Efficient Machine Translation with Model Pruning and Quantization

Maximiliana Behnke[†] **Nikolay Bogoychev**[†] **Alham Fikri Aji**[†] **Kenneth Heafield**[†]
Graeme Nail[†] **Qianqian Zhu**[†] **Svetlana Tchistiakova**[†] **Jelmer van der Linde**[†]
Pinzhen Chen[†] **Sidharth Kashyap**[‡] **Roman Grundkiewicz**[†§]
[†]University of Edinburgh [‡]Intel Corporation [§]Microsoft
{maximiliana.behnke, n.bogoych, a.fikri, kenneth.heafield, graeme.nail, qianqian.zhu,
stchisti, jelmer.vanderlinde, pinzhen.chen, rgrundki}@ed.ac.uk,
sidharth.n.kashyap@intel.com

Abstract

We participated in all tracks of the WMT 2021 efficient machine translation task: single-core CPU, multi-core CPU, and GPU hardware with throughput and latency conditions. Our submissions combine several efficiency strategies: knowledge distillation, a simpler simple recurrent unit (SSRU) decoder with one or two layers, lexical shortlists, smaller numerical formats, and pruning. For the CPU track, we used quantized 8-bit models. For the GPU track, we experimented with FP16 and 8-bit integers in tensorcores. Some of our submissions optimize for size via 4-bit log quantization and omitting a lexical shortlist. We have extended pruning to more parts of the network, emphasizing component- and block-level pruning that actually improves speed unlike coefficient-wise pruning.

1 Introduction

This paper describes the University of Edinburgh's submission to Sixth Conference on Machine Translation (WMT2021) Efficiency Task[1], which measures performance on latency and throughput on both CPU and GPU, in addition to translation quality. Our submission focused on the trade-off between these metrics and quality.

Our submission builds upon the work of last year's submission (Bogoychev et al., 2020). We trained our models in a teacher-student setting (Kim and Rush, 2016), using Edinburgh's En-De system submitted to the WMT2021 news translation task as the teacher model. For the students, we used a Simpler Simple Recurrent Unit (SSRU) (Kim et al., 2019) decoder, used a target vocabulary shortlist, and experimented with pruning the student models by removing component- and block-level parameters to improve speed. We further experimented with quantizing into smaller numerical formats, including fixed point 8-bit quantization on the CPU, and both 8-bit and log based 4-bit quantization on the GPU, as well as post-quantization fine-tuning of 4-bit quantized models.

For running our experiments, we improved upon the Marian (Junczys-Dowmunt et al., 2018) machine translation framework by incorporating speed ups for 8-bit matrix multiplication operations, optimizations for pruning neural network parameters on Intel CPUs, and exploring tensorcores on the GPU.

1.1 Efficiency Shared Task

The WMT21 efficiency shared task consists of two sub-tasks: throughput and latency. Systems should translate English to German under the constrained conditions of the WMT21 news task. For each task, systems are provided 1 million lines of raw English input with at most 150 space-separated words. The throughput task receives this input directly. The latency task, introduced this year, is fed input one sentence at a time, waiting for the translation output before providing the next sentence.

Throughput is measured on multi-core CPU or GPU system, and latency is measured on single-core CPU or GPU systems. The CPU-based evaluations use an Intel Ice Lake system via Oracle Cloud BM.Optimized3.36, while the GPU-based use a single A100 via Oracle Cloud BM.GPU4.8.

Entries to both tasks are measured on quality, approximated via BLEU score (Papineni et al., 2002), speed, model size, Docker image size, and memory consumption. We did not optimise specifically for the latency task beyond configuring the relevant batch sizes to one. We used Ubuntu 20.04 based images for our systems, with standard Ubuntu for CPU-only systems and NVIDIA's Ubuntu-based CUDA-11.4 docker for GPU-capable systems. Docker images were created using multi-stage builds, with model disk size reduced by compression with xzip.

[1] http://statmt.org/wmt21/efficiency-task.html

Proceedings of the Sixth Conference on Machine Translation (WMT), pages 780–785

2 Training teacher models

We used Edinburgh's En↔De systems submitted to the WMT 2021 news translation task as teacher models (Chen et al., 2021). We trained transformer-big models (Vaswani et al., 2017), using a shared 32K SentencePiece (Kudo and Richardson, 2018) vocabulary, built in three stages: corpus filtering, back-translation and fine-tuning. The models achieved 29.90 and 51.78 BLEU on En→De and De→En WMT 2021 test respectively (scored by the task organizers, with multiple references).

