The NiuTrans System for the WMT21 Efficiency Task

Chenglong Wang[1], Chi Hu[1], Yongyu Mu[1], Zhongxiang Yan[1], Siming Wu[1], Yimin Hu[1], Hang Cao[1], Bei Li[1], Ye Lin[1], Tong Xiao[1,2] and Jingbo Zhu[1,2]
[1]NLP Lab, School of Computer Science and Engineering, Northeastern University
[2]NiuTrans Research, Shenyang, China
clwang1119@gmail.com,huchinlp@gmail.com
{xiaotong,zhujingbo}@mail.neu.edu.cn

Abstract

This paper describes the NiuTrans system for the WMT21 translation efficiency task[1]. Following last year's work, we explore various techniques to improve the efficiency while maintaining translation quality. We investigate the combinations of lightweight Transformer architectures and knowledge distillation strategies. Also, we improve the translation efficiency with graph optimization, low precision, dynamic batching, and parallel pre/post-processing. Putting these together, our system can translate 247,000 words per second on an NVIDIA A100, being 3$\times$ faster than our last year's system. Our system is the fastest and has the lowest memory consumption on the GPU-throughput track. The code, model, and pipeline will be available at NiuTrans.NMT[2].

1 Introduction

Large and deep Transformer models have dominated machine translation (MT) tasks in recent years (Vaswani et al., 2017; Edunov et al., 2018; Wang et al., 2019; Raffel et al., 2020). Despite their high accuracy, these models are inefficient and difficult to deploy (Wang et al., 2020a; Hu et al., 2021; Lin et al., 2021b). Many efforts have been made to improve the translation efficiency, including efficient architectures (Li et al., 2021a,b), quantization (Bhandare et al., 2019; Lin et al., 2020), and knowledge distillation (Li et al., 2020; Lin et al., 2021a).

This work investigates efficient Transformers architectures and optimizations specialized for different hardware platforms. In particular, we study deep encoder and shallow decoder Transformer models and optimize them for both GPUs and CPUs. Starting from an ensemble of three deep Transformer teacher models, we train various student models via sequence-level knowledge distillation (SKD) (Hinton et al., 2015; Li et al., 2021a; Kim and Rush, 2016) and data augmentation (Shen et al., 2020). We find that using a deep encoder (6 layers) and a shallow decoder (1 layer) gives reasonable improvements in speed while maintaining high translation quality. We improve the student model's efficiency by removing unimportant components, including the FFN sub-layers and multi-head mechanism. We also explore other model-agnostic optimizations, including graph optimization, dynamic batching, parallel pre/post-processing, 8-bit matrix multiplication on CPUs, and 16-bit computation on GPUs.

Section 2 describes the training procedures of the deep teacher models. Then, Section 3 presents various optimizations for reducing the model size, improving model performance and efficiency. Finally, Section 4 details the accuracy and efficiency results of our submissions for the shared efficiency task.

2 Model Overview

Following Hu et al. (2020), Li et al. (2021a) and Lin et al. (2021a), we use the SKD method to train our models. Our experiments also show that the SKD method can obtain better performance than the word-level knowledge distillation (WKD) method, similar to Kim and Rush (2016). Therefore, all of student models are optimized by using the interpolated SKD method (Kim and Rush, 2016), and trained on data generated from the teacher models.

2.1 Deep Transformer Teacher Models

Recently, researchers have explored deeper models to improve the translation quality (Wang et al., 2019; Li et al., 2020; Dehghani et al., 2019; Wang et al., 2020b). Inspired by them, we employ deep Transformers as the teacher models. More specifically, we train three teachers with different configurations, including *Deep-30*, *Deep-12-768*, and *Skipping Sublayer-40*. We also utilize Li et al. (2019)'s

[1]http://statmt.org/wmt21/efficiency-task.html
[2]https://github.com/NiuTrans/NiuTrans.NMT

Student Model	Param.	BLEU
Student-6-6-8	96M	33.2
Student-6-1-8	42M	33.0
Student-6-1-1	42M	32.9

Table 1: Reference BLEU scores for the student models on *newstest20*. 6-6-8 means that the model contains 6 encoder layers and 6 decoder layers with 8 attention heads. Other hyper-parameters are the same as the vanilla Transformer.

ensemble strategy to boost the teachers.

Deep-30 Transformer Model: We set the number of encoder layers to 30 in the Transformer model. Other hyper-parameters are identical to the vanilla Transformer.

