Algorithm 1 lists the process of Candidate Soups. We generate the final translation while looking for the common subsequence.

First, for the input candidate results set $\boldsymbol{R}$ and the corresponding log-probability score set $\boldsymbol{S}$, we will remove the adjacent repeated tokens and their corresponding scores for each sentence. Then we initialize a pointer set I that each pointer points to position 0 for each candidate translations and use these pointers to traverse simultaneously. If all the current pointers point to the same token, the token is added to the final translation, and all pointers are moved one step to the right. Otherwise, Candidate Soups will look for the next sequence of pointers I^* that satisfies the above conditions and move all pointers there. At the same time, the segment with the highest average log-probability score among all segments generated by the pointer traverse from I to I^* is added to the final translation.

Experimental results show that Candidate Soups can significantly improve final translation quality, requiring only 3 to 7 candidate translations. Moreover, the time required by Candidate Soups is almost negligible compared to the inference time of the NAT model.

3.3 Example

Figure 4 shows how Candidate Soups fuses the valuable information in each candidate translation to generate high-quality translations during the traversal process.

First, the NAT model predicts three candidate translations for the input sentence by introducing different lengths (t = 0). Afterward, through the traversal of the pointers, Candidate Soups found that the first two tokens ("The Republican") in the candidate results were the same, so they were added to the final translation (t = 1). When there is a disagreement between candidate results, Candidate Soups will find the next token ("extend") that all candidate translations predict in common and get three different segments ("authorities were quick," "authorities were quick to," and "and the authority"). Then Candidate Soups will select the candidate segment with the highest average log-probability scores ("authorities were quick to") and add it to the final translation (t = 2). Similarly, in the subsequent traversal process, if the tokens are predicted jointly by all candidate results, Candidate Soups will add them to the final translation (t = 3, t = 5). Otherwise, Candidate Soups will select the tokens segment with the highest log-probability score to join the final translation (t = 4). Ultimately, we get higher-quality translations by combining all valuable information in the candidate results.

From this example, we can find that only selecting an independent candidate result as the final translation is not effective enough for the NAT model. Because different lengths introduce uncertainty into the NAT model, there is diversity among candidate translations, but the previous methods do not take advantage of this. Proudly, through the certainty and confidence of the NAT model's output, Candidate Soups makes full use of the candidate results, takes the essence and removes the dross, and further improves the final translation quality without affecting the inference speed.

4 Experiments

In this section, we first introduce the settings of our experiments in Section 4.1, then report the main results in Section 4.2. Ablation experiments and analysis are presented in Section 4.3.

4.1 Experimental Setup

Dataset and Evaluation We evaluate our method on the two most recognized machine translation benchmarks: WMT'14 English–German (4.0M sentence pairs)[2] and WMT'16 English–Romanian (610K pairs)[3]. We use BLEU (Papineni et al., 2002) to evaluate the translation quality. And the beam size is set to 5 for AT model during inference. Moreover, for a fair comparison, we obtain the Huang et al. (2021) open-source corpus whose tokenization and vocabulary are the same as previous work: Zhou et al. (2020) for WMT'14 EN–DE which contains 39.8k subwords, and Lee et al. (2018) for WMT'16 EN–RO which contains 34.6k subwords. Both the implementation and evaluation of our method are performed using the open source fairseq[4] (Ott et al., 2019).

Knowledge Distillation Using AT model's output to train the NAT model can significantly improve the performance of the NAT model. Following previous work (Gu et al., 2018; Lee et al., 2018; Ghazvininejad et al., 2019), we also employ sequence-level knowledge distillation for all datasets. All the distillation data we use is open sourced by Huang et al. (2021).

Hyperparameters Our model architecture is Transformer-base (Vaswani et al., 2017): a 6-layer encoder and a 6-layer decoder, 8 attention heads per layer, 512 attention modules dimensions, 2048 feedforward modules hidden dimensions. We adopt the Adam optimizer (Kingma and Ba, 2015) with $\beta = (0.9, 0.98)$. To train the models, we use a batch size of 64K tokens, with a maximum 300K updates. For regularization, we use dropout (WMT'14 EN-DE: 0.1, WMT'16 EN-RO: 0.3), 0.01 weight decay and 0.1 label smoothing.

