Efficient and High-Quality Neural Machine Translation with OpenNMT

Guillaume Klein, Dakun Zhang, Clément Chouteau, Josep Crego, Jean Senellart

SYSTRAN, 5 rue Feydeau, 75002 Paris, France
`firstname.lastname@systrangroup.com`

Abstract

This paper describes the OpenNMT submissions to the WNGT 2020 efficiency shared task. We explore training and acceleration of Transformer models with various sizes that are trained in a teacher-student setup. We also present a custom and optimized C++ inference engine that enables fast CPU and GPU decoding with few dependencies. By combining additional optimizations and parallelization techniques, we create small, efficient, and high-quality neural machine translation models.

1 Introduction

This paper describes the OpenNMT (Klein et al., 2017) submissions to the Workshop on Neural Generation and Translation 2020 efficiency shared task. For WNMT 2018, we explored training and optimizations of small LSTM translation models combined with a customized runtime (Senellart et al., 2018). While this resulted in interesting decoding speed, there was still room for improvements in terms of quality, memory usage, and overall efficiency.

For this 2020 edition, we focus on the standard Transformer architecture (Vaswani et al., 2017) that is now commonly used in production machine translation systems. Similar to our first participation, we train smaller models using the teacher-student technique (Kim and Rush, 2016). We experiment with several encoder and decoder sizes following the work by Hongfei et al. (2020) which shows that reducing the number of decoder layers can improve decoding speed at a very limited accuracy cost.

We also keep the approach of running the models with a custom C++ runtime. This year we present CTranslate2[1], an optimized and production-grade inference engine for OpenNMT models that enables fast CPU and GPU decoding with few dependencies. This library implements several optimizations for decoding neural machine translation models such as 8-bit quantization, parallel translations, caching, and dynamic target vocabulary reduction.

Section 2 of this paper describes the data preparation and the training procedures we apply to train the candidate models. Section 3 presents the various optimizations we implemented to reduce model size and improve runtime efficiency. Finally, Section 4 details the accuracy and efficiency results achieved by the submitted models.

2 Teacher-student training

We train our systems using a teacher-student approach (Kim and Rush, 2016). First, a large model (the teacher) is trained on all available bilingual data, including synthetic data such as back-translations of monolingual target sentences (Sennrich et al., 2016; Edunov et al., 2018) and translations of monolingual source sentences (Zhang and Zong, 2016). Model ensembles are also typically used to build stronger teacher systems.

Then, a small model (the student) is trained by means of minimizing the loss between the student and teacher systems with the goal of distilling the knowledge of the teacher (Kim and Rush, 2016; Zhang et al., 2018) into a smaller model with comparable accuracy results. Crego and Senellart (2016) show that student models can even outperform to some extent their teacher counterparts.

Knowledge distillation is an effective approach to reduce the model size, thus lowering memory and computation requirements.

2.1 Teacher system

As suggested in the task description and given the limited amount of time available, we use Face-

[1] `https://github.com/OpenNMT/CTranslate2`

book's WMT 2019 system as our teacher model (Ng et al., 2019). The system is trained as an ensemble of big Transformer models for both directions, English-German and German-English. Table 1 shows the BLEU (Papineni et al., 2002) evaluation results of this model over *newstest* public evaluation datasets.

	newstest2018	newstest2019
Facebook WMT 2019	49.1	42.1
Microsoft-Marian	48.3	44.9

Table 1: Evaluation of the teacher system on the English-German *newstest* files as reported by SacreBLEU (Post, 2018). The results for Microsoft-Marian are reported for comparison and retrieved from the WMT matrix[2].

2.2 Training data

We limit our training data to the WMT 2019 English-German translation task[3]. Table 2 summarizes the data provided by the task organizers which consist of more than 38M parallel sentences and 808M monolingual English sentences.

Corpora		# sents
Parallel	Europarl v9	1,838,568
	Common Crawl corpus	2,399,123
	News Commentary v14	338,285
	Wiki Titles v1	1,305,141
	Document-split Rapid	1,531,261
	ParaCrawl v3	31,358,551
	Total	38,770,929
Mono (Eng)	news-crawl 2007-2018	199,900,557
	news-discuss 2011-2018	605,540,239
	europarl-v9	2,295,044
	news-commentary-v14	545,919
	Total	808,281,759

Table 2: English-German parallel data and English monolingual data provided by the WMT 2019 translation task.

