Findings of the WMT 2021 Shared Task on Efficient Translation

Kenneth Heafield[†] **Qianqian Zhu**[†] **Roman Grundkiewicz**[†§]
[†]University of Edinburgh
10 Crichton Street
Edinburgh, Scotland EH8 9AB
[§]Microsoft
1 Microsoft Way
Redmond, WA 98052, USA
{Kenneth.Heafield,Qianqian.Zhu,rgrundki}@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 53 submissions by 4 teams. There were 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 5–20 ms, and models fit in 7.5–150 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 fourth edition of the task, expanding upon previous editions (Heafield et al., 2020; Hayashi et al., 2019; Birch et al., 2018).

Participants built English→German machine translation systems following the constrained data condition of the 2021 Workshop on Machine Translation news translation task. For translation quality measurement, we use the same news-focused WMT21 test set and human evaluation protocol as the news task. However, human assessment was conducted separately from the evaluation of the news task submissions.

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 five conditions in total: GPU Batch (for throughput), GPU Latency, 1 CPU Core Batch, 1 CPU Core Latency, and 36 CPU cores Batch. 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.

	Edinburgh	HuaweiTSC	NiuTrans	TenTrans
GPU Batch	✓		✓	✓
GPU Latency	✓			
1 Core Batch	✓			
1 Core Latency	✓	✓		
36 Cores Batch	✓		✓	

Table 1: Participation in each of the hardware and batching conditions. Core refers to CPU hardware with 1 core or all 36 cores.

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 four participants and the conditions they submitted to.

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, 53 combinations of models, hardware, and batching were benchmarked.

Proceedings of the Sixth Conference on Machine Translation (WMT), pages 639–651
November 10–11, 2021.

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. In practice, most submissions used only one core while NiuTrans's submissions used the CPU cores to parallelize preprocessing and postprocessing.

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 amoritize 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.

Filler data was gathered from parallel corpora and gender bias challenge sets: WMT news test sets from 2008 through 2021 (Barrault et al., 2020), 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 capped sentence lengths at 150 space-separated tokens, except for the WMT 2021 test set to preseve the ability to evaluate with it. 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 including additional tests	14,938
Khresmoi Summary Test v2	1,000
IWSLT 2019	2,278
SimpleGen	2,664
WinoMT	3,888
TED 2020 v1	293,562
Tilde RAPID 2019	654,995
Total	1,010,362
Deduplicated	1,000,000

Table 2: Corpora used for input text.

Input text and tools to extract test sets from system outputs are available at `http://data.statmt.org/heafield/wmt21-testdata.tar.xz`.

The input file has 1,000,000 lines, 19,951,184 space-separated words, and 124,257,215 bytes (most of which are characters since the file is English in UTF-8). This is an average of 20 words per sentence compared to 15 words per sentence the previous year (Heafield et al., 2020) due to raising the cap from 100 to 150 tokens per sentence and the lengthy text in the RAPID corpus.

Teams were responsible for their own tokenization and detokenization. We provided raw UTF-8 English input text with one sentence per line.

4 Metrics

4.1 Cost

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 told to separate their Docker

	Edinburgh	HuaweiTSC	NiuTrans	TenTrans
GPU Batch	3/10		4/4	4/4
GPU Latency	0/11			
1 Core Batch	0/6			
1 Core Latency	3/6	4/4		
36 Cores Batch	0/6		0/2	
Total	6/39	4/4	4/6	4/4

Table 3: Number of submissions by participant and condition (cores refers to the CPU hardware). The number after / is all submissions by the participant. The number before / is how many participants selected for focused human evaluation based on automatic metrics.

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 included in results but small relative to the cost of translation. 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`), though participants were encouraged to prioritize shipping one container for multiple hardware conditions over the size of the container.

4.2 Quality

Translation quality is measured on the WMT 2021 news test set. The automatic metrics are COMET (Rei et al., 2020) `wmt20-comet-da` from version `1.0.0rc6`, BLEU from sacrebleu (Post, 2018) `nrefs:3|case:mixed|eff:no|tok:13a|smooth:exp|version:2.0.0`, and chrF also from sacrebleu. We use references A, C, and D because the organizers found postedited DeepL output in reference B. COMET does not natively support multiple references so we averaged as recommended by the authors.[1] We also averaged chrF across references. Results were presented to participants[2] who were encouraged to whittle down systems for a focused human evaluation. HuaweiTSC and TenTrans included all of their submissions. NiuTrans included their GPU submissions but not their CPU submissions that have lower automatic scores than Edinburgh's. This left GPU Batch and 1 Core Latency as the only conditions with multiple teams. Edinburgh kept systems that have competitors and are near the Pareto frontier. The number of submissions evaluated is shown in Table 3. Out of 53 submissions, we ran direct assessment on 18.

For human evaluation, as a source of the absolute quality measure we used document-level source-based direct assessments (DA) (Graham et al., 2013; Cettolo et al., 2017) following the procedure established at the WMT20 News Translation Task (Barrault et al., 2020). We also conducted contrastive evaluation using segment-level pairwise direct assessments (Novikova et al., 2018; Sakaguchi and Van Durme, 2018), because it can be a better discriminative tool for measuring relative quality difference between pairs of systems. We compared the 18 systems using source-based direct assessment and 58 pairs of systems with contrastive direct assessment. In total, we gathered 21,487 and 20,416 direct assessment scores in standard and contrastive campaigns respectively. All annotations were made by bilingual native German speakers with a translation or linguistics background. Annotations were collected using Appraise[3] (Federmann, 2018).

5 Results

All submissions are shown in Table 4. Source-based direct assessment scores appear for the submissions in the focused human evaluation with the number of wins against other systems (including those in other conditions), raw direct accessment score, and z-score after standardizing annotator scores to mitigate differences in annotator scores. Scores were averaged ("Ave.") across sentences. Rows are sorted by COMET because only some submissions have human assessment.

The system ranking based on the standard DA is presented in Table 5. Systems are ordered by the number of respective wins against other systems and average DA z-score. Ordering solely by z-scores would produce three clusters with all systems within a cluster considered tied according to Wilcoxon rank-sum test with $p < 0.05$.

[1] https://github.com/Unbabel/COMET/issues/20

[2] Only reference A was available at the time.

