Findings of the WMT 2022 Shared Task on Efficient Translation

Kenneth Heafield Biao Zhang Graeme Nail Jelmer van der Linde Nikolay Bogoychev
University of Edinburgh
10 Crichton Street
Edinburgh, Scotland EH8 9AB
{Kenneth.Heafield,b.zhang,graeme.nail,jelmer.vanderlinde,n.bogoych}@ed.ac.uk

Abstract

The machine translation efficiency task challenges participants to make their systems faster and smaller with minimal impact on translation quality. How much quality to sacrifice for efficiency depends upon the application, so participants were encouraged to make multiple submissions covering the space of trade-offs. In total, there were 76 submissions from 5 teams. The task covers GPU, single-core CPU, and multi-core CPU hardware tracks as well as batched throughput or single-sentence latency conditions. Submissions showed hundreds of millions of words can be translated for a dollar, average latency is 3.5–25 ms, and models fit in 7.5–900 MB.

1 Introduction

The efficiency task complements the collocated news task by challenging participants to make their machine translation systems computationally efficient. This is the fifth edition of the task, expanding upon previous editions (Heafield et al., 2021, 2020; Hayashi et al., 2019; Birch et al., 2018).

Participants built English→German machine translation systems following a constrained data condition. The data condition follows the constrained **2021** Workshop on Machine Translation news translation task. This year, to reduce the barrier to entry, organisers provided an ensemble of teacher systems, as well as cleaned data and distilled output from the teacher ensemble. Participants were required to use the provided teacher systems, but were free to distil additional data from the constrained condition. The SentencePiece vocabulary used by the teachers was also made available.

For translation quality measurement, we use the news-focused WMT22 dataset, and the systems are ranked according to the COMET (Rei et al., 2020) automatic metric. We also evaluate systems on BLEU and chrF for additional reference.

	Throughput		**Latency**	
	CPU-ALL	GPU	CPU-1	GPU
CUNI	1	1	1	1
ECNU	1	1	1	1
Edinburgh	15	11	15	11
HuaweiTSC	5		5	
RoyalFlush				6

Table 1: Number of systems submitted by each participant for the different hardware and batching conditions. CPU-ALL refers to the 36-core hardware setting.

Submissions are made as Docker containers so we can consistently measure their performance in terms of quality, speed, memory usage, and disk space. We run the containers in three different hardware environments: one GPU, one CPU core, and multiple CPU cores. Systems were tested for throughput by providing 1 million sentences upfront to allow batching and parallelization. We also tested for latency with a program that drip-feeds one input sentence, waits for the translation, and then provides the next input sentence. There were four conditions in total: GPU throughput, GPU Latency, 1 CPU Core Latency, and 36 CPU cores throughput. We did not measure latency in a multi-core CPU setting because the test hardware has 36 cores and overhead for 36 threads is larger than the cost of arithmetic for the small tensors in optimized models. We also did not measure throughput on a single CPU core as we found that setting to be a somewhat unrealistic real world scenario.

Participants were free to choose which conditions to participate in. The condition was passed to the Docker container as command line arguments. Table 1 shows the five participants and the number of systems they submitted to each of the conditions.

Machine translation is used in a range of settings where users might choose different trade-offs between quality and efficiency. For example, a high-frequency trading system might prefer the lowest latency at the expense of quality given that the output will only be read by a machine. Conversely, in a post-editing scenario the personnel costs outweigh many computational costs. Therefore there is not a single best system, but a range of options that trade between quality and efficiency.

We emphasize the Pareto frontier: the fastest systems at each level of quality, or the smallest systems at each level of quality. To explore the Pareto frontier, participants were encouraged to make multiple submissions covering the range of trade-offs. In total, 76 combinations of models, hardware, and batching were benchmarked.

2 Hardware

We chose modern hardware to encourage exploiting new hardware features. The GPU is an NVidia A100 from the Oracle Cloud `BM.GPU4.8` instance. The instance has eight GPUs and we limited Docker to using only one GPU. The GPU machine has an AMD EPYC 7542 CPU with all cores allowed.

The CPU-only condition used a dual-socket Intel Xeon Gold 6354 from Oracle Cloud `BM.Optimized3.36` with a total of 36 cores. For the single-core CPU track, we reserved the entire machine then ran Docker with `-cpuset-cpus=0`. In the 36-core CPU track, participants were free to configure their own CPU sets and affinities.

The Oracle Cloud machines are bare metal servers, meaning there was no shared tenancy, no virtualization, and the test machines were otherwise quiescent.