We use the following data to be translated by the Facebook's WMT 2019 teacher system: (a) English part of the bilingual data, (b) English part of ParaCrawl v3, and (c) English monolingual data.

Before translation, data is cleaned following several rules: sentences that are empty or longer than 100 tokens without considering tokenization are filtered out. We also use the language identification (LID) toolkit `langid` (Lui and Baldwin, 2012) to further clean ParaCrawl and the English monolingual corpora which are known to contain a large number of noisy sentences. Nearly 5% of the sentences are discarded by the LID toolkit.

The cleaned data is then translated by the teacher model and the resulting synthesized parallel data is used to train the student systems[4].

2.3 Vocabulary

We build a joint subword segmentation model from the synthesized parallel data using SentencePiece (Kudo and Richardson, 2018). The vocabulary size is set to $32,000$ tokens. We removed the non-latin characters before building the vocabulary.

2.4 Student models

We train 4 different student systems based on the Transformer architecture (Vaswani et al., 2017). The candidate configurations are presented in Table 3. In addition to the base Transformer configuration, we train 3 model variants with different number of encoder layers N_{Enc}, decoder layers N_{Dec}, hidden size d_{model}, and feed-forward network size d_{ff}. We share both the source and target word embeddings and softmax weights in the 3 variants while the base configuration considers them as separate weights.

2.5 Student training

Since the amount of synthetic data is relatively large, we define an epoch as a random sampling of 5M sentences. We set the sampling weights of the selected data (a), (b), and (c) to 5, 2, and 2 respectively. That is, we consider a larger number of sentences synthesized from the English part of the bilingual data than from ParaCrawl or from the monolingual English data set.

We use the OpenNMT-tf[5] toolkit to train our student systems. Training is run on a single NVIDIA Tesla V100 GPU with an effective batch size of 25,000 tokens for the early epochs. Just before the final release, we train 10 additional epochs with a larger batch size by increasing the gradient update delay by a factor of 16 (Ott et al., 2018). Figure 1 shows the comparison with a larger batch size. We achieve an additional 0.1 to 0.2 BLEU using this

[2]`http://matrix.statmt.org/`

[3]`http://statmt.org/wmt19/translation-task.html`

[4]Due to the long decoding time of the teacher system, the English monolingual data was partially translated. The final data pool used for training consists of: (a) 7.4M bilingual data, (b) 26.1M ParaCrawl data, and (c) 127M English monolingual data.

[5]`https://github.com/OpenNMT/OpenNMT-tf`

Transformer	N_{Enc}	N_{Dec}	h	d_{model}	d_{ff}	# params	newstest2018	newstest2019
Base	6	6	8	512	2048	93324544	46.7	43.0
(4:3 2xFFN)	4	3	8	256	2048	18221568	43.2	40.8
(6:3)	6	3	8	256	1024	16123904	43.0	40.3
(4:3)	4	3	8	256	1024	14544384	42.0	39.7

Table 3: Transformer configurations and their BLEU scores on *newstest2018* and *newstest2019*. Evaluation is performed without inference optimizations using OpenNMT-tf and a beam size of 4.

technique. Finally, we average the weights of the last 10 checkpoints to produce the final models.

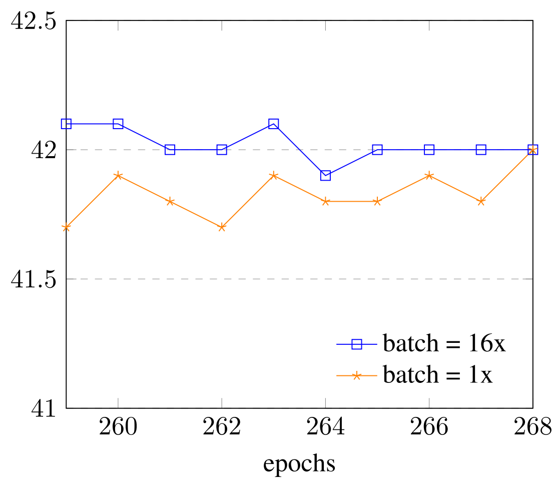

Figure 1: BLEU evaluations on larger batch size on *newstest2018*.