Base Models Our Candidate Soups is a general algorithm that can be applied to various NAT models. Therefore, to evaluate whether our proposed method can perform well on different NAT models, we selected the following four base models:

(1) Vanilla NAT (Gu et al., 2018), which predicts length instead of fertility sequence.

(2) CMLM (Ghazvininejad et al., 2019), whose training strategy follows a masked language model approach similar to BERT (Devlin et al., 2019). And it can perform iterative decoding during inference. We trained one CMLM model for each translation task and respectively iterated decoding once and iterated decoding five times as two baselines.

(3) GLAT (Qian et al., 2021), which trains NAT model step-by-step in a curriculum learning manner.

(4) GLAT & DSLP (Huang et al., 2021), whose decoder layers can get the prediction result of the previous layer.

The prediction patterns and performance of these base models are quite different, so through them,

[2] https://www.statmt.org/wmt14.

[3] https://www.statmt.org/wmt16.

[4] https://github.com/pytorch/fairseq.

Row#	Model	WMT'14 EN–DE	WMT'14 DE–EN	WMT'16 EN–RO	WMT'16 RO–EN	Speedup
1	Transformer (teacher)	27.48	31.21	33.70	34.05	1×
2	Vanilla NAT	21.14	24.65	29.14	28.93	15.6×
3	w/ NPD	23.44	27.25	31.70	30.93	7.9×
4	w/ Candidate Soups	**25.29**	**28.50**	**32.75**	**32.22**	7.8×
5	CMLM_1	19.44	23.16	27.61	28.33	15.6×
6	w/ NPD	21.72	26.00	30.54	31.34	7.9×
7	w/ Candidate Soups	**23.51**	**27.67**	**32.13**	**32.70**	7.8×
8	CMLM_5	26.37	30.05	32.30	30.73	5.3×
9	w/ NPD	26.94	30.68	33.07	33.68	4.4×
10	w/ Candidate Soups	**27.80**	**31.21**	**33.42**	**34.06**	4.4×
11	GLAT	24.95	28.80	31.29	31.93	15.6×
12	w/ NPD	26.19	30.64	32.46	33.38	7.9×
13	w/ Candidate Soups	**27.59**	**30.95**	**33.22**	**33.73**	7.8×
14	GLAT & DSLP	25.41	29.28	32.32	32.37	14.8×
15	w/ NPD	26.68	30.69	33.32	33.67	7.7×
16	w/ Candidate Soups	**27.72**	**30.98**	**33.71**	**34.11**	7.6×
	Average Improvement	2.92	2.67	2.51	2.91	–

Table 1: Applying Candidate Soups to four different base NAT models, which shows the generality of our algorithm. Translation quality is evaluated in BLEU. Speedup is relative to the AT teacher. All results are achieved by us, except Transformer (teacher), which is obtained from Huang et al. (2021). CMLM_k refers to k iterations of progressive generation. Here, we consider k = 1 and k = 5. Both NPD and Candidate Soups use the AT model to re-score, and the number of candidate results is 5.

we can verify whether Candidate Soups can be applied to various NAT models. In the future, we will test Candidate Soups on more NAT models, such as CTC (Libovickỳ and Helcl, 2018) and CTC+VAE (Gu and Kong, 2021).

4.2 Main Results

Generality of Candidate Soups Table 1 shows the performance improvement of our method for four base models on four translation tasks. Here, our number of candidate results is set to 5[5]. And we use the Transformer-Base as the architecture of the AT model for re-scoring[6]. The results show that for each NAT model and translation task, using Candidate Soups can achieve an average of **2.51-2.92** BLEU higher than the base model. Even compared with the NPD, Candidate Soups improves BLEU by an average of **0.76-1.39** BLEU, which is a considerable improvement in machine translation task. Impressively, using the Candidate Soups on a strong baseline (GLAT & DSLP) can achieve superior performance than the AT teacher. Furthermore, when AT models are used for re-scoring, they can perform parallel decoding as fast as training (Gu et al., 2018). So the inference latency is only roughly doubled, which is still much faster than the AT model.

