# Speeding Up Transformer Decoding via an Attention Refinement Network

**Kaixin Wu**[1], **Yue Zhang**[2,3], **Bojie Hu**[1], **Tong Zhang**[1]
[1]Tencent Minority-Mandarin Translation, Beijing, China
[2]School of Engineering, Westlake University
[3]Institute of Advanced Technology, Westlake Institute for Advanced Study
{danielkxwu, bojiehu, zatozhang}@tencent.com
zhangyue@westlake.edu.cn

## Abstract

Despite the revolutionary advances made by Transformer in Neural Machine Translation (NMT), inference efficiency remains an obstacle due to the heavy use of attention operations in auto-regressive decoding. We thereby propose a lightweight attention structure called *Attention Refinement Network* (ARN) for speeding up Transformer. Specifically, we design a weighted residual network, which reconstructs the attention by reusing the features across layers. To further improve the Transformer efficiency, we merge the self-attention and cross-attention components for parallel computing. Extensive experiments on ten WMT machine translation tasks show that the proposed model yields an average of 1.35× faster (with almost no decrease in BLEU) over the state-of-the-art inference implementation. Results on widely used WMT14 En→De machine translation tasks demonstrate that our model achieves a higher speed-up, giving highly competitive performance compared to AAN and SAN models with fewer parameter numbers[1].

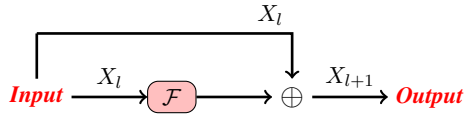



Figure 1: The information transformation between the input and output after through a sub-layer $\mathcal{F}$, where $X_{l+1} = X_l + \mathcal{F}(X_l)$

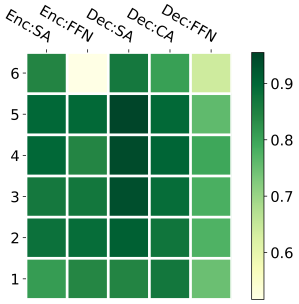



Figure 2: The cosine similarity of the input and output for every sub-layer over validation set on WMT14 En→De translation task. "Enc", "Dec", "SA", "CA", "FFN" represent Encoder, Decoder, Self-attention, Cross-attention and Feed-forward network, respectively. Darker cells denote more similar.

## 1 Introduction

Transformer (Vaswani et al., 2017) has become the dominant approach in the NMT literature, which achieves superior translation performance and efficiency due to its well-designed attention mechanism. The highly parallelizable architecture enables Transformer to capture the dependency among positions over the entire sequence parallelly for a faster training step. However, the inference efficiency remains a bottleneck for Transformer. In inference, Transformer follows an auto-regressive generation paradigm and generates the target words one by one on the decoder side. The heavy use of dot-product attention operations even further slows Transformer efficiency. In addition, there are a large number of attention sub-layers in the multi-layer stacked Transformer, which makes it prone to redundancy. As shown in Figure 2, we observe redundant computation in Transformer attention sub-layer, especially the attention in the decoder. This inspires us to explore a lightweight attention structure to speed up Transformer decoding.

Many efforts have been dedicated to accelerating the decoding process of Transformer. AAN (Zhang et al., 2018) adopts an average strategy to avoid computing the correlations over the entire input word. However, this method requires a complicated network and only focuses on the decoder-side self-attention. SAN (Xiao et al., 2019) reuses the attention results among layers, but it requires a model to learn which layers should be allowed to share. Besides, another representative

[1]https://github.com/Kaixin-Wu-for-Open-Source/ARN



*Proceedings of the 29th International Conference on Computational Linguistics*, pages 5109–5118
October 12–17, 2022.

approach (Gu et al., 2017; Guo et al., 2019; Wang et al., 2019) follows another line to abandon the auto-regressive generation property and produces target sequences in parallel, thus it fails to model the word dependencies.

We investigate an alternative lightweight attention structure for Transformer acceleration. As shown in Figure 3, the main idea is to reconstruct the attention via a weighted combination of high-level and low-level features, rather than the standard dot-product function. The two parts represent the attention results of the current layer and its previous layer, respectively. This weight, a learned matrix under the guidance of the two input features, which determines how many low-level features can be selected to fuse. Consequently, we refer to our model as ***A**ttention **R**efinement **N**etwork*, or ARN in short. Our combination process is computationally inexpensive, which allows us to employ less computation to obtain the attention results of adjacent layers and combine them to approximate the original attention results. ARN structure can be viewed as a weighted residual network, which acquires the attention results by reusing the features across layers. This way introduces low-level features into the attention sub-layer to further enhance model confidence, which is an improvement over SAN (Xiao et al., 2019). In addition, this method can be applied to both self-attention and cross-attention on the decoder side and we merge the above two components to further improve decoding efficiency. Moreover, ARN structure is simple and it is easy to implement. As another "bonus", ARN requires fewer parameters, so it is faster to train and maintains a smaller memory footprint.

Extensive experiments on ten WMT translation tasks show that ARN achieves an average of $1.35\times$ faster with performance on par with a strong baseline. Compared to AAN and SAN baselines, our model gives a higher speed-up as well as highly competitive performance with fewer parameter numbers on widely used WMT14 En→De translation tasks.

