The NiuTrans System for WNGT 2020 Efficiency Task

Chi Hu[†]   **Bei Li**[†]   **Ye Lin**[†]   **Yinqiao Li**[†]
Yanyang Li[†]   **Chenglong Wang**[†]
Tong Xiao[†‡]   **Jingbo Zhu**[†‡]
[†]NLP Lab, Northeastern University, Shenyang, China
[‡]NiuTrans Reasearch, Shenyang, China
huchinlp@gmail.com, libei_neu@outlook.com,
{xiaotong,zhujingbo}@mail.neu.edu.com,

Abstract

This paper describes the submissions of the NiuTrans Team to the WNGT 2020 Efficiency Shared Task. We focus on the efficient implementation of deep Transformer models (Wang et al., 2019; Li et al., 2019) using NiuTensor[1], a flexible toolkit for NLP tasks. We explored the combination of deep encoder and shallow decoder in Transformer models via model compression and knowledge distillation. The neural machine translation decoding also benefits from FP16 inference, attention caching, dynamic batching, and batch pruning. Our systems achieve promising results in both translation quality and efficiency, e.g., our fastest system can translate more than 40,000 tokens per second with an RTX 2080 Ti while maintaining 42.9 BLEU on *newstest2018*.

1 Introduction

In recent years, the Transformer model and its variants (Vaswani et al., 2017; Shaw et al., 2018; So et al., 2019; Wu et al., 2019; Wang et al., 2019) have established state-of-the-art results on machine translation (MT) tasks. However, achieving high performance requires an enormous amount of computations (Strubell et al., 2019), limiting the deployment of these models on devices with constrained hardware resources.

The efficiency task aims at developing MT systems to achieve not only translation accuracy but also memory efficiency or translation speed across different devices. This competition constraints systems to translate 1 million English sentences within 2 hours. Our goal is to improve the quality of translations while maintaining enough speed. We participated in both CPUs and GPUs tracks in the shared task.

Our system was built with NiuTensor, an open-source tensor toolkit written in C++ and CUDA

based on dynamic computational graphs. NiuTensor is developed for facilitating NLP research and industrial deployment. The system is lightweight, high-quality, production-ready, and incorporated with the latest research ideas.

We investigated with a different number of encoder/decoder layers to make trade-offs between translation performance and speed. We first trained several strong teacher models and then compressed teachers to compact student models via knowledge distillation (Hinton et al., 2015; Kim and Rush, 2016). We find that using a deep encoder (up to 35 layers) and a shallow decoder (1 layer) gives reasonable improvements in speed while maintaining high translation quality. We also optimized the Transformer model decoding in engineering, such as caching the decoder's attention results and using low precision data type.

We present teacher models and training details in Section 2, then in Section 3 we describe how to obtain lightweight student models for efficient decoding. Optimizations for the decoding across different devices are discussed in Section 4. We show the details of our submissions and the results in Section 5. Section 6 summarizes this paper and describes future work.

2 Deep Transformer Teachers

2.1 Deep Transformer Architectures

Recent years have witnessed the success of transformer-based models in MT tasks. Many works (Dehghani et al., 2019; Zhang et al., 2019; Li et al., 2020) focus on designing new attention mechanisms and Transformer architectures. Shaw et al. (2018) extended the self-attention to consider the relative position representations or distances between words. Wu et al. (2019) replaced the self-attention components with lightweight and dynamic convolutions. Deep Transformer mod-

---

[1]https://github.com/NiuTrans/NiuTensor

els also attracted a lot of attention. Wang et al. (2018) proposed a multi-layer representation fusion approach to learn a better representation from the stack. Wang et al. (2019) analyzed the high risk of gradient vanishing or exploring in the standard Transformer, which place the layer normalization (Ba et al., 2016) after the attention and feed-forward components. They showed that a deep Transformer model can surpass the big one by proper use of layer normalization and dynamic combinations of different layers. In their method, the input of layer $l + 1$ is defined by:

$$x_{l+1} = \mathcal{G}(y_0, \ldots, y_l) \tag{1}$$

$$\mathcal{G}(y_0, \ldots, y_l) = \sum_{k=0}^{l} W_k^{(l+1)} \text{LN}(y_k) \tag{2}$$

where y_l is the output of the $l_t h$ layer and W is the weights of different layers.

We employed the dynamic linear combination of layers Transformer architecture incorporated with relative position representations as our teacher network, call it Transformer-DLCL-RPR.