We used sequence-level knowledge distillation (Kim and Rush, 2016) to synthesize forward, backward, and backward-forward translations using the teachers. We filtered the synthesized parallel data using handcrafted rules[2], followed by removing bottom 5% according to cross-entropy per word on the generated side using KenLM (Heafield et al., 2013).

3 Knowledge distillation

We ran experiments using different combinations of teacher-synthesized corpora. One variant included all of the synthesized data: parallel, monolingual backward and forward as well as backward-forward (Aji and Heafield, 2020b). Another variant excludes only the fully-synthetic monolingual backward-forward data, while the final variant used parallel data only. All student models were trained using a validation set consisting of the subset of sentences in the English-German WMT test sets from 2015–2019 that were originally in English. Training concluded after reaching 20 consecutive validations without an improvement in BLEU score. The student models used the same shared vocabulary as the teacher ensemble. During decoding, we used a lexical shortlist (Schwenk et al., 2007; Le et al., 2012; Devlin et al., 2014) of the top 50 most probable alignments, combined through a union with the top 50 most frequent vocabulary items. Other than this, we used the default training hyperparameters from Marian for the transformer-base model.

Each of the student models used transformer encoders (Vaswani et al., 2017) and RNN-based decoders with Simpler Simple Recurrent Unit (SSRU) (Kim et al., 2019). Several different architectures were explored; these differ in the number of encoder and decoder blocks as well as in the sizes of the embedding and FFN layers. Further to this, some of our transformer architectures use a modified attention matrix of shape $(d_{\text{emb}}, n_{\text{head}} \times d_{\text{head}})$ rather than the typical $(d_{\text{emb}}, d_{\text{emb}})$. In all cases we use 8 transformer heads per layer, and set $d_{\text{head}} = 32$ across all modified attention models.

The student architectures are summarized in Table 1. A baseline comparison of student models trained on all synthesized data can be seen in Table 1.

3.1 Pruning

Attention is a crucial part of the transformer architecture, but it is also computationally expensive. Research has shown that many heads can be pruned after training; with further work suggesting that pruning during training can be less damaging to quality. Feedforward layers are also expensive and could be reduced.

Among many experiments, we applied group lasso regularisation to sparsify and prune 12–1.tiny and 12–1.micro architectures. We follow the directions set by Behnke and Heafield (2021). We tried two pruning settings: *rowcol-lasso* and *head-lasso*. Both prune feedforward and attention layers in the encoder. *rowcol-lasso* regularised individual connections (rows and columns) and removed an entire attention head if at least half of its connections are dead. *head-lasso* applied lasso to a whole head submatrix. Due to the scale of the task, we had no opportunity to grid-search for the best pruning hyperparameters, thus the experiments are as close to 'out-of-the-box' usage as they can be. We control pruning with $\lambda = 0.5$ for both methods. The models were pretrained for 50k updates and regularised for 150k, after which the models were sliced and trained until convergence. The results are presented in Tab. 2.

head-lasso left attention layers almost completely unpruned, focusing on removing connections from feedforward layers instead. *rowcol-lasso* was much more aggressive in both layers at the cost of quality. Behnke and Heafield (2021) have shown that group lasso pruning results in a better quality model than training the same exact architecture from scratch. To further optimise the models, they were quantised to work within 8bit representation. However, we observe that the smaller a model is, the larger the quality drop after its quantisation. Additional finetuning allows us to recover at least partially from the quantisation

[2] https://github.com/browsermt/students/tree/master/train-student/clean

Model	Depth Enc	Dec	Dimensions Emb.	FFN	Att.	Heads	Params.	Size	BLEU WMT20	BLEU WMT21	COMET WMT20	COMET WMT21	Speed (s)
teacher x 3	6	6/6/8	1024	4096	1024	16	619.0M	1.59GB	38.3	28.8	56.8	50.8	-
12-1.large	12	1	1024	3072	256	8	130.5M	498MB	37.6	28.7	54.0	47.7	92.2
12-1.base	12	1	512	2048	256	8	51.1M	195MB	36.7	28.2	50.7	44.1	38.9
12-1.tiny	12	1	256	1536	256	8	22.0M	85MB	36.1	27.6	48.2	41.9	19.2
12-1.micro	12	1	256	1024	256	8	18.6M	72MB	35.4	27.6	46.2	40.2	17.1
8-4.tied.tiny	8	4	256	1536	256	8	17.8M	69MB	35.7	27.8	50.3	43.9	30.4
6-2.tied.tiny	6	2	256	1536	256	8	15.7M	61MB	34.9	27.4	47.4	42.1	18.6
6-2.base	6	2	512	2048	512	8	42.7M	163MB	37.7	28.7	54.3	48.5	56.2
6-2.tiny	6	2	256	1536	256	8	16.9M	65MB	35.8	27.4	50.2	44.5	19.2