## 2 Standard Transformer Attention

Standard Transformer follows the popular encoder-decoder paradigm, which consists of a 6-layer encoder and a 6-layer decoder. The overall architecture only contains stacked attention and feed-forward networks (FFN) in Transformer. There are three types of attention mechanisms: the encoder-side self-attention, the decoder-side self-attention and the cross-attention. On the encoder side, each layer follows the order of operations that could be defined as: *self-attention* → FFN. Similarly, the decoder side follows the way: *self-attention* → *cross-attention* → FFN.

The attention model in Transformer is scaled dot-product attention. See Figure 3 (a) for an illustration of the standard attention. The input of attention is a tuple of $(Q_l, K_l, V_l)$, where $Q_l \in \mathbb{R}^{m\times d}$ and $K_l, V_l \in \mathbb{R}^{n\times d}$ are the matrices of corresponding queries, keys and values of the $l$-th layer. For encoder or decoder self-attention, $m = n$ represents the source or target sequence length. For cross-attention, $m$ and $n$ are the target sequence length and source sequence length, respectively. $d$ is the dimension of the hidden representation. We first compute the attention distribution via a scaled dot-product and softmax operations.

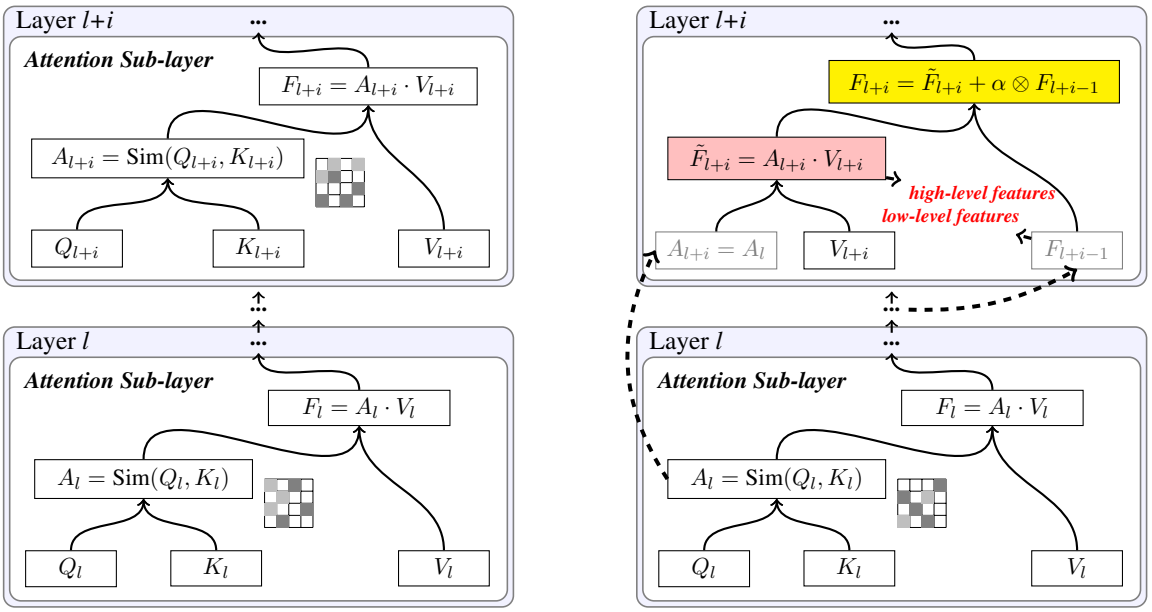
$$\begin{aligned} A_l &= \text{Sim}(Q_l, K_l) \\ &= \text{Softmax}(\frac{Q_l \cdot {K_l}^T}{\sqrt{d}}) \end{aligned} \tag{1}$$

where $A_l$ is an $m \times n$ matrix, which represents the degree of relevance between different positions of queries and values. The ouput of attention is a weighted sum of values, and it can be defined as:

$$F_l = A_l \cdot V_l \tag{2}$$

where $Q_l, K_l, V_l$ are all generated by a linear transformation. In self-attention (encoder or decoder), the three parts share the same source, which comes from the output of its previous layer. While in cross-attention, the difference is that the $K_l$ and $V_l$ from the output of encoder side. $F_l$ is the attention results of the $l$-th attention sub-layer, which is then fed into the next sub-layer.

Note that the matrix multiplications in Eq. 1 and Eq. 2 are computationally expensive. This is even worse for inference because the operation is repeated until an end symbol is reached due to the auto-regressive generation property. We also compute the cosine similarity of the input and output for every sub-layer in Transformer. Figure 1 is the definition of a sub-layer $\mathcal{F}$ and Figure 2 shows the similarity results. As can be seen that the input and output of the attention sub-layer are very similar, especially the attention in the decoder. Specifically, we observe that the decoder-side self-attention presents the highest similarity compared to other sub-layers, followed by the cross-attention.




(a) Standard Transformer Attention

(b) Attention Refinement Network (ARN)

Figure 3: Comparison of the standard attention model and ARN model

Most of their similarities are more than 90%, and some even more than 95%. All these show the possibility of removing redundant computation in Transformer and lead us to learn a lightweight attention structure.

