# An in-depth Study of Neural Machine Translation Performance

Simla Burcu Harma[1], Mario Drumond[2], Babak Falsafi[2], and Oğuz Ergin[1]

[1] TOBB University of Economics and Technology / Söğütözü, Söğütözü Cd. No:43, Technology Center, 217. 06510 Çankaya/Ankara
s.harma@etu.edu.tr oergin@etu.edu.tr
https://www.kasirgalabs.com/en
[2] École polytechnique fédérale de Lausanne / EPFL-Faculté IC Parallel Systems Architecture Lab (PARSA) INJ 234 - Station 14 1015 Lausanne
mario.drumond@epfl.ch babak.falsafi@epfl.ch
https://parsa.epfl.ch

**Abstract.** With the rise of deep learning and rapidly increasing popularity of it, neural machine translation (NMT) has become one of the major research areas. Sequence-to-sequence models are widely used in NTM tasks, and one of the state-of-the-art models, the Transformer, has also encoder-decoder architecture with an additional attention mechanism. Despite a substantial amount of research in improving NMT models' translation qualities and speeds, to the best of our knowledge, none of them gives a detailed performance analysis of each step in a model. In this paper we analyze the Transformer model's performance and translation quality in different settings. We conclude that beam search is the bottleneck of the NMT inference and analyze beam search's effect on the performance and quality in detail. We observe that the beam size is one of the largest contributors to the Transformer's execution time. Additionally, we observe that the beam size only affects BLEU score at word level, and not at token level, indicating a mismatch between the beam search internal evaluation metrics and the end-to-end metrics used to evaluate models. We also show that the vocabulary size has a major role in the performance of the beam search.

**Keywords:** Neural machine translation · Performance analysis · Beam search.

## 1 Introduction

Machine translation has become a pivotal datacenter workload. As more online services cater for uses distributed worldwide, processing user generated text and generate more text as a result, machine translation is part of most online services [4]. Following the advances in neural networks and natural language processing, neural machine translation (NMT) has become the most successful machine translation method. Unfortunately, neural machine translation computational requirements are orders of magnitude higher than traditional online

services, forcing service providers to increase the compute power of their infrastructure.

Microservices [5] and acceleration [2,4,7] are the two main techniques used by service providers to cope with the computational requirements of DNN services like machine translation. Microservices shift services into a group of loosely-coupled smaller rather than monolithic services, in order to satisfy the performance, availability and flexibility constraints. Service providers also adopt accelerators like GPUs, FPGAs or TPUs in order to improve energy efficiency, enabling a more load to be accommodated under the same power constraints.

The throughput observed of NMT services, therefore, has a large impact over how online service providers design their systems. Unfortunately, the computational performance of NMT services has a poorly understood, non-trivial relationship to the quality of the translation generated. Each model feature are a large number of parameters that affect both throughput and accuracy. While prior work has partially explored the design space formed by such parameters, there are no in-depth studies allow for a better understanding of the relationship between quality and throughput.

In this work, we present an in-depth study of NMT performance, making the following contributions:

- We profile a inference NMT microservice based on OpenNMT [10], showing that the majority of processing time is dedicated to the NMT system, with negligible time spent in external communication
- We provide a breakdown of the execution time in CPU and GPU bound NMT systems. We show how much time is dedicated to each of the components.
- We study the effect of each of the major system parameters over throughput.

## 2 A primer on neural machine translation

Sequence-to-sequence models map varying sizes of fixed-length inputs to fixed-length outputs (Sutskever, 2014) for tasks like machine translation, speech recognition, and video captioning. These models have two parts: the encoder, witch takes an input sequence and produces an intermediate vector, and the decoder, which takes that intermediate vector as input and outputs the target sequence.

One of the state-of-the-art models for NMT is the Transformer [20]. The Transformer has the encoder-decoder structure. Both the encoders and decoders consist of stacked self-attention and point-wise, fully connected feed forward neural network layers (see Figure 1).

Both the encoding and decoding components are seen in Figure 1, composed by stacks of $N$ layers each. The encoder processes sentences in groups of tokens while the decoder take the output of the encoder and the previous outputs as inputs and outputs a group of token. The $N$ parameter is crucial to both throughput and accuracy. As $N$ increases, the translation time should increase not linearly but close to the linear trend.

