MobileNMT: Enabling Translation in 15MB and 30ms

Ye Lin[1]*, Xiaohui Wang[2], Zhexi Zhang[2], Mingxuan Wang[2], Tong Xiao[1,3]†, Jingbo Zhu[1,3]
[1]NLP Lab, School of Computer Science and Engineering,
Northeastern University, Shenyang, China
[2] ByteDance
[3]NiuTrans Research, Shenyang, China
{linye2015}@outlook.com
{wangxiaohui.neo,zhangzhexi,wangmingxuan.89}@bytedance.com
{xiaotong,zhujingbo}@mail.neu.edu.cn

Abstract

Deploying NMT models on mobile devices is essential for privacy, low latency, and offline scenarios. For high model capacity, NMT models are rather large. Running these models on devices is challenging with limited storage, memory, computation, and power consumption. Existing work either only focuses on a single metric such as FLOPs or general engine which is not good at auto-regressive decoding. In this paper, we present MobileNMT, a system that can translate in 15MB and 30ms on devices. We propose a series of principles for model compression when combined with quantization. Further, we implement an engine that is friendly to INT8 and decoding. With the co-design of model and engine, compared with the existing system, we speed up 47.0× and save 99.5% of memory with only 11.6% loss of BLEU. The code is publicly available at https://github.com/zjersey/Lightseq-ARM.

1 Introduction

As a classic subfield of natural language processing, neural machine translation (NMT) has achieved great success in recent years. Most of the studies focus on improving the accuracy of large machine translation systems, ignoring whether such models are easy to be deployed in real-world scenarios.

Here we adopt four metrics to evaluate whether an NMT model is deployment-friendly. (1) **Model size** is the most important metric in model compression (Han et al., 2016). (2) **Floating-point operations (FLOPs)** is commonly used to evaluate computational complexity in neural architecture design. (3) **Memory** or **Memory mapped I/O (MMI/O)** reflects the memory requirements of the real running system. (4) **Decoding speed** depends on many realistic factors such as engine implementation and the power of avaliable processors.

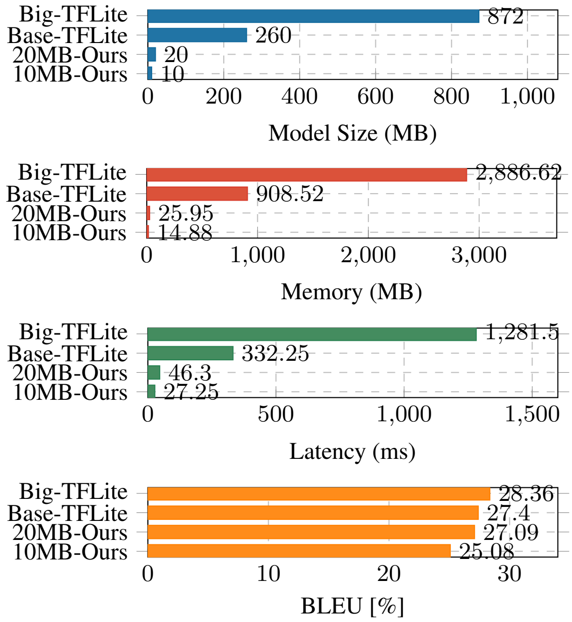


Figure 1: These metrics are measured on Google Pixel 4. Each result is the average of 200 runs on a sample of src/tgt length 30.

In this paper, we propose MobileNMT, a Transformer-based machine translation system that can translate in 15MB and 30ms. First, we propose three principles for designing parameter-limited MT models: 1) To compress embedding, reducing vocabulary size is simple and effective compared to embedding factorization; 2) To compress the encoder and decoder, reducing the model width is much more efficient in computation and memory than cross-layer parameter sharing; 3) Encoder depth is very important to ensure accuracy. To achieve higher accuracy, we adjust the training hyperparameters according to the newly designed structure, and adopt sequence-level knowledge distillation. For industrial deployment, we optimize general matrix multiplication (GEMM) and memory in our own inference engine and use the 8-bit integer for storage and computation. As shown in Table 1, the 10MB MobileNMT achieves 88.4%

*This work is done during the internship at ByteDance.
†Corresponding author.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 5: Industry Track, pages 368–378
July 10-12, 2023 ©2023 Association for Computational Linguistics