2.6 Evaluation

We list the number of parameters of the 4 trained models in Table 3 and their evaluation scores on the English-German *newstest2018* and *newstest2019* before any inference optimizations. The results correlate well with the expectation that more model parameters lead to better performance. The base Transformer model achieves better results on *newstest2019* than the Facebook's WMT 2019 model used as a teacher (43.0 vs. 42.1). This confirms the finding in Crego and Senellart (2016) that student systems can sometimes outperform their corresponding teacher networks.

3 Inference optimizations

All models are converted and executed with CTranslate2. We use the version 1.10.0 of the library.

3.1 CTranslate2 technical overview

CTranslate2 is a standalone C++ library that implements the complete logic of executing and decoding neural machine translation models with a focus on Transformer variants. This custom implementation supports CPU and GPU execution with the goal of being faster, lighter, and more customizable than a general-purpose deep learning framework. Key features of this project include model quantization, parallel translations, dynamic memory usage, and interactive decoding. Some of these features are difficult to implement effectively with standard deep learning frameworks and are the motivation for this project.

The CPU runtime is backed by Intel MKL, a popular math computation library optimized for Intel processors. We specialize operators with BLAS routines and Vector Mathematical functions whenever possible to benefit from vectorization. We also use the caching allocator provided by `mkl_malloc` and align allocated memory to 64 bytes. Other operations not available in Intel MKL are implemented in plain C++ using the STL and OpenMP.

The GPU runtime minimally requires the cuBLAS and Thrust libraries. Basic transformations are defined using Thrust while more complex layers such as layer normalization and softmax are using CUDA kernels ported from PyTorch (Paszke et al., 2019). We also integrate a caching allocator from the CUB library to reuse previously allocated buffers and minimize device synchronization.

3.2 8-bit quantization (CPU)

Quantization is a standard technique to reduce the model size in memory and accelerate its execution. We quantize the weights of linear and embedding layers to 8-bit signed integers after completing training. Experimental results show that model quantization can achieve high translation accuracy without making the training quantization-aware. We use the equation from Wu et al. (2016) to compute the quantized weight W^Q from the original weight W:

Quantization	Model size
None	373MB
16-bit	187MB
8-bit	94MB

Table 4: Effect of weight quantization on the model size on disk. The model is a base Transformer without shared embeddings.

$$
\begin{aligned}
s_i &= \max_j |W_{i,j}| \\
W^Q_{i,j} &= \left\lfloor \frac{127}{s_i} W_{i,j} \right\rceil
\end{aligned}
\tag{1}
$$

Table 4 shows the effect of weight quantization on the final model size.

On CPU, we dynamically quantize the input of the linear layer using Equation 1, multiply the quantized input and weight with MKL's `cblas_gemm_s8u8s32` function, and dequantize the result before adding the bias term. In addition, we employ two notable techniques:

Weights pre-packing. On model load, we replace the quantized linear weights with the packed representation returned by MKL's packed GEMM API.

Unsigned compensation term. In row major mode, Intel MKL expects the input matrix a to be unsigned while the quantization Equation 1 produces signed values. To overcome this constraint, we shift a to the 8-bit unsigned domain and add a compensation term c to the output matrix. This compensation term only depends on the quantized weight matrix and can be computed once:

$$
c_i = -128 \times \sum_{j=1}^{k} W^Q_{i,j} \tag{2}
$$

On GPU, 8-bit computation is disabled as our implementation still requires some efficiency improvements regarding repetitive quantization and dequantization. In this case the weights are dequantized on load to single precision floating points.

3.3 Greedy decoding

To maximize speed and reduce memory usage, we use greedy search instead of beam search. During decoding, we also skip the final softmax layer and simply get the maximum from the output logits.