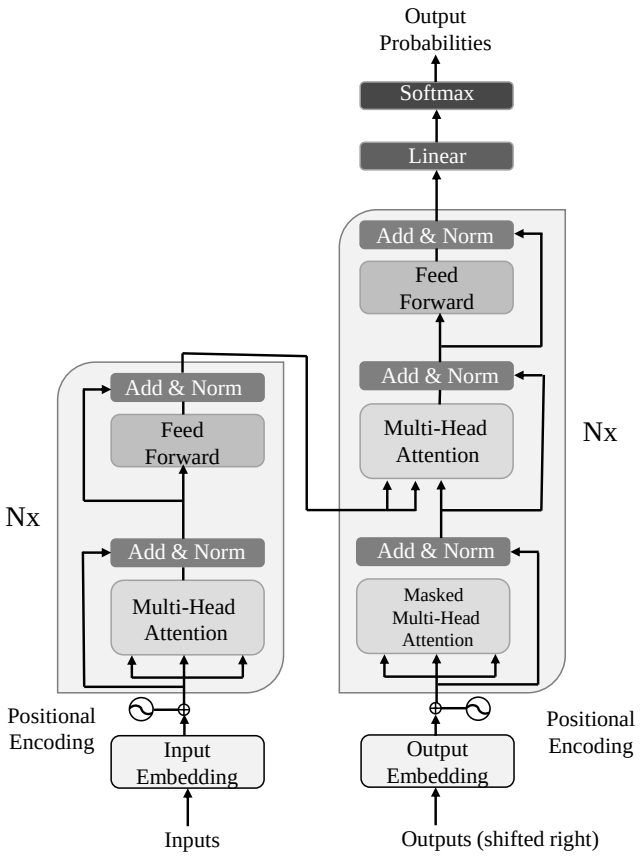


**Fig. 1.** Transformer model architecture

The self-attention mechanism allows the model to associate the words with each other when needed (for example finding which word "it" refers to in a sentence). As seen in Figure 1, Transformer uses Multi-Head Attention which means there are multiple attention heads ($h$), each focuses on a different position by using different parameters. This provides information from different representation subspaces at different positions for the model. The number of heads $h$ also introduces a throughput vs. accuracy trade-off.

The inputs to the transformer model go through pre-processing before execution start. First, words are converted into tokens in a process called tokenization and then tokens are mapped into a vectorial space to generate embeddings. The encoder take embeddings as inputs and the decoder generates them as outputs. The last linear and softmax layers of the transformer converts the embeddings probabilities for each token in the vocabulary. These layers are called Generator layers. After that, the model searches through all the possible output sequences in order to find the most likely sequence.

However, the vocabulary has tens of thousands of tokens and the search problem is exponential in the sequence length. One approach to overcome this problem is greedy search, where the token with the highest probability is chosen every time. Another approach, most commonly used, is the beam search, expanded version of the greedy search, which keep the $k$ most likely tokens at each step. It is a sequential process that generates the output token-by-token keeping the $k$ active candidate tokens at each step and cumulatively adds their probabilities to build paths. Beam search introduces two sources of inefficiencies. First, it incurs unnecessary work because the decoder has to process $k$ times more tokens than necessary. Second, it is an irregular sequential process that combines dense matrix multiplications with control heavy decisions after each token is generated. As such, it complicates the use of accelerators, as it forces the execution flow to go in and out of the accelerator multiple times during a single inference pass.

In this paper, we perform a detailed time analysis of the Transformer model, and show that beam search is the bottleneck. In order to explain the effect of beam search on the performance and the translation quality, we perform a number of experiments. We also show the importance of the vocabulary size on the performance of the beam search.

## 3 Methodology

We analyze the OpenNMT-py Transformer implementation, which is a PyTorch port of OpenNMT, an open-source neural machine translation system. We use OpenNMT-py's pretrained English-German translation model: Base Transformer configuration with standard training options. The model is trained with WMT'17 train and validation data with shared SentencePiece, which is a tokenizer implementing subword units (e.g., byte-pair-encoding (BPE) [17] and unigram language model [12] We conduct our experiments on English-German neural machine translation task using WMT'17 test data (newstest2017) on Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz and NVIDIA GeForce GTX TITAN Xp GPU. We use only one dataset since the dataset used does not affect performance. The dataset consists of 3004 sentences/lines and all the experiments, except the one reported in Table 1, are run with the complete dataset.

## 4 Evaluation and Results

### 4.1 Transformer with Different Configuration Values

We built a DNN microservice based on OpenNMT and measured end to end performance. We observed that the network communication time is small compared to the model execution time, and does not become a bottleneck. As such, the bottleneck for these systems is the neural network model itself. Thus, in order to speed up an NMT microservice one needs to focus on the NMT model.