[3] https://github.com/AppraiseDev/Appraise

NVIDIA A100 GPU Batch

		Human			Automatic			Seconds		Disk MB		RAM MB	
Team	Variant	Win	Ave.	Ave. z	COMET	BLEU	chrF	Wall	CPU	Model	Docker	CPU	GPU
Edinburgh	base	17	90.3	0.352	0.527	55.25	61.54	140	152	150	455	1725	36140
Edinburgh	tiny11	14	85.9	0.185	0.492	52.74	60.52	115	120	60	364	1622	36092
Edinburgh	2.12-2.tied.tiny.heads-0.3				0.473	52.36	60.32	126	130	59	363	1618	36090
Edinburgh	2.6-2.tied.tiny.heads-0.3				0.459	51.52	60.00	116	120	53	357	1611	36088
Edinburgh	2.12_1.tiny.heads-0.3				0.445	52.20	60.25	117	121	62	366	1620	36092
Edinburgh	2.12_1.micro.heads-0.3				0.440	51.73	60.02	117	121	60	364	1617	36092
Edinburgh	2.8-4.tied.tiny.4bit				0.432	50.20	59.47	140	144	8	355	1639	29054
NiuTrans	6_1_512	9	83.5	0.057	0.423	50.05	59.96	95	377	73	303	2447	4254
NiuTrans	12_1_512	4	88.8	0.016	0.422	50.50	59.83	124	411	109	335	2458	4356
Edinburgh	2.12_1.tiny.4bit	6	85.6	0.104	0.422	51.78	59.86	118	122	10	357	1659	29062
NiuTrans	6_1_0	4	80.4	-0.019	0.384	49.78	59.71	94	400	72	302	2467	3998
Edinburgh	3.12_1.micro				0.382	50.40	59.29	116	121	66	370	1627	36094
NiuTrans	3_1_512	3	85.6	-0.035	0.354	48.72	59.25	81	380	55	287	2475	4134
Edinburgh	2.12_1.micro.rowcol-0.5				0.352	48.73	58.59	107	110	42	346	1603	36082
TenTrans	tea-20_6-h512-ffn4096	3	81.1	-0.046	0.335	46.26	57.19	456	638	643	1804	2380	25318
TenTrans	stu-20_1-h512-ffn2048	2	81.8	-0.104	0.291	45.89	57.06	340	528	355	1272	2120	17126
TenTrans	stu-10_1-h512-ffn2048	2	82.5	-0.138	0.263	44.88	56.89	280	458	234	1049	2006	17128
TenTrans	stu-20_1-h256-ffn1024	2	84.3	-0.091	0.238	44.34	56.68	257	443	114	829	1864	17126

NVIDIA A100 GPU Latency

		Human			Automatic			Seconds		Disk MB		RAM MB	
Team	Variant	Win	Ave.	Ave. z	COMET	BLEU	chrF	Wall	CPU	Model	Docker	CPU	GPU
Edinburgh	base				0.527	55.25	61.54	16851	16859	150	455	1573	36140
Edinburgh	tiny11				0.491	52.80	60.55	15101	15102	60	364	1247	36092
Edinburgh	2.12_1.base.4bit				0.476	53.81	60.86	15239	15243	22	369	1653	38174
Edinburgh	2.12-2.tied.tiny.heads-0.3				0.473	52.39	60.32	18269	18271	59	363	1233	36090
Edinburgh	2.6-2.tied.tiny.heads-0.3				0.460	51.60	59.99	17204	17205	53	357	1216	36088
Edinburgh	2.12_1.tiny.heads-0.3				0.445	52.13	60.22	13839	13841	62	366	1241	36092
Edinburgh	2.12_1.micro.heads-0.3				0.436	51.66	59.99	13952	13952	60	364	1236	36092
Edinburgh	2.8-4.tied.tiny.4bit				0.431	50.26	59.49	26635	26637	8	355	1264	29054
Edinburgh	2.12_1.tiny.4bit				0.419	51.79	59.87	13876	13878	10	357	1299	29062
Edinburgh	3.12_1.micro				0.379	50.40	59.34	13944	13945	66	370	1251	36094
Edinburgh	2.12_1.micro.rowcol-0.5				0.352	48.73	58.61	13665	13665	42	346	1184	36082

1 Core Ice Lake CPU Batch

		Human			Automatic			Seconds		Disk MB		RAM MB
Team	Variant	Win	Ave.	Ave. z	COMET	BLEU	chrF	Wall	CPU	Model	Docker	CPU
Edinburgh	base				0.520	54.72	61.36	11067	11066	45	63	1569
Edinburgh	3.12_1.large				0.485	53.71	60.89	30342	30338	129	386	2428
Edinburgh	tiny11				0.464	52.24	60.17	5108	5107	21	468	621
Edinburgh	4.12_1.tiny.rowcol-0.5.ft8				0.328	48.34	58.33	3288	3287	52	302	1040
Edinburgh	4.12_1.micro.rowcol-0.5.ft8				0.326	48.97	58.41	3497	3497	17	302	912
Edinburgh	4.12_1.micro.rowcol-0.5				0.318	47.66	58.01	4046	4045	53	338	781

1 Core Ice Lake CPU Latency

		Human			Automatic			Seconds		Disk MB		RAM MB
Team	Variant	Win	Ave.	Ave. z	COMET	BLEU	chrF	Wall	CPU	Model	Docker	CPU
Edinburgh	base	13	88.3	0.205	0.465	53.53	60.69	16815	16814	45	63	542
HuaweiTSC	base	7	90.3	-0.019	0.450	53.00	60.82	14939	14937	37	53	377
Edinburgh	3.12_1.large				0.430	52.95	60.34	40518	40514	129	386	1175
Edinburgh	tiny11	3	81.4	-0.008	0.413	51.18	59.63	9272	9272	21	468	241
HuaweiTSC	sm9	4	86.1	-0.001	0.391	50.58	59.74	8866	8865	20	36	206
HuaweiTSC	sm6	2	77.5	-0.025	0.338	48.75	58.85	7714	7713	17	33	173
Edinburgh	4.12_1.micro.rowcol-0.5	0	84.0	-0.444	0.257	47.56	57.88	6343	6343	53	338	342
HuaweiTSC	tiny	0	81.9	-0.363	0.197	44.20	56.84	5138	5138	10	27	107
Edinburgh	4.12_1.tiny.rowcol-0.5.ft8				-0.073	37.43	56.33	8148	8147	52	302	335
Edinburgh	4.12_1.micro.rowcol-0.5.ft8				-0.173	37.67	55.71	7564	7563	17	302	239

36 Core Ice Lake CPU Batch

		Human			Automatic			Seconds		Disk MB		RAM MB
Team	Variant	Win	Ave.	Ave. z	COMET	BLEU	chrF	Wall	CPU	Model	Docker	CPU
Edinburgh	base				0.519	54.69	61.35	500	17790	45	63	28630
Edinburgh	3.12_1.large				0.484	54.02	60.92	1509	53528	129	386	34903
Edinburgh	tiny11				0.465	52.17	60.16	237	8434	21	468	15594
NiuTrans	6_1_512				0.430	50.08	60.02	520	36015	146	142	57636
NiuTrans	3_1_512				0.358	48.53	59.34	417	28727	109	126	56415
Edinburgh	4.12_1.tiny.rowcol-0.5.ft8				0.336	48.38	58.37	159	5682	52	302	18606
Edinburgh	4.12_1.micro.rowcol-0.5.ft8				0.329	48.95	58.42	167	5948	17	302	15825
Edinburgh	4.12_1.micro.rowcol-0.5				0.318	47.98	58.16	184	6540	53	338	16469

Table 4: All submissions. Human source-based DA is shown for selected submissions. Total time measured in seconds is equivalent to microseconds/sentence because the input is 1 million sentences.