In conclusion, the above experimental results show that Candidate Soups is a general approach that can significantly improve translation quality while maintaining fast inference speed.

Comparing with the State of the Art To evaluate the best performance Candidate Soups can achieve, we compare our best variant (GLAT+DSLP+Candidate Soups) with previous state-of-the-art NAT models, including Iterative NAT and Fully NAT. As shown in Table 2, compared with the Iterative NAT, we produce a very competitive translation quality with approximately 2×-4× faster inference speed. Compared with Fully NAT, our best variant is better than all existing models in two translation tasks (EN→DE, RO→EN) and is close to the current state-of-the-art performance in the remaining two translation tasks. More encouragingly, our approach even performed better than AT teacher on three translation tasks and achieved very comparable performance on the remaining one translation tasks, which extensively validated the effectiveness of Candidate Soups.

In addition, we also try to use two smaller AT models for re-scoring to accelerate the inference speed further. These two models have the same hyperparameters as Transformer-base, except for the number of layers of decoder and encoder. AT

[5]If not specified below, the default number of candidate results is 5, and both NPD and Candidate Soups will be re-scored by AT teacher.

[6]If not specified below, the default architecture of AT model for re-scoring is Transformer-Base.

Models		Iter.	Speedup	WMT'14		WMT'16	
				EN-DE	DE-EN	EN-RO	RO-EN
AT	Transformer *base* (teacher)	N	1.0×	**27.48**	**31.21**	**33.70**	**34.05**
Iterative NAT	InsT (Stern et al., 2019)	≈log N	4.8×	27.41	-	-	-
	CMLM (Ghazvininejad et al., 2019)*	10	1.7×	27.03	30.53	33.08	33.31
	LevT (Gu et al., 2019)	Adv.	4.0×	27.27	-	-	33.26
	JM-NAT (Guo et al., 2020)*	10	5.7×	27.69	**32.24**	33.52	33.72
	DisCO (Kasai et al., 2020)*	Adv.	3.5×	27.34	31.31	33.22	33.25
	SMART (Ghazvininejad et al., 2020b)*	10	1.7×	27.65	31.27	-	-
	Imputer (Saharia et al., 2020)*	8	3.9×	**28.20**	31.80	**34.40**	**34.10**
	Multi-Task NAT (Hao et al., 2021)*	10	1.7×	27.98	31.27	33.80	33.60
	RewriteNAT (Geng et al., 2021)*	Adv.	-	27.83	31.52	33.63	34.09
Fully NAT	Vanilla NAT (Gu et al., 2018)	1	15.6×	17.69	21.47	27.29	29.06
	DCRF (Sun et al., 2019)	1	10.4×	23.44	27.22	-	-
	Flowseq (Ma et al., 2019)	1	1.1 ×	23.72	28.39	29.73	30.72
	ReorderNAT (Ran et al., 2020)	1	16.1×	22.79	27.28	29.30	29.50
	AXE (Ghazvininejad et al., 2020a)*	1	15.3×	23.53	27.90	30.75	31.54
	ENGINE (Tu et al., 2020)	1	15.3×	22.15	-	-	33.16
	Imputer (Saharia et al., 2020)*	1	18.6×	25.80	28.40	32.30	31.70
	AlignNART (Song et al., 2021)	1	13.4×	26.40	30.40	32.50	33.10
	OAXE (Du et al., 2021)	1	15.3×	26.10	30.20	32.40	33.30
	CTC+VAE (Gu and Kong, 2021)	1	16.5×	**27.49**	**31.10**	**33.79**	**33.87**
	GLAT+NPD (Qian et al., 2021)	1	7.9×	26.55	31.02	32.87	33.51
	GLAT+DSLP (Huang et al., 2021)	1	14.9×	25.69	29.90	32.36	33.06
Ours	GLAT+DSLP+Candidate Soups (AT 6E-6D)	1	7.6×	**27.72**	**30.98**	**33.71**	**34.11**
	GLAT+DSLP+Candidate Soups (AT 4E-2D)	1	10.1×	27.51	30.79	33.58	34.03
	GLAT+DSLP+Candidate Soups (AT 3E-1D)	1	11.5×	27.46	30.69	33.65	34.01