## 3 Attention Refinement Network (ARN)

The proposed method for speeding up Transformer concentrates on the decoder-side attention because the decoder is the bottleneck of Transformer inference.

**Overall Architecture.** The proposed ARN module is shown in Figure 3 (b). We assume that the decoder contains $L = M \times N$ layers. The decoder is simply divided into $M$ parts equally, and each part contains $N$ layers. For each ARN module, the bottom layer is the dot-product attention same as used in standard Transformer, and the next $N - 1$ layers are the lightweight attention composed of the two inputs. One is an approximate attention result of the current layer by reusing the attention weights within adjacent layers, and the other is the up-sampled attention results from its previous layer. Since the upper layer contains more semantic information, we refer to the two inputs high-level features and low-level features, respectively. $\alpha$ is a learned weight matrix, which is used to represent the fusion degree of low-level features. In contrast to Figure 3 (a), ARN reconstructs the attention via a feature fusion of the two input sources to replace the original dot-product attention. ARN can be regarded as a weighted residual network, which sums to obtain the attention results by reusing the features across layers. The ARN module is applied to both self-attention and cross-attention for speeding up Transformer.

### 3.1 The Model

**Weighted Residual Network.** For each ARN module, the attention is reconstructed via a weighted sum of the two input sources.

$$F_{l+i} = \tilde{F}_{l+i} + \alpha \otimes F_{l+i-1} \quad \text{for} \quad i \in [1, N-1] \tag{3}$$

where $\tilde{F}_{l+i}$ and $F_{l+i-1}$ are the high-level features and low-level features, respectively. $F_{l+i-1}$ is the attention results of the previous layer, and we reuse it as our low-level features of the current layer. $\otimes$ denotes the element-wise multiplication. $\tilde{F}_{l+i}$ is defined as:

$$\begin{aligned} \tilde{F}_{l+i} &= A_{l+i} \cdot V_{l+i} \\ &= A_l \cdot V_{l+i} \end{aligned} \tag{4}$$

where $A_l$ denotes the attention weights of the $l$-th layer, its calculation process is shown in Eq. 1. Previous work show that the attention weights are redundant and the adjacent layers share similar distributions (Michel et al., 2019; Behnke and

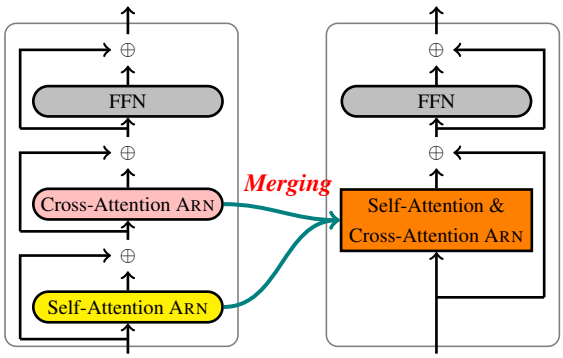



Figure 4: Merging self-attention and cross-attention in ARN decoder layer.

| Model | Complexity per Step |
|---|---|
| Decoder self-attention | $O(n \cdot d^2 + n^2 \cdot d)$ |
| Decoder self-attention ARN | $O(n^2 \cdot d)$ |
| Cross-attention | $O(m \cdot d^2 + m^2 \cdot d)$ |
| Cross-attention ARN | $O(m^2 \cdot d)$ |

Table 1: Computation complexity of different attention structures in a decoding step. $m$ and $n$ are the source sentence length and target sentence length, $d$ is the dimension of the hidden representation.

Heafield, 2020; Xiao et al., 2019). Thus, we reuse the attention weights of adjacent lower layers to construct the high-level features. $\alpha$ is a learned matrix, which is obtained from the following process:

$$\begin{aligned}\alpha &= G(F_{l+i-1}, \tilde{F}_{l+i}) \\ &= \text{ReLU}(\frac{\mathbf{W}^{\alpha}_{l+i} \cdot \text{Max}(F_{l+i-1}, \tilde{F}_{l+i})}{\sqrt{d}})\end{aligned} \quad (5)$$

Here, $\text{Max}(\cdot)$ is an element-wise operation and $\mathbf{W}^{\alpha}_{l+i}$ is a learnable parameter. According to Eq. 5, $\alpha$ is a sparse weight matrix obtained from the supervision of high-level and low-level features. This weight matrix highlights important semantic information of low-level features, and it determines how many low-level features can be selected for fusion. In other words, the upper layer features can be refined through the fusion of the lower layer important features iteratively.