Team	Variant	Win	Ave.	Ave. z	Time (s)	Condition
Edinburgh	base	17	90.3	0.352	140	GPU Batch
Edinburgh	tiny11	14	85.9	0.185	115	GPU Batch
Edinburgh	base	13	88.3	0.205	16815	1 Core Latency
NiuTrans	6_1_512	9	83.5	0.057	95	GPU Batch
HuaweiTSC	base	7	90.3	-0.019	14939	1 Core Latency
Edinburgh	2.12_1.tiny.4bit	6	85.6	0.104	118	GPU Batch
NiuTrans	12_1_512	4	88.8	0.016	124	GPU Batch
HuaweiTSC	sm9	4	86.1	-0.001	8866	1 Core Latency
NiuTrans	6_1_0	4	80.4	-0.019	94	GPU Batch
Edinburgh	tiny11	3	81.4	-0.008	9272	1 Core Latency
NiuTrans	3_1_512	3	85.6	-0.035	81	GPU Batch
TenTrans	tea-20_6-h512-ffn4096	3	81.1	-0.046	456	GPU Batch
HuaweiTSC	sm6	2	77.5	-0.025	7714	1 Core Latency
TenTrans	stu-20_1-h256-ffn1024	2	84.3	-0.091	257	GPU Batch
TenTrans	stu-20_1-h512-ffn2048	2	81.8	-0.104	340	GPU Batch
TenTrans	stu-10_1-h512-ffn2048	2	82.5	-0.138	280	GPU Batch
HuaweiTSC	tiny	0	81.9	-0.363	5138	1 Core Latency
Edinburgh	4.12_1.micro.rowcol-0.5	0	84.0	-0.444	6343	1 Core Latency

Table 5: System ranking based on the standard direct assessment (DA) human evaluation. The rows are ordered by the number of respective wins against other systems, followed by the DA z-score. Systems within a cluster are considered tied according to Wilcoxon rank-sum test $p < 0.05$ with standard DA.

Figure 1 shows the trade-off between quality and speed of batched translation submissions. Since source-based DA is available for select GPU submissions, we include that comparison; the other plots rely on COMET to approximate quality. Each plot shows the Pareto frontier as a black staircase to highlight the best combinations of quality and speed. In Figure 2, we combine GPU and 36 Core CPU speed by using Oracle Cloud pricing. The GPU is cheaper for throughput-oriented tasks that allow batching.

Latency is shown in Figures 3 and 4. HuaweiTSC and Edinburgh were the two participants and shared the Pareto frontier. While the GPU is cheaper for throughput, both CPU and GPU entries appear on the Pareto frontier for latency. In fact, the lowest latencies are achieved by single-core CPU submissions, likely due to the overhead of launching small kernels on a GPU.

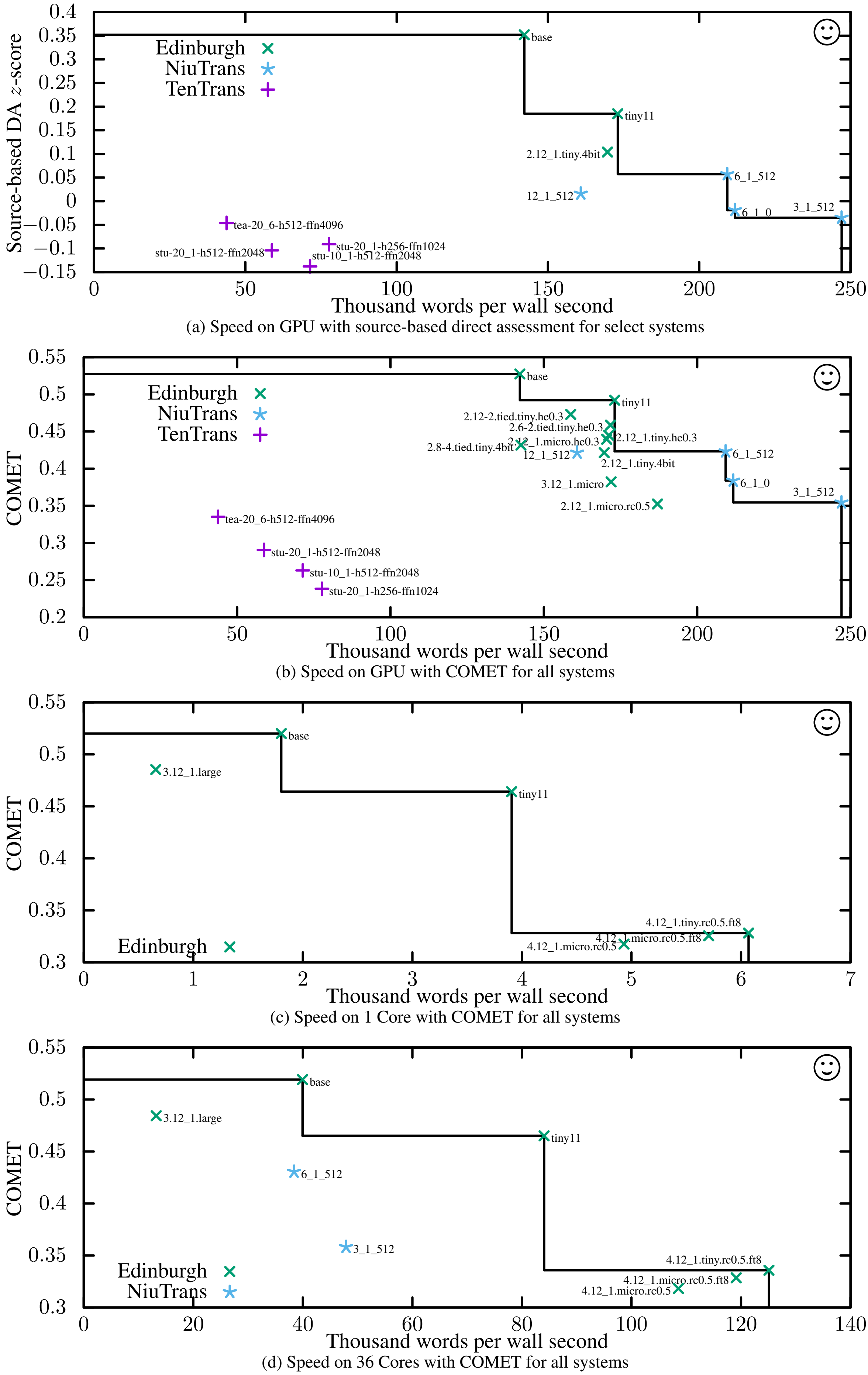
Model sizes at rest on disk appear in Figures 5 and 6. Participants were allowed to compress their models using their own tools and standard tools like `xz`. The entire Pareto frontier consists of Edinburgh submissions, resting partly on 4-bit integer compression. Docker image sizes, which include model and software, appear in Figure 7. HuaweiTSC optimized their image size well. 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 8. 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. NiuTrans is the clear winner on GPU RAM consumption and curiously the clear loser on CPU RAM consumption.

