Edinburgh's submission to the WMT 2022 Efficiency task

Nikolay Bogoychev[†] **Biao Zhang**[†], **Maximiliana Behnke**[†] **Graeme Nail**[†]
Jelmer van der Linde[†] **Sidharth Kashyap**[‡] **Kenneth Heafield**[†]
[†]University of Edinburgh [‡]Intel Corporation
{n.bogoych, biao.zhang, maximiliana.behnke, graeme.nail, jelmer.vanderlinde
kenneth.heafield}@ed.ac.uk, sidharth.n.kashyap@intel.com

Abstract

We participated in all tracks of the WMT 2022 efficient machine translation task: single-core CPU, multi-core CPU, and GPU hardware with throughput and latency conditions. Our submissions explore a number of efficiency strategies: knowledge distillation, a simpler simple recurrent unit (SSRU) decoder with one or two layers, shortlisting, deep encoder, shallow decoder, pruning and bidirectional decoder. For the CPU track, we used quantized 8-bit models. For the GPU track, we used FP16 quantisation. We explored various pruning strategies and combination of one or more of the above methods.

1 Introduction

This paper describes the University of Edinburgh's submission to Seventh Conference on Machine Translation (WMT2022) 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 (Behnke et al., 2021). We trained our models in a teacher-student setting (Kim and Rush, 2016), using the data provided by the organisers. 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 used 8-bit quantisation for the CPU submission and FP16 quantisation for the GPU submission. We further experimented with IBDecoder (Zhang et al., 2020).

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 profiler aided optimisation of various components.

[1]http://statmt.org/wmt22/efficiency-task.html

1.1 Efficiency Shared Task

The WMT22 efficiency shared task consists of two sub-tasks: throughput and latency. Systems should translate English to German under the constrained conditions, where the teacher model and the distilled data are provided. 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 in WMT21, 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 COMET score (Rei et al., 2020), 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 22.04 based images for our systems, with standard Ubuntu for CPU-only systems and NVIDIA's Ubuntu-based CUDA-11.7 docker for GPU-capable systems. Docker images were created using multi-stage builds, with model disk size reduced by compression with xzip.

2 Knowledge distillation

We used the provided distilled data to build different student systems. The provided data was distilled through two different processes; for the monolingual input, distilled data was generated using a beam-size of 6. For the parallel data, the teacher ensemble was used to produce the 6 best candidate translations for each input sentence, the candidate most similar to the parallel reference in BLEU score was kept as the distilled sentence. We trained student models using just the provided parallel data and identical systems using parallel+monolingual data. Our early comparisons showed that the full corpora produced higher quality student systems according to automatic metrics; submitted systems therefore use both parallel and monolingual data.

The student models were trained using a validation set consisting of the subset of sentences in the English-German WMT test sets from 2014–2019 that were originally in English. Training concluded after reaching 20 consecutive validations without an improvement in BLEU score. The student models all used the provided shared SentencePiece vocabulary. We used the default training hyperparameters from Marian for the transformer-base model with the learning rate reduced to 0.0002.

We explored a number of different configuration in order to find the optimal system on the Pareto frontier for speed-quality. We experimented with the following configurations:

Deep encoders/Shallow decoders The majority of the computational cost of the machine translation system falls to the decoder. We can therefore increase drastically the number of encoder layers and decrease the decoder layers without noticeable drop in quality (Kong et al., 2021).

Tied decoder layers Since matrix multiplication is a memory bound problem, we can increase the number of decoder layers, as long as we don't add extra parameters. Tied decoder layers allow us to maintain the same memory footprint and keep all the traversed matrices in cache.

SSRU We replace the self-attention in the decoder with an RNN using the less computation and memory intensive cell SSRU.

Reduced model dimensions We reduce the model dimensions, using several presets. See table 1 for details.

Wide embeddings We increased the size of the embedding dimension to match the FFN dimension. While this produces models that are strictly larger than their non-wide equivalent, the initial increased capacity can yield competitive systems using fewer layers.

Fewer heads We reduced the number of attention heads for some of the smaller models. These have the same number of parameters, but intermediate computations have different shape inputs.

3 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.