3 Input Text

To amortize loading time, avoid starving highly parallel submissions, and reduce the ability to cheat, we benchmark systems on 1 million sentences of input. The test set is hidden inside these 1 million sentences, shuffled with filler sentences. Many filler sentences are drawn from parallel corpora to check that systems are in fact translating all sentences, though we do not consider scores on noisy corpora reliable enough to report. The composition of this set changes each year and is decided after the submission deadline.

The filler data was gathered from parallel corpora and gender bias challenge sets: WMT news test sets from 2008 through 2022 (Akhbardeh et al., 2021), the additional test inputs in WMT 2021, Khresmoi summary test v2 (Dušek et al., 2017), IWSLT 2019 (Jan et al., 2019), SimpleGen (Renduchintala et al., 2021), WinoMT (Stanovsky et al., 2019), TED 2020 (Reimers and Gurevych, 2020), and Tilde RAPID 2019 (Rozis and Skadiņš, 2017). We limit sentence lengths to 150 space-separated tokens. Because WMT 2020 includes excessively long segments that are actually concatenated sentences, we also added sentence split versions of WMT 2020 and WMT 2021, though the difference on WMT 2021 was minor. Source sentences were concatenated, deduplicated, and shuffled. The Tilde RAPID corpus was clipped to make a total of 1 million deduplicated lines. Counts are shown in Table 2.

Corpus	**Sentences**
WMT 08–19	32,477
WMT 20 under 150 tokens	1,416
WMT 20 sentence split	2,048
WMT 21 sentence split	1,096
WMT 21 inc. additional tests	14,938
WMT 22	2,037
Khresmoi Summary Test v2	1,000
IWSLT 2019	2,278
SimpleGen	2,664
WinoMT	3,888
TED 2020 v1	293,562
Tilde RAPID 2019	663,922
Total	1,021,326
Deduplicated	1,000,000

Table 2: Summary of corpora used for the input text.

Input text and tools to extract test sets from system outputs are available at https://data.statmt.org/wmt22/efficiency-task/wmt22-testdata.tar.xz.

The input file is 1,000,000 lines, consisting of 19,926,744 space-separated words, or 124,186,772 bytes of English text in UTF-8. This is a mean of 19.9 words per sentence and is comparable to the previous year (Heafield et al., 2021). Teams were responsible for their own tokenization and detokenization; for this they were permitted to use the SentencePiece vocabulary provided with the teacher system, or to implement an alternative. We provided raw UTF-8 English input text with one sentence per line.

4 Metrics

4.1 Resources

Time was measured with wall (real) time reported by `time` and CPU time reported by the kernel for the process group. We do not measure loading time because it is small compared to translating 1 million sentences, some tools load lazily, and it is easily gamed by padding loading time.

Peak RAM consumption was measured using `memory.max_usage` in bytes from the kernel for the CPU and by polling `nvidia-smi` for the GPU. Swap was disabled.

Participants were instructed to separate their Docker images into model and code files so that models could be measured separately from the relatively noisy size of code and libraries. A model was defined as "everything derived from data: all model parameters, vocabulary files, BPE configuration if applicable, quantization parameters or lookup tables where applicable, and hyperparameters like embedding sizes." Code could include "simple rule-based tokenizer scripts and hard-coded model structure that could plausibly be used for another language pair." They were also permitted to use standard compression tools such as `xz` to compress models; decompression time was excluded in results. We report size of the model directory captured before the model ran. We also measured the total size of the Docker image (after compressing with `xz`).

4.2 Quality

Translation quality is measured on the WMT 2022 news test set. The automatic metrics are COMET (Rei et al., 2020) from `unbabel-comet` version `1.1.3` with the pretrained model `wmt20-comet-da`, BLEU from sacrebleu (Post, 2018) `nrefs:1|case:mixed|eff:no|tok:13a|smooth:exp|version:2.3.1`, and chrF also from sacrebleu.

5 Results

The results of the task evaluation for the latency scenario are presented in Table 3, and those for throughput are presented in Table 4. Results are separated by the different hardware conditions and within each hardware setting the results are ordered by their COMET score, which is shown to have closer correspondence to human evaluation as compared to BLEU and ChrF (Freitag et al., 2021).

Figure 1 shows the trade-off between quality and speed of batched translation submissions separated by hardware environment. Each plot shows the Pareto frontier as a black staircase to highlight the best combinations of quality and speed. While GPU systems (Figure 1a) achieve higher throughput compared to CPU systems (Figure 1b), this ignores pricing differences between these compute options. In Figure 2, we combine GPU and 36 Core CPU speed by using Oracle Cloud pricing. Despite the less expensive per-hour pricing of CPU, GPU is cheaper for throughput-oriented tasks that allow batching.

