Deep Encoder, Shallow Decoder: Reevaluating the Speed-Quality Tradeoff in Machine Translation

Jungo Kasai♡[*] Nikolaos Pappas♡ Hao Peng♡
James Cross♣ Noah A. Smith♡♢

♡Paul G. Allen School of Computer Science & Engineering, University of Washington
♣Facebook AI ♢Allen Institute for AI
{jkasai,npappas,hapeng,nasmith}@cs.washington.edu
jcross@fb.com

Abstract

State-of-the-art neural machine translation models generate outputs *autoregressively*, where every step conditions on the previously generated tokens. This sequential nature causes inherent decoding latency. *Non-autoregressive* translation techniques, on the other hand, parallelize generation across positions and speed up inference at the expense of translation quality. Much recent effort has been devoted to non-autoregressive methods, aiming for a better balance between speed and quality. In this work, we re-examine the trade-off and argue that transformer-based autoregressive models can be substantially sped up without loss in accuracy. Specifically, we study autoregressive models with encoders and decoders of varied depths. Our extensive experiments show that given a sufficiently deep encoder, a *one-layer* autoregressive decoder yields state-of-the-art accuracy with comparable latency to strong non-autoregressive models. Our findings suggest that the latency disadvantage for autoregressive translation has been overestimated due to a suboptimal choice of layer allocation, and we provide a new speed-quality baseline for future research toward fast, accurate translation.

1 Introduction

Fast, accurate machine translation is a fundamental goal with a wide range of applications both in research and production. State-of-the-art neural machine translation systems generate translations *autoregressively* where words are predicted one-by-one conditioned on all previous words (Sutskever et al., 2014; Bahdanau et al., 2015; Vaswani et al., 2017). This sequential property causes inherent latency in inference since multiple tokens in each sentence cannot be generated in parallel. A flurry of recent work developed ways to (partially) parallelize the decoder with *non-autoregressive* machine translation (NAT, Gu et al., 2018), thereby speeding up decoding during inference. NAT tends to suffer in translation quality because parallel decoding requires conditional independence assumptions and prevents the model from properly capturing the highly multimodal distribution of target translations (Gu et al., 2018).

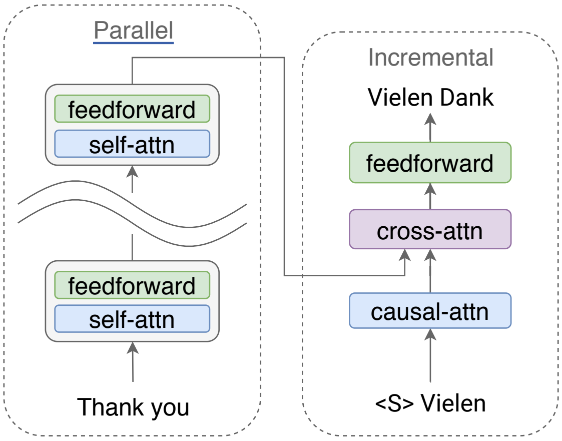

Figure 1: Deep encoder, shallow decoder.

Recent work proposed methods to mitigate this multimodality issue, including iterative refinement (e.g., Lee et al., 2018; Ghazvininejad et al., 2019; Gu et al., 2019b; Kasai et al., 2020) and modeling with latent variables (e.g., Ma et al., 2019; Shu et al., 2020). These approaches modify the decoder transformer to find a balance between decoding parallelism and translation quality. In this work, however, we adopt a contrasting strategy to the speed-quality trade-off. The standard transformer for machine translation is typically assumed to have the same number of encoding and decoding layers (Vaswani et al., 2017). Observing that the encoder transformer is inherently parallel, we place most of the model capacity in the encoder while keeping the decoder minimal, to accelerate inference. A resulting autoregressive transformer with a deep

[*] Work partially done at Facebook AI.

encoder and a shallow decoder (Fig. 1) achieves a substantial latency improvement over the standard transformer configuration, without sacrificing performance.

We provide extensive speed-quality comparisons between iterative NAT models and autoregressive models with varying numbers of encoder and decoder layers. In particular, we use two types of latency measures for translation and discuss their relation to computational complexity. The two measures reflect two possible scenarios in application by feeding one sentence at a time or as many words as possible into the GPU memory. The first scenario is designed to simulate, for example, instantaneous machine translation that translates text (or even speech) input from users. This is where current NAT models shine – we can make full use of parallelism across decoding positions in a GPU. For this reason, much prior work in NAT only measures latency using this metric (Gu et al., 2018, 2019b; Kasai et al., 2020; Li et al., 2020). The second scenario aims at a situation where we want to translate a large amount of text as quickly as possible. In this case, we see that autoregressive models run faster than NAT models by a large margin. Computation at each time step is large enough to exploit parallelism in a GPU, which cancels out the benefit from parallel NAT decoding. Further, autoregressive models can reduce latency by caching all hidden states from the previous positions (Ott et al., 2019) and computing each step in linear complexity with respect to the sequence length. NAT models necessitate a fresh run of quadratic self and cross attention in every decoding iteration.

Interestingly, if we apply the layer allocation strategy of deep encoder and shallow decoder to NAT models, we fail to retain the original translation quality from 6 layers each (§5.1). This suggests that departure from autoregressive decoding necessitates more computational capacity in the decoder side, and our strategy is effective specifically for autoregressive models. Our analysis demonstrates that the decoder in NAT models requires more capacity because it needs to learn to reorder words for the target (§6). Since the configuration of deep encoder and shallow decoder is specifically effective for autoregressive models, we need to re-establish where autoregressive transformers sit in the spectrum of the speed-quality trade-off for future work in fast, accurate machine translation.

2 Transformer and Parallelism

The transformer architecture (Vaswani et al., 2017) differs from recurrent neural networks such as LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Cho et al., 2014) in its parallel structure. Here we review the architecture and discuss its implications for fast machine translation.