Deep-12-768 Transformer Model: This model modifies the number of encoder layers, hidden sizes and embedding sizes to 12, 3072 and 768. Such a setting makes the Transformer model deeper and wider. Other hyper-parameters are the same as vanilla Transformer.

Skipping Sublayer-40 Transformer Model: This model uses a simple training procedure that samples one streaming configuration in each iteration (Li et al., 2021a). The number of encoder layers is 40 and model's other setups are same as Li et al. (2021a).

We adopt the relative position representation (RPR) (Shaw et al., 2018) to further improve the teacher models and set the key's relative length to 8.

2.2 Lightweight Transformer Student Models

Although the ensemble teacher model delivers excellent performance, our goal is to learn lightweight models. The natural idea is to compress knowledge from an ensemble into the lightweight model using knowledge distillation (Hinton et al., 2015). We employ sequence-level knowledge distillation on the ensemble teacher model described in Section 2.1.

Seqence-level Knowledge Distillation The SKD will make a student model mimic the teacher's behaviors at the sequence level. Moreover, the method considers the sequence-level distribution specified by the model over all possible sequences $\mathbf{t} \in T$. Following Kim and Rush (2016), the loss function of SKD method for training students is

$$\mathcal{L}_{\text{SKD}} \approx -\sum_{\mathbf{t}\in\mathcal{T}} \mathbf{1}\{\mathbf{t} = \hat{\mathbf{y}}\} \log p(\mathbf{t} \mid \mathbf{s}) \quad (1)$$

$$= -\log p(\mathbf{t} = \hat{\mathbf{y}} \mid \mathbf{s}) \quad (2)$$

where $\mathbf{1}\{\cdot\}$is the indicator function, $\hat{\mathbf{y}}$ is the output of teacher model using beam search, $\mathbf{s}$ symbolizes the source sentence and $p(\cdot|\cdot)$ denotes the conditional probability. We use the ensemble teacher model to generate multiple translations of the raw English sentences. In particular, we collect the 5-best list for each sentence against the original target to create the synthetic training data. However, we select only 12 million synthetic data to train our student models to reduce training costs. We find that student models will not have better performance when increasing the number of training data.

Fast Student Models As suggested in Hu et al. (2020), the bottleneck of translation efficiency is the decoder part. Hence, we accelerate the decoding by reducing the number of decoder layers and removing multi-head mechanism[3]. Inspired by Hu et al. (2021), we design the lightweight Transformer student model with one decoder layer. We further remove the multi-head mechanism in the decoder's attention modules. Table 1 shows that the Transformer student model with one decoder layer and one decoder attention head can achieve similar translation quality to the baseline. Therefore, we train four different student models based on the Transformer architecture with one decoder layer and one decoder attention head. Those student models are described in detail in the Table 2. Besides, experiments show that adding more encoder layers cannot improve the performance when the student model has 12 encoder layers. Therefore, our submissions have 12 encoder layers at most.

2.3 Data and Training Details

Our data is constrained by the condition of the WMT 2021 English-German news translation task[4], and we use the same data filtering method as Zhang et al. (2020). We select 20 million pairs to train our teacher models after filtering all official released parallel datasets (without official synthetic datasets). The data is tokenized with Moses tokenizer (Koehn et al., 2007), and jointly Byte-Pair

[3]Although the multi-head mechanism does not increase the parameter of the model, it brings non-negligible computational costs.

[4]https://www.statmt.org/wmt21/translation-task.html

Student Model	N-Enc	Dim-FFN	Param.	Speedup	newstest18	newstest19	newstest20
Student-12-1-512	12	512	56M	2.0×	45.3	41.7	33.2
Student-6-1-512	6	512	38M	2.3×	44.5	41.0	32.7
Student-6-1-0	6	0	37M	2.4×	43.9	40.6	32.4
Student-3-1-512	3	512	28M	2.6×	42.8	40.0	31.5

Table 2: N-Enc is the number of encoder layers and Dim-FFN denotes the feed-forward network (FFN) size. The Speedup and BLEU results are measured on a TITAN V GPU. The Speedup is calculated comparing with our ensemble teacher model. The student model has not FFN component in the decoder when the Dim-FFN is 0. Evaluation is performed without inference optimizations and with a beam size of 1.

Teacher Model	Param.	BLEU
Deep-30	138M	32.8
Deep-12-768	170M	33.3
Skipping Sublayer-40	171M	33.1
Ensemble	479M	33.4

Table 3: Results on *newstest20*-Teacher Models. We train our teacher models with the RPR and back-translation.