The all-hardware latency Pareto frontier is shown in Figure 3. This year all participants submitted systems to the latency task. This year, for the first time, the semi-autoregressive GPU system by RoyalFlush dominates the lower quality settings of the latency Pareto frontier, with Edinburgh GPU systems having won on some higher quality systems.

Model sizes at rest on disk appear in Figures 4a. Participants were allowed to compress their models using their own tools and standard tools like `xz`. The Pareto frontier consists of almost entirely Edinburgh submissions, with HuaweiTSC producing several systems on the lower quality settings, due to their 4-bit compression models. Docker image sizes, which include model and software, appear in Figure 4b, where the Pareto frontier is dominated by Edinburgh submissions. Conversely, some others opted to optimize other metrics and included large Linux installations. We compressed all docker images with `xz` before measuring.

Memory (RAM) consumption appears in Figure 5. GPU memory consumption reflects batch size and some participants set a large batch size to maximize speed. Optimizing speed for multi-socket CPU machines implies having a copy of the model in RAM close to each socket, so memory consumption is larger beyond simply having temporary space for more batches. Finally, participants may have sorted the entire 118 MB input file in RAM to form batches of equal length sentences. RoyalFlush is the clear winner on the GPU latency RAM consumption, and HuaweiTSC is the winner of CPU latency RAM consumption.

6 Conclusion

Using the highest quality system in this evaluation, translating 124,186,772 characters took 283

NVIDIA A100 GPU Latency

		Automatic			Seconds		Disk MB		RAM MB
Team	**Variant**	COMET	BLEU	chrF	**Wall**	**CPU**	**Model**	**Docker**	**GPU**
Edinburgh	6-1.base.wide-gpu	0.542	34.50	61.90	15051	15141	900	2316	37961
Edinburgh	12_1.large-gpu	0.541	34.10	61.60	14116	14186	624	2039	37555
Edinburgh	6-2.base-gpu	0.528	33.80	61.50	16548	16584	171	1587	37181
Edinburgh	12_1.base-gpu	0.518	33.90	61.40	13081	13118	225	1641	37211
RoyalFlush	royalflush_hrt_e20d1_k2	0.512	33.80	61.50	6008	6051	345	869	2021
Edinburgh	6-1.base-gpu	0.507	33.50	61.10	12665	12698	159	1574	37175
RoyalFlush	royalflush_hrt_e12d1_k2	0.498	33.90	61.40	5437	5472	259	781	1973
Edinburgh	8-4.tied.tiny-gpu	0.462	32.40	60.10	24126	24157	84	1500	37133
RoyalFlush	royalflush_hrt_e20d1_k3	0.458	33.40	61.10	4706	4752	345	870	2021
Edinburgh	6-2.micro.4h-gpu	0.454	31.70	59.80	15003	15031	74	1489	37129
Edinburgh	6-2.tied.tiny-gpu	0.443	31.50	59.50	15236	15261	77	1492	37129
ECNU	ecnu-mt	0.432	33.20	60.70	25306	25338	492	15680	4989
Edinburgh	6-2.micro.1h-gpu	0.432	31.30	59.20	14789	14817	73	1489	37129
RoyalFlush	royalflush_hrt_e12d1_k3	0.430	33.30	60.90	4093	4129	257	783	1973
Edinburgh	ib-6-2-tiny-gpu	0.388	31.10	59.40	12624	12653	81	1496	37133
RoyalFlush	royalflush_hrt_e20d1_k4	0.376	33.00	60.80	4024	4064	343	866	2021
Edinburgh	ib-12_1-tiny-gpu	0.373	31.90	59.80	10763	10793	99	1515	37141
RoyalFlush	royalflush_hrt_e12d1_k4	0.342	32.60	60.30	3409	3443	259	783	1973
CUNI	cuni-large-ende	0.250	30.80	59.10	8327	8410	856	1676	1875