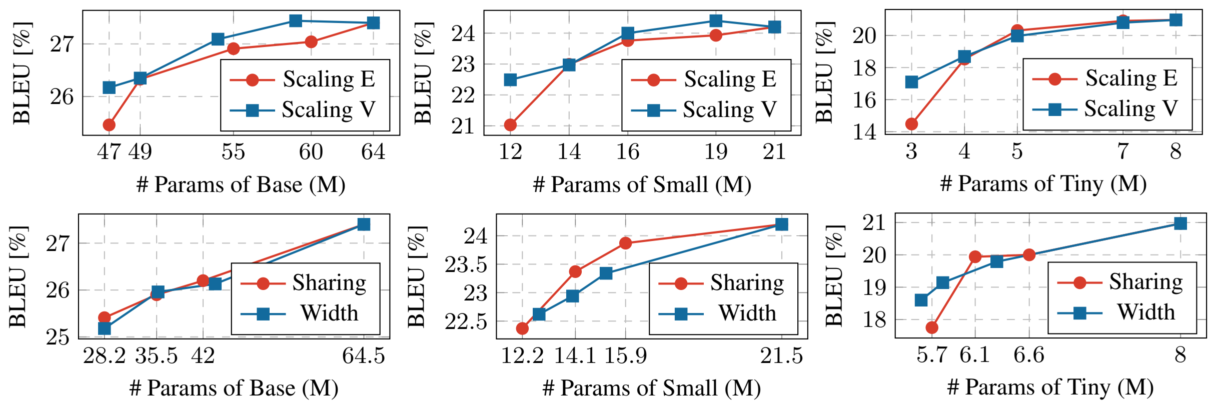


Figure 2: Model performance of different methods in Section 2 and Section 3 (Scaling E: scaling embedding dimension; Scaling V: scaling vocabulary size; Sharing: cross-layer parameter sharing; Width: reducing model width). Scaling V performs better than Scaling E. Width performs nearly the same with Sharing.

performance of Transformer-big with only 1.1% size and runs 47.0× faster on decoding, which can be easily deployed and used.

Our contributions are summarized as follows:

- We propose three principles for parameter-limited MT models to make more efficient use of computation and memory resources.
- We adjust training strategies according to the newly designed structure to achieve higher translation accuracy.
- We develop a mobile inference engine to bridge the gap between industrial practice and theoretical research.

2 Architecture Design Principles

For model compression and acceleration, most studies focus on a single metric such as model size or FLOPs, without considering the real-world applications. In this section, we consider four metrics including model size, FLOPs, memory usage, and decoding speed, and then propose three design principles for parameter-limited MT models. We choose Transformer (Appendix A) as our baseline because of its great success in machine translation.

2.1 Embedding Compression

The vocabulary size V usually reaches tens of thousands in NMT models (Akhbardeh et al., 2021). The parameters can reach tens of millions and greatly affect the overall parameter efficiency.

Embedding Factorization (Scaling E). For model compression, embedding factorization has been widely studied (Lan et al., 2020; Grave et al., 2017; Baevski and Auli, 2019). To decouple the embedding dimension E and hidden dimension H, it additionally introduces a trainable transformation weight $W^T \in \mathbb{R}^{E\times H}$, where $E \leq H$. After factorization, the embedding parameters will be decreased from $O(V \times H)$ to $O(V \times E + E \times H)$.

Module	Dim	Base		Small		Tiny	
Embed	Vocab	40,000	×1	40,000	×1	40,000	×1
	Embed	N/A		N/A		N/A	
	Hidden	512		256		128	
Encoder	Hidden	512	×6	256	×6	128	×6
	Head	8		4		2	
	FFN	2048		1024		512	
Decoder	Hidden	512	×6	256	×6	128	×6
	Head	8		4		2	
	FFN	2048		1024		512	
Params		64.5M		21.5M		8.0M	

Table 1: The detailed settings of Base, Small and Tiny.