We expand upon our work from the previous year on the group lasso regularisation. We build upon the standard group lasso with a novel approach of *aided regularisation*. The idea behind it is to use supplementary information to scale the penalties per layer to steer them towards a specific behaviour. In practice, it means adding a new scalar γ alongside an already existing λ:

$$E(batch) = \frac{1}{|batch|}\left(\sum_{x \in batch} CE(x) + \lambda \sum_{l \in layers} \gamma_l^{batch} R(l)\right)$$

As shown in the equation above, each layer has its individual γ, which gets updated after every backpropagation pass. In order to avoid sudden shits in γ between individual batches, which could make a ratio between perplexities and penalties even more unstable, γ are exponentially smoothed as training progresses:

$$\gamma^j \leftarrow \alpha\gamma^j + (1-\alpha) * \gamma^{j-1}$$

After every batch i, we calculate a local scalar γ_j for each layer j based on information gathered during this specific update, which then updates a smoothed global scalar. α is a constant used in exponential average that controls the contribution of a new element in a sequence towards the overall average. We use $\alpha = 1\text{e}{-4}$ in my experiments.

We explore *gradient-aided regularisation* which *scales penalties based on layer gradients*. γ scalars should increase as gradients stop flowing through a layer since it indicates that this layer does not contribute to training as much, possibly stopping learning altogether. A layer with small gradients is a good candidate to be regularised more aggressively and vice versa.

With W_i being a regularised layer and ∇W as accumulated gradients in a model, the gradient-aided γ function is defined as:

	Layers		Dimensions			Size		Quality			Speed
Model	Encoder	Decoder	Emb.	FFN	Att. heads	Params	Disk	BLEU	chrF	COMET	Time
Teacher	6	6	1024	4096	16	627.5M	3.19GB	43.37	67.39	0.5908	—
Large	12	1	1024	3072	8	171.4M	654MB	44.26	68.06	0.5901	170.4
Base	12	1	512	2048	8	57.9M	222MB	44.06	67.94	0.5842	57.7
Tiny	12	1	256	1536	8	22.0M	85MB	43.32	67.36	0.5516	23.4
Micro	12	1	256	1024	8	18.6M	72MB	43.00	67.16	0.5389	20.9
Base	6	2	512	2048	8	42.7M	163MB	44.04	67.90	0.5879	50.5
Tiny	6	2	256	1536	8	16.9M	65MB	42.76	67.12	0.5538	19.6
Tied.Tiny	6	2	256	1536	8	15.7M	61MB	42.72	67.08	0.5470	17.7
Tied.Tiny	8	4	256	1536	8	17.8M	69MB	43.22	67.38	0.5621	23.0
Base.Wide	12	1	2048	2048	8	401.5M	1.50GB	43.82	67.74	0.5773	395.4
Base.Wide	6	2	2048	2048	8	283.8M	1.1GB	44.28	68.14	0.5979	374.7

Table 1: Architectures for the different student models. The number of encoder/decoder layers are reported with the size of the embedding and FFN layers, the total number of parameters and the model size on disk. Quality and speed evaluated and averaged across WMT16–19.

	Layers		Sparsity		Quality			Speed
Model	Encoder	Decoder	Attention	FFN	BLEU	chrF	COMET	Time
Base	12	1	0%	0%	44.06	67.94	0.5842	57.7
+ pruning	12	1	63%	20%	43.92	67.86	0.5825	44.6
+ pruning + ft8bit	12	1	63%	20%	43.68	67.66	0.5710	18.6
Tiny	12	1	0%	0%	43.32	67.36	0.5516	23.4
+ pruning	12	1	74%	72%	41.54	66.16	0.4882	12.3
+ pruning + ft8bit	12	1	74%	72%	41.02	65.70	0.4615	5.8
Tied Tiny	8	4	0%	0%	43.22	67.38	0.5621	23.0
+ pruning	8	4	46%	20%	42.98	67.22	0.5584	19.3
+ pruning + ft8bit	8	4	46%	20%	42.36	66.78	0.5393	10.0

Table 2: The evaluation of student models pruned with aided regularisation and quantised to 8-bits. Both quality and speed has been averaged over WMT16–20 testsets. Quantised models were finetuned shortly to help recover quality.

$$\gamma_i = -log\left(\frac{\|\frac{\partial W_i}{\partial E}\|_2}{\|\nabla W\|_2}\right)$$

We follow the training regime outlined by Behnke et al. (2021):

1. Pretraining (50k batches)
2. Regularise (200/300k batches)
3. Slice and converge (200k+ batches)

All on-going training statistics including the learning rate and Adam optimiser were refreshed after each step. The results are presented in Tab. 2. The results include quantised inference with models finetuned for the best quality performance. The 12-1.Base model was regularised for 300k batches with $\lambda = 0.05$. The 12-1.Tiny model was regularised for 200k batches with $\lambda = 0.5$. Both aforementioned models were pruned in the encoder only. The 8-4.Tiny.Tied model was regularised for 200k batches with $\lambda = 0.3$ with both encoder and decoder layers being penalised.