1 Core Ice Lake CPU Latency

		Automatic			Seconds		Disk MB		RAM MB
Team	**Variant**	COMET	BLEU	chrF	**Wall**	**CPU**	**Model**	**Docker**	**CPU**
Edinburgh	6-1.base.wide-cpu	0.517	33.90	61.50	79230	79234	162	212	2487
Edinburgh	12_1.large-cpu	0.516	33.70	61.30	51991	51995	121	171	1537
Edinburgh	12_1.base_efh_0.05	0.513	33.80	61.40	37183	37190	176	1176	1337
Edinburgh	6-2.base-cpu	0.509	33.30	61.00	18101	18102	32	82	542
Edinburgh	12_1.base_efh_0.05_ft8	0.507	33.50	61.20	14669	14679	156	217	1256
Edinburgh	6-1.base-cpu	0.496	33.10	60.90	13383	13385	29	79	533
Edinburgh	12_1.base-cpu	0.494	33.70	61.20	19100	19102	44	94	640
HuaweiTSC	huawei.cpu.base.docker	0.485	34.00	61.10	15743	15741	40	112	254
HuaweiTSC	huawei.cpu.sm.docker	0.455	32.90	60.30	9955	9954	22	94	162
Edinburgh	8-4.tied.tiny_efh_0.3_ft8	0.444	31.80	59.70	13360	13361	36	97	459
Edinburgh	ib-12-4-micro-cpu	0.442	31.90	59.90	12071	12072	18	68	328
Edinburgh	8-4.tied.tiny-cpu	0.439	31.60	59.60	14090	14090	15	65	270
ECNU	ecnu-mt	0.434	33.20	60.70	327823	327764	492	14469	4900
Edinburgh	6-2.micro.4h-cpu	0.418	30.90	59.20	8916	8917	13	63	247
HuaweiTSC	huawei.cpu.t12.docker	0.417	32.20	59.70	7591	7590	15	87	122
Edinburgh	6-2.micro.1h-cpu	0.383	29.90	58.40	8632	8632	13	63	256
Edinburgh	6-2.tied.tiny-cpu	0.378	30.00	58.50	9371	9372	13	63	257
Edinburgh	ib-6-3-tiny-cpu	0.372	30.40	58.80	9258	9258	15	65	302
Edinburgh	12_1.tiny_efh_0.5_ft8	0.371	30.00	58.50	6590	6592	30	91	374
HuaweiTSC	huawei.cpu.t6.docker	0.315	30.20	58.30	5871	5870	11	84	100
CUNI	cuni-large-ende	0.250	30.80	59.10	335787	335806	856	1676	4857
HuaweiTSC	huawei.cpu.ex.docker	0.128	26.30	55.10	6286	6285	7	80	70

Table 3: Results of system evaluation on the latency task. Total time measured in seconds is equivalent to microseconds/sentence because the input is 1 million sentences.

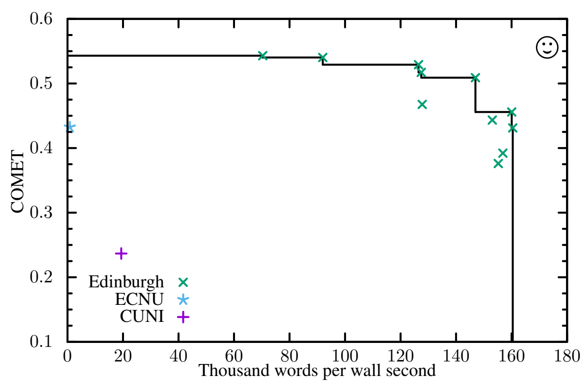
NVIDIA A100 GPU Batch

		Automatic			Seconds		Disk MB		RAM MB
Team	**Variant**	COMET	BLEU	chrF	**Wall**	**CPU**	**Model**	**Docker**	**GPU**
Edinburgh	6-1.base.wide-gpu	0.543	34.60	61.90	283	349	900	2316	37961
Edinburgh	12_1.large-gpu	0.540	34.10	61.60	217	262	624	2039	37555
Edinburgh	6-2.base-gpu	0.529	33.80	61.50	158	169	171	1587	37181
Edinburgh	12_1.base-gpu	0.517	33.90	61.50	156	172	225	1641	37211
Edinburgh	6-1.base-gpu	0.509	33.40	61.10	136	146	159	1574	37175
Edinburgh	8-4.tied.tiny-gpu	0.468	32.50	60.20	156	161	84	1500	37133
Edinburgh	6-2.micro.4h-gpu	0.456	31.90	59.90	125	128	74	1489	37129
Edinburgh	6-2.tied.tiny-gpu	0.443	31.50	59.50	130	134	77	1492	37129
ECNU	ecnu-mt	0.432	33.20	60.70	23600	23643	492	15680	5719
Edinburgh	6-2.micro.1h-gpu	0.431	31.30	59.20	124	128	73	1489	37129
Edinburgh	ib-6-2-tiny-gpu	0.392	31.10	59.50	127	132	81	1496	37133
Edinburgh	ib-12_1-tiny-gpu	0.376	32.10	59.90	128	134	99	1515	37141
CUNI	cuni-large-ende	0.237	30.80	59.10	1029	1115	856	1676	4179