Table 1: Architectures for the different student models. The number of encoder/decoder layers are reported with the size of the embedding, attention and FFN layers, the total number of parameters, the model size on disk, quality in both BLEU and COMET as well as speed on WMT21 testset. The first and second groups use a modified attention matrix shape, with second group consisting of tied models. The third group uses the typical shape attention matrices.

damage. Evaluating on the latest testset WMT21, our pruned models are 1.2–1.7× faster at the cost of 0.6–1.3 BLEU. With quantisation, those models are 1.9–2.7× faster losing 0.9–1.7 BLEU in comparison to the unpruned and unquantised baselines.

3.2 Fixed Point 8-bit Quantization

Quantizing fp32 models into 8-bit integers is a known strategy to reduce decoding time, specifically on CPU, with a minimal impact on quality (Kim et al., 2019; Bhandare et al., 2019; Rodriguez et al., 2018). This year's submission closely follows the quantization scheme of last year's work (Bogoychev et al., 2020).

Quantization entails computing a scaling factor to collapse the range of values to $[-127, 127]$. For parameters, this scaling factor is computed offline using the maximum absolute value but activation tensors change at runtime. This year, we changed from computing a dynamic scaling factor on the fly for activations to computing a static scaling factor offline. We decoded the WMT16-20 datasets and recorded the scaling factor $\alpha(A_i) = 127/max(|A_i|)$ for each instance A_i of an activation tensor A. Then, for production, we fixed the scaling factor for activation tensor A to the mean scaling factor plus 1.1 standard deviation: $\alpha(A) = \mu(\{\alpha(A_i)\}) + 1.1 * \sigma(\{\alpha(A_i)\})$. These scaling factors were baked into the model file so that statistics were not computed at runtime.

Quantization does not extend to the attention layer, which is still computed in fp32. The reason being is that in the attention layer, both the A and B matrices of the GEMM operation would need to be quantized at runtime, which makes the quantization too expensive. We note that we only perform the GEMM operations in 8-bit integers.

3.3 Log 4-bit Quantization

We further quantize the models with log based 4-bit quantization (Aji and Heafield, 2020a). In this case, model weights are represented in a 16 unique quantization centers in a form of $S * 2^k$. S is a scaling factor that is optimized to minimize the MSE of the quantized weight to the actual weight. Following Aji and Heafield (2020a), we only perform 4-bit quantization on non-bias layers.

Unfortunately, the hardware used is not designed to perform native 4-bit operations. Therefore, our 4-bit quantization experiment is used solely for model compression purposes, in which we can reduce the model size to be 8x smaller. To perform inference, we de-quantize the 4-bit model back to fp32 representation, therefore does not achieve any speed up over the vanilla fp32 models.

3.4 Quantization fine-tuning

Quantizing models degrades the quality, especially on smaller architectures. Therefore, after applying quantization, we fine-tune the model under the quantized weight. We find that lowering the learning rate to 0.0001 yields better model quality. Moreover, for 4-bit models, we also find that doubling the warm-up duration helps.

Our 8-bit quantization models mainly aim for speed improvement. Therefore, we apply 8-bit quantization to pruned models to further boost the speed. As shown in Table 2, 8-bit inference achieves significant speedup. However, fine-tuning is necessary to restore the quality degradation.

