HW-TSC's Participation in the WMT 2021 Efficiency Shared Task

Hengchao Shang[1], **Ting Hu**[2], **Daimeng Wei**[1], **Zongyao Li**[1],
Jianfei Feng[2], **Zhengzhe Yu**[1], **Jiaxin Guo**[1], **Shaojun Li**[1],
Lizhi Lei[1], **Shimin Tao**[1], **Hao Yang**[1], **Jun Yao**[2], **Ying Qin**[1],
[1]Huawei Translation Service Center, Beijing, China
[2]Huawei Noah's Ark Lab, Hong Kong, China
{shanghengchao,huting35,weidaimeng,lizongyao,
fengjianfei1,yuzhengzhe,guojiaxin1,lishaojun15,
leilizhi,taoshimin,yanghao30,yaojun97,qinying}@huawei.com

Abstract

This paper presents the submission of Huawei Translation Services Center (HW-TSC) to WMT 2021 Efficiency Shared Task. We explore the sentence-level teacher-student distillation technique and train several small-size models that find a balance between efficiency and quality. Our models feature deep encoder, shallow decoder and light-weight RNN with SSRU layer. We use Huawei Noah's Bolt[1], an efficient and light-weight library for on-device inference. Leveraging INT8 quantization, self-defined General Matrix Multiplication (GEMM) operator, shortlist, greedy search and caching, we submit four small-size and efficient translation models with high translation quality for the one CPU core latency track.

1 Introduction

Transformer and its variants (Vaswani et al., 2017; Shaw et al., 2018; So et al., 2019; Dehghani et al., 2019) have become benchmark models in the domain of machine translation. A lot of innovations and engineering optimizations (Tay et al., 2020) in this area are based on Transformer. In general, to train a high-quality translation model, a large amount of data is required. Expensive training and deployment costs pose great challenges to scenarios where hardware are limited or the deployment environment is complex. This task aims to explore a solution that balances efficient decoding and high-quality translation. We focus on the one CPU latency track, which can better demonstrate the capability of our model and inference framework. We explore a balance between speed and quality, and ensure efficient memory usage and light-weight inference framework capability at the same time. We finally submit four models of different sizes.

We use knowledge distillation (Hinton et al., 2015) to train small models. The teacher models come from our WMT 2021 News Shared Task. For the sake of efficient decoding, our models have only 1-layer decoder. However, the number of encoding layers vary. Such settings lead to a great increase of inference efficiency while ensuring the translation quality (Wang et al., 2019).

All of our experiments are conducted based on fariseq (Ott et al., 2019), including the training of teacher and student models, as well as the generation of distillation data.

We use Huawei Noah's Bolt as the inference library. Bolt is a universal deep learning library featuring light weight and high speed. For the CPU task, we realize INT8 quantization inference and efficient GEMM operator, which is faster than Intel oneDNN[2]. With other engineering optimization strategies, we achieve a significant improvement in terms of inference efficiency.

Section 2 describes the teacher-student knowledge distillation process. Section 3 introduces how we optimize inference for this task. Section 4 presents the final result of our submissions.

2 Teacher to Student Knowledge Distillation

Sentence-level distillation (Kim and Rush, 2016; Freitag et al., 2017) have been demonstrated effective for machine translation tasks. First of all, we train a large teacher model that emphasizes translation quality. Then, we translate the source side of the training data and generate a synthetic parallel corpus, as synthetic data is easier for model fitting than real parallel data. Finally, we train student models using the synthetic data, hoping to minimize model sizes while ensuring equal translation quality as the teacher model. We use KD refer to knowledge distillation.

[1]https://github.com/huawei-noah/bolt

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

Proceedings of the Sixth Conference on Machine Translation (WMT), pages 781–786
November 10–11, 2021.

2.1 Teacher Model

As suggested in the task description, we select four iteration models for the third round and also the models before the final round of fine-tuning. All the models adopt back translation (Edunov et al., 2018) and forward translation (Wu et al., 2019) techniques. Our final ensembled model gained 39.7 BLEU on the WMT 2020 test set. The settings of the four models vary. We make sure that the model sizes are similar by adjusting hyperparameters such size of embedding, encoder layers, decoder layers, ffn size, etc. For more details about our teacher model, please refer to our system report for WMT 2021 News Shared Task.