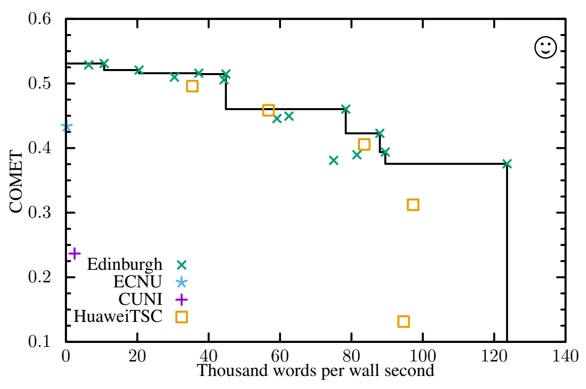
36 Core Ice Lake CPU Batch

		Automatic			Seconds		Disk MB		RAM MB
Team	**Variant**	COMET	BLEU	chrF	**Wall**	**CPU**	**Model**	**Docker**	**CPU**
Edinburgh	12_1.large-cpu	0.531	33.90	61.40	1864	65214	121	171	57879
Edinburgh	6-1.base.wide-cpu	0.529	34.10	61.60	3121	108057	162	212	77379
Edinburgh	12_1.base_efh_0.05	0.521	34.00	61.50	972	34532	176	1176	32754
Edinburgh	6-2.base-cpu	0.516	33.50	61.20	535	18982	32	82	24467
Edinburgh	12_1.base_efh_0.05_ft8	0.514	33.70	61.40	445	15571	156	217	22373
Edinburgh	12_1.base-cpu	0.510	34.00	61.40	656	23159	44	94	33434
Edinburgh	6-1.base-cpu	0.506	33.30	61.00	450	15795	29	79	23520
HuaweiTSC	huawei.cpu.base.docker	0.496	34.10	61.30	562	36577	40	112	17513
Edinburgh	8-4.tied.tiny_efh_0.3_ft8	0.460	31.90	59.80	254	8909	36	97	16473
HuaweiTSC	huawei.cpu.sm.docker	0.459	32.90	60.30	351	21437	22	94	12461
Edinburgh	8-4.tied.tiny-cpu	0.450	31.90	59.80	319	11041	15	65	13880
Edinburgh	ib-12-4-micro-cpu	0.446	32.00	60.00	337	11781	18	68	16707
ECNU	ecnu-mt	0.434	33.20	60.70	88463	2059785	492	14469	2103
Edinburgh	6-2.micro.4h-cpu	0.423	30.90	59.30	227	7925	13	63	11154
HuaweiTSC	huawei.cpu.t12.docker	0.406	31.80	59.60	238	13532	15	87	5797
Edinburgh	6-2.micro.1h-cpu	0.394	30.00	58.50	223	7671	13	63	10526
Edinburgh	6-2.tied.tiny-cpu	0.390	30.30	58.50	244	8559	13	63	12804
Edinburgh	ib-6-3-tiny-cpu	0.381	30.50	58.90	266	9280	15	65	13464
Edinburgh	12_1.tiny_efh_0.5_ft8	0.376	30.20	58.60	161	5531	30	91	11843
HuaweiTSC	huawei.cpu.t6.docker	0.312	30.20	58.40	205	11147	11	84	7166
CUNI	cuni-large-ende	0.237	30.80	59.10	8243	295751	856	1676	138539
HuaweiTSC	huawei.cpu.ex.docker	0.131	26.20	55.20	211	11495	7	80	7458

Table 4: Results of system evaluation on the throughput task. Total time measured in seconds is equivalent to microseconds/sentence because the input is 1 million sentences.



(a) Speed on GPU with COMET for all systems



(b) Speed on 36 Cores with COMET for all systems

Figure 1: Speed and quality of batched submissions. The staircase shows the Pareto frontier.