Table 2: Performance comparison between our variant and previous state-of-the-art NAT models. All results reported are quoted from respective papers. **Iter.** is the number of decoding iterations, Adv. denotes adaptive, * denotes models trained with distillation data from Transformer-big. The Speedup is measured on WMT'14 En-DE test set with batch size 1. AT mE-nD refers to the AT model for re-scoring that has m encoder layers and n decoder layers.

4E-2D contains 4 encoder layers and 2 decoder layers, and AT 3E-1D contains 3 encoder layers and 1 decoder layer. Moreover, they were trained using the same distillation data as the NAT model. Surprisingly, even when the small AT models were used for re-scoring, our method maintained a similar performance to the previous model (AT 6E-6D), and its inference speed was **10.1**×-**11.5**× that of the AT model. This result further proves that Candidate Soups can well balance the trade-off between translation quality and inference speed.

4.3 Ablation Study and Analysis

Influence of the Candidate Number In order to analyze the effect of the candidate translation number on the Candidate Soups, we conduct experiments with different candidate numbers. Figure 5 shows the relationship between translation quality and the number of candidate results. Specifically, with the increased candidate numbers, the quality of the translation generally maintains a growth trend. Especially when the candidate results number is less than 4, the Candidate Soups performance is significantly improved when the number increases. However, when the number increases to a certain threshold, the quality of the translation begins to fluctuate, even showing a slight downward trend. We guess this is because when the number is larger than the threshold, the quality of the candidate translations may gradually decrease due to the gap between the predicted length and the actual length becoming larger. Furthermore, there may be duplication between the candidate results. Therefore, using 3-7 candidate translations is enough for Candidate Soups to significantly improve the quality of the final translation.

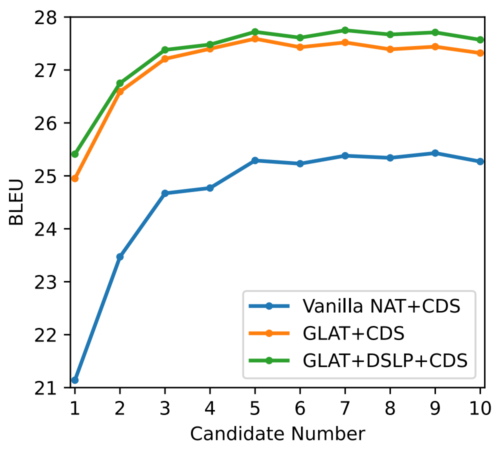


Figure 5: Translation quality under different number of candidate results on WMT'14 EN-DE.

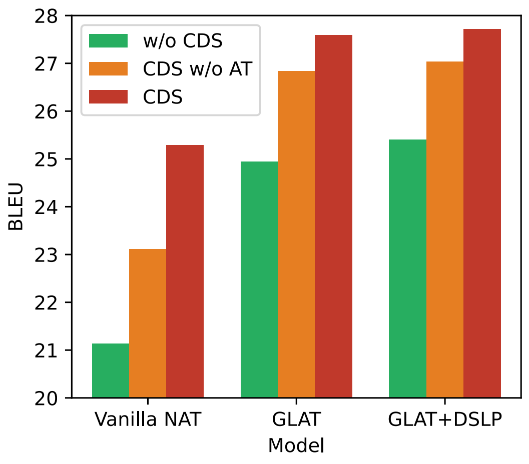


Figure 6: Performance of NAT model that with or without AT teacher re-scoring on WMT'14 EN-DE.