2.2 Training Details

We followed the constrained condition of the WMT 2019 English-German news translation task and used the same data filtering method as (Li et al., 2019). We also normalized punctuation and tokenized all sentences with the Moses tokenizer (Koehn et al., 2007). The training set contains about 10M sentences pairs after processed. In our systems, the data was tokenized, and jointly byte pair encoded (Sennrich et al., 2016) with 32K merge operations using a shared vocabulary. After decoding, we removed the BPE separators and de-tokenize all tokens.

We trained four teacher models using *newstest2018* as the development set with fairseq (Ott et al., 2019). Table 1 shows the results of all teacher models and their ensemble, where we report SacreBLEU (Post, 2018) and the model size. The difference between teachers is the number of encoder layers and whether they contain a dynamic linear combination of layers. All teachers have 6 decoder layers, 512 hidden dimensions, and 8 attention heads. We shared the source-side and target-side embeddings with the decoder output weights. The maximum relative length was 8, and the maximum position for both source and target was 1024. We used the Adam optimizer (Kingma

Model	Param.	BLEU
Transformer-35-6	152M	43.3
Transformer-35-6+DLCL	152M	43.7
Transformer-40-6	168M	44.5
Transformer-40-6+DLCL	168M	43.9
Ensemble	640M	45.5

Table 1: Results on *newstest18* - Teacher Models. 35-6 means that the model contains 35 encoder layers and 6 decoder layers.

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 footprint. Each model was trained on 8 RTX 2080Ti GPUs for up to 21 epochs. We batched sentence pairs by approximate length and limited input/output tokens per batch to 2048/GPU. Following the method of (Wang et al., 2019), we accumulated every two steps for a better batching. This resulted in approximately 56000 tokens per training batch. The learning rate was decayed based on the inverse square root of the update number after 16000 warm-up steps, and the maximum learning rate was 0.002. Furthermore, we averaged the last five checkpoints in the training process for all models.

As shown in Table 1, the best single teacher model achieves 44.5 BLEU (beam size 4) on *newstest2018*. Then we obtained an improvement of 1 BLEU via a simple ensemble strategy used in (Li et al., 2019).

3 Lightweight Student Models

After the training of deep Transformer teachers, we compressed the knowledge in an ensemble into a single model through knowledge distillation (Hinton et al., 2015; Kim and Rush, 2016). Then we analyzed the decoding time of each part in the deep Transformer. We further pruned the encoder and decoder layers to improve the decoding efficiency.

3.1 Knowledge Distillation

Knowledge distillation approaches (Hinton et al., 2015; Kim and Rush, 2016) have proven successful in reducing the size of neural networks. They learn a smaller student model to mimic the original teacher network by minimizing the loss between the student and teacher output. We applied the sequence-level knowledge distillation on the teacher ensemble described in Section 2. We used the ensemble to generate multiple translations of
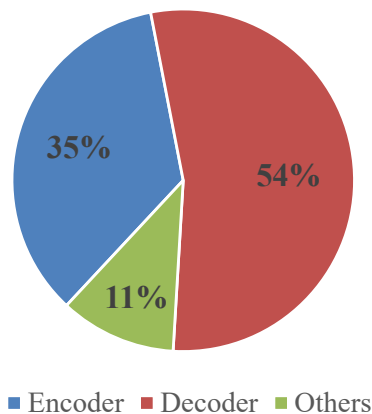
Figure 1: Profiling of the throughput during inference on *newstest2018* using a 35-6 model.

Model	Param.	Speedup	BLEU
Teacher-40-6	168M	1x	44.5
Student-35-6	152M	1.1x	44.6
Student-35-1	131M	1.6x	44.3
Student-18-1	77M	2.0x	43.4
Student-9-1	49M	2.4x	42.9
Student-tiny	25M	2.9x	37.2

Table 2: Results on *newstest18*. The students were trained by sequence-level knowledge distillation. The tiny setting keeps the 9-1 model's configurations except for using a model size of 256. We report the translation speed on a single 2080Ti.

the raw English sentences. In particular, we collected the 4-best list for each sentence against the original target to create the synthetic training data. Our base student model consists of 35 encoder layers and six decoder layers (call it 35-6) with nearly 150M parameters. It achieves 44.6 BLEU on the test set.