2.1 Architecture

An autoregressive transformer (Vaswani et al., 2017) consists of an encoder and a decoder. Each encoder layer takes as input a sequence of vectors $\mathbf{X}_{\texttt{in}}$, and outputs $\mathbf{X}_{\texttt{out}}$:[1]

$$\begin{aligned}\mathbf{X}_{\texttt{self}} &= \text{self-attention}(\mathbf{X}_{\texttt{in}}) + \mathbf{X}_{\texttt{in}}\\ \mathbf{X}_{\texttt{out}} &= \text{feed-forward}(\mathbf{X}_{\texttt{self}}) + \mathbf{X}_{\texttt{self}}\end{aligned}$$

A decoder layer takes as input a sequence of vectors $\mathbf{Y}_{\texttt{in}}$ and encoded source tokens $\mathbf{X}_{\texttt{src}}$ from the final encoder layer:

$$\begin{aligned}\mathbf{Y}_{\texttt{self}} &= \text{causal-attention}(\mathbf{Y}_{\texttt{in}}) + \mathbf{Y}_{\texttt{in}}\\ \mathbf{Y}_{\texttt{cross}} &= \text{cross-attention}(\mathbf{Y}_{\texttt{self}}, \mathbf{X}_{\texttt{src}}) + \mathbf{Y}_{\texttt{self}}\\ \mathbf{Y}_{\texttt{out}} &= \text{feed-forward}(\mathbf{Y}_{\texttt{cross}}) + \mathbf{Y}_{\texttt{cross}}\end{aligned}$$

Here causal-attention denotes a variant of self attention that only attends to the prefix (i.e., $\mathbf{Y}_{\texttt{self},i}$ only attends to $\mathbf{Y}_{\texttt{in},\leq i}$). During training one can parallelize computation across positions both in the encoder and decoder, resulting in linear complexity in sequence length. At inference time, the decoder generates outputs sequentially, and thus computation cannot be parallelized over positions. This sequential nature of autoregressive decoding causes inherent latency, with complexity quadratic in sequence length.

2.2 Deep Encoder, Shallow Decoder

Since its first proposal (Vaswani et al., 2017), much prior work has assumed that the transformer architecture in machine translation has the same numbers of encoder and decoder layers, including top-performing systems in recent WMT competitions (Edunov et al., 2018; Pinnis et al., 2018; Ng et al., 2019). We challenge this convention and explore pairing deep encoders with a shallow decoder. As we will show in later experiments, this *deep-shallow* configuration retains translation accuracy, but can substatially reduce decoding time.

[1] Layer normalization (Ba et al., 2016) is applied after attention and feed forward. We suppress this for brevity.

This is because at inference time, the encoder only accounts for a minor part of the latency overhead since its computation can be easily parallelized over input positions; on the other hand, the speedup gains from a lightweight decoder are substantial. Several prior works explored the use of deep encoders and shallow decoders to improve translation accuracy (Barone et al., 2017; Wang et al., 2019a). Here, we study the impact of such architectures from the perspective of a speed-quality trade-off.

3 Latency in Machine Translation

In this section, we present two types of latency for machine translation to target two different scenarios in application: S_1 and $S_{\max}$. We then discuss complexity differences between autoregressive translation (AT) and non-autoregressive translation (NAT) models and how their computational complexity affects their S_1 and $S_{\max}$ latency. Our analysis shows that under the same layer configuration, NAT models improve S_1 over AT models by parallelizing the decoder computation. A *deep-shallow* AT model reduces the complexity from the decoder's sequential computation, and achieves competitive S_1 to those NAT models.

3.1 Latency Measures

We use two translation latency metrics:

- **S_1** measures the speed to translate one sentence at a time. It aligns with applications like instantaneous machine translation that translates text input from users immediately.
- **$S_{\max}$** measures the speed to translate in mini-batches as large as the hardware allows. This is closer to the scenarios where one wants to translate a large amount of text.

Both metrics measure wall-clock time speedups relative to an AT baseline with a 6-layer encoder and decoder, following prior work (Gu et al., 2018; Li et al., 2020).

3.2 Complexity Analysis

Seen in Table 1 is a complexity analysis of different types of transformer layers and full translation models. Assume that the source and target lengths are both N for simplicity. T denotes the number of iterations in an iterative NAT method where $T < N$. We use incremental decoding for AT models (Ott et al., 2019) where the model states from previously generated tokens are cached and reused. In this case, the total complexity in one AT decoder layer will be $\mathcal{O}(N^2)$. NAT decoding with T iterative steps will have cross and self attention of quadratic complexity. Since iterative NAT models run fresh transformer passes in each iteration (Lee et al., 2018; Ghazvininejad et al., 2019, 2020b; Kasai et al., 2020; Saharia et al., 2020), we will have complexity of $\mathcal{O}(N^2T)$ per layer. Some of these operations can be parallelized over N target positions on a GPU, resulting in reduction in time complexity (Harris, 2007, column "w/ parallelization"). Assuming the parallelization over all N positions, each encoder layer only costs $\mathcal{O}(N)$. Similarly, one NAT decoder layer with T iterations can be computed in $\mathcal{O}(NT)$. The AT decoder layer still costs $\mathcal{O}(N^2)$ due to its sequential nature.

	Total Complexity	w/ parallelization
By Layer		
Enc. Layer	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$
AT Dec. Layer	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
NAT Dec. Layer	$\mathcal{O}(N^2T)$	$\mathcal{O}(NT)$
Full Model		
AT Enc-E Dec-D	$\mathcal{O}(N^2(E+D))$	$\mathcal{O}(N(E+ND))$
AT Enc-E Dec-1	$\mathcal{O}(N^2E)$	$\mathcal{O}(N(E+N))$
NAT Enc-E Dec-D	$\mathcal{O}(N^2(E+DT))$	$\mathcal{O}(N(E+DT))$

Table 1: Complexity analysis of transformers. N: source/target length; T: # NAT iterations.