Many of the systems tied on standard DA and contrastive DA helps us pull them apart by directly comparing system outputs. Table 6 shows detailed results of contrastive DA including average scores, respective deltas between two systems and the outcome of significance testing. For groups of systems for which we evaluated each system from a group against each other system from the same group, we created separate rankings based solely on pairwise comparisons within the group, presented in Table 7.

6 Conclusion and Future Tasks

Using the highest quality system in this evaluation, translating 124,257,215 characters took 140 seconds on an A100 GPU that costs \$3.05/hr in a cloud. That is \$0.001/million characters. By comparison, Google Translate's cost is \$20/million



(a) Speed on GPU with source-based direct assessment for select systems

(b) Speed on GPU with COMET for all systems

(c) Speed on 1 Core with COMET for all systems

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

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

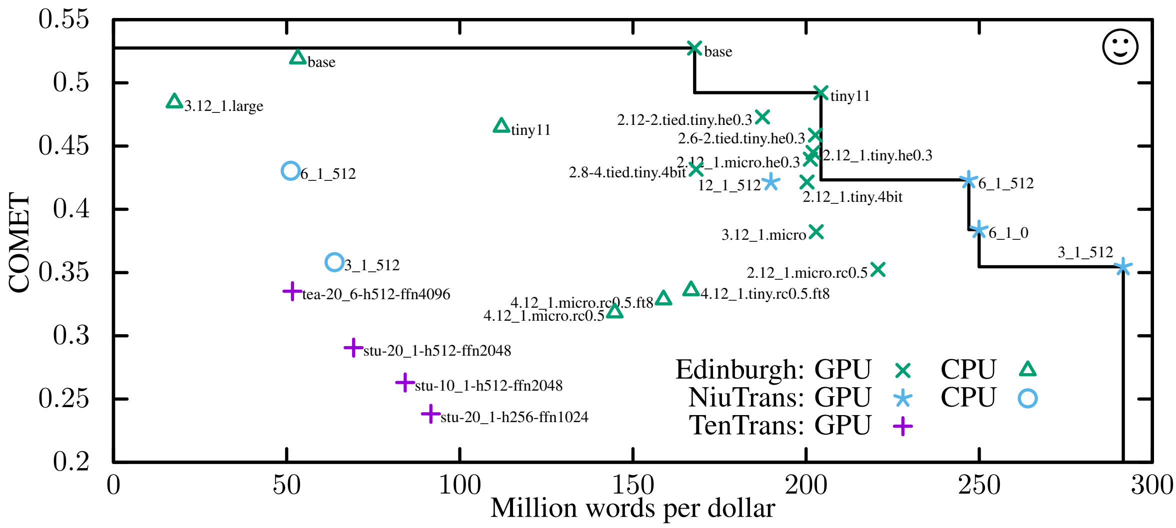


Figure 2: Cost of batched translation for an A100 GPU at $3.05/hr or 36 Cores of CPU at $2.7/hr on Oracle Cloud.

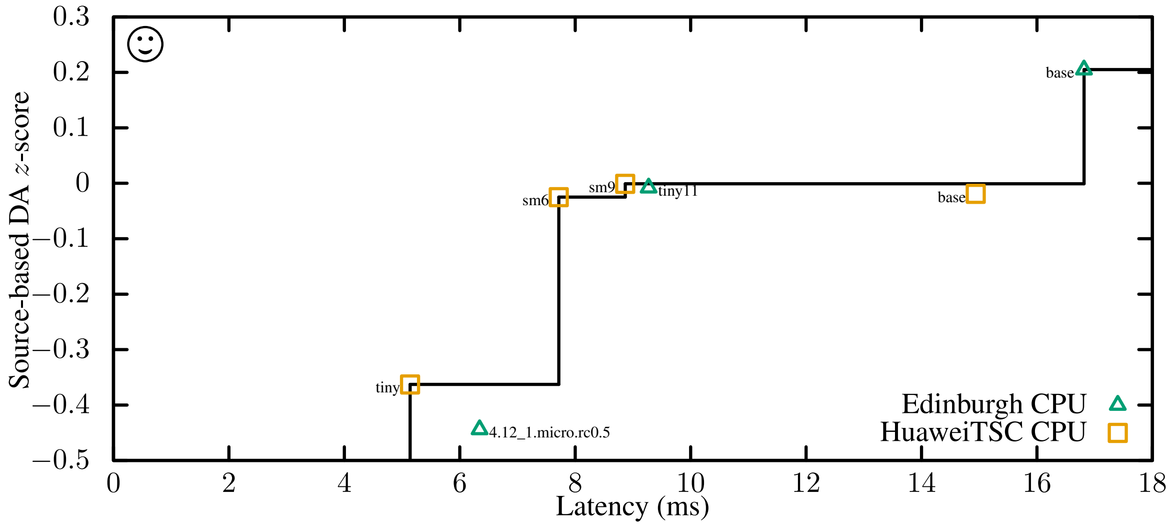


Figure 3: Latency of select CPU systems with source-based direct assessment. Contrastive direct assessment (Table 7) insignificantly ranked HuaweiTSC's base > Edinburgh's tiny11 > HuaweiTSC's sm9.

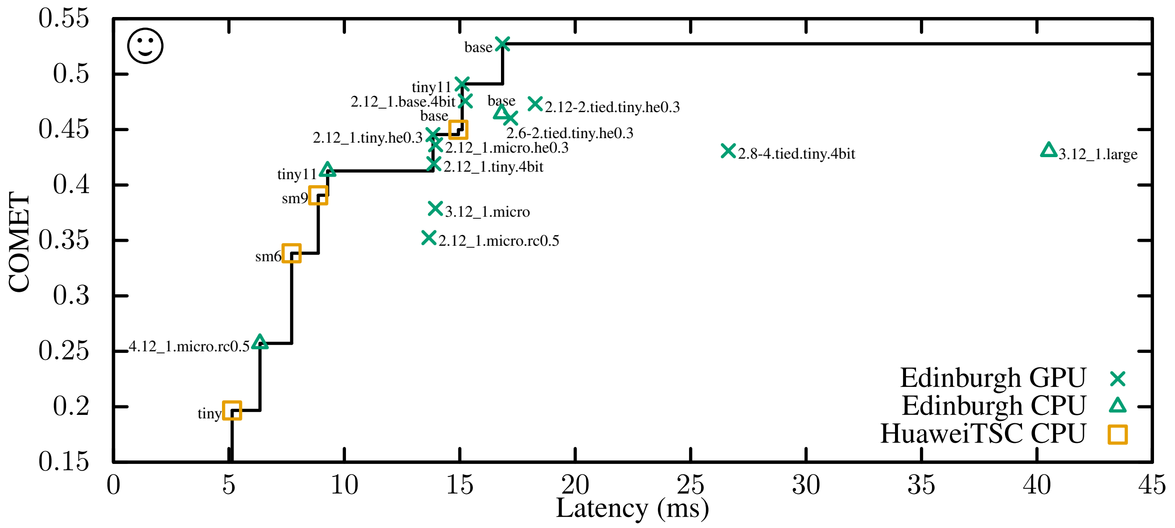


Figure 4: Latency of combined CPU and GPU systems with COMET scores. To improve the scale of the graph, low-quality variants 4.12_1.tiny.rowcol-0.5.ft8 and 4.12_1.micro.rowcol-0.5.ft8 from Edinburgh are not shown. Their respective COMET scores are -0.073 and -0.173.