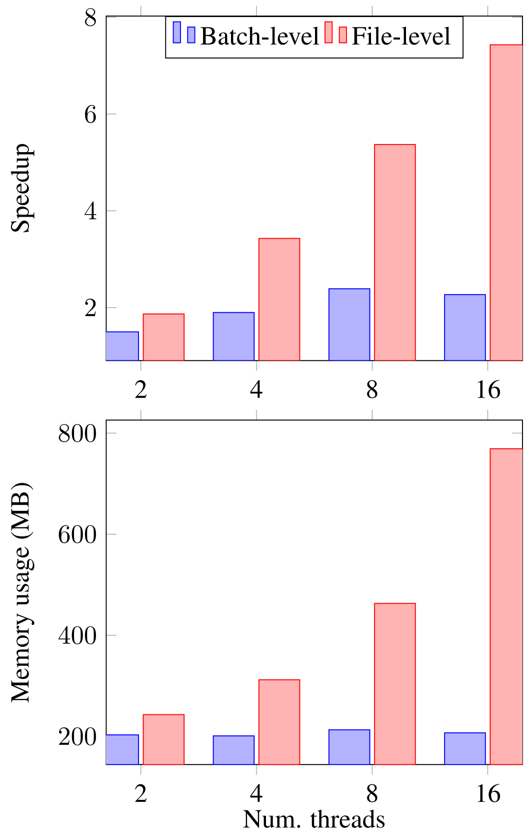

Figure 2: Speedup and memory usage for a base Transformer model when increasing the number of threads for batch translation, either at the batch level (left blue bars) or at the file level (right red bars).

3.4 Decoder projections caching

We apply the common technique of caching linear projections in the Transformer decoder layers. In particular, at step t the decoder self-attention layers compute $Attention(Q_t W^Q, Q_{1..t} W^K, Q_{1..t} W^V)$. As the matrix $Q_{1..t-1}$ is constant, we only compute $Q_t W^K$ and $Q_t W^V$ and concatenate the results to previous projections before calling the attention.

We also cache the encoder output projections KW^K and VW^V in the encoder-decoder attention layers as K and V remain constant during decoding.

For both cases, we transpose the matrices to delimit the attention heads before saving them in the cache.

3.5 File-level parallelism (CPU)

Figure 2 compares the observed speedup when increasing the number of threads at the batch level–the number of OpenMP threads–or at the file level–the number of batches processed in parallel. We use the same batch size in both cases.

As the number of threads increases, the first approach looses efficiency because not all operators within the model scale linearly and some of them are not parallelized at all. On the other hand, the second approach continues to improve as we add more threads because all batches are independent and the full decoding can be executed in parallel. However, the duplicated internal state of parallel translators increases memory usage. To mitigate this issue, we share the static model data among all parallel translators and read and write batches in a streaming manner while ensuring that the original order is preserved.

Given the large number of CPU cores available for this task, we chose to exploit parallelism at the file level to maximize the overall throughput. The number of parallel translators is set to the number of physical cores. Each translator is using a single thread so the decoding algorithm is executed sequentially and without OpenMP.

3.6 Sorted and dynamic batches

When setting the maximum batch size to N tokens, each consumer reads $8N$ contiguous tokens, sorts the sentences from the longest to the shortest, and then splits by batch of N tokens before running the model. The correct order is restored when returning the translation results. This local sorting makes the batches contain sentences of similar sizes which reduces the amount of padding and increases the computation efficiency.

We use $N = 6000$ for the GPU task, $N = 512$ for the single-core CPU task, and $N = 256$ for the multi-core CPU task.

During decoding we remove finished translations from the batch to avoid unnecessary computation. We also exploit the prior knowledge that short sentences finish early: by moving shorter sentences at then end of the batch, we reduce memory copies when updating the decoder cache in place.

3.7 Target vocabulary reduction

We generate a static source-target vocabulary mapping using the technique described in Senellart et al. (2018). We first train an alignment model with `fast_align` to align source and target words. To increase the coverage of this mapping, we build a phrase table from these alignments to extract the N-best translation hypotheses of 1-gram, 2-gram, ..., n-gram source sequences and include all target words in the mapping. We set $n = 3$ to generate the vocabulary mapping that are included in the models of this submission.

	Speed	BLEU
OpenNMT-tf	214.0	26.00
CTranslate2	242.5	26.00
+ int8	845.2	25.88
+ local sorting	1054.0	25.88
+ packed GEMM	1167.4	25.84
+ vocabulary reduction	1687.1	25.42

Table 5: Single-core greedy decoding speed (target tokens per second) for a base Transformer model. The BLEU scores are computed on an undisclosed test set and show the impact on quality (if any) of the enabled optimization.