Reducing Vocabulary Size (Scaling V). A more direct way to compress embedding is to reduce the vocabulary size V. To reduce the risk of out-of-vocabulary words, here we adopt Byte-Pair Encoding (BPE) (Sennrich et al., 2016; Ott et al., 2018; Ding et al., 2019; Liu et al., 2020). For most studies on machine translation, the adopted BPE merge operations range from 30∼40K (Ding et al., 2019). Volt proves that we can find a well-performing vocabulary with higher BLEU and smaller BPE merge operations (Xu et al., 2021). Experiments in Lin et al. (2021)'work also show that smaller vocabularies may be better.

Reducing Vocabulary Size Performs Better. To compare the two embedding compression methods, here we select three baseline models of different sizes. The model settings are shown in Table 1. As shown in Table 2, the parameters and FLOPs are almost the same in these two methods. As shown

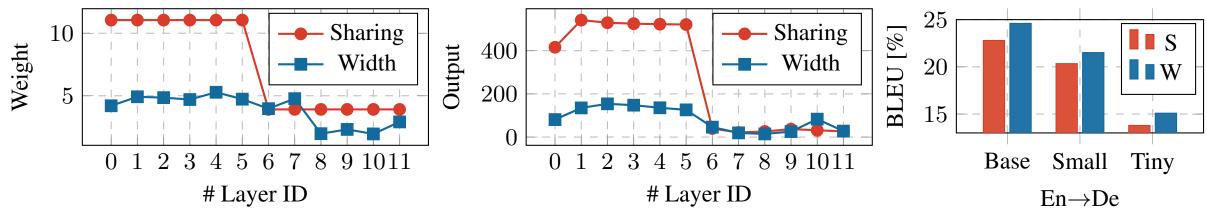


Figure 3: The left two figures show weight and output ranges for each layer. The right figure shows the model performance of Post Training Quantization (PTQ) in cross-layer parameter sharing vs. reducing model width. These figures show that reducing model width is more quantization-friendly than cross-layer parameter sharing.

Metric	Scaling E			vs.	Scaling V		
	Base	Small	Tiny		Base	Small	Tiny
Params (M)	47	12	3		47	12	3
FLOPs (G)	1.41	0.38	0.11		1.41	0.38	0.11
MMI/O (M)	48	15	6		47	14	5
BLEU	25.46	21.03	14.48		26.17	22.49	17.10
Metric	Sharing			vs.	Width		
	Base	Small	Tiny		Base	Small	Tiny
Params (M)	28	12	6		28	12	6
FLOPs (G)	1.95	0.65	0.24		0.85	0.38	0.17
MMI/O (M)	66	24	10		30	15	7
BLEU	25.41	22.37	17.75		25.18	22.62	18.60

Table 2: Parameters, FLOPs, and model performance (FLOPs and MMI/O are estimated on a sample with src/tgt length of 30.). For embedding compression, reducing vocabulary size (Scaling V) is more simple and effective. For encoder/decoder compression, reducing model width (Width) is more efficient in computation and memory.

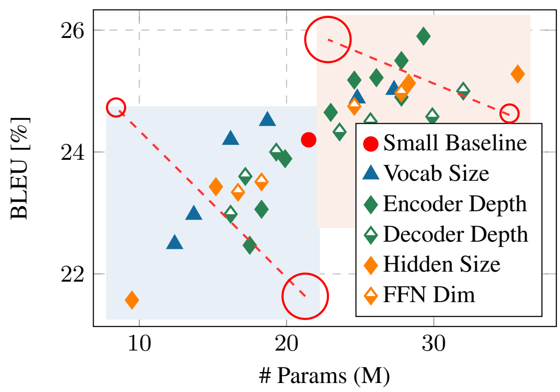


Figure 4: Performance (BLEU) vs. parameters (M). Different marks denote different dimensions. Points near large red circles have a greater impact on model performance than points near small red circles. Encoder depth can be considered as the most important dimension.

in the first row of Fig. 2, compared to reducing vocabulary size, the model with embedding factorization performs poorly in most cases, especially when the parameters are limited.

2.2 Encoder/Decoder Compression

For encoder and decoder compression, here we compare models with cross-layer parameter sharing and model width reduction.

Cross-Layer Parameter Sharing (Sharing). The most widespread use of parameter sharing is in convolutional neural networks (Long et al., 2015). In recent years, it has also been investigated on NLP and NLU tasks. Among them, cross-layer parameter sharing can provide stronger nonlinearity along the model depth while keeping the parameters unchanged (Dehghani et al., 2019; Takase and Kiyono, 2021; Lan et al., 2020).