Encoded (BPE) (Sennrich et al., 2016) with 32K merge operations using a shared vocabulary. After decoding, we remove the BPE separators and detokenize all tokens with Moses detokenizer (Koehn et al., 2007).

Teacher Models Training We train three teacher models using *newstest19* as the development set with `Fairseq` (Ott et al., 2019). We share the source-side and target-side embeddings with the decoder output weights. We use the Adam optimizer (Kingma and Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.997$ and $\epsilon = 10^{-8}$ as well as gradient accumulation due to the high GPU memory footprints. Each model is trained on 8 TITAN V GPUs for up to 11 epochs. The learning rate is decayed based on the inverse square root of the update number after 1,6000 warm-up steps, and the maximum learning rate is 0.002. After training, we average the last five checkpoints in the training process for all models. Similar to Zhang et al. (2020), we train our teacher models with a round of back-translation with 12 million monolingual data selected from the News crawl and News Commentary. We train three De→En models with the same method and model setup to generate pseudo-data. Table 3 shows the results of all teacher models and their ensemble, where we report SacreBLEU (Post, 2018) and the model size. Our final ensemble teacher model can achieve a BLEU score of 33.4 on *newstest20*.

Student Models Training The training settings for student models are the same for the teacher models, except its learning rate is 0.0007 and warmup-updates is 8000. In addition, we also use the cutoff method (Shen et al., 2020) to boost our student models[5] and we train our student model with 21 epochs. Table 2 shows the results of all student models. Our student model yields a significant speedup (2×-2.6×) with modest sacrifice in terms of BLEU (0.2-0.9 on *newstest20*).

2.4 Interpretation of Results

After training the final student models, we evaluate their BLEU scores on the English-German *newstest20*, *newstest19*, and *newstest18* before any inference optimization. Results show that the student models can achieve very similar performance to the teachers. For instance, the *Student-12-1-512* model delivers a loss of 0.2 BLEU score compared to the ensemble of teacher models.

3 Optimizations for Decoding

Our optimizations for decoding are implemented with NiuTensor [6]. The optimizations can be divided into three parts, including optimizations for CPUs, GPUs, and device-independent techniques.

3.1 Optimizations for GPUs

For the GPU-based decoding, we mainly explore dynamic batching and FP16 inference.

Dynamic Batching Unlike the CPU version, the easiest way to reduce the translation time on GPUs is to increase the batch size within a specific range. We implement a dynamic batching scheme that maximizes the number of sentences in the batch while limiting the number of tokens. This strategy

[5] `https://github.com/stevezheng23/fairseq_extension/tree/master/examples/translation/augmentation`

[6] `https://github.com/NiuTrans/NiuTensor`

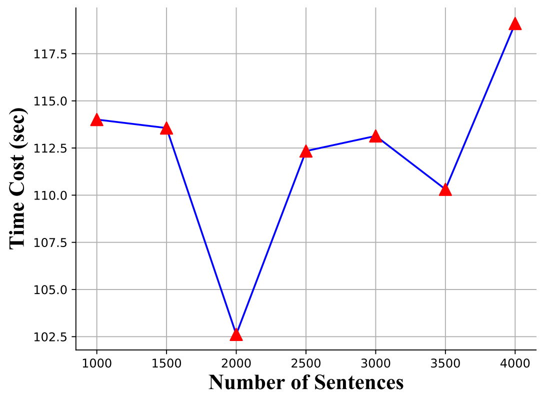


Figure 1: Results on Student-6-1-512 model. The time cost is measured on an Intel Xeon Gold 6240 CPU with 100,000 lines of raw English sentences with an averaged length of 18 words.

significantly accelerates the inference compared to a fixed batch size when the sequence length is short.

FP16 Inference Since the Tesla A100 GPU supports calculations under FP16, our systems execute almost all operations in 16-bit floating-point. To escape overflow, we convert the data type before and after the softmax operation in the attention modules. We also reorder some operations for numerical stability. For instance, we apply the scaling operation (dived by $\sqrt{d_k}$) to the query instead of the attention weights. To accelerate our systems further, we replace the vanilla layer normalization with the L1-norm (Lin et al., 2020). Also, we find that removing the multi-head mechanism (by setting the head to 1) in the student models significantly improves the throughput without performance loss.

3.2 Optimizations for CPUs

We employ the *Student-6-1-512* and *Student-3-1-512* models as our CPU submissions. Two methods are discussed to speed up the decoding for our CPU systems.