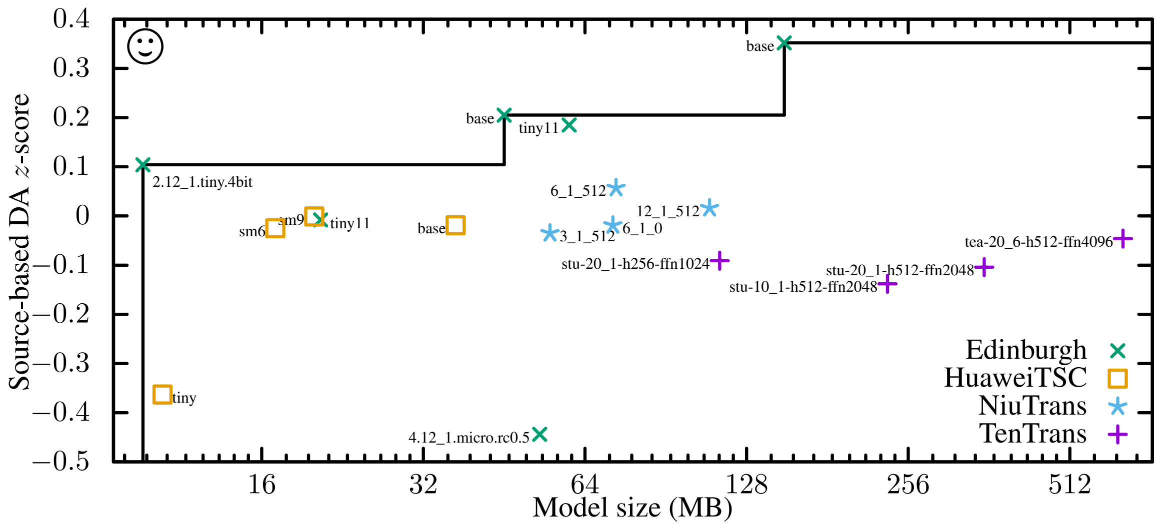


Figure 5: Model sizes of select systems in the human evaluation with source-based DA. Because selection for human evaluation focused on speed (and not model size), this is missing the smallest model, Edinburgh's 2.8-4.tied.tiny.4bit and a few other Pareto optimal systems identified by automatic metrics.

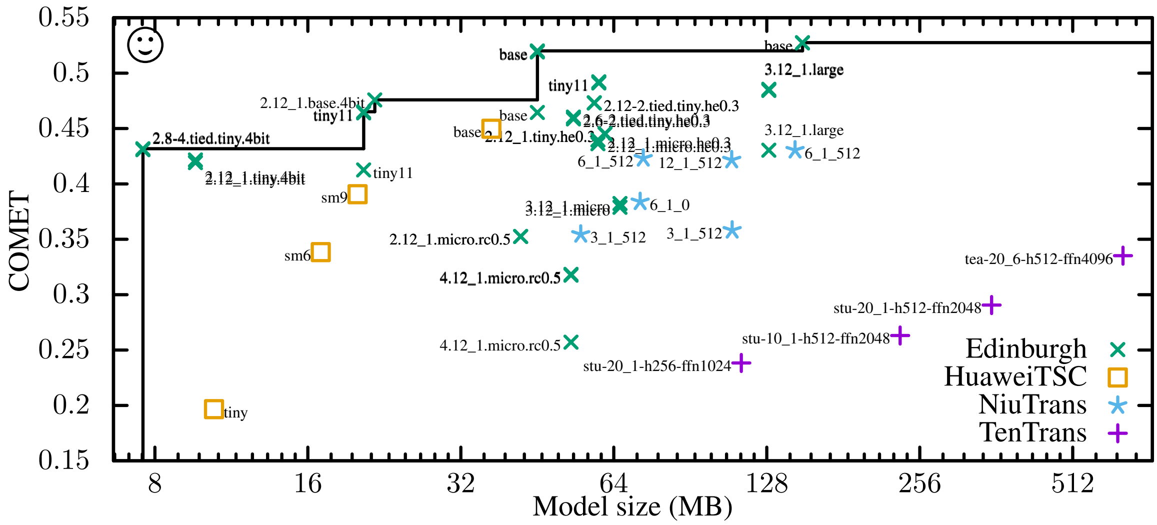


Figure 6: All model sizes with quality by COMET. Because models had slightly different output in different hardware conditions, the same variant label can appear multiple times like a shadow. Low-quality variants 4.12_1.tiny.rowcol-0.5.ft8 and 4.12_1.micro.rowcol-0.5.ft8 from Edinburgh are omitted for scale.