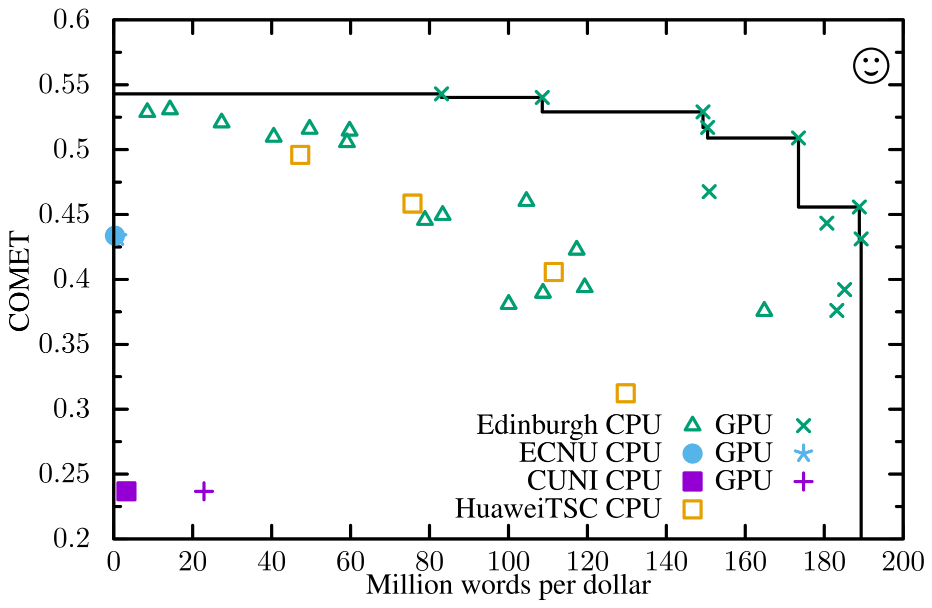


Figure 2: Cost of batched translation for an A100 GPU at \$3.05/hr or 36 Cores of CPU at \$2.7/hr on Oracle Cloud. For readability, we omit systems with a COMET score less than 0.2.

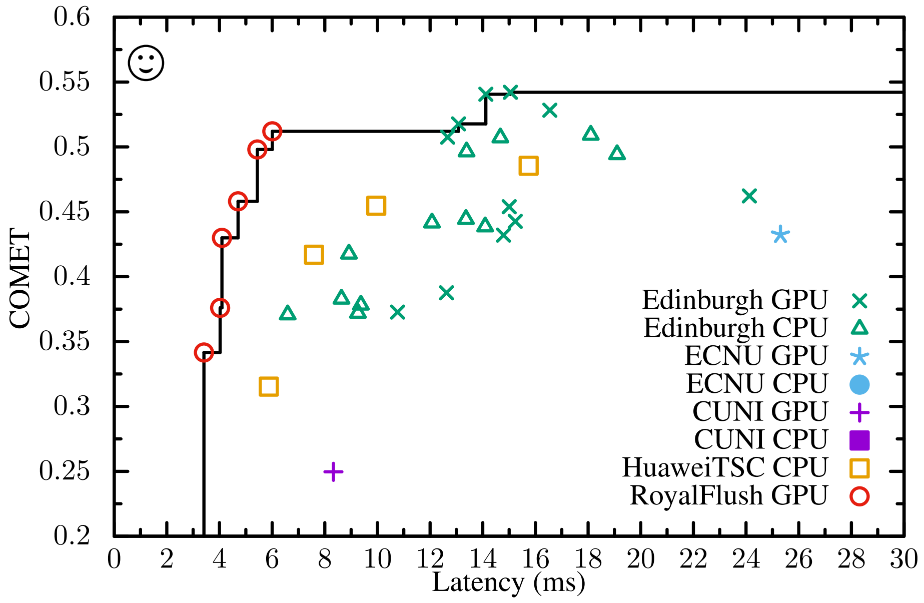
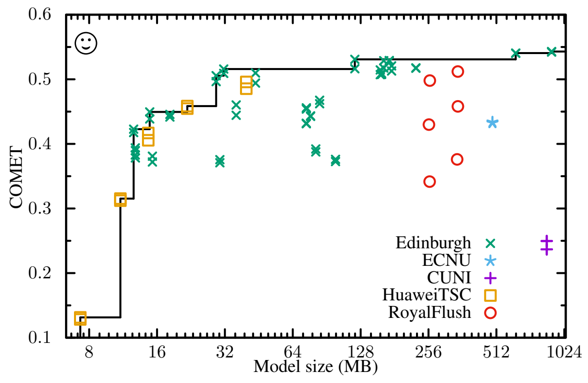
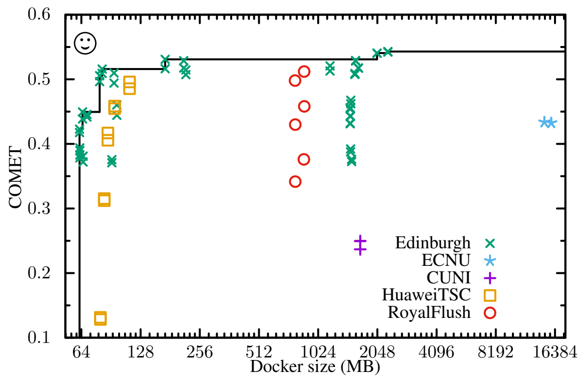


Figure 3: Measured latency for CPU and GPU systems with COMET scores.