3.2 Fast Student Models

Although the deep model can obtain high-quality translations, its speed is not satisfactory. For example, it costs 6.7 seconds to translate 2998 sentences on a 2080Ti GPU using a 35-6 model with the greedy search. Statistics show that the most time-consuming part of the decoding process is the decoder, as presented in Figure 1, so the most efficient optimization is to use a lightweight decoder. To make a comparison, we kept the 35 encoder layers and reduced the decoder layer to 1. In practice, we copied the bottom layers' parameters from big models to small models for initialization. Then we trained the small models as usual. Similar to (Wang et al., 2019), the encoder has a more significant influence on the translation quality than the decoder. Reducing the number of decoder layers brings us a speedup of more than 30% with a slight loss of 0.3 BLEU.

We further compressed the model by shrinking the encoder. Unless otherwise stated, the following student models have only one decoder layer. We copied the bottom layer parameters from big models to initialize small models to stabilize the training. We trained two small models with an 18-layer encoder and a 9-layer encoder, respectively. Table 2 shows the comparison of different teachers and students. Compared with the 35-1 model,

cutting off half of the encoder layer reduces the parameters by nearly half and gives a speedup of 20% with a decrease of 0.2 BLEU. The 9-1 model is the fastest model we run on the GPU. It can translate *newstest2018* within 3 seconds on a 2080Ti GPU and obtain 42.9 BLEU.

All models mentioned above can translate 1 million sentences on the GPU in 2 hours. However, using a CPU to achieve this goal is not easy, so we need smaller models. We set the 9-1 model size to 256 for the CPU version, namely 9-1-tiny, which has only half the 9-1 model parameters. This model achieves 37.2 BLEU on *newstest2018* and reduces 90% parameters compared to the 35-6 model.

4 Optimizations for Decoding

4.1 General Optimizations

First, we discuss some device-independent optimization methods.

Caching We can cache the output of the top layer of the encoder and each step of the decoder since we use an autoregressive model. More specifically, we cache the linear transformations for keys and values before the self-attention and cross-attention layers.

Faster Beam Search Beam search is a common approach in sequence decoding. The standard beam search strategy generates the target sequence in a self-regression manner and keeps a fixed amount of active candidates during decoding. We adopt a basic strategy to accelerate beam search: the search ends when any candidate predicts the EOS symbol, and there are no candidates with higher scores. This strategy brings us up to a 20% speedup on the WMT test set. Other threshold-based pruning strategies (Freitag and Al-Onaizan, 2017) are not appropriate due to the complex hyper-parameters.

Batch Pruning   The length of target sequences may vary for different sentences in a batch, which makes the computation inefficient. We prune the finished hypotheses in a batch during decoding but only gain little accelerations on CPUs.

4.2   Optimizing for GPUs

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

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 implemented a dynamic batching scheme that maximizes the number of sentences in the batch while limiting the number of tokens. This strategy significantly accelerates decoding compared to using a fixed batch size when the sequence length is short.

FP16 Inference   Since the Tesla T4 GPU supports calculations under FP16, our systems execute almost all operations in 16-bit floating-point. All model parameters are stored in FP16, which reduces the model size on disk by half. We tried to run all operations at a 16-bit floating-point. However, in our test, some particular inputs will cause numerical instability, such as large batch size or sequence length. To escape overflow, we convert the data type around some potentially problematic operations, i.e., all operations related to $reduce\_sum$.

4.3   Optimizing for CPUs

As mentioned above, the goal we set for the CPU version is to translate 1 million sentences in 2 hours. We used the same settings as the 9-1 model except that the model size is 256 and therefore sacrifice about 6 BLEU on the WMT test set. We employed two methods to speed up the decoding on CPUs.

Using of MKL   To make the full use of the Intel architecture and to extract the maximum performance, the NiuTensor framework is optimized using the Intel Math Kernel Library for basic operators. We can take advantage of this convenience with only minor changes to the configuration.

Decoding in Parallel   The target machine in this task has 96 logical processors (with hyperthreading) and 192 GB RAM so that we can run our multi-threading system. We split the input into several parts according to the number of lines and start multiple processes to translate simultaneously. Then we merge each part of translations to one file in the original order.

4.4   Other Optimizations

In addition to the methods above, we also tried to find the optimal settings for our system.

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 we use greedy search in all submissions.