During decoding, we consider all 1-gram, 2-gram, and 3-gram sequences in the input batch and select the target tokens that are likely to appear in the translation according to the pretrained mapping as well as the 50 most frequent target tokens. These candidates are used to mask the weights of the final linear layer and effectively reduce its computational cost.

3.8 Docker images

The Docker images entrypoint is a small C++ main function that wraps the CTranslate2 and SentencePiece libraries and sets the decoding options that are relevant for this task.

We submit separate Docker images for CPU and GPU to only include the required dependencies. The images are based respectively on `ubuntu:18.04` and `nvidia/cuda:10.2-base-ubuntu18.04`. Without the model, the CPU image size is 104MB and the GPU image size is 210MB.

4 Optimization results

Table 5 shows the impact of selected optimizations when decoding a base Transformer model on a single CPU core. The CTranslate2 library combined with few optimizations can lead to a $8\times$ speedup with limited accuracy loss over a baseline TensorFlow program.

Finally, Table 6 summarizes the global impact of the optimizations described above that we compare against a baseline beam search decoding with OpenNMT-tf. For a base Transformer model, single-core CPU translation is $13\times$ faster while only loosing 0.8 BLEU points and GPU translation is $7\times$ faster for the same quality.

Transformer variant	Time (s)	BLEU
Baseline (single-core CPU)		
Base	522.1	43.0
(4:3 2xFFN)	251.5	40.8
(6:3)	238.7	40.3
(4:3)	238.0	39.7
Optimized (single-core CPU)		
Base	39.5	42.2
(4:3 2xFFN)	11.2	39.8
(6:3)	10.1	39.5
(4:3)	8.8	38.7
Optimized (multi-core CPU)[6]		
Base	5.2	42.0
(4:3 2xFFN)	2.5	39.7
(6:3)	2.5	39.3
(4:3)	2.3	38.5
Baseline (GPU)		
Base	57.6	43.0
(4:3 2xFFN)	40.7	40.8
(6:3)	41.6	40.3
(4:3)	42.1	39.7
Optimized (GPU)		
Base	7.7	43.0
(4:3 2xFFN)	4.0	40.1
(6:3)	3.9	39.9
(4:3)	3.8	39.0

Table 6: Time in seconds to translate *newstest2019* and BLEU scores as returned by SacreBLEU. The time includes model loading and tokenization. *Baseline* models are decoded with OpenNMT-tf using a beam of size 4; *Optimized* models are decoded with the final images submitted for this task. The runs were executed on a *c5.metal* AWS instance for CPU and a *g4dn.xlarge* instance for GPU.

5 Conclusion

We demonstrated that the OpenNMT ecosystem can be used to train efficient and high-quality neural machine translation models. The training frameworks–OpenNMT-tf and OpenNMT-py–include all features and procedures that are commonly applied to reach competitive translation scores. This year we presented CTranslate2, an optimized and production-grade inference engine for OpenNMT models that enables fast CPU and GPU decoding with few dependencies. By combining several optimizations and parallelization techniques, the library can drastically improve decoding speed and reduce memory usage over a general-purpose deep learning toolkit.

[6]The difference in BLEU score with the single-core runs comes from the smaller batch size which changes the candidates selected for reducing the target vocabulary.

References

Josep Maria Crego and Jean Senellart. 2016. Neural machine translation from simplified translations. *CoRR*, abs/1612.06139.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium. Association for Computational Linguistics.

Xy Hongfei, Deyi Xiong, Joseph van Genabith, and Liu Qiuhui. 2020. Analyzing word translation of transformer layers. *arXiv preprint arXiv:2003.09586v1*.

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.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Marco Lui and Timothy Baldwin. 2012. langid.py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 System Demonstrations*, pages 25–30, Jeju Island, Korea. Association for Computational Linguistics.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook fair wmt19 news translation task submission. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 314–319, Florence, Italy. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of* *the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary 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 H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

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, Belgium, Brussels. 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. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Dakun Zhang, Josep Crego, and Jean Senellart. 2018. Analyzing knowledge distillation in neural machine translation. In *15th International Workshop on Spoken Language Translation*.

Jiajun Zhang and Chengqing Zong. 2016. Exploiting source-side monolingual data in neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545, Austin, Texas. Association for Computational Linguistics.