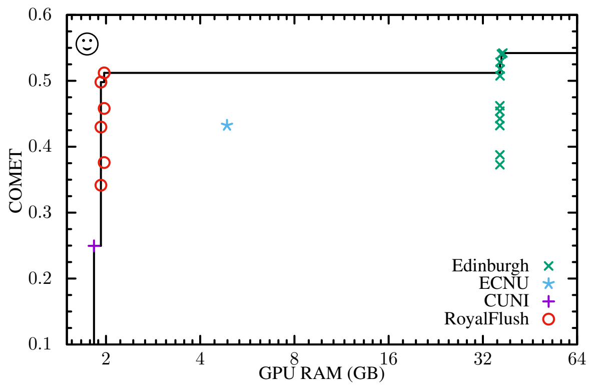


(a) Compressed model size.

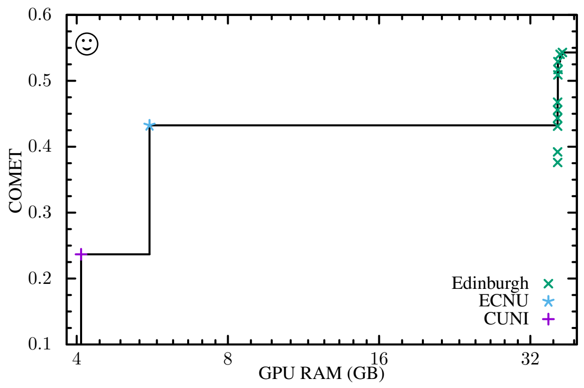


(b) Compressed Docker image size.

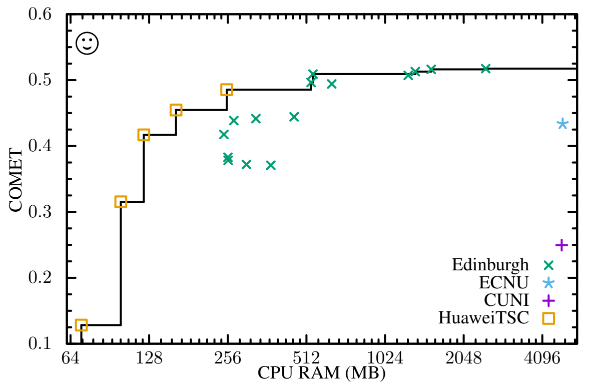
Figure 4: COMET score of systems as a function of model size, and Docker image size. Sizes are reported after compression with `xz`, and are shown on a logarithmic scale. Some participants did not seek to prune image size and included large Linux installations.



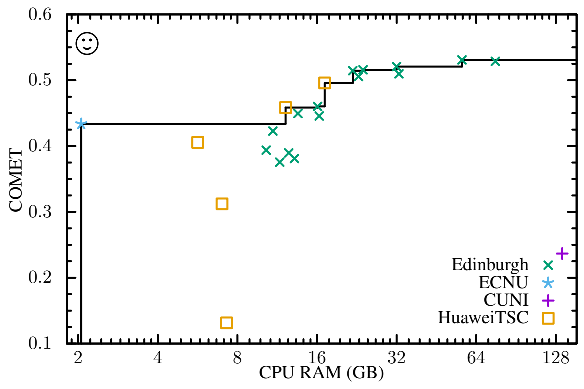
(a) GPU memory consumption with latency.



(b) GPU memory consumption with batching.



(c) 1 Core CPU memory consumption with latency.



(d) 36 Core CPU memory consumption with batching.

Figure 5: RAM consumption of all submissions on a logarithmic scale. Some participants used large batches to favor speed over memory consumption.

seconds on an A100 GPU that costs $3.05/hr in a cloud. That is $0.002/million characters. By comparison, Google Translate's cost is $20/million characters.[1]

In terms of translation throughput cost per $ spent, the GPU submissions are better value for money, provided that enough sentences can be fed to the GPU continuously.

The GPU latency track had been intended to attract non-autoregressive machine translation submissions in their ideal condition with a large GPU and no batch to parallelize. For the first time this year, we had a mix of autoregressive, semi-autoregressive and non-autoregressive systems:

- CUNI submitted a fully non-autoregressive system based on connectionist-temporal-classification (CTC) networks (Helcl et al., 2022).
- Edinburgh submitted bidirectional decoder based semi-autoregressive system (Zhang et al., 2020). This system generates two tokens at an autoregressive step at a time from both sides of the sentences.
- RoyalFlush submitted a semi-autoregressive system based on their novel hybrid regressive translation framework (HRT). They first perform a coarse-grained autoregressive pass that generates some words in the target sentence, with gaps of up to several words in between. Afterwards a second, non-autoregressive pass fills in all the missing words.

The RoyalFlush system proves extremely well suited to the GPU latency task, dominating the pareto frontier in the lower quality setting, even outperforming CPU systems, which have traditionally won this task.