The Use of MKL We use the Intel Math Kernel Library (Wang et al., 2014) to optimize our NiuTensor framework, which helps our systems to make the full use of the Intel architecture and to extract the maximum performance.

8-bit Matrix Multiplication with Packing We implement 8-bit matrix multiplication using the open-source library FBGEMM (Khudia et al., 2021). Following Kim et al. (2019), we quantize each column of the weight matrix separately with different scales and offsets. Scale and offsets for weight matrix are calculated by:

$$b_{scale}[j] = \frac{14\sigma_j}{255} \quad (3)$$

$$b_{zeropoint}[j] = \frac{127 - (\bar{x}_j + 7\sigma_j)}{b_{scale}[j]} \quad (4)$$

where σ_j and $\bar{x}_j$ refers to average and standard deviation for the j-th column. The quantization parameters for the input matrix is calculated by:

$$a_{scale} = \frac{x_{max} - x_{min}}{255} \quad (5)$$

$$a_{zeropoint} = \frac{255 - x_{max}}{a_{scale}} \quad (6)$$

where x_{max} and x_{min} are the maximum and minimum values of the matrix respectively. With FBGEMM API, we also execute the packing operation to change the layout of the matrices into a form that uses the CPU more efficiently. We pre-quantize and pre-pack all the weight matrices to avoid repeated operation during inference.

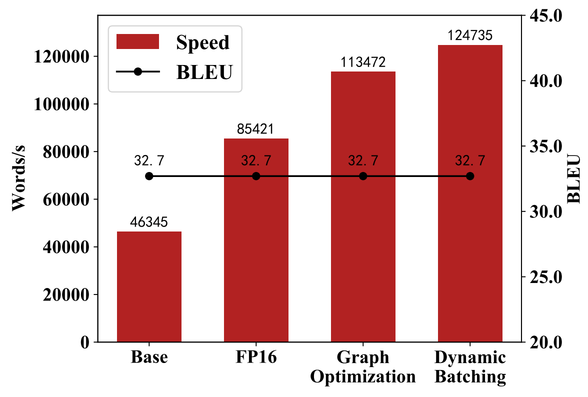
where x_{max} and x_{min} are the maximum and minimum values of the matrix, respectively. We also execute the packing operation to change the layout of the matrices into a form that uses the CPU more efficiently. We pre-quantize and pre-pack all the weight matrices to avoid repeated operation during inference.

3.3 Other Optimizations

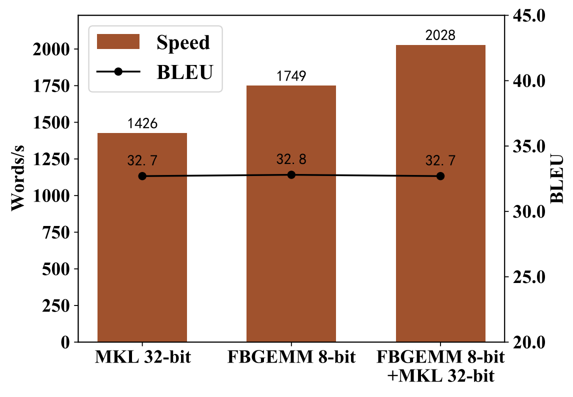
Furthermore, we explore other device-independent methods to optimize our systems. Those methods help our systems to achieve obvious speed-up without translation precision loss.

Graph Optimization A neural net can be represented by a directed acyclic graph (DAG), where the nodes represent tensors and the connections represent operations. We optimize our system by simplifying the computational graph of the models. The optimizations for the graph are detailed as follows:

- Computation optimization. We prune all redundant operations and reorder some operations in the computational graph. For instance, we remove the *log-softmax* operation in the output layer when using greedy search. We also extract the *transpose* operations from matrix multiplications to the begin of decoding.



(a) Performance of our GPU system



(b) Performance of our CPU system

Figure 2: BLEU on *newstest20* versus words per second (Words/s) with different optimizations on a TITAN V GPU and Intel Xeon Gold 5118 CPUs. Result of decoding speed is measured with 0.1 M sentences (average length is 18). When the GPU system is running, it will use all free CPUs on the device.

- Memory optimization. We reuse all possible nodes to minimize the memory consumption. We also reduce the memory allocation or movement with an efficient memory pool. Moreover, we sort the source sentences in descending order of length and detect the peak memory footprint before decoding.