	BLEU		COMET		Sparsity		
	WMT20	WMT21	WMT20	WMT21	Att.	FFN	Speed (s)
12-1.tiny	36.1	27.6	48.2	41.9	0%	0%	19.2
+ head-lasso pruning	34.7	27.0	42.9	38.8	3%	75%	14.5
+ 8bit quantisation	33.9	26.2	38.8	33.6	3%	75%	9.3
+ finetuning	34.1	26.7	39.8	33.0	3%	75%	9.3
+ rowcol-lasso pruning	33.8	26.3	39.3	34.2	68%	73%	11.6
+ 8bit quantisation	32.9	25.6	33.7	28.7	68%	73%	6.9
+ finetuning	32.9	26.0	35.7	31.3	68%	73%	7.1
12-1.micro	35.4	27.6	46.2	40.2	0%	0%	17.1
+ head-lasso pruning	34.6	26.7	43.0	35.4	3%	72%	14.1
+ 8bit quantisation	33.4	26.0	36.7	31.2	3%	72%	9.2
+ finetuning	33.7	26.5	38.3	33.3	3%	72%	9.2
+ rowcol-lasso pruning	34.3	26.4	40.7	35.1	60%	59%	12.0
+ 8bit quantisation	32.7	25.5	34.2	29.1	60%	59%	7.5
+ finetuning	33.3	25.9	35.2	30.5	60%	59%	7.5

Table 2: 8-bit model performance. BLEU score is calculated from WMT20. Speed is measured on a single core CPU with a mini-batch of 32. We experimented with two types of pruning. Head pruning removes entire heads. Row and column pruning removes entire rows or columns of matrices, resulting in a smaller matrix.

	BLEU		COMET		
	WMT20	WMT21	WMT20	WMT21	Size
12-1.base	37.1	28.3	51.5	45.1	195MB
+ 4bit	36.3	27.7	50.0	43.2	25MB
12-1.tiny	36.0	28.0	47.5	42.5	85MB
+ 4bit	35.0	27.6	42.4	38.3	11MB
8-4.tied.tiny	35.7	27.5	49.4	43.6	69MB
+ 4bit	34.2	26.4	44.4	38.2	9MB

Table 3: 4-bit model performance. BLEU score is calculated from WMT20. All the quantized models include fine-tuning. The inference is done in 32fp, therefore their speed are comparable.

We apply 4-bit quantization solely for size efficiency. Therefore, we quantize non-pruned models since they give better size to quality trade-off, compared to pruned models. The performance of 4-bit models can be seen in Table 3.

4 Software improvements

4.1 CPU

We built our work using the Marian machine translation framework, making some improvements on top of the submission from last year: We used predominantly *intgemm*[3] for our 8-bit GEMM operations, including for the shortlisted output layer. All parameter matrices are quantized to 8-bit offline and the activations get quantized dynamically before a GEMM operation. We only perform the GEMM operation and the following activation in 8-bit integer mode. Right after a GEMM operation, the output is de-quantized back to fp32. More formally we perform $dequantize(\sigma(A * B + bias))$, where the addition of the $bias$, the activation function[4] σ, and the de-quantization are applied in a streaming fashion to prevent a round trip to memory.

Furthermore we make use of Intel's *DNNL*[5] for our pruned models, as it performs better than *intgemm* for irregular sized matrices. Unfortunately, *DNNL* doesn't support streaming de-quantization, bias addition or activation function application.

For the CPU_ALL throughput track, we swept configurations of multiple processes and threads on the platform, settling on 4 processes with 9 threads each. The input text is simply split into 4 pieces and parallelized (Tange, 2011) over processes. The mini-batch sizes did not impact performance substantially and 32 was chosen as the mini-batch size. The Hyperthreads available on the platform were not put into use as the compute on each was saturated by the efficient threads. Each process is bound to 9 cores assigned sequentially and to the memory domain corresponding to the socket with those cores using numactl. Output from the data parallel run is then stitched together to produce the final translation.

[3] https://github.com/kpu/intgemm

[4] We only support ReLU activation for now

[5] https://github.com/oneapi-src/oneDNN

mini-batch	master fp32	master fp16	ours fp32	ours fp16	ours 8-bit
32	1160s	1151s	740s	731s	732s
64	696s	636s	426s	400s	416s
128	475s	430s	261s	246s	261s
256	320s	296s	181s	160s	169s
512	282s	241s	147s	127s	133s
768	285s	225s	139s	120s	123s
1024	277s	218s	136s	117s	120s
1132	277s	216s	135s	**116s**	119s
BLEU	33.47	33.43	33.48	33.42	33.26

Table 4: Comparison between the **master** branch of marian-dev, **our** branch and **our** best 8-bit integer tensorcore work for GPU decoding. For grid search we used last year's submission model and tested on 1 million sentences from last year's WNGT competition (Heafield et al., 2020).