The quality gap becomes larger the harsher pruning is. The base transformer model with 12 pruned encoder layers gets $1.3\times$ faster at the cost of 0.0017 COMET point. Applying quantisation on the top of it makes translation $3.1\times$ faster in exchange of 0.13 COMET points.

We applied regularisation onto both encoder and decoder with the "8-4" tied tiny transformer architecture. This pruned and quantised model speeds up by a factor of $2.3\times$ at a 0.8 BLEU drop.

The most aggressive pruning among the presented results is a tiny transformer with 12 encoder layers with more than 70% parameters removed. This model is $4\times$ faster in comparison to its baseline with 3.3 BLEU and 0.09 COMET points drop.

We note that quantisation struggles with quality on smaller models, both when trained from scratch or pruned. Fine-tuning rectifies the problem to some degree, but quality is sacrificed for faster translation in the end.

4 Fixed Point 8-bit Quantisation

Quantising 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 quantisation scheme of last year's work (Behnke and Heafield, 2021).

Quantisation 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. To compute a scaling factor for them, 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.

We used predominantly *intgemm*[2] for our 8-bit GEMM operations, including for the shortlisted output layer. All parameter matrices are quantised to 8-bit offline and the activations get quantised dynamically before a GEMM operation. We only perform the GEMM operation and the following activation in 8-bit integer mode. 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 σ, and the de-quantisation are applied in a streaming fashion to prevent a round trip to memory.

Furthermore we make use of Intel's *DNNL*[3] for our pruned models, as it performs better than *intgemm* for irregular sized matrices. Unfortunately, *DNNL* doesn't support streaming de-quantisation, bias addition or activation function application.

Quantisation 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 quantised at runtime, which makes the quantisation too expensive. We note that we only perform the GEMM operations in 8-bit integers.

[2]`https://github.com/kpu/intgemm`

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

Similar to previous' year's submission, we performed quantisation fine-tuning for some 8-bit models, where we perform a small amount of training with low learning rate and a damaged GEMM implementation that simulates the quantised output. We found that this helps regain some quality, especially in smaller models.

5 Shortlisting

The single most expensive computation in machine translation is the cost of the output layer. We can reduce the computation if we only take into account likely output tokens, reducing the output layer size from 32000 to something much more manageable like 500-2000. We used IBM model based shortlisting (Kim et al., 2019).

This lexical shortlist is straightforward to work with, but it is limiting in the sense that it doesn't capture well idioms and favours more literal translations. The hyper-parameters that control the size of this shortlist are: the number of most frequently targeted words included, and the number of probable translations for each token in the input. This year we increased the number of most-frequent and aligned tokens to `100,100` (from `50,50` in the previous year) in order to improve quality.

The shortlist is built using alignment models trained on a specific corpora. The total number of tokens in a shortlist considered is influenced by the size of the current batch: The shortlist produced is the union of probable translations for each input token and overall most-likely candidates. In latency scenarios, where batches are a single sentence, a small shortlist is more detrimental to the quality than for larger batches, such as in throughput scenarios, that benefit from inclusion of more candidate tokens. Similarly, this approach benefits when inputs are batched.

6 IBDecoder

The sequential nature of autoregressive decoding forms an inference bottleneck, hurting decoding parallelisation and latency. A popular method to break this bottleneck is to allow the parallel prediction of multiple target tokens per step through semi- or non-autoregressive modelling (Gu et al., 2018; Wang et al., 2018) with a quality tradeoff.

We experimented with Interleaved Bidirectional Decoder (Zhang et al., 2020), a variant of semi-autoregressive decoder that predicts target tokens from the left-to-right and the right-to-left directions