We analyze the Transformer model in detail and in this scope, we first perform the time analysis of the different configurations of Transformer in OpenNMT-py

**Table 1.** Transformer's BLEU scores and translation times with different configuration values for 100 lines of input text

| 1 | | $N$ | $d_{model}$ | $d_{ff}$ | $h$ | $d_k$ | $d_v$ | BLEU | 100 lines |
|---|---|---|---|---|---|---|---|---|---|
| 2 | base | 6 | 512 | 2048 | 8 | 64 | 64 | 25.8 | 43.816 s |
| 3 | Change in | | | | 1 | 512 | 512 | 24.9 | 27.371 s |
| 4 | number of | | | | 4 | 128 | 128 | 25.5 | 52.202 s |
| 5 | attention | | | | 16 | 32 | 32 | 25.8 | 60.981 s |
| 6 | heads | | | | 32 | 16 | 16 | 25.4 | 73.278 s |
| 7 | Change in | 2 | | | | | | 23.7 | 26.857 s |
| 8 | number of | 4 | | | | | | 25.3 | 35.845 s |
| 9 | layers | 8 | | | | | | 25.5 | 94.582 s |
| 10 | | | 256 | | | 32 | 32 | 24.5 | 37.398 s |
| 11 | Change in | | 1024 | | | 128 | 128 | 26 | 96.560 s |
| 12 | model size | | | 1024 | | | | 25.4 | 60.178 s |
| 13 | | | | 4096 | | | | 26.2 | 64.702 s |

on CPU and compare the results with the different parameters for inference with 100 lines of input text (see Table 1). Here, the model is re-trained for each of the configurations. Vaswani et. al. [20] reported the change in BLEU scores with different configuration values in the Transformer model. However, to the best of our knowledge there is no work on the performance of these configurations.

The translation time increases almost linearly as $N$ increases as expected, until $N = 8$, then there is a sharp increase (see Figure 2). The same trend is also observed for the GPU (see Table 2).

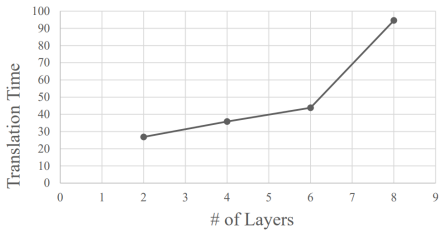


**Fig. 2.** Translation Time vs. Number of layers

**Table 2.** The change in model size with the number of layers

| **N** | **100 lines in GPU** | **Params ($\times 10^6$)** | **Model size(KB)** |
|---|---|---|---|
| 2 | 8.105 s | 36 | 137.330 |
| 4 | 8.590 s | 50 | 190.735 |
| 6 | 8.710 s | 65 | 247.955 |
| 8 | 14.156 s | 80 | 305.176 |

The reason behind this is that the model size increases with $N$ (see Table 2). The cache sizes in the computer used for the experiments are L1: 32K, L2: 256K, L3: 30720K. So, until N=6 the model fits in L2, but for N=8 it needs to go to L3 and it results in a large run-time increase.

Table 1 shows that decreasing the number of attention heads (h) results in a high performance gain and not much quality loss (see line 3). We also observe that when we change the number of layers ($N$), the BLEU decreases, but there are large performance gains (see lines 2, 9 and 10).

Model performance is affected by the change of configuration values, requiring a detailed analysis of the whole model to understand its behavior. Thus, we obtain the time breakdown of the entire model with the Base Transformer on both CPU and GPU.

### 4.2 Transformer Performance in Detail

As shown in the Table 3, decoder takes 60% of the total translation time in CPU, and 41% in GPU. The generator and beam search together take 32% of the total translation time in CPU, and 45% in GPU. The decoder takes a smaller fraction of execution time on the GPU because it is composed mostly of large matrix multiplications. However, beam search is a sequential process as stated before and the time portion of beam search on GPU is larger.