S_1 is dominated by the complexity after parallel reduction; a GPU typically has enough memory to parallelize all operations across N target positions in a NAT decoder layer. This means that a NAT model with an E-layer encoder and D-layer decoder has an advantage over an AT model with the same layer configuration because $T < N$ and $\mathcal{O}(N(E + DT)) < \mathcal{O}(N(E + ND))$. However, NAT and AT models have similar complexity when the AT model only uses one decoder layer ($\mathcal{O}(N(E + DT))$ vs. $\mathcal{O}(N(E + N))$). This results in comparable S_1 latency between NAT and deep-shallow AT models. In the case of $S_{\max}$, total complexity without parallelization is also at stake since an AT decoder can make crucial use of a GPU by simply parallelizing over the batch instances and offsets NAT's benefit. We observe that NAT costs much more total complexity than AT because of the T factor from the decoder: $\mathcal{O}(N^2(E + DT))$. Indeed we will see in a later section that NAT models yield much slower $S_{\max}$ than AT models.

4 Experiments

We conduct extensive experiments on standard benchmark datasets of varying sizes. We compare latency across non-autoregressive and autoregressive models and show that autoregressive models with a deep encoder and shallow decoder provide a substantially better balance between speed and quality than standard autoregressive models with the encoder and decoder of equal total depth.

4.1 Baselines and Comparison

Prior work has proposed various approaches to non-autoregressive machine translation (NAT). These methods must seek a balance in the speed-quality trade-off: the more parallelization is introduced into a model, the more the output quality deteriorates because of a stronger conditional independence assumption. Some approaches require external models to achieve competitive accuracy such as candidate rescoring with an autoregressive model (Sun et al., 2019; Li et al., 2020) and a reordering module to align input word order to the target (Ran et al., 2019). Given this complication in much recent work, latency comparisons among NAT models present challenges. In this work, we focus on comparisons with iteration-based approaches because they perform competitively to autoregressive models without any external system. Specifically, we use two strong iteration-based NAT models from recent work (Ghazvininejad et al., 2019; Kasai et al., 2020). See §7 for descriptions of more prior work on NAT.

CMLM The conditional masked language model (Ghazvininejad et al., 2019) predicts randomly masked target tokens given observed target tokens as well as the source, similar to masked language models for contextual word representations (Devlin et al., 2019; Liu et al., 2019). CMLM is used for iterative NAT by the mask-predict inference. Following Ghazvininejad et al. (2019, 2020b), we use 4 and 10 iterations and length beam 5 where 5 most probable lengths are chosen and each of those candidates is decoded in parallel until we select the one with the best score at the end.

DisCo The disentangled context transformer (Kasai et al., 2020) is an efficient alternative to CMLM. DisCo predicts every target token given an arbitrary subset of the rest of the reference tokens. Following Kasai et al. (2020), we use their parallel easy-first inference, and set the maximum number of iterations and length beam to be 10 and 5 respectively.

Distillation Following previous work on non-autoregressive translation (e.g., Ghazvininejad et al., 2019; Kasai et al., 2020; Saharia et al., 2020), we apply sequence-level knowledge distillation (Kim and Rush, 2016) by training every model in all directions on translations produced by a standard left-to-right transformer model (transformer large for EN-DE, EN-ZH, EN-FR and base for EN-RO). We assess the impact of distillation in §6 and demonstrate that distillation is important, especially for non-autoregressive models. Notice that we apply distillation to all configurations, including autoregressive models, for fair comparisons.[2]

4.2 Experimental Setup

We experiment with 7 translation directions from four datasets of various training data sizes: WMT14 EN-DE (4.5M pairs), WMT16 EN-RO (610K), WMT17 EN-ZH (20M), and WMT14 EN-FR (36M, EN→FR only). These datasets are all encoded into subwords by BPE (Sennrich et al., 2016).We follow the preprocessing and data splits by previous work (EN-DE: Vaswani et al., 2017; EN-RO: Lee et al., 2018; EN-ZH: Hassan et al., 2018; Wu et al., 2019; EN-FR: Gehring et al., 2017). We evaluate performance with BLEU (Papineni et al., 2002) for all directions, except that we use SacreBLEU (Post, 2018) for EN→ZH following a previous protocol (Ghazvininejad et al., 2019, 2020b; Kasai et al., 2020).[3] For all autoregressive models, we apply beam search decoding with beam size 5 and length penalty 1.0. S_1 and $S_{\max}$ wall-clock time speedups (§3) for all models are evaluated on the same single Nvidia V100 GPU with 16GB memory, with CUDA 10.1, cuDNN 7.6.3, and PyTorch version 1.4.0 (Paszke et al., 2019). We apply half-precision inference (Ott et al., 2019), and found it speeds up $S_{\max}$ for non-autoregressive models by 30+%, but not S_1, in line with previous observations (Kim et al., 2019).

Hyperparameters We generally follow the hyperparameters of the base sized transformer (Vaswani et al., 2017): 8 attention heads, 512 model dimensions, and 2048 hidden dimensions for both the encoder and decoder. The dropout rate is tuned

[2] Several works in the NAT literature only apply distillation to NAT models, which undermines comparability.

[3] SacreBLEU hash: BLEU+case.mixed+lang.en-zh+numrefs.1+smooth.exp+test.wmt17+tok.zh+version.1.3.7.