	Layers		Size		Quality			Speed
Model	Encoder	Decoder	Params	Disk	BLEU	chrF	COMET	Time
Base	12	1	57.9M	222MB	44.06	67.94	0.5842	57.7
+ IBDecoder	12	1	57.9M	221MB	43.84	67.74	0.5605	51.6
+ 8bit quantisation	12	1	57.9M	221MB	43.50	67.48	0.5412	28.6
Base + IBDecoder	12	1	57.9M	221MB	43.84	67.74	0.5605	51.6
+ pruning	12	1	57.9M	168MB	43.50	67.48	0.5340	42.1
+ 8bit quantisation	12	1	57.9M	168MB	43.26	67.28	0.5166	18.8
Tiny	6	2	16.9M	65MB	42.76	67.12	0.5538	19.6
+ IBDecoder	6	2	16.9M	65MB	41.88	66.64	0.5074	17.1
+ 8bit quantisation	6	2	16.9M	65MB	41.00	65.94	0.4628	9.8
Tiny + IBDecoder	6	3	18.1M	69MB	42.48	66.98	0.5275	19.6
+ 8bit quantisation	6	3	18.1M	69MB	41.62	66.32	0.4971	11.2
Micro + IBDecoder	12	4	21.4M	82MB	42.96	67.28	0.5475	25.3
+ 8bit quantisation	12	4	21.4M	82MB	42.56	67.08	0.5338	15.4

Table 3: The evaluation of IBDecoder models and their 8-bit quantisation (without finetuning). Both quality and speed has been averaged over WMT16–20 testsets.

simultaneously. Zhang et al. (2020) showed that words from different directions are more loosely dependent thus their parallel generation hurts quality less. IBDecoder produces one word in each direction at a time, thus halving the total decoding steps and approximately doubling speed.

The efficiency gains from IBDecoder decrease when using deep encoders and shallow decoders (Tab. 3). In general, IBDecoder delivers a speed-up over our baseline system and is competitive at BLEU scores but much worse at COMET scores. IBDecoder shows higher sensitivity to model sizes, where reducing model size dramatically hurts its performance regardless of BLEU or COMET. We also tried to initialise IBDecoder from the baseline system which unfortunately doesn't help. IBDecoder also benefits from pruning and quantisation in speed, but at the cost of losing COMET.

7 Quality issues

Quantisation applied to small models, especially those that were pruned, struggles with maintaining the quality. For example, as can be seen in Tab. 2, quantisation on top of pruned models damages the quality from 0.1 to 0.3 COMET points. This gap is more evident in smaller architectures such as Tiny or Tied Tiny. We hypothesise that the fewer parameters there are in a model, the more difficult it is to optimise through pruning and/or quantisation, or using a bidirectional generation.

IBDecoder suffers from the pruning and quantisation particularly on the COMET scores as shown in Tab. 3.

We compared several sentences that showed little difference in BLEU but significant difference in the COMET scores and tried to see what went wrong (Table 4). We can see that the IBDecoder is prone to pathological repetitions (Example 1) and even more so when quantised (Examples 3 and 4). Those repetitions, especially long ranged one don't hurt the BLEU score, but they get heavily penalised by the COMET score (Example 3).

It seems quantisation doesn't always result in a a worse transaltion. In the second example the quantised IBDecoder produces a more complicated, but overall much better translation than the IBDecoder, which also suffers from a repetition error. This suggests that the model is quite brittle and very susceptible to small changes.

8 Software improvements

We built our work using the Marian machine translation framework, making some improvements on top of the submission from last year:

AVX512 inrinsics We implemented hand crafted intrinsics for various arithmetic operations, resulting in .5% improvement in performance.

Max element We identified via a profiler that the *max element* implementation was taking more time than usual so we implemented a hand optimised version resulting in 5% performance improvement. More details are available in Appendix A.

Thread configuration For the CPU_ALL throughput track, we swept configurations of multiple processes and threads on the platform, settling

Reference	Biotechnische Anwendungen
Baseline	Biotech-Anwendungen (0.6632)
IBDecoder	Anwendungen in der-Anwendungen (-1.4205)
IBDecoder-Quant	Anwendungen in der-Anwendungen (-1.4205)
Reference	Die nächste Show findet am 9. Oktober in San Francisco statt. Am 16. März 2020 wird die Band ihre UK-Tournee in Manchester eröffnen.
Baseline	Ihre nächste Show ist am 9. Oktober in San Francisco und die Band wird ihre UK-Tour in Manchester am 16. März 2020 eröffnen. (0.7350)
IBDecoder	Ihr nächster Auftritt ist am 9. Oktober in San Francisco und eröffnet eröffnet ihre UK-Tour in Manchester am 16. März 2020. (-0.1582)
IBDecoder-Quant	Ihre nächste Show findet am 9. Oktober in San Francisco statt, wo die Band ihre UK-Tournee in Manchester am 16. März 2020 eröffnen wird. (0.7387)
Reference	Die Herzogin von York schrieb auf Twitter: „ Ich kenne die Gefühle einer Mutter, deshalb weine ich vor Freude. Ich freue mich sehr über diese sensationellen Neuigkeiten
Baseline	Die Herzogin von York schrieb auf Twitter: "Ich weiß, was eine Mutter fühlt, also habe ich Tränen der Freude. (0.4349)
IBDecoder	Die Herzogin von York schrieb auf Twitter: "Ich weiß, was eine Mutter fühlt, also habe ich Tränen der Freude. (0.4351)
IBDecoder-Quant	Die Herzogin von York schrieb auf Twitter: "Ich weiß, was eine Mutter Freude, also habe ich Tränen der Freude, also habe ich Tränen der Freude. (-0.9820)
Reference	Meghan Markle bezüglich des Kurzauftritts bei Suits „nie gefragt“
Baseline	Meghan Markle wurde nach Suits Cameo "nie gefragt" (0.1344)
IBDecoder	Meghan Markle wurde "niemals" nach Suits Cameo gefragt (0.0509)
IBDecoder-Quant	Meghan Markle wurde "nicht gefragt" nach Suits Cameo gefragt (-1.1194)