2.2 Traning data

We comply with the constrained condition and use only data from the WMT 2021 En-De News Task. The size of parallel data after filtering is around 80M. In terms of monolingual data, we only use the news-crawl corpus with 230M sentences. So the size of English data we obtained is around 310M. In our teacher-student distillation experiment, we translate all English sentences from the parallel corpus and only 80M sentences sampled from the English monolingual data. Thus, the ratio of real parallel data to synthetic parallel data is 1:2.

When translating English sentences from the parallel corpus, we generate four candidates using the teacher model. Then we calculate the TER scores between those candidates and the corresponding German reference from the parallel corpus and select the candidate with the lowest TER score as the translation result. When translating monolingual data, the beam size is set to 4. For all translation results, we conduct data filtering with language identification using FastText (Joulin et al., 2017). We also delete sentences of which the source side has less than 5 tokens and those with repeated translated segments. The final sizes of our training data are as follow: 79M real parallel data, 73M synthetic data generated from the source side of parallel data, and 76M synthetic data generated from monolingual sentences. Table 1 summarizes the details of data we use.

Type	Corpora	Size
Parallel	Europarl v10	1.8M
	News Commentary v16	0.4M
	Tilde Rapid corpus	1.6M
	Wiki Titles v3	1.4M
	Common Crawl	2.4M
	ParaCrawl v7.1	82.6M
	WikiMatrix	6.2M
	Totle	96.5M
	Filtered	79.4M
Mono	news-crawl	230M
	For KD Translate	80M
KD	Parallel	79.4M
	Parallel En Translated	73.8M
	Mono En Translated	76.8M
	Total	230M

Table 1: Our training data details. The training data consists of three parts: filtered Parallel corpus, Parallel En translated then filtered and part of news-crawl En translated then filtered.

2.3 Vocabulary

We build a joint subword segmentation model from the synthesized parallel data using SentencePiece (Kudo and Richardson, 2018). The vocabulary size is set to 25,000 tokens. We employ SentencePiece regularization (Kudo, 2018) during data processing. We integrate SentencePiece into the training code and perform subword segmentation on the source side via sampling. Such strategy can improve model quality and robustness.

2.4 Student Model

The standard Transformer with self-attention in decoder has a drawback: decoding complexity increase as the decoding length increases. To address this issue, we refer to some light-weight RNNs, such as SRU (Lei et al., 2018) and SSRU (Kim et al., 2019). Based on our previous experiments and experience, we find that under the teacher-student distillation setting, SSRU models cam basically satisfies the translation quality requirements. As a result, all our student models replace the self-attention layer with SSRU layer on the decoder side. The encoder is still the standard Transformer architecture (Vaswani et al., 2017). We train three sizes of model during our experiment: base, small, and tiny, with different hidden sizes and filter sizes. They all have deep encoders/shallow decoders an architecture capable of increasing speed and maintain quality (Kasai et al., 2021; Wang et al., 2019). All models share the source and target word embeddings and softmax weights.

2.5 Traning

Our distillation experiments are based on fairseq. We also integrate SentencePiece into training. The sampling size is set to 64 and smoothing parameter to 0.1 for subword regularization. All our models are trained using 8 Nvidia Tesla V100 for two days with a batch size of 4096. Because the student models have relatively small capacities, regularization techniques such as dropout and label smoothing are not used. The other parameters use the default fairseq parameters. We save models every 1000 steps and average the last 10 checkpoints to produce the final models.

2.6 Evaluation

We use WMT 2019 and 2020 News Task test sets to measure our models with SacreBLEU (Post, 2018). We use the 12-1 base configuration model as our baseline model, which achieves 38.02 BLEU on 2020 test set, 1.7 BLEU lower than our teacher model. In general, more parameters means better translation quality. The BLEU score of the small.12 model is about 2-2.5 lower than that of the base.12 model, and the BLEU score of the tiny.2 model is also about 2-2.5 lower than that of the small.6 model. For details about parameter settings and BLEU results, see Table 2.

3 Inference Optimizations

For CPU optimization, we use Bolt v1.3.0. Bolt is a standalone open-source deep learning acceleration library. v1.3.0 will be available in September 2021.

3.1 Bolt technical overview

As a universal deployment tool for neural networks, Bolt aims to be faster and lighter. Key features of Bolt include extremely high performance, low-bit inference, widely compatible model converter and low memory usage. Bolt has a standalone C++ runtime, therefore Bolt can perform fast inference without any third-party dependencies. Bolt supports most of the NLP and CV models inference on x86 and ARM CPU as well as MALI GPU. We apply assembly-level optimizations to ensure computing performance and memory accessing efficiency. The operators of Bolt are capable of achieving high throughput near the peak of hardware.