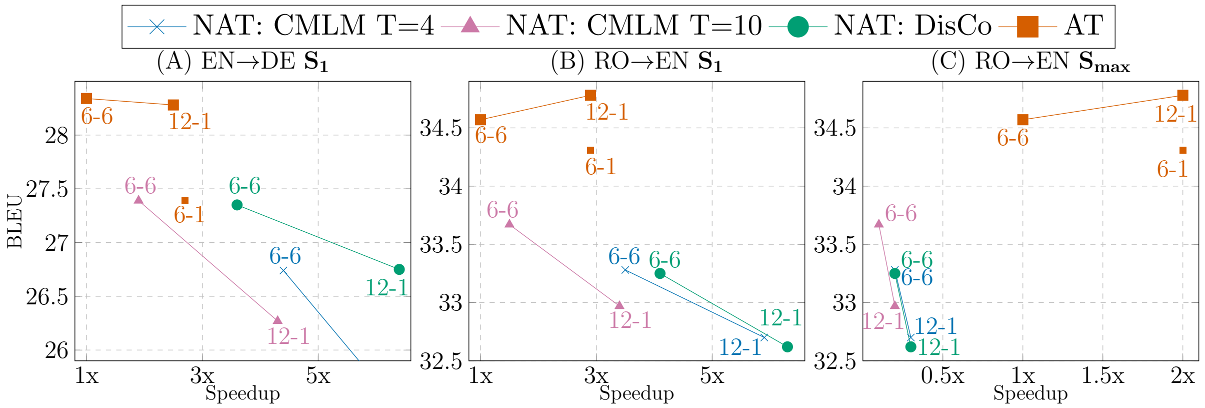

Figure 2: BLEU and speed comparisons with varying numbers of encoder and decoder layers on the test data. 12-1 denotes 12 and 1 encoder and decoder layers respectively. S_{max} for EN→DE has similar patterns to RO→EN S_{max} in (C). AT deep-shallow (12-1) finds a balanced middle ground in the trade-off. See the appendix for more results.

from $[0.1, 0.2, 0.3]$ based on development BLEU performance. We apply weight decay with 0.01 and label smoothing with $\varepsilon = 0.1$. We train with a batch size of approximately 65K tokens, using Adam (Kingma and Ba, 2015) with $\beta = (0.9, 0.98)$ and $\varepsilon = 10^{-6}$. The EN→FR model is trained for 500K updates, while others for 300K (Kasai et al., 2020). Dev. BLEU is measured at the end of each epoch, and we average the 5 best checkpoints to obtain the final model (Vaswani et al., 2017). We use mixed precision training (Micikevicius et al., 2018), and implement all models with `fairseq` (Ott et al., 2019). Further details are described in the appendix.

5 Results and Discussion

We provide in-depth results comparing performance and speedup across autoreogressive and non-autoregressive models.

5.1 Deep Encoder, Shallow Decoder

Fig. 2 shows translation speed-quality trade-off curves of CMLM, DisCo, and AT models on EN→DE and RO→EN test data. For each model we plot the results of configurations with varying encoder and decoder depths. For brevity, we denote by E-D a model with an E-layer encoder and a D-layer decoder. All speedups are measured with respect to the AT 6-6 baseline (§3).

Firstly, under the 6-6 configuration, the AT model outperforms both CMLM and DisCo by a considerable margin in BLEU, but it achieves the slowest S_1. Using a single-layer decoder, AT 6-1 gains a substantial S_1 speedup (2.6x for EN→DE and 2.9x for RO→EN), but this comes at a cost of BLEU: 28.34 vs. 27.39 for EN→DE, and 34.57 vs. 34.31 for RO→EN. AT 12-1 lands on a balanced middle ground: it yields similar BLEU to AT 6-6, but its S_1 is more than 2.5 times faster. Notably, AT 12-1 achieves even faster S_1 than that of CMLM 6-6 model with 10 iterations. In contrast, 12-1 NAT models generally suffer in BLEU compared to the 6-6 configuration; e.g., 26.75 (DisCo 12-1) vs. 27.35 (DisCo 6-6) in EN→DE.

Model		**E-D**	**BLEU**	$\mathbf{S_1}$	$\mathbf{S_{max}}$
		WMT17 EN→ZH			
CMLM	T=4	6-6	33.58	**3.5x**	0.2x
CMLM	T=10	6-6	34.24	1.5x	0.1x
DisCo		6-6	34.63	2.5x	0.2x
AT Deep-Shallow		12-1	34.71	2.7x	**1.7x**
AT		6-6	**35.06**	1.0x	1.0x
		WMT17 ZH→EN			
CMLM	T=4	6-6	22.56	**3.8x**	0.2x
CMLM	T=10	6-6	23.76	1.7x	0.1x
DisCo		6-6	23.83	2.6x	0.2x
AT Deep-Shallow		12-1	**24.22**	2.9x	**1.8x**
AT		6-6	24.19	1.0x	1.0x
		WMT14 EN→FR			
CMLM	T=4	6-6	40.21	**3.8x**	0.2x
CMLM	T=10	6-6	40.55	1.7x	0.1x
DisCo		6-6	40.60	3.6x	0.2x
AT Deep-Shallow		12-1	**42.04**	2.8x	**1.9x**
AT		6-6	41.98	1.0x	1.0x

Table 2: BLEU and speed comparisons with varying numbers of encoder (E) and decoder (D) layers on large bitext. Best performance is bolded.

	WMT14 EN–DE				WMT16 EN–RO				WMT17 EN–ZH			
Models	→DE	T	→EN	T	→RO	T	→EN	T	→ZH	T	→EN	T
CMLM	25.9	4	29.9	4	32.5	4	33.2	4	32.6	4	21.9	4
	27.0	10	31.0	10	33.1	10	33.3	10	33.2	10	23.2	10
Lev. Transformer	27.3	>7	–	–	–	–	33.3	>7	–	–	–	–
DisCo	27.3	4.8	31.3	4.2	33.2	3.3	33.2	3.1	34.6	5.4	23.8	5.9
SMART	27.0	4	30.9	4	–	–	–	–	33.4	4	22.6	4
	27.6	10	31.3	10	–	–	–	–	34.1	10	23.8	10
Imputer	28.0	4	31.0	4	34.3	4	34.1	4	–	–	–	–
	28.2	8	31.3	8	34.4	8	34.1	8	–	–	–	–
AT Enc6-Dec6	**28.3**	N	**31.8**	N	**34.6**	N	34.6	N	**35.1**	N	**24.2**	N
AT Deep-Shallow	**28.3**	N	**31.8**	N	33.8	N	**34.8**	N	34.7	N	**24.2**	N

Table 3: BLEU comparisons with iterative NAT methods. T indicates the average # iterations. CMLM: Ghazvininejad et al. (2019); Lev. Transformer: Gu et al. (2019b); DisCo: Kasai et al. (2020); SMART: Ghazvininejad et al. (2020b); Imputer: Saharia et al. (2020). Best performance is bolded.