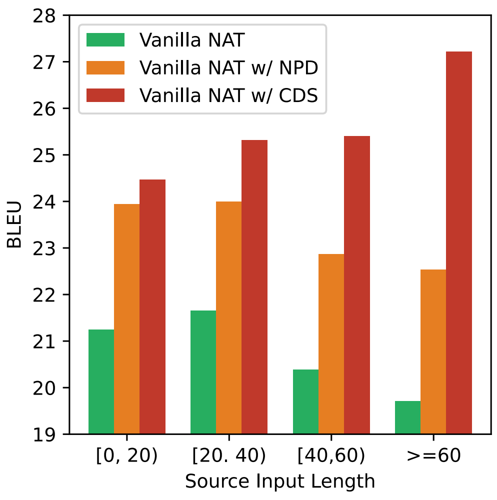


Figure 7: Performance under different source input length on WMT'14 EN-DE.

Influence of the Autoregressive Teacher To analyze the effect of whether using the AT model to re-score on our proposed method, we conducted experiments on the WMT'14 EN-DE dataset. Figure 6 demonstrates that Candidate Soups has a different dependence on AT model in different NAT models. For a model with weaker performance, such as Vanilla NAT, when AT model is not used for re-scoring, Candidate Soups' performance degrades significantly. However, for GLAT and GLAT+DSLP, which can produce high-quality translations, Candidate Soups can still increase approximately **1.89-1.63** BLEU even without re-scoring with the AT model. Notably, using Candidate Soups in this case hardly increases the inference time of the NAT model. Moreover, after using AT model for re-scoring, the effect of Candidate Soups can be further improved, which increases **2.64** and **2.31** BLEU on the GLAT and GLAT+DSLP, respectively. In addition, the results in Table 2 show that we can further improve the inference speed on the premise of guaranteeing translation quality by using a smaller AT model for re-scoring.

Influence of the Source Input Length To analyze the influence of source sentence length on Candidate Soups' performance, we divide the source sentence after BPE into different intervals by length and calculate the BLEU score of each interval. The histogram of results is presented in Figure 7. It can be seen that the performance of Vanilla NAT degrades significantly as the length of the source sentence increases. Although NPD can improve the overall translation quality, the translation quality of long source sentences is still inferior. However, Candidate Soups can dramatically improve Vanilla NAT's performance and enables long sentences to achieve much higher BLEU than short. Impressively, the BLEU score of the source sentence length ranging from 40 to 60 increases by **5.01**, and the BLEU score of the source sentences longer than 60 increases by **7.51**.

We believe this is because the NAT model tends to generate more uncertain and diverse candidate results for longer source sentences. This feature enables Candidate Soups to obtain more useful information in the candidate results to generate higher-quality translations. These experimental results further verify the potential of the Candidate Soups in translating complex long sentences. More experimental results and analyses are presented in the Appendix B.

5 Conclusion

In this paper, we propose "Candidate Soups," which can discover and fuse valuable information from multiple candidate translations based on model uncertainty. This approach is general and can be applied to various NAT models. Extensive experimental results prove that the translation quality of the NAT model can be significantly improved by using Candidate Soups, especially for long sentences that are difficult to translate. And the trade-off between translation quality and inference speed is well controlled and balanced by Candidate Soups. Furthermore, our best variant can achieve better results on three translation tasks than the AT teacher while maintaining NAT's high-speed inference.

Limitations

Although our proposed method can significantly improve the performance of non-autoregressive translation (NAT) models, it relies on trained autoregressive translation (AT) models to a certain extent. Not using the AT model for re-scoring can lead to poorer quality of translations generated by Candidate Soups, especially when using it for the poorer performing NAT model. Although using a small AT model is sufficient for Candidate Soups to achieve decent performance, it still results in a drop in inference speed and more GPU resources being used for translation. In addition, the performance of the AT model may limit the upper bound of the Candidate Soups' capability. Therefore, we will explore new methods that can be effective without AT re-score in the future.