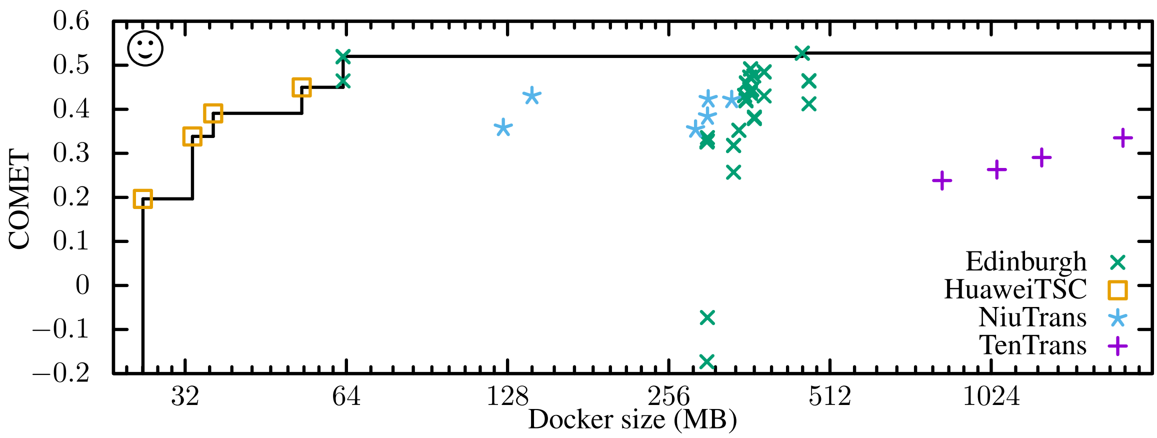
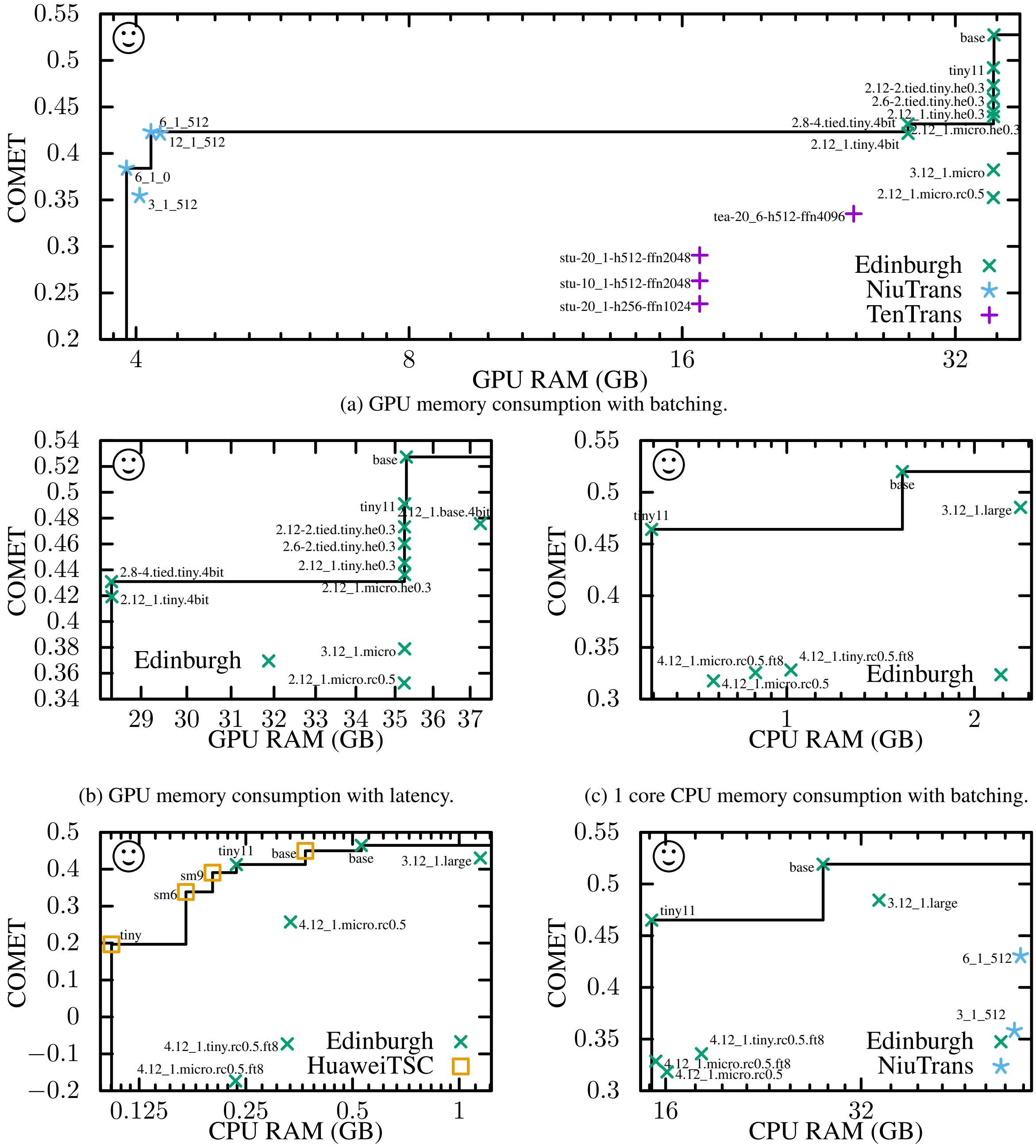


Figure 7: Size of all Docker images after compression with `xz` on a logarithmic scale. Some participants did not seek to prune image size and included large Linux installations. Labels are not shown due to crowding.



(a) GPU memory consumption with batching.

(b) GPU memory consumption with latency.

(c) 1 core CPU memory consumption with batching.

(d) 1 core CPU memory consumption with latency.

(e) 36 core CPU memory consumption with batching.

Figure 8: RAM consumption of all submissions on a logarithmic scale. Some participants used large batches to favor speed over memory consumption.