**Table 3.** Transformer's time breakdown for each layer in CPU and GPU with 3004 lines of text.

| | **Layer** | **CPU** | **CPU Fraction** | **GPU** | **GPU Fraction** |
|---|---|---|---|---|---|
| **Encoder** | Multi-Head Attention | 7 s | 0.56% | 1.8 s | 0.33% |
| | Normalization | 1.1 s | 0.09% | 0.2 s | 0.04% |
| | Feed Forward NN | 7.5 s | 0.60% | 0.8 s | 0.15% |
| **Decoder** | Masked Multi-Head Attention | 358 s | 28.70% | 87.3 s | 16.22% |
| | Normalization | 23.8 s | 1.91% | 11.9 s | 2.21% |
| | Multi-Head Attention | 242.9 s | 19.47% | 63.3 s | 11.76% |
| | Normalization | 22.8 s | 1.83% | 12.1 s | 2.25% |
| | Feed Forward NN | 102.3 s | 8.20% | 48 s | 8.92% |
| **Generator** | Generator | 119.3 s | 9.56% | 2 s | 0.37% |
| **Beam Search** | Beam Search | 276 s | 22.12% | 238 s | 44.22% |
| **TOTAL** | | 1247.6 s | | 538.19 s | |

### 4.3 Beam Search

The beam search process includes the following steps: When beam search is applied to the translation of a sentence, we only have the token probabilities in the first iteration. Beam search chooses the $k$ tokens with the highest probabilities. Starting from the second iteration, beam search adds the current token probabilities to the previous sum, then operates the same on the new scores.

Unfortunately, beam search presents three performance bottlenecks. First, it is a sequential process, that forces examples to be processed one by one. Second, it incurs a lot of extra unnecessary work like decoding words that won't be used in the end. Finally, it is irregular, and thus, it either requires implementation

in high level (slow) languages or it has high engineering costs if implemented in low level (fast) languages.

For these reasons, beam search is the bottleneck of the model. Improving other parameters result in modest throughput increase, while beam search incurs a 5× throughput decrease (when the beam size is 5). So we investigate the beam search process further.

### 4.4 Beam Size vs. Performance & BLEU Scores

Increasing the beam size results in longer translation time, but also affects the translation quality. In order to see the relation, we vary the beam size, and measure the translation times and evaluate BLEU scores in both word-level (after we detokenize the output) and token-level (without any detokenization) (Table 4). The beam size has a high impact on performance but a small impact over quality. Additionally, while beam size affects the BLEU score calculated using words, it does not affect the BLEU score calculated with tokens. We theorize that because beam search optimizes results at token level (i.e. it maximizes the likelihood of the model predicting the correct tokens), it requires smaller beam sizes to reach better results. Unfortunately, translation quality is measured with word level BLEU. As such, the beam search process does not have the means to properly search the sentence space, as it uses incomplete information to make decisions. We believe this insight is important as it indicates that a word focused beam search could reach better word-level BLEU scores.

**Table 4.** Beam size vs. performance in CPU & BLEU scores for 3004 lines of text

| Beam Size | Translation Time | Word-level BLEU | Token-level BLEU |
|---|---|---|---|
| 2 | 665.51 s | 27.58 | 33.77 |
| 3 | 856.17 s | 27.88 | 33.93 |
| 4 | 1087.96 s | 27.99 | 33.99 |
| 5 | 1247.60 s | 28.09 | 33.97 |
| 10 | 2092.67 s | 28.07 | 33.69 |
| 15 | 2931.36 s | 27.83 | 33.37 |

In addition to these, we also measured line-by-line translation times for 3004 lines of text. The results show that the translation time for a sentence is dominated by the lengths, in number of tokens, of the target or predicted sentences, whichever is longest. When we divide the sentence translation times to sentence lengths, the average translation time of a token, is generally around 0.02 for the longer one. The translation time of a token in target sentences, in predicted sentences and the minimum of these two are usually around 0.02. The number of sentences having more than 0.025 ratio are 74 for target sentences, 327 for predicted sentences, and 39 for the minimum of the two.

### 4.5 Vocabulary Size in Beam Search

As shown above, beam search is the bottleneck of the neural machine translation. There has been a significant amount of research on how to speedup this process. One popular method is reducing the vocabulary size [16, 19]. Using a smaller vocabulary results in faster translation but the current research does not provide the reason behind this.

We know that beam search uses the generator's output of probabilities, adds these probabilities to the previous cumulative sums on the active paths, and chooses the top $k$. However, choosing the top k tokens from a huge list takes a long time and so do all the operations containing the vocabulary. In order to understand this better we provide time breakdowns of both cases.

A pseudocode of the important steps of beam search in OpenNMT-py for Transformer and its line-by-line time analysis are given in Table 5.