Ethics Statement

Our work has potentially positive implications for various non-autoregressive machine translation applications. It is a general method that can be applied to virtually all existing non-autoregressive translation models to improve their performance while maintaining their high inference speed. Our work can facilitate the implementation of non-autoregressive translation models in commercial companies and humanitarian translation services in the future and promote cultural exchanges between different languages and different races.

Acknowledgements

This work was supported by NSFC grants (No. 62136002), National Key RD Program of China (2021YFC3340700) and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805.

Cunxiao Du, Zhaopeng Tu, and Jing Jiang. 2021. Order-agnostic cross entropy for non-autoregressive machine translation. *arXiv preprint arXiv:2106.05093*.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *ArXiv*, abs/1506.02142.

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, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. 2020a. Aligned cross entropy for non-autoregressive machine translation. In *International Conference on Machine Learning*, pages 3515–3523. PMLR.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. *arXiv preprint arXiv:1904.09324*.

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

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, Online. Association for Computational Linguistics.

Jiatao Gu, Changhan Wang, and Jake Zhao. 2019. Levenshtein transformer. *arXiv preprint arXiv:1905.11006*.

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.

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.

Chenyang Huang, Hao Zhou, Osmar R Zaïane, Lili Mou, and Lei Li. 2021. Non-autoregressive translation with layer-wise prediction and deep supervision. *arXiv preprint arXiv:2110.07515*.

Jungo Kasai, James Cross, Marjan Ghazvininejad, and Jiatao Gu. 2020. Non-autoregressive machine translation with disentangled context transformer. In *International Conference on Machine Learning*, pages 5144–5155. PMLR.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *EMNLP*.

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

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. Flowseq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4282–4292.

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 the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

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 (Volume 1: Long Papers)*, pages 1993–2003, Online. Association for Computational Linguistics.

Qiu Ran, Yankai Lin, Peng Li, and Jie Zhou. 2020. Learning to recover from multi-modality errors for non-autoregressive neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3059–3069.

Qiu Ran, Yankai Lin, Peng Li, and Jie Zhou. 2021. Guiding non-autoregressive neural machine translation decoding with reordering information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13727–13735.

Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-autoregressive machine translation with latent alignments. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1098–1108.

Raphael Shu, Jason Lee, Hideki Nakayama, and Kyunghyun Cho. 2020. Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8846–8853.

David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. *ArXiv*, abs/1901.11117.

Jongyoon Song, Sungwon Kim, and Sungroh Yoon. 2021. Alignart: Non-autoregressive neural machine translation by jointly learning to estimate alignment and translate. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1–14.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. Mass: Masked sequence to sequence pre-training for language generation. In *ICML*.

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

Shuo Sun, Hongxu Hou, Nier Wu, Ziyue Guo, and Chaowei Zhang. 2020. Multi-reward based reinforcement learning for neural machine translation. In *Proceedings of the 19th Chinese National Conference on Computational Linguistics*, pages 984–993, Haikou, China. Chinese Information Processing Society of China.

Tianxiang Sun, Xiangyang Liu, Wei Zhu, Zhichao Geng, Lingling Wu, Yilong He, Yuan Ni, Guotong Xie, Xuanjing Huang, and Xipeng Qiu. 2022. A simple hash-based early exiting approach for language understanding and generation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2409–2421, Dublin, Ireland. Association for Computational Linguistics.

Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin, and Zhi-Hong Deng. 2019. Fast structured decoding for sequence models. In *NeurIPS*.

Lifu Tu, Richard Yuanzhe Pang, Sam Wiseman, and Kevin Gimpel. 2020. ENGINE: Energy-based inference networks for non-autoregressive machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2819–2826, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. In *EMNLP*.

Chunting Zhou, Graham Neubig, and Jiatao Gu. 2020. Understanding knowledge distillation in non-autoregressive machine translation. *ArXiv*, abs/1911.02727.