Better decoding configurations   As mentioned earlier, our GPU versions use a large batch size, but the number on the CPU is much smaller. We use a fixed batch size (number of sentences) of 512 on the GPU and 64 on the CPU. We also set the number of processes on the CPU as 24 and use 2 MKL threads for each process. The maximum sequence length is 120 for the source and 200 for the target.

Profile-guided optimization   To further improve our systems' efficiency, we identified and optimized the performance bottlenecks in our implementation. There are many off-the-shelf tools for performance profiling such as the gprof[2] for C++ and the nvprof[3] for CUDA. We run our systems on the WMT test set for ten times and collect profile data for all functions. Figure 2(a) shows the profiling results for different operations on GPUs before optimizing. Before optimizing, the most time-consuming functions on CPUs is pre-processing and post-processing. We gain 2x speedup on CPUs by using multi-threads for Moses (4 threads) and replacing the Python subword tool with the C++ implementation[4].

For GPU-based decoding, the bottleneck is matrix multiplication and memory management. Therefore we use a memory pool to control allocation/deallocation, which dynamically allocates blocks during decoding and releases them after the translation finished. Compared with the on-the-fly mode, this strategy significantly improves the efficiency of our systems by up to 3x speedup and slightly increases the memory usage. We further remove the $log\_softmax$ in the output layer for greedy search and other data transfers with a slight acceleration of about 10%. Figure 2(b) shows the statistics of optimized operations. The data type conversion overhead takes about 12% of the decoding time.

---

[2]https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_node/gprof_toc.html
[3]http://docs.nvidia.com/cuda/profiler-users-guide/index.html
[4]https://github.com/glample/fastBPE

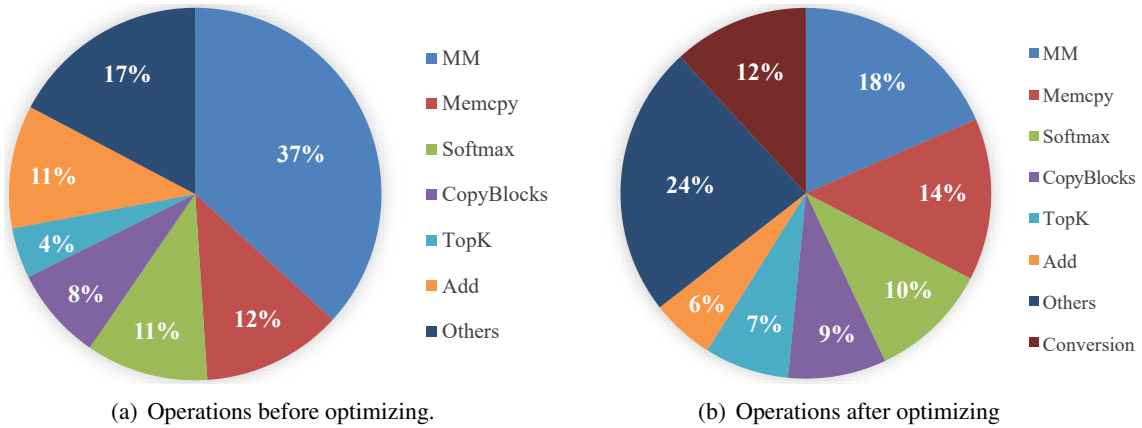(a) Operations before optimizing.

(b) Operations after optimizing

Figure 2: Profiling results of all operations during inference before or after optimizing on *newstest2018* using a 9-1 model on a 2080Ti. We performed decoding for ten times to get more convincing results. Before optimizing, the decoding time is 76.9 seconds. The combination of different optimizations reduces the time to 24.9 seconds. MM is matrix multiplication, and CopyBlocks is used in the tensor copy.

5 Submissions and Results

We submitted five systems to this shared task, one for the CPU track and four for the GPU track, summarized as Table 3. We report file sizes, model architectures, configurations, metrics for translation, including BLEU on *newstest2018* and the real translation time on a combination of test sets. The BLEU and translation time were measured by the shared-task organizers on AWS c5.metal (CPU) and g4dn.xlarge (GPU) instances.