Stronger System			Weaker System			Stronger	Weaker		
Team	**Variant**	**Condition**	**Team**	**Variant**	**Condition**	**DA Score**	**DA Score**	**Delta**	p**-val**
Edinburgh	base	GPU Latency	Edinburgh	base	GPU Batch	92.2	92.2	0.0	
Edinburgh	base	GPU Batch	Edinburgh	tiny11	GPU Batch	74.9	74.7	0.2	
Edinburgh	tiny11	GPU Batch	Edinburgh	tiny11	GPU Latency	86.6	86.4	0.2	
Edinburgh	tiny11	GPU Batch	Edinburgh	2.12_1.tiny.4bit	GPU Batch	85.6	83.6	1.9	
NiuTrans	12_1_512	GPU Batch	NiuTrans	6_1_0	GPU Batch	78.1	75.2	2.8	**
NiuTrans	12_1_512	GPU Batch	NiuTrans	3_1_512	GPU Batch	67.7	65.1	2.6	*
NiuTrans	12_1_512	GPU Batch	NiuTrans	6_1_512	GPU Batch	65.9	65.1	0.8	
NiuTrans	6_1_0	GPU Batch	NiuTrans	3_1_512	GPU Batch	86.7	85.8	0.9	
NiuTrans	6_1_512	GPU Batch	NiuTrans	3_1_512	GPU Batch	79.7	78.2	1.4	
NiuTrans	6_1_512	GPU Batch	NiuTrans	6_1_0	GPU Batch	82.7	82.6	0.0	
TenTrans	stu-10_1-h512-ffn2048	GPU Batch	TenTrans	tea-20_6-h512-ffn4096	GPU Batch	85.4	83.3	2.1	
TenTrans	stu-10_1-h512-ffn2048	GPU Batch	TenTrans	stu-20_1-h256-ffn1024	GPU Batch	83.6	83.3	0.3	
TenTrans	stu-10_1-h512-ffn2048	GPU Batch	TenTrans	stu-20_1-h512-ffn2048	GPU Batch	82.0	79.8	2.2	
TenTrans	tea-20_6-h512-ffn4096	GPU Batch	TenTrans	stu-20_1-h256-ffn1024	GPU Batch	73.9	63.2	10.7	***
TenTrans	stu-20_1-h512-ffn2048	GPU Batch	TenTrans	stu-20_1-h256-ffn1024	GPU Batch	66.9	66.3	0.6	
TenTrans	tea-20_6-h512-ffn4096	GPU Batch	TenTrans	stu-20_1-h512-ffn2048	GPU Batch	88.4	88.0	0.4	*
Edinburgh	base	GPU Batch	TenTrans	tea-20_6-h512-ffn4096	GPU Batch	84.4	78.9	5.5	**
Edinburgh	base	GPU Batch	TenTrans	stu-20_1-h256-ffn1024	GPU Batch	88.7	82.1	6.6	***
Edinburgh	tiny11	GPU Batch	TenTrans	tea-20_6-h512-ffn4096	GPU Batch	88.0	81.1	6.9	**
Edinburgh	tiny11	GPU Batch	TenTrans	stu-20_1-h256-ffn1024	GPU Batch	72.1	57.6	14.5	***
Edinburgh	base	GPU Batch	NiuTrans	6_1_512	GPU Batch	87.4	74.7	12.7	***
Edinburgh	base	GPU Batch	NiuTrans	3_1_512	GPU Batch	83.0	73.7	9.3	***
Edinburgh	tiny11	GPU Batch	NiuTrans	6_1_512	GPU Batch	68.6	65.8	2.8	
Edinburgh	tiny11	GPU Batch	NiuTrans	3_1_512	GPU Batch	91.8	87.4	4.4	***
TenTrans	tea-20_6-h512-ffn4096	GPU Batch	NiuTrans	6_1_512	GPU Batch	67.2	65.9	1.3	
TenTrans	tea-20_6-h512-ffn4096	GPU Batch	NiuTrans	3_1_512	GPU Batch	89.2	87.3	1.9	***
NiuTrans	6_1_512	GPU Batch	TenTrans	stu-20_1-h256-ffn1024	GPU Batch	94.6	93.5	1.1	**
NiuTrans	3_1_512	GPU Batch	TenTrans	stu-20_1-h256-ffn1024	GPU Batch	84.4	82.3	2.1	
Edinburgh	base	GPU Latency	HuaweiTSC	base	1 Core Latency	91.4	86.9	4.6	***
Edinburgh	base	GPU Latency	HuaweiTSC	sm9	1 Core Latency	77.3	69.7	7.6	***
Edinburgh	base	GPU Latency	HuaweiTSC	sm6	1 Core Latency	86.0	77.6	8.4	***
Edinburgh	base	GPU Latency	HuaweiTSC	tiny	1 Core Latency	90.8	77.2	13.6	***
Edinburgh	tiny11	GPU Latency	HuaweiTSC	base	1 Core Latency	89.3	84.2	5.1	**
Edinburgh	tiny11	GPU Latency	HuaweiTSC	sm9	1 Core Latency	88.5	83.2	5.4	***
Edinburgh	tiny11	GPU Latency	HuaweiTSC	sm6	1 Core Latency	92.9	89.2	3.7	***
Edinburgh	tiny11	GPU Latency	HuaweiTSC	tiny	1 Core Latency	82.4	73.7	8.7	***
Edinburgh	base	1 Core Latency	Edinburgh	tiny11	1 Core Latency	67.5	65.0	2.5	**
Edinburgh	base	1 Core Latency	Edinburgh	4.12_1.micro.rowcol-0.5	1 Core Latency	66.9	62.2	4.8	***
Edinburgh	tiny11	1 Core Latency	Edinburgh	4.12_1.micro.rowcol-0.5	1 Core Latency	81.1	74.5	6.7	***
HuaweiTSC	base	1 Core Latency	HuaweiTSC	sm9	1 Core Latency	87.5	85.0	2.5	*
HuaweiTSC	base	1 Core Latency	HuaweiTSC	sm6	1 Core Latency	89.2	86.0	3.2	**
HuaweiTSC	base	1 Core Latency	HuaweiTSC	tiny	1 Core Latency	94.5	86.4	8.2	***
HuaweiTSC	sm9	1 Core Latency	HuaweiTSC	sm6	1 Core Latency	68.8	68.0	0.9	
HuaweiTSC	sm9	1 Core Latency	HuaweiTSC	tiny	1 Core Latency	90.2	85.8	4.3	***
HuaweiTSC	sm6	1 Core Latency	HuaweiTSC	tiny	1 Core Latency	79.3	73.2	6.1	***
HuaweiTSC	base	1 Core Latency	Edinburgh	base	1 Core Latency	84.7	84.6	0.1	
Edinburgh	base	1 Core Latency	HuaweiTSC	sm9	1 Core Latency	78.5	74.8	3.7	*
Edinburgh	base	1 Core Latency	HuaweiTSC	sm6	1 Core Latency	89.0	85.7	3.3	**
Edinburgh	base	1 Core Latency	HuaweiTSC	tiny	1 Core Latency	87.9	79.8	8.2	***
HuaweiTSC	base	1 Core Latency	Edinburgh	tiny11	1 Core Latency	90.7	90.4	0.2	
Edinburgh	tiny11	1 Core Latency	HuaweiTSC	sm9	1 Core Latency	81.5	78.9	2.5	
Edinburgh	tiny11	1 Core Latency	HuaweiTSC	sm6	1 Core Latency	90.9	90.1	0.7	
Edinburgh	tiny11	1 Core Latency	HuaweiTSC	tiny	1 Core Latency	85.9	77.9	8.0	***
HuaweiTSC	base	1 Core Latency	Edinburgh	4.12_1.micro.rowcol-0.5	1 Core Latency	89.4	81.9	7.5	***
HuaweiTSC	sm9	1 Core Latency	Edinburgh	4.12_1.micro.rowcol-0.5	1 Core Latency	93.4	90.8	2.6	
HuaweiTSC	sm6	1 Core Latency	Edinburgh	4.12_1.micro.rowcol-0.5	1 Core Latency	84.8	82.0	2.8	*
HuaweiTSC	tiny	1 Core Latency	Edinburgh	4.12_1.micro.rowcol-0.5	1 Core Latency	84.6	82.1	2.4	*

Table 6: Results of the pairwise contrastive direct assessment human evaluation for each evaluated system pair. The stronger system on the left is considered better than the weaker system on the right according to the Wilcoxon rank-sum test with p < 0.05 for $*$, p < 0.01 for $**$, p < 0.001 for $***$.