Wei Zhu, Xiaoling Wang, Yuan Ni, and Guotong Xie. 2021a. GAML-BERT: Improving BERT early exiting by gradient aligned mutual learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3033–3044, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Wei Zhu, Xiaoling Wang, Xipeng Qiu, Yuan Ni, and Guo Tong Xie. 2021b. Autotrans: Automating transformer design via reinforced architecture search. In *NLPCC*.

A Background

A.1 Autoregressive Translation

The autoregressive translation (AT) model achieves sort-of-the-art performance on multiple machine translation tasks (Song et al., 2019; Sun et al., 2020). Given a source sentence $X = (x_1, x_2, \ldots, x_n)$ and the target sentence $Y = (y_1, y_2, \ldots, y_m)$, the AT model decomposes the target distribution of translations according to the chain rule:

$$p_{\mathrm{AT}}(Y \mid X; \theta) = \prod_{t=1}^{m} p\left(y_t \mid y_{<t}, X; \theta\right) \tag{1}$$

where $y_{<t}$ denotes generated previous tokens before the t^{th} position. During the training process, the AT model is trained via the teacher-forcing strategy that uses ground truth target tokens as previously decoded tokens so that the output of the decoder can be computed in parallel.

However, during inference, the AT model still needs to generate translations one by one from left to right until the token that represents the end **[EOS]** is generated. Although AT model has good performance, its autoregressive decoding method dramatically reduces the decoding speed and becomes the main bottleneck of its efficiency.

A.2 Non-Autoregressive Translation

To improve the inference speed, the non-autoregressive translation (NAT) model is proposed (Gu et al., 2018), which removes the order dependency between target tokens and can generate target words simultaneously:

$$p_{\mathrm{NAT}}(Y \mid X; \theta) = \prod_{t=1}^{m} p\left(y_t \mid X; \theta\right) \tag{2}$$

where m denotes the length of the target sentence. Generally, NAT models need to have the ability to predict the length because the entire sequence needs to be generated in parallel. A common practice is to treat it as a classification task, using the information from the encoder's output to make predictions.

However, this superior decoding speed is achieved at the cost of significantly sacrificing translation quality. Because NAT is only conditioned on source-side information, but AT can obtain the strong target-side context information provided by the previously generated target tokens, there is always a gap in the performance of NAT compared with AT.

A.3 Noisy Parallel Decoding

Noisy parallel decoding (NPD) (Gu et al., 2018) is a stochastic search method that can draw samples from the length space and compute the best translation for each length as a candidate result. Then NPD selects the translation with the highest average log-probability as the final result:

$$Y_{NPD} = \underset{Y^m}{\operatorname{argmax}} \frac{1}{m} \sum_{t=1}^{m} \log p_{NAT}\left(y_t^m \mid X; \theta\right) \quad (3)$$

where Y^m is the translation predicted by the NAT model based on the length m. NPD also can use the AT model to identify the best translation:

$$Y_{NPD} = \underset{Y^m}{\operatorname{argmax}} \frac{1}{m} \sum_{t=1}^{m} \log p_{AT}\left(y_t^m \mid y_{<t}^m, X; \theta\right)$$

Notably, when an AT model is used for re-scoring, it can be decoded in parallel as it does at training time. Moreover, since all search samples can be computed independently, even with AT model for re-scoring, the latency of the NPD process is only doubled compared to computing a single translation.

B Additional Analysis Experiment

B.1 Influence of the Knowledge Distillation

Compared with the original data, the distillation data generated by the AT model has less noise and is more deterministic, which can effectively alleviate the multimodality problem of the NAT model. Therefore, almost all the existing NAT model adopts the method of Knowledge Distillation (KD) for training. However, generating distillation data tends to consume significant computing resources and time, and using distillation data to train NAT models may limit the translation capabilities of NAT models.