Parallel Execution We use the GNU Parallel (Tange, 2011) for our systems to perform tasks in parallel. More specifically, we split the standard input into several lines and deliver them via the pipeline. The method is used to accelerate pre-processing, post-processing, and decoding on CPUs. We also find that the system decoding speed/memory is strongly correlated with the number of lines per task. To find the best number of lines for each run, we measure the time cost in different setups against the number of lines. Figure 1 shows that 2000 is a relatively good choice, and the *Student-6-1-512* model can translate 100,000 sentences in 102.6s on CPUs under this setup.

Better Decoding Configurations As aforementioned, our GPU versions use a large batch size, but the batch size on the CPU is much smaller. To be more clear, there is sentence batch (*sbatch*) and word batch (*wbatch*) in our systems, and they restrict the number of sentences and number of words in a mini-batch to not be greater than *sbatch* and *wbatch*, respectively. In our GPU systems, we set the *sbatch*/*wbatch* to 3072/64000. For our CPU systems, the number of processes is managed by the Parallel tool, which is more efficient and accurate. Moreover, We use one MKL thread for each process and set the *sbatch*/*wbatch* to 128/2048.

Greedy Search In the practice of knowledge distillation, we find that our systems are insensitive to the beam size. It means that the translation quality is good enough even using greedy search in all submissions.

Fast Data Preparation We use the fastBPE[7], a faster C++ version of subword-nmt[8], to speed the BPE process. Moreover, we also use the fast-mosestokenizer[9] for tokenization.

3.4 Results after Optimizations

Figure 2 plots the *Student-6-1-512* model's performance with different decoding optimizations. All results show that our optimizations can significantly speed up our system without losing BLEU. What is interesting about the BLEU is that we can achieve additional improvements of 0.4/0.1 BLEU points on the GPU/CPU through decoding optimizations in all our experiments. We also measure other models after decoding optimizations and find their performance is similar to the *Student-6-1-512* model.

4 Submissions and Results

4.1 Submissions

For the GPU track submissions, our GPU systems are compiled with CUDA 11.2. We set the num-

[7]https://github.com/glample/fastBPE
[8]https://github.com/rsennrich/subword-nmt
[9]https://github.com/mingruimingrui/fast-mosestokenizer

ber of decoder layers and the number of our decoder attention head to 1 as described in Section 2.2 for all our GPU systems. We see a speedup of more than 6× on the GPU system created by *Student-12-1-512* model and a slight decrease of only 0.2 BLEU on the *newstest20* compared to the deep ensemble model. The system is named as ***Base-GPU-System*** in following part. We continue to reduce the number of encoder layers for more accelerations, and the GPU system with *Student-6-1-512* model reduces the translation time by one-four with only six encoder layers compared to the *Base-GPU-System*. Our fastest GPU system consists of three encoder layers and one decoder layer, which achieves 31.5 BLEU on the *newstest20* with GPU and 1.6× speedup compared to the *Base-GPU-System*. We also employ the *Student-6-1-0* model to create a GPU system that can achieve the 1.3× speedup compared to *Base-GPU-System*. Our systems are compiled in the 11.2.1-devel-centos7 docker image, an NVIDIA open-source image[10]. We copy the executables, dependence tools, and model files to the 11.2.1-base-centos7 docker image (final submission). In this way, we ensure all of our system docker images can be executed by the organizers successfully and reduce the docker images size.

For the CPU track submissions, we use the test machine, which has 18 virtual cores. Our CPU version is compiled with MKL static library, and the executable file is 23MiB. Also, we use the 8-bit matrix multiplication with packing to speed the matrix multiplication in the network. We use the *Student-3-1-512* and *Student-6-1-512* models in our CPU systems, and they respectively achieve 31.5 and 32.8 BLEU on *newstest20*. For our CPU docker images, we use the base-centos7 docker image[11] to deploy our CPU MT systems.

Furthermore, all submissions are tested with different cases, including dirty data, empty input, and very long sentences. The test results show that our systems can run successfully with exceptional inputs.