3.2 8-bit Quantization

To accelerate translation tasks on Intel CPU and reduce the model size, Bolt uses linear symmetric quantization (Bhandare et al., 2019) to quantize the weights and part of the activations to 8-bit signed integers. Then Bolt converts the activations to 8-bit unsigned integers by adding 128 because of the limitation of Intel SIMD instructions. To ensure the correctness of matrix multiplication, Bolt applies extra integer offsets which can be obtained offline to the results.

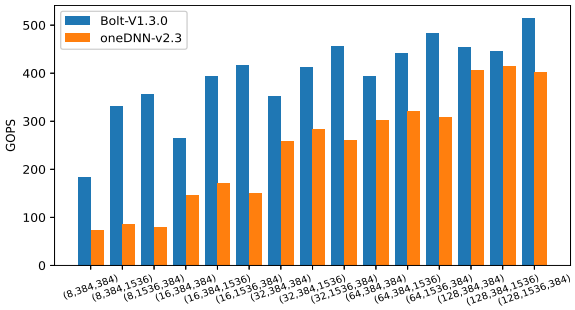

Figure 1: single thread u8s8s32 gemm performance of Bolt v1.3.0 and oneDNN-v2.3 tested on Intel Xeon Gold 6266C CPU, the reported sizes are frequently used in translation task.

The most time-consuming operation of translation tasks is GEMM, Bolt has implemented u8s8s32 gemm kernel, which is faster than Intel oneDNN (MKL-DNN). The u8s8s32 gemm performance of Bolt and oneDNN are shown in Figure 1. We present two key-points of the implementation:

Weights Offline Packing. Assuming the layout of GEMM weights is [N, K], Bolt chunks the weights in the K direction first, and then rearranges the data as NKNxK4 layout, where x is the chunk-size of N direction, x is in {8, 16, 32, 48}. The one that can divide N will be selected.

Highly Efficient Computation. Bolt quantizes bias to 32-bit signed integers, and then adds the offset value obtained offline to bias as the new bias, which is used to initialize the accumulation register of computation for saving addition operations. Bolt uses AVX512 VNNI instructions to perform u8s8s32 matrix multiplication. We have highly optimized the assembly to well utilize the register sources and ensure the instruction efficiency, and we also use memory optimization techniques such as cache-blocking, prefetching and memory alignment. All elements of the product of matrices are 32-bit signed integers. These intermediate data could be efficiently quantized to 8-bit unsigned integers or de-quantized to floating point numbers in registers for the next layer.

Model	Emb.	FFN	Head	Depth	Params(M)	Size(MB)	wmt19	wmt20
Teacher*4	1024	4096	16	25/6	514	2000	46.71	39.70
Base.12	512	2048	8	12/1	53	210	44.65	38.02
Small.12	384	1536	6	12/1	33	132	42.56	35.77
Small.9	384	1536	6	9/1	28	112	42.17	35.73
Small.6	384	1536	6	6/1	22	88	41.15	34.49
Tiny.2	256	1024	4	6/2	13	52	38.92	31.93
Tiny	256	1024	4	6/1	12	48	37.22	30.54

Table 2: Results of Distillation Training. The translation quality deteriorates as the model size decreases: the BLEU score of small model is 2-2.5 lower than that of the base model, and the BLEU score of the tiny model is also about 2-2.5 lower than that of the small model.

3.3 Greedy decoding and Caching

To maximize speed and reduce memory usage, we use greedy search instead of beam search. During decoding, we also skip the final softmax layer and simply get the maximum from the output logits.

Due to the autoregressive model, we also cache the linear transformations for keys and values before the self-attention and cross attention layers.

3.4 Shortlist and Online Quantification

When decoding, we also use the mapping relationship between source and target tokens, a.k.a shortlist, which finds the best matched target token via the source input. Such strategy decreases the dimensions of softmax_weight, which can significantly improve the decoding efficiency while ensuring that the quality is only slightly influenced.

We use fastalign[3] (Dyer et al., 2013) to construct the mapping relationship. During inference, a small target token set is obtained via querying the mapping dynamically, which conflicts with our offline quantization matrix technique. As a result, we try two schemes: a) abandon shortlist and use offline quantification instead; b) keep shortlist and quantify the reduced matrix online. In our experiments, we find that the efficiency of the two versions depends on the size of the target tokens. Because the larger the matrix, the greater the cost of online quantization, and the multiplicative benefits of dimension reduction are offset. For example, on a model we tested, when the input is fixed to 47 tokens, the time cost of a) is 46 ms; the time cost of b) is 68 ms when the size is set to 25000; and the time cost is 48 ms when the size is set to 2000.