In order to analyze whether our proposed method can still be effective in the scenarios where knowledge distillation is not used, we conducted experiments on the WMT'14 EN-DE dataset. Figure 8 shows the performance of Candidate Soups on the NAT model that does not use knowledge distillation. For Vanilla NAT and GLAT, the performance trained with raw data is significantly reduced compared to that trained with knowledge distillation. However, after using Candidate Soups, the BLEU of Vanilla NAT and GLAT respectively increased by **4.54** and **4.88**, which was only 5.68 and **1.28** lower than the performance with knowledge distillation. We believe that this significant performance improvement may be since NAT models without knowledge distillation may produce more diverse candidate translations, thus enabling Candidate Soups to fully play its role and obtain higher-quality translations from different candidate translations. The experimental results show that the Candidate Soups can significantly improve the performance of the NAT model without knowledge distillation, which proves the potential of the Candidate Soups in this scenario.

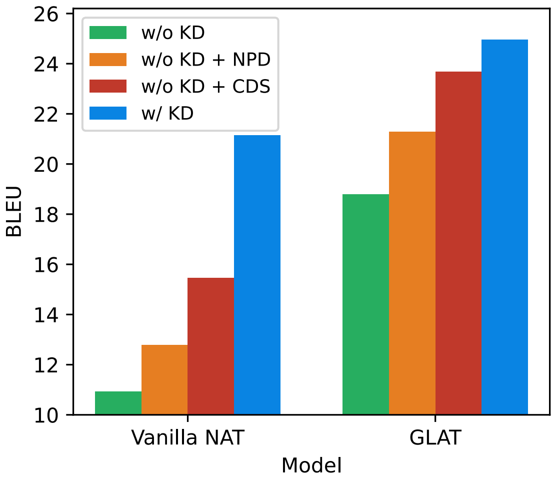


Figure 8: Performance of NAT model without Knowledge Distillation (KD) on WMT'14 EN-DE.

B.2 Influence of introducing uncertainty methods

In addition to introducing uncertainty through length, we propose two other methods for generating different candidate translations:

- Use the prediction results of different decoder layers. DSLP (Huang et al., 2021) is a general method that can be applied to various NAT models, and it needs to predict the translations in each decoder layer. Therefore, Candidate Soups can be combined with DSLP, and any NAT model using DSLP can use Candidate Soups to fuse the results of different layers. In this experiment, we use the results generated by the last 5 layers of the decoder.

- Generate different translations by maintaining dropout during inference (Gal and Ghahramani, 2016). Even with the same input, the model can produce different outputs since

Model	**WMT'14**	
	EN-DE	DE-EN
GLAT+DSLP	25.41	29.28
w/ Candidate Soups(Length)	27.72	30.98
w/ Candidate Soups(Layer)	26.77	29.96
w/ Candidate Soups(Dropout)	26.66	30.01

Table 3: Performance of Candidate Soups using different methods of introducing uncertainty. Length means introducing uncertainty with different lengths. Layer means using the prediction results of different decoder layers. Dropout means maintaining dropout during inference. The number of candidate translations is 5.

dropout activates different neurons each time. In this experiment, the dropout probability at inference is set to 0.02.

Table 3 shows the performance of Candidate Soup under three different ways of introducing uncertainty. The experimental results show that, compared with the other two methods, when uncertainty is introduced by length, Candidate Soups improves the translation quality more significantly. We speculate that this is because the length uncertainty can ensure that the generated translations are more diverse under the premise of high quality.

However, for layer uncertainty, the quality of the translations produced by the first layer will be significantly lower than that of the last layer. These low-quality candidate translations are of little help to Candidate Soups and even affect the performance of Candidate Soups. For dropout uncertainty, the candidate translations generated will be affected by the dropout probability. On the one hand, if the dropout probability is set too high, it may reduce the overall quality of the candidate translations. On the other hand, the generated candidate translations will be less diverse if the dropout probability is low. So we further need to spend time searching for the optimal dropout probability setting for different NAT models and tasks. However, these two methods can still achieve about 1 BLEU improvement on the strong baseline (GLAT+DSLP), and their generalization ability is stronger than the length-based method. In addition, Candidate Soups can also be used as a new model ensemble method to enhance the final translation quality by using the output from multiple NAT models. We will discuss this in future work.