**Merging Self-attention and Cross-attention.** For the self-attention and cross-attention in a decoder layer, the formula can be simply defined as:

$$F_l = Q_l + \text{SelfAttn}(Q_l) + \text{CrossAttn}(\tilde{Q}_l, R) \quad (6)$$

where $\tilde{Q}_l = Q_l + \text{SelfAttn}(Q_l)$, $Q_l$ is the input of $l$-th layer and $R$ denotes the hidden reprensentation of encoder. As shown in Figure 2, the input and output of attention sub-layer in the decoder are very similar, especially the self-attention. Thus, we can regard that $\tilde{Q}_l \approx Q_l$, the $F_l$ can be rewritten as follows:

$$F_l = Q_l + \underbrace{\text{SelfAttn}(Q_l) + \text{CrossAttn}(Q_l, R)}_{\textit{merge for parallel computing}} \quad (7)$$

The Eq. 7 provides the conditions for us to do parallel computing. The merging process is shown in Figure 4. Original cross-attention relies on the output of self-attention and it needs to wait until self-attention calculation is completed. We thereby merge the self-attention and cross-attention into a single one to further improve the decoding efficiency. This way is faster because the model can compute the attention results of self-attention and cross-attention simultaneously. Thus, the formula Eq. 2 and Eq. 4 can be rewritten as follows:

$$\begin{aligned}F_l &= F^S_l + F^C_l \\ &= A^S_l \cdot V^S_l + A^C_l \cdot V^C_l\end{aligned} \quad (8)$$

$$\tilde{F}_{l+i} = [A^S_l; A^C_l] \cdot [V^S_{l+i}; V^C_{l+i}] \quad (9)$$

Here, $F^S_l$, $F^C_l$, $A^S_l$, $A^C_l$, $V^S_l$, $V^C_l$ represent self-attention results, cross-attention results, self-attention weights, cross-attention weights, self-attention values and cross-attention values of the $l$-th layer, respectively. $[;]$ denotes the concatenate operation. The main idea is that we sum the self-attention and cross-attention representations as a whole to decode in ARN model, instead of computing them separately. Similarly, the Eq. 5 also share the strength of parallelization.

### 3.2 Decoding Complexity

We investigate how the ARN accelerates Transformer compared to the original dot-product attention. For each decoding step, the self-attention first connects all positions with a constant number of sequentially executed operations (Eq. 1, $O(n \cdot d^2)$), and then obtains atention results via a weighted sum operations (Eq. 2, $O(n^2 \cdot d)$). Thus, the computation complexity of self-attention is $O(n \cdot d^2 + n^2 \cdot d)$. Similarly, the cross-attention is $O(m \cdot d^2 + m^2 \cdot d)$. The ARN model is fast because the weighted combination process (Eq. 3 and Eq. 5) are all element-wise operations, which require less computation and the Eq. 4 occupies a major computation. The self-attention and cross-attention adopting ARN method are $O(n^2 \cdot d)$ and

$O(m^2 \cdot d)$, respectively. See Table 1 for the details. In particular, $m$ and $n$ are smaller than the representation dimensionality $d$, which is most often the case with sentence representations used by NMT models. Moreover, the self-attention and cross-attention components are merged for higher parallelization. All these enable the ARN model to enjoy greater decoding efficiency.

## 4 Experiments

### 4.1 Experimental Settings

**Datasets.** We evaluate our proposed model on WMT14 and WMT17 translation tasks. The details follow as:

- WMT14 En→{De, Fr}. For En→De task, we choose newstest2013 as validation set and newstest2014 as test set. For En→Fr task, we validate the system on the combination newstest2012 and newstest2013 as validation set and test it on newstest2014.

- WMT17 En↔{Fi, De, Cs, Ru}. For validation, we concatenate the data of newstest2014-2016. For test, we choose newstest2017.

Table 2 shows the statistics of these datasets. For all datasets, we tokenize every sentence with Moses tokenizer (Koehn et al., 2007) and use byte pair encodings (BPE) with 32K split operations for subword segmentation (Sennrich et al., 2015). We remove sentences with more than 250 subword units. For WMT14 En→De translation task, we share the source and target vocabularies. We report the case-sensitive tokenized BLEU using *multi-bleu.perl*

**Implementation Detail.** For all machine translation tasks, our systems are based on an open-source implementation of *fairseq-py*[2]. We replicate the model setup of Vaswani et al. (2017). The standard implementation of Transformer baseline consists of a 6-layer encoder and a 6-layer decoder. The embedding size and FFN hidden size are 512 and 2048, respectively. The number of attention heads is set to 8. Dropout and label smoothing are used as regularization, both set to 0.1. We adopt Adam (Kingma and Ba, 2014) optimizer with an adaptive learning rate schedule as described in Vaswani et al. (2017), the warmup step and learning rate are 4K and $7\times e^{-4}$, respectively. All experiments are trained on 8 NVIDIA Tesla V100 GPUs with mixed-precision training and a batch size of 4096 tokens per GPU. For widely used WMT14 translation tasks, all models are trained for 100K steps as provided by Vaswani et al. (2017). For all WMT17 tasks, we stop training until the model no longer improves on the validation set. We average parameters of the last 5 checkpoints to obtain the final model. In inference, all models are decoded with half-precision on a single V100 GPU. By default, the batch size of decoding is set to 1 for avoiding invalid computations on padding. The beam size is 4 and length penalty is set to 0.6. All speed testing are based on the state-of-the-art inference implementation of Transformer with attention caching[3].

| Source | Lang. | Train Set | | Valid. Set | | Test Set | |
|---|---|---|---|---|---|---|---|
| | | sent. | word | sent. | word | sent. | word |
| WMT14 | En→De | 4.5M | 220M | 3000 | 110K | 3003 | 114K |
| | En→Fr | 35M | 2.2B | 26K | 1.7M | 3003 | 155K |
| WMT17 | En↔Fi | 2.6M | 108M | 8870 | 330K | 3002 | 110K |
| | En↔De | 5.9M | 276M | 8171 | 356K | 3004 | 128K |
| | En↔Cs | 52M | 1.2B | 8658 | 354K | 3005 | 118K |
| | En↔Ru | 25M | 1.2B | 8819 | 391K | 3001 | 132K |

Table 2: Data statistics (# of sentence pairs and # of words, M=million, B=billion, K=kilo)

### 4.2 Baselines

We compare our proposed ARN model with the following baselines:

- **Transformer**(Vaswani et al., 2017) is the most widely-used NMT system with self-attention mechanism.