4.2 Results

Our systems for the GPU-throughput track are the fastest overall submissions. Specifically, the *Student-3-1-512* system can translate about 250 thousand words per second and achieve 25.5 BLEU on *newstest21*. We attribute this to the comparison of the performance of our teacher model on WMT21. In the CPU track, our system also has competitive performance. Our fastest CPU system created by *Student-3-1-512* model can translate about 48 thousand words per real second via 36 CPU cores and can achieve 25.5 BLEU. We find that reducing the number of encoder layers for student model achieves lower BLEU scores at a similar speed for our CPU systems. Moreover, we compare the cost-effective of GPU and CPU decoding in terms of millions of words translated per dollar according to the official evaluation results. We find that highly-effective GPU decoding is about to out-compete CPU-bound decoding in terms of cost-effective. Noteworthy, our GPU system with *Student-3-1-512* model can translate 300M words per dollar with acceptable quality. Also, all of our GPU systems have the lowest RAM consumption (about 4 GB) to official test compared with the submissions of other participants.

[10]https://hub.docker.com/r/nvidia/cuda
[11]https://hub.docker.com/_/centos

5 Conclusion

We have described our systems for the WMT21 shared efficiency task. We have explored various efficient Transformer architectures and optimizations specialized for both CPUs and GPUs. We have shown that a lightweight decoder and proper optimizations for different hardware can significantly accelerate the translation process with slight or no loss of translation quality. Our fastest GPU system with three encoder layers and one decoder layer is 11× faster than the deep ensemble model and lose 1.9 BLEU points.

Acknowledgements

This work was supported in part by the National Science Foundation of China (Nos. 61876035 and 61732005), the National Key R&D Program of China (No.2019QY1801), and the Ministry of Science and Technology of the PRC (Nos. 2019YFF0303002 and 2020AAA0107900). The authors would like to thank the anonymous reviewers for their comments and suggestions.

References

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale.

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

Chi Hu, Bei Li, Yinqiao Li, Ye Lin, Yanyang Li, Chenglong Wang, Tong Xiao, and Jingbo Zhu. 2020. The NiuTrans system for WNGT 2020 efficiency task. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 204–210, Online. Association for Computational Linguistics.

Chi Hu, Chenglong Wang, Xiangnan Ma, Xia Meng, Yinqiao Li, Tong Xiao, Jingbo Zhu, and Changliang Li. 2021. Ranknas: Efficient neural architecture search by pairwise ranking.

Daya Khudia, Jianyu Huang, Protonu Basu, Summer Deng, Haixin Liu, Jongsoo Park, and Mikhail Smelyanskiy. 2021. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

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 *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 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, Prague, Czech Republic. Association for Computational Linguistics.

Bei Li, Yinqiao Li, Chen Xu, Ye Lin, Jiqiang Liu, Hui Liu, Ziyang Wang, Yuhao Zhang, Nuo Xu, Zeyang Wang, et al. 2019. The niutrans machine translation systems for wmt19. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 257–266.

Bei Li, Ziyang Wang, Hui Liu, Quan Du, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2021a. Learning light-weight translation models from deep transformer. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 13217–13225. AAAI Press.

Bei Li, Ziyang Wang, Hui Liu, Yufan Jiang, Quan Du, Tong Xiao, Huizhen Wang, and Jingbo Zhu. 2020. Shallow-to-deep training for neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 995–1005, Online. Association for Computational Linguistics.

Yanyang Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2021b. An efficient transformer decoder with compressed sub-layers. *CoRR*, abs/2101.00542.

Ye Lin, Yanyang Li, Tengbo Liu, Tong Xiao, Tongran Liu, and Jingbo Zhu. 2020. Towards fully 8-bit integer inference for the transformer model. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3759–3765. ijcai.org.

Ye Lin, Yanyang Li, Ziyang Wang, Bei Li, Quan Du, Tong Xiao, and Jingbo Zhu. 2021a. Weight distillation: Transferring the knowledge in neural network parameters. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2076–2088. Association for Computational Linguistics.

Ye Lin, Yanyang Li, Tong Xiao, and Jingbo Zhu. 2021b. Bag of tricks for optimizing transformer efficiency.

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.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations.

Dinghan Shen, Mingzhi Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. 2020. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818*.

O. Tange. 2011. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™*, pages 167–188. Springer.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. Hat: Hardware-aware transformers for efficient natural language processing.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Qiang Wang, Tong Xiao, and Jingbo Zhu. 2020b. Training flexible depth model by multi-task learning for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4307–4312, Online. Association for Computational Linguistics.

Yuhao Zhang, Ziyang Wang, Runzhe Cao, Binghao Wei, Weiqiao Shan, Shuhan Zhou, Abudurexiti Reheman, Tao Zhou, Xin Zeng, Laohu Wang, et al. 2020. The niutrans machine translation systems for wmt20. In *Proceedings of the Fifth Conference on Machine Translation*, pages 338–345.