Reducing Model Width (Width). Since model depth has been proven to be important in natural language processing tasks such as machine translation (Devlin et al., 2019; Liu et al., 2020; Wang et al., 2022; Liu et al., 2020), here we keep the depth unchanged and reduce the model width.

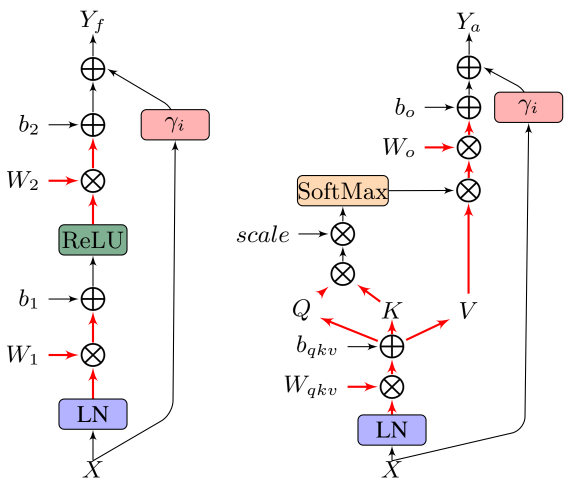
Reducing Model Width is More Efficient and Quantization-Friendly. In the second row of Fig. 2, these two methods perform nearly the same. However, Table 2 shows that there is a large difference in FLOPs and MMI/O, which means reducing model width is much more efficient in computation and memory. Since it is necessary to quantize these models for greater compression, we further compare the weights and output ranges of the two methods in Fig. 3. It can obviously be observed that models with parameter sharing have larger ranges of values for both weight and output, which is not quantization-friendly. The right figure also verifies this: when we apply post-training quantization (PTQ) (Sung et al., 2015; Banner et al., 2019; Choukroun et al., 2019) to these two methods, cross-layer parameter sharing performs poorly.

2.3 Deep Encoder and Shallow Decoder

Fig. 4 studies how different dimensions affect the Transformer performance. In order to analyze the impact of each dimension separately, here we only change one specific dimension and keep the others

Module	Dim	MobileNMT-10MB		MobileNMT-20MB	
Embed	Vocab	8,000		8,000	
	Embed	N/A	×1	N/A	×1
	Hidden	256		384	
Encoder	Hidden	256		384	
	Head	4	×12	6	×12
	FFN	512		768	
Decoder	Hidden	256		384	
	Head	4	×2	6	×2
	FFN	512		768	
Params		≈10M		≈20M	

Table 3: The detailed settings of MobileNMT.



(a) FFN Quantizer (b) Attention Quantizer

Figure 5: Running examples of the FFN and attention quantizers. Here red lines denote values that will be quantized, black lines denote values with full precision.

unchanged. The point on the left of the Small Baseline ● represents scaling one dimension down, while the point on the right represents scaling one dimension up. We can see that Encoder Depth ◆ is more important than other dimensions, which is consistent with the related work on large-scale models (Wang et al., 2019, 2022). Based on the above discussion, we finally build a deep encoder and a shallow decoder, while reducing the vocab size and model width. Two MobileNMT models of different sizes are built here and the detailed settings are shown in Table 3.

3 Training Strategies

3.1 Pre-Training with Knowledge Distillation

In order to improve the performance of compressed models, recent studies distill knowledge from a well-trained full-precision teacher network to a student network (Mishra and Marr, 2018) or directly use a quantized teacher network (Kim et al., 2019). Here we adopt sequence-level knowledge distillation because it has shown to be effective for NMT tasks. The most basic full-precision Transformer-base model is adopted as the teacher.

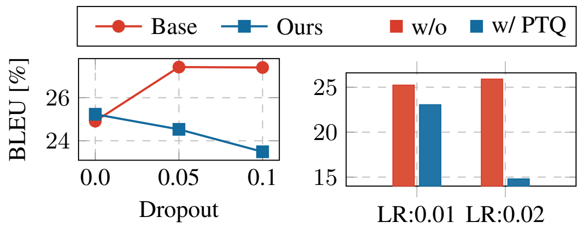


Figure 6: The left part shows performance of different dropouts on base model vs. MobileNMT. The right part shows performance before vs. after PTQ. Removing dropout from MobileNMT can lead to significant performance improvement. While larger learning rates can also improve model performance, the model will become quantization-unfriendly.