- **AAN**(Zhang et al., 2018) is a classic lightweight attention NMT model, which leverages an average attention network for inference acceleration.

- **SAN**(Xiao et al., 2019) is another lightweight attention NMT model via sharing attention results among layers. For convenience, we simply adopt the sharing strategy per 2 layers, which can maintain a relatively high speed-up at the trade-off on BLEU performance.

We re-implement all the above baseline systems, and their experimental settings are consistent with our ARN model.

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

[3] An engineering optimization technique, which cache the attention output of previous positions and then reuse it in following steps.

| Model | BLEU | $\Delta_{\text{BLEU}}$ | Speed | $\Delta_{\text{Speed}}$ | #Param |
|---|---|---|---|---|---|
| Transformer | 27.34 | 0.00 | 110.55 | 0.00% | 58.7M |
| ARN$_2$ | 27.35 | +0.01 | 134.91 | +22.04% | 55.0M |
| ARN$_3$ | 27.26 | -0.08 | 149.89 | +35.59% | 53.7M |
| ARN$_6$ | 26.86 | -0.48 | 159.62 | +44.39% | 52.4M |

Table 3: BLEU scores [%] and translation speeds (token/sec) on WMT14 En→De task for different sharing policies. ARN$_n$ means that adopt a ARN strategy every $n$ layer.

| Source | Lang. | Model | BLEU | $\Delta_{\text{BLEU}}$ | Speed | $\Delta_{\text{Speed}}$ |
|---|---|---|---|---|---|---|
| WMT14 | En→De | Transformer | 27.34 | 0.00 | 110.55 | 0.00% |
| | | ARN | 27.26 | -0.08 | 149.89 | +35.59% |
| | En→Fr | Transformer | 39.73 | 0.00 | 106.18 | 0.00% |
| | | ARN | 39.56 | -0.17 | 149.58 | +40.87% |
| WMT17 | En→Fi | Transformer | 21.50 | 0.00 | 107.49 | 0.00% |
| | | ARN | 21.47 | -0.03 | 144.62 | +34.54% |
| | Fi→En | Transformer | 25.06 | 0.00 | 110.13 | 0.00% |
| | | ARN | 25.14 | +0.08 | 149.24 | +35.51% |
| | En→De | Transformer | 28.66 | 0.00 | 109.02 | 0.00% |
| | | ARN | 28.46 | -0.20 | 147.38 | +35.19% |
| | De→En | Transformer | 34.56 | 0.00 | 115.37 | 0.00% |
| | | ARN | 34.38 | -0.18 | 151.56 | +31.37% |
| | En→Cs | Transformer | 23.94 | 0.00 | 111.03 | 0.00% |
| | | ARN | 23.62 | -0.32 | 148.77 | +33.99% |
| | Cs→En | Transformer | 29.94 | 0.00 | 111.14 | 0.00% |
| | | ARN | 29.92 | -0.02 | 145.62 | +31.02% |
| | En→Ru | Transformer | 30.69 | 0.00 | 110.40 | 0.00% |
| | | ARN | 30.24 | -0.45 | 146.99 | +33.14% |
| | Ru→En | Transformer | 34.22 | 0.00 | 106.72 | 0.00% |
| | | ARN | 33.94 | -0.28 | 148.67 | +39.31% |
| Avg. | | Transformer | 29.56 | 0.00 | 109.80 | 0.00% |
| | | ARN | 29.40 | -0.16 | 148.23 | +35.00% |

Table 4: BLEU scores [%] and translation speeds (token/sec) on WMT14 and WMT17 translation tasks.

| Model | BLEU | $\Delta_{\text{BLEU}}$ | Speed | $\Delta_{\text{Speed}}$ | #Param |
|---|---|---|---|---|---|
| Transformer | 27.34 | 0.00 | 110.55 | 0.00% | 58.7M |
| AAN | 27.16 | -0.18 | 116.47 | +5.36% | 70.7M |
| SAN | 27.17 | -0.17 | 131.72 | +19.15% | 55.0M |
| ARN | 27.26 | -0.08 | 149.89 | +35.59% | 53.7M |

Table 5: Comparison of different attention models on WMT14 En→De translation task.