4.2 GPU

For our GPU submission we built up on top of last year's submission, applying experimental GPU optimisations on top of the marian-dev master tree[6] and exploring tensorcore[7] applicability using CUTLASS.[8]

Tensorcores can in theory drastically increase the performance of our computations and were enabled for all of our fp16 experiments. Tensorcores can also improve speed when doing 8-bit integer operations, so we implemented 8-bit integer GPU decoding similar to our CPU scheme. We found that shortlisting doesn't improve the performance, so we didn't use it.

We found that while fp16 decoding works fairly well and delivers good performance improvements for decoding, especially when using a really large mini-batch size. We performed a large parameter sweep on a RTX 3090, as shown on Table 4. Unfortunately, we found no setting in which tensorcore 8-bit integer decoding outperforms the fp16 baseline, likely due to the overhead of quantisating the activations beforehand.

5 Conclusion

We participated in all tracks of the WMT 2021 efficiency tracks and we submitted multiple systems that have different trade-offs between speed and translation quality. We performed ample hyperparameter tuning and exploration in order to take advantage of GPU tensorcores for decoding, but unfortunately we couldn't beat our optimised fp16 baseline. For the CPU submission we used 8bit integer decoding and a combination of pruned and non-pruned system, together with a lexical shortlist in order to reduce the computational cost of the largest GEMM in decoding – the output layer.

[6] https://github.com/marian-nmt/marian-dev/pull/743

[7] https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/

[8] https://github.com/NVIDIA/cutlass

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council (grant EP/L01503X/1), EPSRC Centre for Doctoral Training in Pervasive Parallelism at the University of Edinburgh, School of Informatics.

This work has been performed using resources provided by the Cambridge Tier-2 system operated by the University of Cambridge Research Computing Service (www.hpc.cam.ac.uk) funded by EPSRC Tier-2 capital grant EP/P020259/1.

References

Alham Fikri Aji and Kenneth Heafield. 2020a. Compressing neural machine translation models with 4-bit precision. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 35–42.

Alham Fikri Aji and Kenneth Heafield. 2020b. Fully synthetic data improves neural machine translation with knowledge distillation. *CoRR*, abs/2012.15455.

Maximiliana Behnke and Kenneth Heafield. 2021. Pruning neural machine translation for speed using group lasso. In *Proceedings of the Six Conference on Machine Translation*, Online. Association for Computational Linguistics.

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.

Nikolay Bogoychev, Roman Grundkiewicz, Alham Fikri Aji, Maximiliana Behnke, Kenneth Heafield, Sidharth Kashyap, Emmanouil-Ioannis Farsarakis, and Mateusz Chudyk. 2020. Edinburgh's submissions to the 2020 machine translation efficiency task. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 218–224, Online. Association for Computational Linguistics.

Pinzhen Chen, Jindřich Helcl, Ulrich Germann, Laurie Burchell, Nicolay Bogoychev, Antonio Valerio Miceli Barone, Jonas Waldendorf, Alexandra Birch, and Kenneth Heafield. 2021. The University of Edinburgh's English-German and English-Hausa submissions to the WMT21 news translation task. In *Proceedings of the Sixth Conference on Machine Translation*, Online. Association for Computational Linguistics.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1370–1380, Baltimore, Maryland. Association for Computational Linguistics.

Kenneth Heafield, Hiroaki Hayashi, Yusuke Oda, Ioannis Konstas, Andrew Finch, Graham Neubig, Xian Li, and Alexandra Birch. 2020. Findings of the fourth workshop on neural generation and translation. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 1–9, Online. Association for Computational Linguistics.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 690–696, Sofia, Bulgaria. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, et al. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121.

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.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *CoRR*, abs/1808.06226.

Hai Son Le, Alexandre Allauzen, and François Yvon. 2012. Continuous space translation models with neural networks. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 39–48, Montréal, Canada. 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.

Andres Rodriguez, Eden Segal, Etay Meiri, Evarist Fomenko, Young Jin Kim, Haihao Shen, and Barukh Ziv. 2018. Lower numerical precision deep learning inference and training.

Holger Schwenk, Marta R. Costa-jussà, and Jose A. R. Fonollosa. 2007. Smooth bilingual n-gram translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 430–438, Prague, Czech Republic. Association for Computational Linguistics.

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, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.