**Table 5.** Beam search time breakdown in CPU with 3004 lines of text. "Subvocab" and "Vocab" stand for "Subvocabulary" and "Vocabulary" respectively

| *Pseudocode* | *Subvocab* | *Vocab* |
|---|---|---|
| If beginning of the sentence: | | |
| beam_scores are the token probabilities | 0.01 s | 0.01 s |
| Else: | | |
| Add the current token probabilities to beam_scores which are the sum of the previous scores for that path | 4.1 s | 11.3 s |
| If there is EOS, stop the path there and don't let it have children | 11.4 s | 13.3 s |
| Find the top k tokens with highest probabilities | 15.8 s | 184.9 s |
| Find and store the indices of these tokens in the vocabulary | 13.9 s | 3.4 s |
| Update the beam search path information stored | 15.5 s | 17.4 s |
| If one of the top k tokens is EOS add this path to the finished path | 12.3 s | 12.8 s |
| *TOTAL BEAM SEARCH TIME* | 82.73 s | 252.78 s |
| *BLEU SCORE* | 17.42 | 28.09 |

When the full vocabulary is used, the most time-taking part is the 3rd step since the vocabulary contains tens of thousands of words/tokens (184.9 s of total 252.78 s). We produce a subvocabulary by using MUSE (Multilingual Unsupervised or Supervised word Embeddings) [14]. MUSE aligns two vector spaces of two languages in order to build a bilingual dictionary. When the input sentence comes, for each word in the input, the trained MUSE model finds the closest n words in the target language's aligned vector space and creates a subvocabulary. This way, the vocabulary size can be reduced to a hundred or less, then this part takes only 15.8 s of total 82.73 s for a test input with 3004 sentences. The time breakdown for the fundamental beam search steps is shown in Table 5(intermediate processes are skipped):

Using a subvocabulary, we obtain 3× speedup in beam search, however, the subvocabulary affects the translation quality.

## 5 Related Work

Beside proposing new models in NMT, many researchers focus on analyzing and improving the existing systems. Junczys-Dowmunt et. al. [8] compared several translation directions on different state-of-the-art systems and developed an efficient NMT decoder. Niehues et. al. [15] separated the modeling and the search space, and analyzed the existing systems in order to measure their sufficiency on the translation quality. Vaswani et. al.[20]'s Transformer is one of the state-of-the-art and the most-studied models in the literature.

There has been extensive work on the analysis, speeding up and improving the translation quality of the Transformer [1, 9, 11, 13, 18] listed main challenges for NMT models which are also applied to the Transformer. They showed that increasing the beam size decreases the translation quality after some point. Focusing on this problem, Yang et. al. [21] analyzed several variations of the beam search and they developed a method increasing the BLEU score for the large beam sizes. Another version of the beam search has been developed by Huang et. al. [6], increasing the BLEU score on Chinese-English translation task.

In addition to its effect on translation quality, Freitag et. al. [3] worked on increasing the performance. They proposed several pruning techniques on beam search and they speed-up their decoder by 43%. Shi et. al. [19] pointed out the importance of the size of the vocabulary used in the translation in order to speed up NMT on GPUs. They offered the word alignment method to shrink the vocabulary and got 2x speedup on the decoder. Senellart et. al. [16] worked on speeding up the Transformer in OpenNMT's Tensorflow port by using several methods including Shi et. al.'s solution.

## 6 Conclusion

In this work, we first built a DNN microservice around Vaswani et. al.'s NMT model Transformer in order to see the bottlenecks. We observed that due to the small input size, the network communication is not the bottleneck, so we need to speed up the model itself in order to speed up this DNN microservice. In order to understand what affects the model, we ran the Transformer with different configuration values and saw that the performance can easily change with the selection of configuration values. After that we performed a detailed time analysis of the Transformer to have a better understanding of its behaviour. We observed that the beam search is the bottleneck and we analyzed the time requirements for each step in the beam search. We also showed that the translation quality metric BLEU score is insensitive to beam size when it is measured in token-level. When a small-sized vocabulary used, the beam search can have 3x speedup however the choice of this subvocabulary is important since it affects translation quality
.