### 4.3 Main Results

We test our approach on the widely used WMT14 En→De translation task. Table 3 reports the various results of BLEU scores and translation speeds, which adopt the simple sharing policy with ARN structure. The speed-up of the ARN$_3$ model is 1.35× with almost no decrease in BLEU. The ARN$_3$ is our ARN baseline system in the following sections. The BLEU will drop significantly if more layers are shared, but ARN model can provide a higher acceleration gain. On the other hand, our model requires fewer parameter numbers compared to Transformer baseline. This result shows that the original Transformer attention does have redundant computation, and simplifying it can achieve greater decoding efficiency. To further verify the effectiveness of our proposed approach, we conduct

| Model | $w/o$ KD | | $w/$ KD | |
|---|---|---|---|---|
| | BLEU | $\Delta_{\text{BLEU}}$ | BLEU | $\Delta_{\text{BLEU}}$ |
| Transformer (**teacher**) | 27.34 | 0.00 | 27.92 | 0.00 |
| AAN | 27.16 | -0.18 | 27.89 | -0.03 |
| SAN | 27.17 | -0.17 | 27.62 | -0.30 |
| ARN | 27.26 | -0.08 | 27.95 | +0.03 |

Table 6: ARN applying in knowledge distillation on WMT14 En→De translation task.

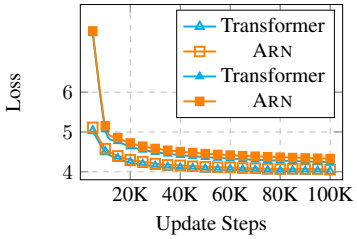



Figure 5: Convergence visualization. Loss vs. update steps on WMT14 En→De translation task (solid marked lines are training loss, hollow marked lines are validation loss).

experiments on ten WMT large-scale translation tasks. As shown in Table 4, the ARN significantly improves the speed for all these translation tasks, its speed-up is 1.35× faster on average. Also, the BLEU only drops 0.16, a very slight decline. Moreover, ARN achieve a stable speed-up, ranging from 31.02% to 40.87%. These results indicate that the ARN model is robust and can further improve the decoding efficiency on widely-range translation tasks.

In addition, we empirically compare the AAN and SAN models on WMT14 En→De translation task shown in Table 5. The three systems present a similar BLEU compared to Transformer baseline. Notably, ARN model achieves the highest speed-up with fewer parameter numbers. Although AAN, SAN and ARN can offer different degrees of decoding acceleration gains, they still suffer from slight performance degradation shown in Table 4 and Table 5. We use the most popular sequence-level knowledge distillation (KD) (Kim and Rush, 2016) for better performance and the Transformer baseline serves as our teacher model. As shown in Table 6, the KD method enables all three attention models get the performance improvements consistent with the Transformer baseline. Furthermore, the performance gap is close between ARN and Transformer baseline, and our model presents a strong generalization ability compared to AAN and SAN baselines.

| Model | BLEU | $\Delta_{\text{BLEU}}$ | Speed | $\Delta_{\text{Speed}}$ | #Param |
|---|---|---|---|---|---|
| Transformer | 27.34 | 0.00 | 110.55 | 0.00% | 58.7M |
| + Self-attention ARN | 27.39 | +0.05 | 117.45 | +6.24% | 56.7M |
| + Cross-attention ARN | 27.33 | -0.01 | 127.58 | +15.40% | 56.7M |
| + Both ARN | 27.27 | -0.07 | 131.87 | +19.29% | 54.7M |
| + Merging | 27.26 | -0.08 | 149.89 | +35.59% | 53.7M |

Table 7: Ablation study on WMT14 En→De translation task.

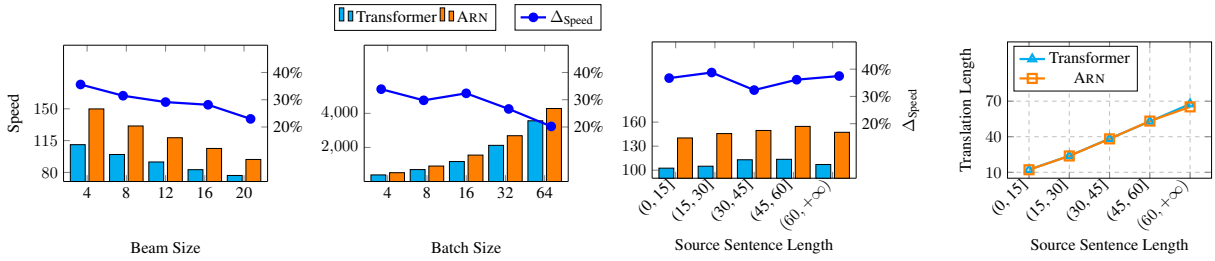



Figure 6: Speed (token/sec) and $\Delta_{\text{Speed}}$ [%] vs. beam size, batch size and source sentence length, translation length vs. source sentence length on WMT14 En→De translation task.

## 5 Analysis

**Analysis on Convergency.** As shown in Figure 5, we plot the loss curves of standard Transformer and ARN model on the training and validation sets, respectively. The two systems converge stably and both of their loss curves maintain a high degree of similarity. They also present similar performance in BLEU as shown in Table 3. All of these verify that the original attention in Transformer can be replaced with an ARN structure and there is no negative impact on model performance.