Interestingly, all NAT models achieve slower $S_{\max}$ than the AT 6-6 baseline: DisCo 6-6: 0.3x; CMLM 6-6 T=10: 0.1x in RO→EN. This is consistent with our complexity analysis in §3.2, where we found that with the same layer allocation, iterative NAT models need more total computation than the AT counterpart. AT 12-1 still gains a considerable speedup over AT 6-6 (2.0x in EN→RO). These results suggest that current NAT models have little advantage when translating a large amount of text, and one should clarify this distinction when discussing translation latency. See the appendix for full results from all four directions.

Table 2 presents results from large bitext, EN↔ZH and EN→FR. We observe similar trends: AT deep-shallow achieves similar BLEU to AT 6-6 while reducing both S_1 and $S_{\max}$ latency substantially. For EN↔ZH, AT deep-shallow has a more S_1 speedup than DisCo (2.7x vs. 2.5x in EN→ZH, 2.9 vs. 2.6 in ZH→EN). Particularly noteworthy is its performance in EN→FR: 42.04 BLEU, a 1.4 point improvement over the best NAT model. These results illustrate that the strategy of having a deep encoder and shallow decoder remains effective in large-scale bitext when the model has to learn potentially more complex distributions from more samples.

Lastly, Table 3 compares AT deep-shallow to recent iteration-based NAT results. All NAT models use the 6-6 configuration with the base size (Vaswani et al., 2017) except that Imputer (Saharia et al., 2020) uses 12 self-attention layers over the concatenated source and target. Overall, our AT deep-shallow models outperform all NAT models. The one exception is EN→RO where Imputer achieves 34.4 points with 8 iterations compared to our 33.8 points. We note, however, latency overhead in each iteration of their model is strictly larger than that of CMLM or DisCo since every iteration involves a fresh run of 12-layer self attention over a concatenation of input and output sequences. As we saw in Fig. 2, AT deep-shallow yields comparable S_1 to CMLM 6-6 with 4 iterations, which would be more than twice as fast as Imputer with 8 iterations.

5.2 Constrained Views

In this section, we present two controlled experiments to compare NAT and autoregressive models more thoroughly.

S_1 Latency Constraint From §5.1 we see that compared to NAT models, AT deep-shallow yields a better translation speed-quality balance—despite being slightly slower in S_1 on some of the datasets, it achieves better BLEU across the board. To confirm this result, we further compare AT deep-shallow against two NAT models, controlling for S_1 latency. More specifically, we experiment with NAT models of varying encoder depths, and pair each with as many decoder layers as possible until it reaches AT 12-1's S_1 latency. Fig. 3 shows the results. For CMLM T=4, CMLM T=10, and DisCo, the best configurations of 12-layer encoders were paired up with 12, 4, and 9 decoder layers respec-

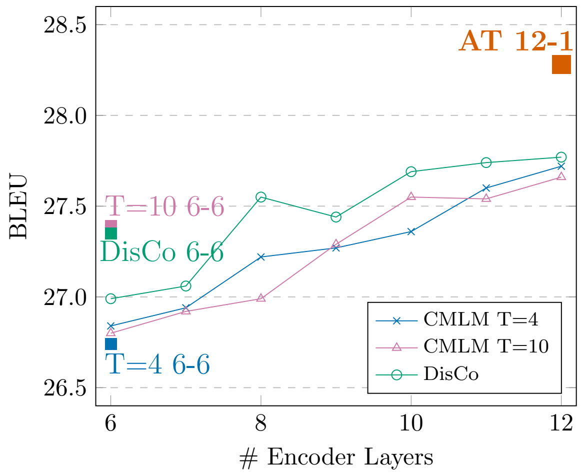

Figure 3: WMT14 EN→DE test results over varying depths of the encoder under the S_1 latency constraint of AT 12-1 ■.

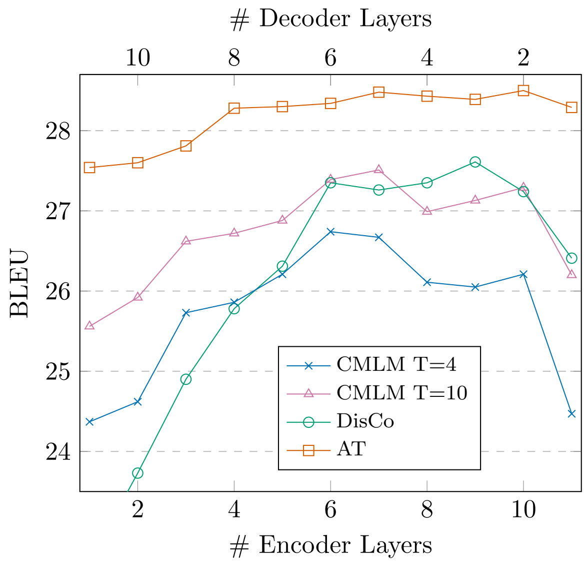

Figure 4: WMT14 EN→DE test results over varying allocation of a total of 12 transformer layers.

tively. All NAT models improve performance as the encoder becomes deeper and surpass the scores of the 6-6 baselines (shown as squares along $x = 6$). Nonetheless, there is still a large performance drop from AT 12-1. This illustrates that the two NAT models are not able to match AT deep-shallow's accuracy under the same S_1 latency budget.

Layer Constraint We can speed up autoregressive translation (AT) by developing a model with a deep encoder and a one-layer decoder. Here we thoroughly compare layer allocation strategies. Shown in Fig. 4 are results of NAT and AT methods under the constraint of 12 transformer layers in total. NAT models perform well when the decoder and encoder are balanced with slight tendency to deep encoders. On the other hand, the AT models perform consistently with 4 or more encoder layers. This confirms that using deep encoders and shallow decoders is more effective in AT models than in NAT ones. Note that the number of parameters in each layer allocation differs since a decoder layer contains 30% more parameters than an encoder layer, due to cross attention (§2.1).

6 Further Analysis

Decoder Depth and Reordering Words From earlier results we see that NAT models need deeper decoders than AT models to perform well. We hypothesize that one reason is that NAT decoders need to learn to adjust to diverging word order between the source and the target: an AT decoder takes as input all preceding tokens and explicitly learns conditional distribution, while a NAT decoder needs to learn target word ordering from scratch.