Table 4: Case study for IBDecoder models. All models are with 12 encoder layers and 1 decoder layer under the base setup. IBDecoder-Quant denotes the quantised system. We show cases where IBDecoder and IBDecoder-Quant performs worse than Baseline and IBDecoder, respectively, and the numbers in bracket shows the COMET scores.

on 4 processes with 9 threads each. The input text is simply split into 4 pieces and parallelised (Tange, 2011) over processes. The mini-batch sizes past 16 did not impact performance substantially but 32 was chosen as the best performing one. The Hyperthreads do not increase performance. 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 output.

For our GPU submission we reused the work from the last year's submission (Behnke et al., 2021) with the improved models.

9 Conclusion

We participated in all tracks of the WMT 2022 efficiency task and we submitted multiple systems that have different trade-offs between speed and translation quality. For the CPU submission we used 8-bit 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 output layer. We also experimented with IBDecoder in both CPU and GPU setting.

Acknowledgements

The computations described in this research were performed using the Baskerville Tier 2 HPC service (https://www.baskerville.ac.uk/). Baskerville was funded by the EPSRC and UKRI through the World Class Labs scheme (EP/T022221/1) and the Digital Research Infrastructure programme (EP/W032244/1) and is operated by Advanced Research Computing at the University of Birmingham.

References

Maximiliana Behnke, Nikolay Bogoychev, Alham Fikri Aji, Kenneth Heafield, Graeme Nail, Qianqian Zhu, Svetlana Tchistiakova, Jelmer van der Linde, Pinzhen Chen, Sidharth Kashyap, and Roman Grundkiewicz. 2021. Efficient machine translation with model pruning and quantization. In *Proceedings of the Sixth Conference on Machine Translation*, pages 775–780, Online. Association for Computational Linguistics.

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.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

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.

Xiang Kong, Adithya Renduchintala, James Cross, Yuqing Tang, Jiatao Gu, and Xian Li. 2021. Multilingual neural machine translation with deep encoder and multiple shallow decoders. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1613–1624, Online. Association for Computational Linguistics.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. 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.

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

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 479–488, Brussels, Belgium. Association for Computational Linguistics.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2020. Fast interleaved bidirectional sequence generation. In *Proceedings of the Fifth Conference on Machine Translation*, pages 503–515, Online. Association for Computational Linguistics.

A Profiler aided optimisation

We used a profiler to identify hotspots for potential software optimisations.

Max element improvements We identified that the *max element* operation takes surprisingly large amounts of runtime during decoding. *max element* is used to select the next word to be produced from the output layer during decoding with beam size of 1. We explored various different implementations and achieved 10X performance improvement compared to the standard library when using GCC and 2X otherwise. This resulted in about 5% performance improvement in the CPU setting. More details about different implementations can be seen on Table 5

	GCC	**clang**
std::max_element	2.670s	0.422s
sequential	1.083s	1.192s
AVX512 max + max_reduce	0.241s	0.215s
AVX512 max_reduce only	0.257s	0.263s
AVX512 cmp_ps_mask	0.188s	0.183s
AVX512 ^+ vectorized overhang	0.210s	0.209s
AVX cmp_ps + movemask	0.218s	0.170s
SSE cmplt_psp + movemask	0.269s	0.205s

Table 5: Performance of *max element* with GCC 11.2 and clang 14 on Intel Cascade lake. For more information check `https://github.com/XapaJIaMnu/maxelem_test`.