## References


1. Chen, M.X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Schuster, M., Shazeer, N., Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Chen, Z., Wu, Y., Hughes, M.: The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics. pp. 76–86 (2018)
2. Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., Heil, S., Patel, P., Sapek, A., Weisz, G., Woods, L., Lanka, S., Reinhardt, S.K., Caulfield, A.M., Chung, E.S., Burger, D.: A Configurable Cloud-Scale DNN Processor for Real-Time AI. In: Proceedings of the 45th International Symposium on Computer Architecture (ISCA). pp. 1–14 (2018)
3. Freitag, M., Al-Onaizan, Y.: Beam Search Strategies for Neural Machine Translation. In: First Workshop on Neural Machine Translation. pp. 56–60 (2017)
4. Hazelwood, K.M., Bird, S., Brooks, D.M., Chintala, S., Diril, U., Dzhulgakov, D., Fawzy, M., Jia, B., Jia, Y., Kalro, A., Law, J., Lee, K., Lu, J., Noordhuis, P., Smelyanskiy, M., Xiong, L., Wang, X.: Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In: Proceedings of the 24th IEEE Symposium on High-Performance Computer Architecture (HPCA). pp. 620–629 (2018)
5. Hoff, T.: Lessons learned from scaling Uber to 2000 engineers, 1000 services, and 8000 Git repositories. http://highscalability.com/blog/2016/10/12/lessons-learned-from-scaling-uber-to-2000-engineers-1000-ser.html, (Date last accessed 16-Aug-2019)
6. Huang, L., Zhao, K., Ma, M.: When to Finish? Optimal Beam Search for Neural Text Generation (modulo beam size). In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 2134–2139 (2017)
7. Jouppi, N.P., Young, C., Patil, N., Patterson, D.A., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., luc Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T.V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C.R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., Yoon, D.H.: In-Datacenter Performance Analysis of a Tensor Processing Unit. In: Proceedings of the 44th International Symposium on Computer Architecture (ISCA). pp. 1–12 (2017)
8. Junczys-Dowmunt, M., Dwojak, T., Hoang, H.: Is Neural Machine Translation Ready for Deployment? A Case Study on 30 Translation Directions. CoRR **abs/1610.01108** (2016)
9. Kaiser, L., Bengio, S., Roy, A., Vaswani, A., Parmar, N., Uszkoreit, J., Shazeer, N.: Fast Decoding in Sequence Models Using Discrete Latent Variables. In: Proceedings of the Thirty-fifth International Conference on Machine Learning (ICML). pp. 2395–2404 (2018)
10. Klein, G., Kim, Y., Deng, Y., Senellart, J., Rush, A.M.: OpenNMT: Open-Source Toolkit for Neural Machine Translation. In: ACL (System Demonstrations). pp. 67–72 (2017)

11. Koehn, P., Knowles, R.: Six Challenges for Neural Machine Translation. In: First Workshop on Neural Machine Translation. pp. 28–39 (2017)
12. Kudo, T.: Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics. pp. 66–75 (2018)
13. Lakew, S.M., Cettolo, M., Federico, M.: A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation. In: Proceedings of the 27th International Conference on Computational Linguistics. pp. 641–652 (2018)
14. Lample, G., Conneau, A., Ranzato, M., Denoyer, L., Jgou, H.: Word translation without parallel data. In: ICLR (Poster) (2018)
15. Niehues, J., Cho, E., Ha, T.L., Waibel, A.: Analyzing Neural MT Search and Model Performance. In: First Workshop on Neural Machine Translation. pp. 11–17 (2017)
16. Senellart, J., Zhang, D., Wang, B., Klein, G., Ramatchandirin, J.P., Crego, J.M., Rush, A.M.: OpenNMT System Description for WNMT 2018: 800 words/sec on a single-core CPU. In: Second Workshop on Neural Machine Translation. pp. 122–128 (2018)
17. Sennrich, R., Haddow, B., Birch, A.: Neural Machine Translation of Rare Words with Subword Units. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (2016)
18. Shazeer, N., Stern, M.: Adafactor: Adaptive Learning Rates with Sublinear Memory Cost. In: Proceedings of the Thirty-fifth International Conference on Machine Learning (ICML). pp. 4603–4611 (2018)
19. Shi, X., Knight, K.: Speeding Up Neural Machine Translation Decoding by Shrinking Run-time Vocabulary. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. pp. 574–579 (2017)
20. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is All you Need. In: Proceedings of the Thirty-first Conference on Neural Information Processing Systems (NIPS). pp. 5998–6008 (2017)
21. Yang, Y., Huang, L., Ma, M.: Breaking the Beam Search Curse: A Study of (Re-)Scoring Methods and Stopping Criteria for Neural Machine Translation. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. pp. 3054–3059 (2018)