To test this hypothesis, we conduct the following controlled experiment in EN→DE translation. We first run the `fast_align` tool (Dyer et al., 2013)[4] on all bitext data (including the test set), and disable the NULL word feature to ensure that every English word is aligned to exactly one German word. We then shuffle the English words according to the order of their aligned German words. When multiple English words are aligned to the same German word, we keep the original English order. We apply the same BPE operations as the original data. Table 4 compares performance on the original and reordered data. AT gains the same improvement regardless of the layer configuration; in contrast, 12-1 NAT benefits more than NAT 6-6. This result supports our hypothesis that word reordering is one reason why NAT models need a deeper decoder. The overall improvements from reordering are consistent with Ran et al. (2019), who found that a NAT model benefits from reordering the source to match the target.

Effects of Distillation We applied sequence-level knowledge distillation (Kim and Rush, 2016) to all models. Here we analyze its effects over the WMT14 EN→DE evaluation data (Table 5). An autoregressive transformer large model (Vaswani et al., 2017) is used as the teacher model. All models benefit from knowledge distillation as indicated by positive Δ, including the AT models. Several

[4] https://github.com/clab/fast_align

Model	Orig.	Reorder	Δ
CMLM Enc-6 Dec-6	27.4	31.7	4.3
CMLM Enc-12 Dec-1	26.3	31.0	4.7
DisCo Enc-6 Dec-6	27.4	31.0	3.6
DisCo Enc-12 Dec-1	26.8	31.6	**4.8**
AT Enc-6 Dec-6	**28.3**	**32.6**	4.3
AT Deep-Shallow (12-1)	**28.3**	**32.6**	4.3

Table 4: WMT14 EN→DE test results using reordered English input. T = 10 for the CMLM models.

recent works only compare NAT models trained with knowledge distillation to AT models trained *without*. Our finding shows that that AT models with knowledge distillation can be an additional baseline for future NAT research. AT deep-shallow deteriorates much less on the raw data compared to the iterative NAT methods, suggesting that our strategy of speeding up autoregressive models is better suited to modeling raw, complex data than the NAT methods.

Model	Raw	Dist.	Δ
CMLM, T = 4	22.3	25.9	**3.6**
CMLM, T = 10	24.6	27.0	2.4
Imputer, T = 4	24.7	27.9	3.2
Imputer, T = 8	25.0	27.9	2.9
DisCo Enc-6 Dec-6	24.8	27.4	2.6
AT Deep-Shallow (12-1)	26.9	**28.3**	1.4
AT Enc-6 Dec-6	**27.4**	**28.3**	0.9

Table 5: WMT14 EN→DE test results in BLEU that analyze the effects of distillation in fast translation methods. All distillation data are obtained from a transformer large. T denotes the number of iterations.

Speedup and Batch Size When decoding with large mini-batches, NAT models can be slower than their AT counterpart (§5.1). Here we further study this effect. Fig. 5 plots the relative speedups of different models' decoding with varying numbers of sentences per batch up to the hardware limit ("max," §3.1). The speedup by NAT models diminishes as the batch size grows: they have similar decoding latency to AT 6-6 with batch size 50, and become slower with larger batch sizes. In contrast, AT deep-shallow achieves consistent speedups over the AT 6-6 baseline.

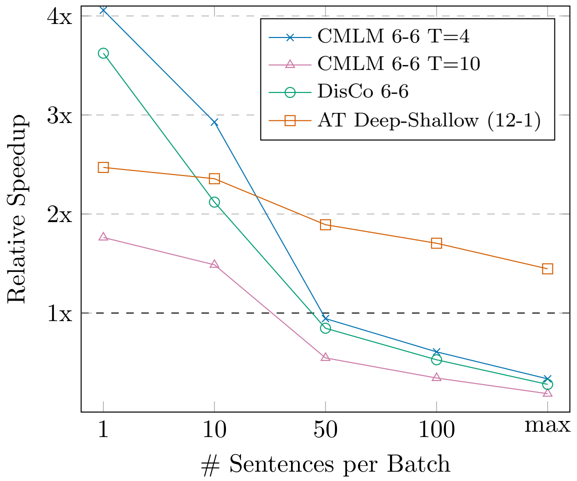

Figure 5: Relative speedup compared to the standard AT Enc-6 Dec-6 with varying batch size. Evaluated on the WMT14 EN→DE test data.

Can we reduce the decoder further? We saw that an autoregressive model with a single-layer decoder and a sufficiently deep encoder can retain the accuracy of the baseline with 6 layers each. One may ask whether we can make the decoder even more compact. Our preliminary experiments showed that we can remove the feed-forward module from the decoder (Fig. 1) without hurting performance. This reduces the S_1 latency by 10%. We leave further exploration to future work.

7 Further Related Work

Non-autoregressive Translation In addition to the work already discussed, several other works proposed to iteratively refine (or insert) output predictions (Mansimov et al., 2019; Stern et al., 2019; Gu et al., 2019a; Chan et al., 2019a,b; Li et al., 2020). Other approaches include adding a light autoregressive module to parallel decoding (Kaiser et al., 2018; Sun et al., 2019; Ran et al., 2019), partially decoding autoregressively (Stern et al., 2018, 2019), rescoring output candidates autoregressively (e.g., Gu et al., 2018), mimicking hidden states of an autoregressive teacher (Li et al., 2019), training with different objectives than vanilla cross-entropy (Libovický and Helcl, 2018; Wang et al., 2019b; Shao et al., 2020; Tu et al., 2020; Saharia et al., 2020; Ghazvininejad et al., 2020a), reordering input sentences (Ran et al., 2019), training on additional data from an autoregressive model (Zhou and Keung, 2020), and modeling with latent variables (Ma et al., 2019; Shu et al., 2020). The approach of adding a light autoregressive module is closest to our method, but note that we pack all *non-autoregressive* computation into the encoder.