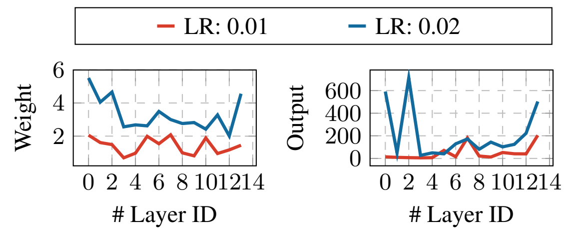


Figure 7: Weight and output ranges for each layer. Larger learning rate will result in larger range of values.

3.2 Quantization

The process of quantizing a transformer model can be divided into two steps: 1) constructing quantizers; 2) applying the quantization-aware training (QAT) (Courbariaux et al., 2015) based on the pre-trained model we have obtained in Section 3.1.

FFN and Attention Quantizers. The original Transformer layer includes two types of sublayers: the attention sublayer and feed-forward network (FFN) (Vaswani et al., 2017). Here we construct the quantizer for each linear in the attention and FFN, and quantize both the weights and activations as shown in Fig. 5. Since most computations are spent on matrix multiplication, all biases and residuals are kept in full precision for accuracy preservation. Since quantization will change the range of network outputs, here we add a learnable weight γ_i to the i-th sublayer to learn how to combine the output and the residual surrounding it.

Quantization-Aware Training. Since MobileNMT only has 10M/20M parameters, quantizing such a small model inevitably results in performance loss, so we perform QAT after constructing

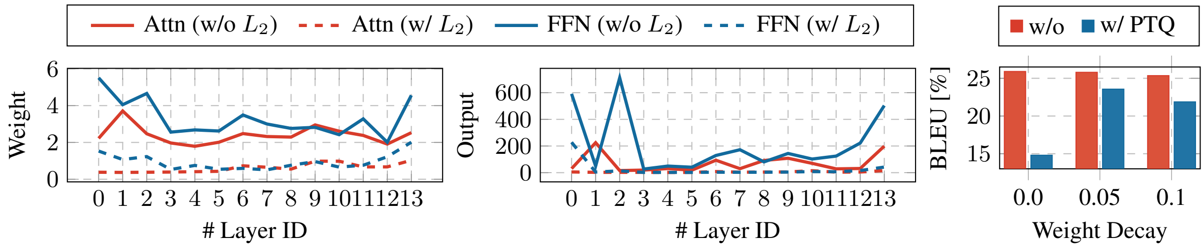


Figure 8: The left two figures show weight and output ranges for each layer. The right figure shows the performance of different L_2 regularizations before vs. after PTQ. Experiments show that L_2 regularization can make the model more quantization-friendly.

the quantizers. Before QAT, we pre-compute all scaling parameters based on a forward running on the pre-trained distillation model obtained in Section 3.1. It takes nearly no additional costs, but provides a good initialization. For engineering development, we choose the uniform quantization scheme because of it is hardware-friendly (Liu et al., 2022). For 8-bit quantization, we use the element-wise quantization (Lee et al., 2021). For lower-bit quantization, such as 4-bit integer, we use the row-wise quantization (Faraone et al., 2018).

3.3 Training Hyperparameters

Compared to the original Transformer model, MobileNMT introduced in Section 2 has fewer parameters and different architectures, so different training hyperparameters are needed.

Removing Dropout. Since our models have fewer parameters, we do not need to impose strong regularizations on them and we remove dropout from the entire model. The left part of Fig. 6 shows that removing dropout will lead to an improvement of almost two BLEU points.

Larger Learning Rate. Here we follow the configuration provided in Wang et al. (2019) with a larger learning rate ($0.01 \rightarrow 0.02$), a larger training batch ($4096 \rightarrow 8192$), and more warmup steps ($4000 \rightarrow 8000$). As shown in the right part of Fig. 6, it can improve model performance by more than 0.5 BLEU points (red bars). However, after PTQ, the model with 0.02 learning rate performs significantly worse than 0.01 (blue bars). As shown in Fig. 7, the network weights and outputs become larger when using a larger learning rate, which is not quantization-friendly.