Since we focus on the one CPU core latency track, the model processes only one input at a time, and the maximum size of target tokens is less than 2000, so we choose b).

3.5 Submitted Docker images

We choose the base image of ubuntu:18.04. Following the task requirements, our startup script is /run.sh. We use C++ to encapsulate our calls to Bolt and models, SentencePiece, as well as our simple pre- and post-processing. Our model is stored in the /model directory, which contains the converted Bolt model, vocabulary, and shortlist files. The compressed file is provided. Due to the simple runtime environment of Bolt, the final SO package is about 2 MB without any third-party dependency. After quantization, the maximum size of our model is less than 60 MB and the minimum size is about 13 MB. Therefore, the final submitted image is about 100 MB.

4 Optimization results

The latency track we participated in is defined as providing one sentence on standard input and flushing then waiting for your system to provide a translation on its standard output (and flush) before providing the next sentence. So we don't use techniques such as batch. We believe such strategy can better demonstrate the capability our model and inference framework.

After the preceding optimizations, the inference speed is significantly improved. Table 3 lists the results. In general, INT8 inference greatly improves performance. Especially, when the model is relatively large, INT8 improves performance by about three times when comparing with FP32. As the model size becomes smaller, the speed improvement becomes less obvious. But our smallest model also has at least a 2x or above speed improvement. Comparing with that of FP32, the average transla-

[3]http://github.com/clab/fast align

Model	Precious	Size	WPS	BLEU
Base.12	FP32	212	237	38.26
	INT8	53	815	38.02
Small.12	FP32	133	411	36.15
	INT8	33	1158	35.77
Small.9	FP32	112	473	35.90
	INT8	28	1295	35.73
Small.6	FP32	88	550	34.53
	INT8	22	1467	34.49
Tiny.2	FP32	52	759	32.06
	INT8	13	1515	31.93
Tiny	FP32	48	1000	31.01
	INT8	12	2096	30.54

Table 3: Optimization results. The test set is WMT 2020 News Task. The unit of size is MB. WPS refers to the source side. The test environment is Intel(R) Xeon(R) Gold 6278C CPU @ 2.60GH. We submit four models: Base.12, Small.9, Small.6 and Tiny.

tion quality of INT8 models decreases less than 0.1 BLEU.

In our small model setting, the translation quality of the 12-1 model is not significantly improved compared with the 9-1 model, but the inference speed decreases by about 25%. Perhaps under the small model setting, the addition of three encoding layers does not bring significant changes to the model quality. Compared with the Tiny.2 model, the size of our Small.6 model doubles, resulting in an increase of 2.5 BLEU. However, the inference speed are almost the same. In addition, the speed of our Tiny model is 30% faster than our Tiny.2 model by dropping a decoder layer. Our result demonstrates that the number of decoding layers has greater impacts on decoding efficiency. As a result, we submit four models: Base.12, Small.9, Small.6 and Tiny.

The above tests on inference speed are performed with Intel(R) Xeon(R) Gold 6278C CPU @ 2.60GHz.

5 Conclusion

In order to produce a translation system with high inference efficiency, we explore sentence-level distillation techniques and train student models with a trade-off between speed and quality by leveraging Deep-Encoder and Shallow-Decoder models. In terms of inference, we use Huawei Noah's Bolt library. Using a series of optimization techniques, such as INT8 inference and custom efficient GEMM operators, we accelerate inference speed by 2 to 3 times. By using shortlist, greedy search and caching, we submit four models with different settings to the efficiency one CPU core latency task, realizing efficiency improvement under different circumstances.

References

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. *arXiv preprint arXiv:1906.00532*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2019. Universal transformers.

Chris Dyer, Victor Chahuneau, and Noah A Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium. Association for Computational Linguistics.

Markus Freitag, Yaser Al-Onaizan, and Baskaran Sankaran. 2017. Ensemble distillation for neural machine translation.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2021. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation.

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*.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 66–75.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 66–71.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

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.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. 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 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Lijun Wu, Yiren Wang, Yingce Xia, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2019. Exploiting monolingual data at scale for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4207–4216, Hong Kong, China. Association for Computational Linguistics.