Optimizing Autoregressive Transformer Prior work has suggested ways to optimize autoregressive transformers for fast inference. For example, Kim et al. (2019) employed layer tying (Dabre and Fujita, 2019; Dehghani et al., 2019) on the transformer decoder and found that it sped up inference on CPUs, but not on a GPU. Shi and Knight (2017) proposed a vocabulary reduction method to speed up the last softmax computation. Zhang et al. (2018) used dynamic programming in an average attention network to accelerate inference. Press and Smith (2018) proposed an eager translation method to avoid attention computation. Reformer (Kitaev et al., 2020) reduced the quadratic complexity of attention computation by locality-sensitive hashing. Some of these methods can be used orthogonally to further facilitate fast inference in a transformer with a deep encoder and shallow decoder.

Rich Encoding, Light Decoding Our experiments suggest that rich features from a deep encoder avoid the need for multiple layers of decoding in machine translation. Wang et al. (2019a) showed that using more encoder transformer layers while keeping 6 decoder layers improves translation quality. Barone et al. (2017) found that RNN-based models with a deep encoder and a shallow decoder can reduce training time with a small performance drop. We took an extreme configuration of a single-layer transformer decoder and focused on inference latency, but all of these results corroborate the benefit of deep encoders. Beyond machine translation, a surprisingly light *decoder* (e.g., multilayer perceptrons) with a powerful *encoder* (e.g., bidirectional LSTMs) has proven successful in structured prediction, such as syntactic and semantic parsing (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017, 2018; Kasai et al., 2018). Generating a target translation is perhaps a more complex task than producing a parse tree, but our results provide further support for the claim that useful distributed representations of natural language can be obtained in a conditionally independent manner.

8 Conclusion and Future Work

We presented extensive empirical studies to demonstrate that autoregressive translation can be dramtically sped up by a simple layer allocation strategy: **deep encoder, shallow decoder**. Compared to strong non-autoregressive models, deep-shallow autoregressive models achieve substantial improvement in translation quality with comparable latency. Our results suggest that layer allocation is an important factor that future work on fast machine translation, particularly non-autoregressive machine translation, should take into consideration. More generally, our work suggests that a better layer allocation between the encoder and decoder might be able to accelerate inference in any sequence-to-sequence task. In particular, a model with a deep encoder and a shallow decoder can be used for large-scale pretraining for sequence generation such as BART (Lewis et al., 2020; Liu et al., 2020) where latency reduction will be key in a wide range of real-world applications.

Acknowledgments

We thank Luke Zettlemoyer, Ofir Press, and Tim Dettmers for their helpful feedback and discussions on this work.

References

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

Antonio Valerio Miceli Barone, Jindřich Helcl, Rico Sennrich, Barry Haddow, and Alexandra Birch. 2017. Deep architectures for neural machine translation. In *Proc. of WMT*.

William Chan, Nikita Kitaev, Kelvin Guu, Mitchell Stern, and Jakob Uszkoreit. 2019a. KERMIT: Generative insertion-based modeling for sequences.

William Chan, Mitchell Stern, Jamie Ryan Kiros, and Jakob Uszkoreit. 2019b. An empirical study of generation order for machine translation.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proc. of SSST-8*.

Raj Dabre and Atsushi Fujita. 2019. Recurrent stacking of layers for compact neural machine translation models. In *Proc. of AAAI*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz. 2019. Universal transformers. In *Proc. of ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL*.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proc. of ICLR*.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proc. of ACL*.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proc. of NAACL*.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proc. of EMNLP*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *Proc. of ICML*.

Marjan Ghazvininejad, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. 2020a. Aligned cross entropy for non-autoregressive machine translation.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke S. Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proc. of EMNLP*.

Marjan Ghazvininejad, Omer Levy, and Luke Zettlemoyer. 2020b. Semi-autoregressive training improves mask-predict decoding.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *Proc. of ICLR*.

Jiatao Gu, Qi Liu, and Kyunghyun Cho. 2019a. Insertion-based decoding with automatically inferred generation order. *TACL*.

Jiatao Gu, Changhan Wang, and Jake Zhao. 2019b. Levenshtein transformer. In *Proc. of NeurIPS*.

Mark Harris. 2007. Optimizing parallel reduction in CUDA.

Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mengnan Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. 2018. Achieving human parity on automatic Chinese to English news translation.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *Proc. of ICLR*.

Łukasz Kaiser, Aurko Roy, Ashish Vaswani, Niki Parmar, Samy Bengio, Jakob Uszkoreit, and Noam Shazeer. 2018. Fast decoding in sequence models using discrete latent variables. In *Proc. of ICML*.

Jungo Kasai, James Cross, Marjan Ghazvininejad, and Jiatao Gu. 2020. Parallel machine translation with disentangled context transformer. In *Proc. of ICML*.

Jungo Kasai, Robert Frank, Pauli Xu, William Merrill, and Owen Rambow. 2018. End-to-end graph-based TAG parsing with neural networks. In *Proc. of NAACL*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proc. of EMNLP*.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proc. of WNGT*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proc. of ICLR*.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proc. of ACL*.

Xiaoya Li, Yuxian Meng, Arianna Yuan, Fei Wu, and Jiwei Li. 2020. LAVA NAT: A non-autoregressive translation model with look-around decoding and vocabulary attention.

Zhuohan Li, Zi Lin, Di He, Fei Tian, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Hint-based training for non-autoregressive machine translation. In *Proc. of EMNLP*.

Jindrich Libovický and Jindrich Helcl. 2018. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proc. of EMNLP*.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard H. Hovy. 2019. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In *Proc. of EMNLP*.

Elman Mansimov, Alex Wang, and Kyunghyun Cho. 2019. A generalized framework of sequence generation with application to undirected sequence models.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. In *Proc. of ICLR*.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 news translation task submission. In *Proc. of WMT*.

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 *NAACL Demonstrations*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proc. of WMT*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proc. of NeurIPS*.

Mārcis Pinnis, Matīss Rikters, and Rihards Krišlauks. 2018. Tilde's machine translation systems for WMT 2018. In *Proc. of WMT*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proc. of WMT*.

Ofir Press and Noah A. Smith. 2018. You may not need attention.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. of EACL*.