L_2 **Regularization.** To solve the above problem, this paper adopts L_2 regularization applied to weight (also called weight decay). It adds the squared magnitude of the network weights as the penalty term to the original loss function and encourage the weights to be smaller. As shown in the left two parts of Fig. 8, with L_2 regularization, both the network weights and output values will become significantly smaller. The right part of Fig. 8 shows the performance of PTQ when applying different degrees of L_2 regularization. The red and blue bars represent the model performance before and after PTQ. We can see that L_2 regularization does improve the model performance after PTQ.

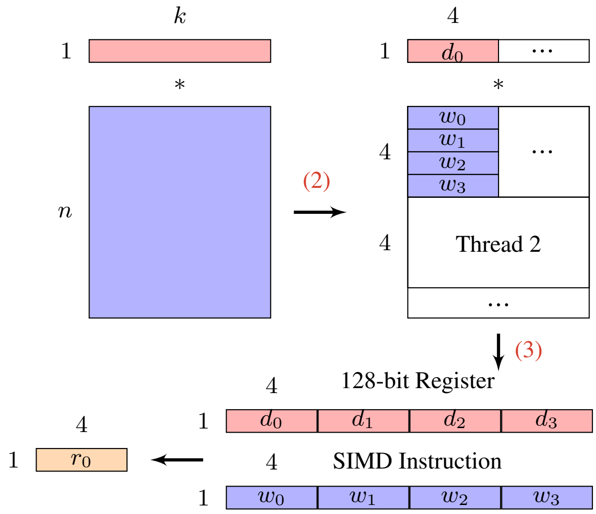


Figure 9: An example of processing multiple integers in a single SIMD instruction.

4 The Engine

This section introduces the detailed implementations of our inference engine.

4.1 GEMM Optimization

According to statistics on the ONNX Runtime platform, general matrix multiplication (GEMM) accounts for 80.44% of the overall decoding time, demonstrating that optimizing GEMM is the key to decoding speed up. We optimize GEMM from three aspects: (1) Replacing 32-bit floating points

	System	Params (M)	Size (MB)	Memory (MB)	Latency (ms)	Test	Valid
En-De	Transformer-big	218 $\uparrow 1\times$	872 $\uparrow 1\times$	2886.6 $\uparrow 1.0\times$	1281.5 $\uparrow 1.0\times$	28.36 Δ-0.00	26.75 Δ-0.00
	Transformer-base	65 $\uparrow 3\times$	260 $\uparrow 3\times$	908.5 $\uparrow 3.2\times$	332.3 $\uparrow 3.9\times$	27.40 Δ-0.96	25.81 Δ-0.94
	Transformer-small	22 $\uparrow 10\times$	88 $\uparrow 10\times$	759.5 $\uparrow 3.8\times$	158.0 $\uparrow 8.1\times$	24.20 Δ-4.61	23.91 Δ-2.84
	Transformer-tiny	8 $\uparrow 27\times$	32 $\uparrow 27\times$	398.9 $\uparrow 7.2\times$	73.0 $\uparrow 17.6\times$	20.97 Δ-7.39	21.53 Δ-5.22
	MobileNMT-20MB	20 $\uparrow 11\times$	20 $\uparrow 44\times$	26.0 $\uparrow 111.2\times$	46.3 $\uparrow 27.7\times$	27.09 Δ-1.27	25.72 Δ-1.03
	MobileNMT-10MB	10 $\uparrow 22\times$	10 $\uparrow 87\times$	14.9 $\uparrow 194.0\times$	27.3 $\uparrow 47.0\times$	25.08 Δ-3.28	24.85 Δ-1.90
En-Fr	Transformer-big	259 $\uparrow 1\times$	1036 $\uparrow 1\times$	2987.6 $\uparrow 1.0\times$	1345.6 $\uparrow 1.0\times$	39.05 Δ-0.00	44.12 Δ-0.00
	Transformer-base	86 $\uparrow 3\times$	344 $\uparrow 3\times$	944.8 $\uparrow 3.2\times$	358.9 $\uparrow 3.7\times$	38.64 Δ-0.41	43.80 Δ-0.32
	Transformer-small	22 $\uparrow 12\times$	88 $\uparrow 12\times$	782.3 $\uparrow 3.8\times$	178.5 $\uparrow 7.5\times$	34.76 Δ-4.29	40.01 Δ-4.11
	Transformer-tiny	8 $\uparrow 32\times$	32 $\uparrow 32\times$	418.8 $\uparrow 7.1\times$	80.3 $\uparrow 16.8\times$	30.36 Δ-8.69	36.01 Δ-8.11
	MobileNMT-20MB	20 $\uparrow 13\times$	20 $\uparrow 52\times$	26.7 $\uparrow 111.9\times$	53.7 $\uparrow 25.1\times$	37.67 Δ-1.38	43.81 Δ-0.31
	MobileNMT-10MB	10 $\uparrow 26\times$	10 $\uparrow 104\times$	15.8 $\uparrow 189.1\times$	28.9 $\uparrow 46.6\times$	36.00 Δ-3.05	41.87 Δ-2.25