For the GPU tracks, our systems were measured on a Tesla T4 GPU. GPU versions were compiled with CUDA 10.1, and the executable file is about 96 MiB. Our models differ in encoder and decoder layers. The base model (35-6) has 35 encoder layers and six decoder layers and achieves 44.6 BLEU on the *newstest2018*. Then we see a speedup of more than one-third and a slight decrease of only 0.2 BLEU by reducing the decoder layer to 1 (35-1). We continue to reduce the number of encoder layers for more accelerations. The 18-1 system reduces the translation time by one-third with only half of the encoder layers compared to the 35-1 model. Our fastest system consists of 9 encoder layers and one decoder layer, which has one-third parameters of the 35-6 model, achieves 40 BLEU on the WMT 2019 test set, and speeds up the baseline by 3x.

For the CPU track, we used the entire machine, which has 96 virtual cores. Our CPU version is compiled with MKL static library, and the executable file is 22 MiB. We used a tiny model for the CPU with 256 hidden dimensions and kept other

hyper-parameters as the 9-1 model in the GPU version. Interestingly, using half of the hidden size significantly reduces the translation quality. The main reason is that the parameters of large models cannot be reused when using smaller dimensions. This also proves that reducing the number of encoder and decoder layers is a more effective compression method. The CPU system achieves 37.2 BLEU on the *newstest2018* and is 1.2x faster than the fastest GPU system.

We made fewer efforts to reduce the model size and memory footprint. Our systems use a global memory pool, and we sort the input sentences in descending order of length. Thus the memory consumption will reach a peak in the early stage of decoding and then decrease. Our base model contains 152 million parameters, and the file size is 291 MiB when stored in 16-bit floats. The docker image size ranges from 724 MiB to 930 MiB for our GPU systems, while the CPU version is 452 MiB. All systems running in docker are slightly slow down, and we plan to improve this in subsequent versions.

6 Conclusion

To maximize the decoding efficiency while ensuring sufficiently high translation quality, we explored different techniques, including knowledge distillation, model compression, and decoding algorithms. The deep encoder and shallow decoder networks achieve impressive performance in both translation quality and speed. We speed up the decoding by 3x with lightweight models and efficient

Model	MiB	Time	BLEU
Student-35-6	305	3166.4	44.6
Student-35-1	264	2023.3	44.3
Student-18-1	156	1355.0	43.4
Student-9-1	99	977.6	42.9
Student-9-1-tiny[†]	67	810.9	37.2

Table 3: Results of all submissions. [†] indicates the CPU system. All student systems were running with greedy search. The time was measured by the organizers on their test set and we only report the BLEU on the $newstest2018$.

implementations.

For the GPU system, we plan to optimize the FP16 inference by reducing the type conversion and applying kernel fusion (Wang et al., 2010) for Transformer models. For the CPU system, we will further speed up the inference by restricting the output vocabulary to a subset of likely candidates given the source (Shi and Knight, 2017; Senellart et al., 2018) and using low precision data type (Bhandare et al., 2019; Kim et al., 2019; Lin et al., 2020).

Acknowledgements

References

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *ArXiv*, abs/1607.06450.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram A. Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *ArXiv*, abs/1906.00532.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. *ArXiv*, abs/1807.03819.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 56–60, Vancouver. Association for Computational Linguistics.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531.

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. *CoRR*, abs/1412.6980.

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, Kai Feng, Hexuan Chen, Tengbo Liu, Yanyang Li, Qiang Wang, Tong Xiao, and Jingbo Zhu. 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, Florence, Italy. Association for Computational Linguistics.

Yanyang Li, Qiang Wang, Tong Xiao, T Liu, and Jingbo Zhu. 2020. Neural machine translation with joint representation. *ArXiv*, abs/2002.06546.

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*.

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.

Jean Senellart, Dakun Zhang, Bo Wang, Guillaume Klein, Jean-Pierre Ramatchandirin, Josep Crego, and Alexander Rush. 2018. OpenNMT system description for WNMT 2018: 800 words/sec on a single-core CPU. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 122–128, Melbourne, Australia. Association for Computational Linguistics.

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. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

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

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

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

G. Wang, Y. Lin, and W. Yi. 2010. Kernel fusion: An effective method for better power efficiency on multithreaded gpu. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications Int'l Conference on Cyber, Physical and Social Computing*, pages 344–350.

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, Fuxue Li, Tong Xiao, Yanyang Li, Yinqiao Li, and Jingbo Zhu. 2018. Multi-layer representation fusion for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3015–3026, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *ArXiv*, abs/1901.10430.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2019. Improving deep transformer with depth-scaled initialization and merged attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 898–909, Hong Kong, China. Association for Computational Linguistics.