**Ablation Study.** To verify the acceleration contributions as well as the performance loss of different components in ARN model, we make an ablation study. As shown in Table 7, the ARN method adopted by the self-attention component achieves a speed-up of 6.24%, while 15.40% for cross-attention. This is because the original cross-attention operation is heavy for its long encoder representation ($V$) and applying the ARN method can bring greater decoding efficiency. When applying the ARN method to both self-attention and cross-attention components, the speed-up is 19.29%. It can achieve a higher acceleration gain (35.59%) when further merging self-attention and cross-attention components. Also, there is almost no effect on BLEU adopting the above different methods.

**Sensitivity Analysis on Speed.** We plot translation speed (in token/sec) and speed-up (in $\Delta_{\text{Speed}}$ [%]) as function of beam size, batch size and source sentence length. As shown in Figure 6, the ARN model can achieve relatively significant speed-up with the different beam sizes and batch sizes. However, the acceleration gains drop slightly with increasing beam size and batch size. This is because the speed-up will be weakened under the large matrix operation of the GPU. In addition, our model is robust and it gets the consistent improvement under the various source sentence length. Furthermore, both the Transformer baseline and our proposed model generate translations with similar lengths as shown on the right in Figure 6. This finding suggests that the acceleration gain comes from the well-designed model structure, not the translation length.

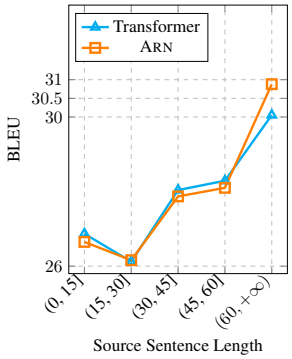



Figure 7: Different source sentence lengths vs. BLEU on WMT14 En→De translation task.

**Analysis on Translation Quality.** It is well-known that the NMT model is difficult to handle

| # | Model | BLEU | $\Delta_{\text{BLEU}}$ | Speed | $\Delta_{\text{Speed}}$ | #Param |
|---|---|---|---|---|---|---|
| 1 | Transformer (6 + 6) | 27.34 | - | 110.55 | - | 58.7M |
| 2 | Deep encoder, shallow decoder Transformer (12 + 2) | 27.38 | 0.00 | 224.48 | 0.00% | 60.7M |
| 3 | + Decoder self-attention ARN | 27.59 | +0.21 | 233.04 | +3.81% | 60.2M |
| 4 | + Decoder cross-attention ARN | 27.32 | -0.06 | 247.33 | +10.18% | 60.2M |
| 5 | + Decoder both ARN | 26.83 | -0.55 | 250.37 | +11.15% | 59.5M |
| 6 | + Merging | 26.82 | -0.56 | 260.73 | +16.15% | 59.4M |
| 7 | + Encoder ARN (using $\text{ARN}_2$) | 27.19 | -0.19 | 235.12 | +4.74% | 57.7M |

Table 8: ARN applying in deep encoder, shallow decoder Transformer on WMT14 En→De translation task.

long-distance dependencies and the issue of under-translation (Tu et al., 2016; Zheng et al., 2019) is prone to occur when translating long sentences. For this, we study the ARN model performance with the different input of sentence lengths as shown in Figure 7. Interestingly, the proposed model performs well when dealing with the longest sentences. Specifically, our model shows relatively poorly on shorter sentences but significantly better performance on longest sentences. This suggests that certain features have a great influence on model performance when translating long sentences, and the phenomenon indicates that ARN structure can better capture the long-distance dependency through the fusion of high-level and low-level features iteratively.

**Deep Encoder, Shallow Decoder Transformer.** Standard Transformer suffers from heavy inference cost due to the multi-layer stacked decoder (6-layer). A popular solution is to balance the encoder and decoder depths for speeding up Transformer (Kasai et al., 2020). We rebuild the Transformer with a deep encoder (12-layer) and a shallow decoder (2-layer) as a stronger baseline. As shown in Table 8, the balanced baseline (12/2) is more than 2× faster without loss in BLEU with similar parameter numbers. When applying ARN method to self-attention and cross-attention on the decoder side, it achieves 3.81% (line 3) and 10.18% (line 4) speed-up, respectively. Also, there is no negative impact on BLEU. The BLEU even increases by 0.21 for adopting ARN on the decoder-side self-attention. Interestingly, the BLEU drop significantly if applying ARN to both self-attention and cross-attention (line 5 and line 6). This is because the decoder is shallow with less parameter redundancy, which enables the model very sensitive when ARN is applied both to the two components. In contrast to the decoder, that encoder suffers from more severe redundancy in the balanced baseline, and the BLEU almost no drop with a higher reduction of parameters (line 7). To summarize, our ARN balanced baseline can achieve a speed-up of 2.24× without sacrificing performance compared to standard Transformer baseline (line 4 vs. line 1).