Finally, we note that in semi-autoregressive models and non-autoregressive models, a small drop in BLEU results in a large drop in COMET compared to an autoregressive system, as evidenced by all teams who submitted any form of non-autoregressive MT to the task. This corroborates the findings of (Helcl et al., 2022) where the large discrepancies between BLEU and COMET were noted. We urge participants in future editions of the task to examine manually the output of their non-autoregressive systems.

[1] https://cloud.google.com/translate/pricing

7 Future tasks

This year's shared task had an increased number of participants, likely due to the organisers providing the distilled data and therefore substantially decreasing the computational cost to participants. We intend to keep this format of the task for future years, in the hopes of attracting even more participants.

German is a high-resource language, which raises the computational cost of participation. We would be interested in also potentially including a medium resource language for distillation so that we can see if the methods that work on high-resource languages generalize well to lower-resource languages, or languages with more morphological complexity.

Last year (Heafield et al., 2021) the organisers suggested that an efficient training shared task would be an interesting natural extension to the efficient translation shared task, however it has proven difficult to set up in practice: we are conscious that the validity of such a task can be easily undermined by participants finding a favorable random seed that fits the training data, or more egregiously by including evaluation data in their training data. We are looking for potential solutions to these problems and we are open to suggestions for next year's edition of the task.

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.

Intel Corporation has supported organization. Cloud credits were provided by the Oracle for Research program.

References

Farhad Akhbardeh, Arkady Arkhangorodsky, Magdalena Biesialska, Ondřej Bojar, Rajen Chatterjee, Vishrav Chaudhary, Marta R. Costa-jussa, Cristina España-Bonet, Angela Fan, Christian Federmann, Markus Freitag, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Leonie Harter, Kenneth Heafield, Christopher Homan, Matthias

Huck, Kwabena Amponsah-Kaakyire, Jungo Kasai, Daniel Khashabi, Kevin Knight, Tom Kocmi, Philipp Koehn, Nicholas Lourie, Christof Monz, Makoto Morishita, Masaaki Nagata, Ajay Nagesh, Toshiaki Nakazawa, Matteo Negri, Santanu Pal, Allahsera Auguste Tapo, Marco Turchi, Valentin Vydrin, and Marcos Zampieri. 2021. Findings of the 2021 conference on machine translation (WMT21). In *Proceedings of the Sixth Conference on Machine Translation*, pages 1–88, Online. Association for Computational Linguistics.

Alexandra Birch, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Yusuke Oda. 2018. Findings of the second workshop on neural machine translation and generation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 1–10, Melbourne, Australia. Association for Computational Linguistics.

Ondřej Dušek, Jan Hajič, Jaroslava Hlaváčová, Jindřich Libovický, Pavel Pecina, Aleš Tamchyna, and Zdeňka Urešová. 2017. Khresmoi summary translation test data 2.0. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Markus Freitag, George Foster, David Grangier, Viresh Ratnakar, Qijun Tan, and Wolfgang Macherey. 2021. Experts, errors, and context: A large-scale study of human evaluation for machine translation. *Transactions of the Association for Computational Linguistics*, 9:1460–1474.

Hiroaki Hayashi, Yusuke Oda, Alexandra Birch, Ioannis Konstas, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Katsuhito Sudoh. 2019. Findings of the third workshop on neural generation and translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 1–14, Hong Kong. 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, Qianqian Zhu, and Roman Grundkiewicz. 2021. Findings of the WMT 2021 shared task on efficient translation. In *Proceedings of the Sixth Conference on Machine Translation*, pages 639–651, Online. Association for Computational Linguistics.

Jindřich Helcl, Barry Haddow, and Alexandra Birch. 2022. Non-autoregressive machine translation: It's not as fast as it seems. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1780–1790, Seattle, United States. Association for Computational Linguistics.

Niehues Jan, Roldano Cattoni, Stuker Sebastian, Matteo Negri, Marco Turchi, Salesky Elizabeth, Sanabria Ramon, Barrault Loic, Specia Lucia, and Marcello Federico. 2019. The IWSLT 2019 evaluation campaign. In *16th International Workshop on Spoken Language Translation 2019*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

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.

Nils Reimers and Iryna Gurevych. 2020. Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4512–4525, Online. Association for Computational Linguistics.

Adithya Renduchintala, Denise Diaz, Kenneth Heafield, Xian Li, and Mona Diab. 2021. Gender bias amplification during speed-quality optimization in neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 99–109, Online. Association for Computational Linguistics.

Roberts Rozis and Raivis Skadiņš. 2017. Tilde MODEL - multilingual open data for EU languages. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 263–265, Gothenburg, Sweden. Association for Computational Linguistics.

Gabriel Stanovsky, Noah A. Smith, and Luke Zettlemoyer. 2019. Evaluating gender bias in machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1679–1684, Florence, Italy. 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.