Qiu Ran, Yankai Lin, Peng Li, and Jie Zhou. 2019. Guiding non-autoregressive neural machine translation decoding with reordering information.

Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-autoregressive machine translation with latent alignments.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proc. of ACL*.

Chenze Shao, Jinchao Zhang, Yang Feng, Fandong Meng, and Jie Zhou. 2020. Minimizing the bag-of-ngrams difference for non-autoregressive neural machine translation. In *Proc. of AAAI*.

Xing Shi and Kevin Knight. 2017. Speeding up neural machine translation decoding by shrinking run-time vocabulary. In *Proc. of ACL*.

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 *Proc. of AAAI*.

Mitchell Stern, William Chan, Jamie Ryan Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: flexible sequence generation via insertion operations. In *Proc. of ICML*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Proc. of NeurIPS*.

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

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proc. of NeurIPS*.

Lifu Tu, Richard Yuanzhe Pang, Sam Wiseman, and Kevin Gimpel. 2020. ENGINE: Energy-based inference networks for non-autoregressive machine translation. In *Proc. of ACL*.

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 *Proc. of NeurIPS*.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019a. Learning deep transformer models for machine translation. In *Proc. of ACL*.

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019b. Non-autoregressive machine translation with auxiliary regularization. In *Proc. of AAAI*.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *Proc. of ICLR*.

Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Accelerating neural transformer via an average attention network. In *Proc. of ACL*.

Jiawei Zhou and Phillip Keung. 2020. Improving non-autoregressive neural machine translation with monolingual data. In *Proc. of ACL*.

A Appendix

A.1 Hyperparameters and Setting

All of our models are implemented in `fairseq` (Ott et al., 2019) and trained with 16 Telsa V100 GPUs CUDA 10.1, and cuDNN 7.6.3. We used mixed precision and distributed training over 16 GPUs interconnected by Infiniband (Micikevicius et al., 2018; Ott et al., 2018). Apart from EN↔ZH where we used separate BPE operations, we tie all embeddings (Press and Wolf, 2017; Inan et al., 2017).

Autoregressive Models We generally follow the hyperparameters chosen in Vaswani et al. (2017); Ghazvininejad et al. (2019); Kasai et al. (2020) regardless of the numbers of encoding and decoding layers. Specifically, we list the hyperparameters in Table 6 for easy replication. All other hyperparamter options are left as default values in `fairseq`.

label smoothing	0.1
# max tokens	4096
dropout rate	[0.1, 0.2, 0.3]
encoder embedding dim	512
encoder ffn dim	2048
# encoder attn heads	8
decoder embedding dim	512
decoder ffn dim	2048
# decoder attn heads	8
max source positions	10000
max target positions	10000
Adam lrate	5e-4
Adam β_1	0.9
Adam β_2	0.98
lr-scheduler	inverse square
warm-up lr	1e-7
# warmup updates	4000
# max updates	300K, 500K (EN→FR)
length penalty	1.0

Table 6: Autoregressive translation `fairseq` hyperparameters and setting.

Non-autoregressive Models We use two strong non-autoregressive translation (NAT) models (CMLM: Ghazvininejad et al. (2019); DisCo: Kasai et al. (2020)). We use their code[5] and generally follow their hyperparameters regardless of the numbers of encoding and decoding layers. Specifically, we list the hyperparameters in Table 7 for easy replication. All other hyperparamter options are left as default values in `fairseq`.

[5] https://github.com/facebookresearch/Mask-Predict

label smoothing	0.1
# max tokens	8192
dropout rate	[0.1, 0.2, 0.3]
encoder embedding dim	512
encoder ffn dim	2048
# encoder attn heads	8
decoder embedding dim	512
decoder ffn dim	2048
# decoder attn heads	8
max source positions	10000
max target positions	10000
Adam lrate	5e-4
Adam β_1	0.9
Adam β_2	0.999
lr-scheduler	inverse square
warm-up lr	1e-7
# warmup updates	10000
# max updates	300K, 500K (EN→FR)

Table 7: Non-autoregressive translation `fairseq` hyperparameters and setting.

B Results

Table 8 provides comparisons of speed and quality in WMT 14 EN−DE and 16 EN−RO datasets.

Model			**WMT14 EN–DE**						**WMT16 EN–RO**					
		E-D	→de	S_1	$\mathrm{S}_{\mathrm{max}}$	→en	S_1	$\mathrm{S}_{\mathrm{max}}$	→ro	S_1	$\mathrm{S}_{\mathrm{max}}$	→en	S_1	$\mathrm{S}_{\mathrm{max}}$
CMLM	T=4	6-6	26.74	4.4x	0.3x	30.75	4.1x	0.3x	33.02	3.6x	0.3x	33.28	3.5x	0.2x
	T=10		27.39	1.9x	0.2x	31.24	1.8x	0.2x	33.33	1.6x	0.1x	33.67	1.5x	0.1x
	T=4	12-1	24.68	**7.6x**	0.4x	29.39	**6.9x**	0.4x	31.90	**6.6x**	0.3x	32.70	5.9x	0.3x
	T=10		26.27	4.3x	0.2x	30.34	4.0x	0.2x	32.36	3.5x	0.1x	32.97	3.4x	0.2x
DisCo		6-6	27.35	3.6x	0.3x	31.31	3.6x	0.3x	33.22	4.0x	0.2x	33.25	4.1x	0.2x
		12-1	26.75	6.4x	0.4x	30.62	6.2x	0.4x	32.60	6.0x	0.3x	32.62	**6.3x**	0.3x
AT		6-6	**28.34**	1x	1x	31.81	1x	1x	**34.60**	1x	1x	34.57	1x	1x
		6-1	27.39	2.7x	**1.4x**	30.80	2.6x	**1.5x**	33.19	3.0x	**2.0x**	34.31	2.9x	**2.0x**
		12-1	28.28	2.5x	**1.4x**	**31.82**	2.5x	1.4x	33.84	2.9x	**2.0x**	**34.78**	2.9x	**2.0x**

Table 8: BLEU and speed comparisons with varying number of encoder (E) and decoder (D) layers.