Team	Variant	Win	Ave.	Ave. z	Time (s)
NiuTrans	12_1_512	1	70.0	0.060	124
NiuTrans	6_1_512	0	77.3	0.017	95
NiuTrans	6_1_0	0	81.6	-0.023	94
NiuTrans	3_1_512	0	78.1	-0.057	81

(a) NiuTrans GPU Throughput

Team	Variant	Win	Ave.	Ave. z	Time (s)
TenTrans	stu-10_1-h512-ffn2048	1	85.9	0.104	280
TenTrans	stu-20_1-h512-ffn2048	0	87.7	-0.011	340
TenTrans	tea-20_6-h512-ffn4096	0	85.6	-0.069	456

(b) Tentrans GPU Throughput

Team	Variant	Win	Ave.	Ave. z	Time (s)
HuaweiTSC	base	2	91.6	0.181	14939
HuaweiTSC	sm9	1	79.5	0.139	8866
HuaweiTSC	sm6	1	75.6	0.005	7714
HuaweiTSC	tiny	0	81.3	-0.250	5138

(c) HuaweiTSC 1 Core Latency

Team	Variant	Win	Ave.	Ave. z	Time (s)
Edinburgh	base	3	83.8	0.214	140
Edinburgh	tiny11	3	75.3	0.106	115
TenTrans	tea-20_6-h512-ffn4096	1	76.3	-0.067	456
NiuTrans	3_1_512	0	83.8	-0.064	81
NiuTrans	6_1_512	0	74.7	-0.087	95
TenTrans	stu-20_1-h256-ffn1024	0	75.8	-0.210	257

(d) GPU Throughput

Team	Variant	Win	Ave.	Ave. z	Time (s)
Edinburgh	base	4	82.1	0.122	16815
Edinburgh	tiny11	2	88.7	0.078	9272
HuaweiTSC	sm9	1	85.6	-0.003	8866
HuaweiTSC	base	0	85.5	0.051	14939
HuaweiTSC	sm6	0	86.8	-0.027	7714
Edinburgh	4.12_1.micro.rowcol-0.5	0	86.9	-0.065	6343
HuaweiTSC	tiny	0	78.5	-0.131	5138

(e) Latency on 1 Core CPU. Total wall Time (s) is the same value as μs/sentence because there are 1 million sentences.

Team	Variant	Win	Ave.	Ave. z	Time (s)	Condition
Edinburgh	tiny11	4	86.7	0.165	15101	GPU Latency
Edinburgh	base	3	89.3	0.238	16851	GPU Latency
HuaweiTSC	sm9	2	79.8	-0.146	8866	1 Core Latency
HuaweiTSC	sm6	0	89.7	-0.155	7714	1 Core Latency
HuaweiTSC	base	0	81.8	-0.161	14939	1 Core Latency
HuaweiTSC	tiny	0	72.8	-0.342	5138	1 Core Latency

(f) Latency on GPU vs 1 Core CPU. Total wall Time (s) is the same value as μs/sentence because there are 1 million sentences.

Table 7: System rankings based on contrastive DA human-evaluation within selected groups of systems. Each system within a group was evaluated against each other system. Systems are ordered by the number of respective wins against other systems and DA z-score.

characters.[4]

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. However, non-autoregressive papers (Libovický and Helcl, 2018; Gu and Kong, 2021) often rely on unreasonably poor autoregressive baselines in order to claim impressive-sounding speedups, when they are in fact slower than optimized autoregressive models seen here. While previous editions of the task did not measure latency, disabling batching is a simple command line modification to systems that existed at the time (Birch et al., 2018) but were omitted as baselines in non-autoregressive literature. All submissions this year are autoregressive.

An efficient training task is a natural extension. The challenge lies in defining proper development and testing conditions. Otherwise, participants will overfit by searching for the random seed that trains the fastest on a particular parallel corpus. Perhaps a parallel corpus could be halved to form development and test sets, but that would reveal the test set by omission and require trusting all participants. One participant was already caught cheating in a past edition of this shared task. Another option is that the test corpus could be a different surprise language pair, which would have the potentially positive effect that it also measures generalizability across languages. An interesting aspect of efficient training is that systems relying on backtranslation (Sennrich et al., 2016) incur substantial inference costs during their training cycle.

The one-month gap between the news task dead-

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

line and the efficient task deadline was too short and some teams noted this reduced the conditions they participated in. In addition, scaffolding would reduce the barrier to participation. This could take the form of providing a trained high-quality model, providing distilled (Kim and Rush, 2016) training data, or even optimized models where only the toolkit code is optimized. Providing this scaffolding would effectively require the organizers to perform the full task before releasing it to participants. If the training and test data are renewed each year as a countermeasure to overfitting and a participant that cheated, this would require more time between the news task and release of the news test set references.

German is a high resource language, which raises the computational cost of participation. A medium resource language would generally reduce training costs and explore whether results apply in this data condition.

The next task should aim to recruit more participants and perhaps separate the organization from one of the participants.

Acknowledgements

This work was conducted within the scope of the Horizon 2020 Research and Innovation Action *Bergamot*, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825303. GPU efficiency was supported by the European Union's Connecting Europe Facility under grant agreement No INEA/CEF/ICT/A2019/1927024, User-Focused Marian. This paper reflects the authors' views.

Intel Corporation has supported organization. The human evaluation was funded by Microsoft. We would like to thank Christian Federmann and Hitokazu Matsushita for their help with conducting human evaluation. Cloud credits were provided by the Oracle for Research program; we thank Rich Pitts for timely delivery.

References

Loïc Barrault, Magdalena Biesialska, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Matthias Huck, Eric Joanis, Tom Kocmi, Philipp Koehn, Chi-kiu Lo, Nikola Ljubešić, Christof Monz, Makoto Morishita, Masaaki Nagata, Toshiaki Nakazawa, Santanu Pal, Matt Post, and Marcos Zampieri. 2020. Findings of the 2020 conference on machine translation (WMT20). In *Proceedings of the Fifth Conference on Machine Translation*, pages 1–55, 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.

Mauro Cettolo, Marcello Federico, Luisa Bentivogli, Niehues Jan, Stüker Sebastian, Sudoh Katsuitho, Yoshino Koichiro, and Federmann Christian. 2017. Overview of the IWSLT 2017 evaluation campaign. In *International Workshop on Spoken Language Translation*, pages 2–14.

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.

Christian Federmann. 2018. Appraise evaluation framework for machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 86–88, Santa Fe, New Mexico. Association for Computational Linguistics.

Yvette Graham, Timothy Baldwin, Alistair Moffat, and Justin Zobel. 2013. Continuous measurement scales in human evaluation of machine translation. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 33–41, Sofia, Bulgaria. Association for Computational Linguistics.

Jiatao Gu and Xiang Kong. 2021. Fully non-autoregressive neural machine translation: Tricks of the trade. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 120–133, Online. Association for Computational Linguistics.

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.

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*.

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.

Jindřich Libovický and Jindřich Helcl. 2018. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021, Brussels, Belgium. Association for Computational Linguistics.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2018. RankME: Reliable human ratings for natural language generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 72–78, New Orleans, Louisiana. Association for Computational Linguistics.

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.

Keisuke Sakaguchi and Benjamin Van Durme. 2018. Efficient online scalar annotation with bounded support. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 208–218, Melbourne, Australia. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

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.