## 6 Related Work

Standard Transformer suffers from the high inference cost due to the auto-regressive generation schema and the heavy use of dot-product attention operations. A classic solution is to generate the entire target sequence at one time by using non-autoregressive inference method (Gu et al., 2017; Guo et al., 2019; Wang et al., 2019). This way offers high decoding efficiency but it is hard to train. Another representative solution adopts a local attention strategy (Kitaev et al., 2020; Beltagy et al., 2020) or simplifies attention structure (Katharopoulos et al., 2020), which can effectively reduce the computation of attention module. But these works are designed for acceleration of very long sequence tasks (e.g., image generation, automatic speech recognition, etc.). NMT inference acceleration methods have been investigated for years, including knowledge distillation (Hinton et al., 2015; Kim and Rush, 2016; Lin et al., 2020; Wang et al., 2021), vocabulary selection (L'Hostis et al., 2016; Sankaran et al., 2017; Shi and Knight, 2017), low-precision computation (Micikevicius et al., 2017; Quinn and Ballesteros, 2018; Aji and Heafield, 2020), kernel fusion (Wu et al., 2021), LayerDrop (Fan et al., 2019) and etc. Compared with the above methods, our work follows another line of work to learn a lightweight attention NMT model and prove its effectiveness.

Zhang et al. (2018) show that the self-attention network is not necessary and a simple averaging is enough. Compared with the above method, our network structure is simple and it is easy to implement. We improve both the self-attention and

cross-attention components. Xiao et al. (2019) observe that the most attention distributions are similar and thus share these distributions among layers. This method can be regarded as a structure pruning for Transformer, which may lead to model performance degradation. Our approach, which fuses low-level features to further enhance model confidence, is an improvement over this method. Li et al. (2021) propose the compressed attention network that simplifies the transformer architecture to achieve a higher parallelism. This method can be regarded as an equivalent transformation of the Transformer structure, it is orthogonal to our research.

ARN structure can be viewed as a weighted residual network, which acquires the attention results by reusing the features across layers. The main idea is similar to (He et al., 2020), but our ARN is different in both motivation and network structure. To our knowledge, we are the first to design a residual network for NMT decoding acceleration, proposing the combination of the features across layers for reconstructing the attention.

## 7 Conclusion

We have investigated ARN, an alternative lightweight attention structure for faster inference of Transformer. Experiments on a range of WMT translation tasks show that ARN offers a significant speed improvement over a strong Transformer baseline without sacrificing translation performance. Results on widely used WMT14 En→De machine translation tasks demonstrate that our model can simultaneously deliver superior acceleration and translation performance with fewer parameters, compared to previous work like AAN and SAN.

## Acknowledgements

We thank the anonymous reviewers for their constructive and thoughtful comments. Thanks to Tong Xiao, Yinqiao Li and Bei Li for their insightful suggestions.

## References

Alham Fikri Aji and Kenneth Heafield. 2020. Compressing neural machine translation models with 4-bit precision. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 35–42.

Maximiliana Behnke and Kenneth Heafield. 2020. Losing heads in the lottery: Pruning transformer attention in neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2664–2674.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Junliang Guo, Xu Tan, Di He, Tao Qin, Linli Xu, and Tie-Yan Liu. 2019. Non-autoregressive neural machine translation with enhanced decoder input. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3723–3730.

Ruining He, Anirudh Ravula, Bhargav Kanagal, and Joshua Ainslie. 2020. Realformer: Transformer likes residual attention. *arXiv preprint arXiv:2012.11747*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network (2015). *arXiv preprint arXiv:1503.02531*, 2.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. *arXiv preprint arXiv:2006.10369*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180.

Gurvan L'Hostis, David Grangier, and Michael Auli. 2016. Vocabulary selection strategies for neural machine translation. *arXiv preprint arXiv:1610.00072*.

Yanyang Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2021. An efficient transformer decoder with compressed sub-layers. *arXiv preprint arXiv:2101.00542*.

Ye Lin, Yanyang Li, Ziyang Wang, Bei Li, Quan Du, Tong Xiao, and Jingbo Zhu. 2020. Weight distillation: Transferring the knowledge in neural network parameters. *arXiv preprint arXiv:2009.09152*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740*.

Jerry Quinn and Miguel Ballesteros. 2018. Pieces of eight: 8-bit neural machine translation. *arXiv preprint arXiv:1804.05038*.

Baskaran Sankaran, Markus Freitag, and Yaser Al-Onaizan. 2017. Attention-based vocabulary selection for nmt decoding. *arXiv preprint arXiv:1706.03824*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Xing Shi and Kevin Knight. 2017. Speeding up neural machine translation decoding by shrinking run-time vocabulary. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 574–579.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Fusheng Wang, Jianhao Yan, Fandong Meng, and Jie Zhou. 2021. Selective knowledge distillation for neural machine translation. *arXiv preprint arXiv:2105.12967*.

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019. Non-autoregressive machine translation with auxiliary regularization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5377–5384.

Kaixin Wu, Bojie Hu, and Qi Ju. 2021. Tentrans high-performance inference toolkit for wmt2021 efficiency task. In *Proceedings of the Sixth Conference on Machine Translation*, pages 795–798.

Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. 2019. Sharing attention weights for fast transformer. *arXiv preprint arXiv:1906.11024*.

Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Accelerating neural transformer via an average attention network. *arXiv preprint arXiv:1805.00631*.

Zaixiang Zheng, Shujian Huang, Zhaopeng Tu, Xin-Yu Dai, and Jiajun Chen. 2019. Dynamic past and future for neural machine translation. *arXiv preprint arXiv:1904.09646*.