Table 4: Results on WMT14 En-De and WMT14 En-Fr tasks. These metrics are measured on Google Pixel 4. Transformer-big/base/small/tiny results are tested on TFLite and MobileNMT-20MB/10MB are tested on our engine. All results are based on a sample with src/tgt length of 30.

with 8-bit integers in GEMM for model quantization. (2) The Arm instruction set we use allows multiple integers to be processed in parallel in a single instruction, which takes full advantage of the processor throughput. (3) To improve the cache hit and the register usage, we adjust the layout of the tensor in memory to ensure that the instruction reads data from continuous space. Specifically, we convert each 4×4 block in the original layout into a contiguous vector of size 16. An example can be seen in Fig. 9.

4.2 Memory Optimization

As shown in Fig. 10 in the appendix C, except for GEMM, other operations account for only 19.56% of the decoding time but will be frequently performed, resulting in a large amount of temporary memory. To improve memory efficiency, we take two strategies: (1) To avoid frequent memory-mapped I/O and footprint, our engine integrates all adjacent fine-grained operations between two GEMM operations into one fused operation. (2) To save temporary memory, different operations are allowed to share the same space, provided that these operations do not interfere with each other at the same time. Through memory sharing, only two 8-bit memory buffers, and one 32-bit buffer need to be pre-allocated in the Transformer encoder to hold intermediate results.

System	Params(M)		FLOPs(G)	BLEU
	w/	w/o		
Transformer-base	65	44	1.9	27.40
DeLighT	37	31.4	-	27.60
Universal Transformer	N/A	7.4	1.9	26.20
Lite Transformer (small)	N/A	2.9	0.2	22.50
Lite Transformer (medium)	N/A	11.7	0.7	25.60
Lite Transformer (big)	N/A	17.3	1.0	26.50
EdgeFormer w/o LA	N/A	8.6	1.8	26.50
EdgeFormer (Adapter-LA)	N/A	9.4	1.8	26.90
EdgeFormer (Prefix-LA)	N/A	8.6	1.9	26.80
MobileNMT-10MB	10	7.9	0.3	25.08
MobileNMT-20MB	20	17.7	0.6	27.09

Table 5: The comparison of MobileNMT with other parameter-efficient Transformers, including DeLighT (Mehta et al., 2021), Universal Transformer (Dehghani et al., 2019), Lite Transformer (Wu et al., 2020) and EdgeFormer (Ge et al., 2022) (Parameters w/ or w/o embedding layer are both provided. FLOPs is estimated on a sample with src/tgt length of 30.).

5 Experiments

5.1 Setups

We evaluate our methods on two WMT benchmarks. For the WMT14 En-De task (4.5M pairs), we choose *newstest-2013* as the validation set and *newstest-2014* as the test set. For the WMT14 En-Fr task (35M pairs), we validate the system on the combination of newstest-2012 and newstest-2013, and test it on newstest-2014. Details of the architecture were introduced in Section 2, and training hyperparameters were introduced in Section 3. For model compression ratio and decoding speed up, we choose Transformer-big as the benchmark ($1.0\times$). Other details of experimental setups are introduced in Appendix D.

5.2 Results

Table 4 shows the results of different systems on WMT14 En-De and En-Fr. Table 5 shows the comparison of MobileNMT with other parameter-efficient methods based on Transformer. MobileNMT-10MB and MobileNMT-20MB are two models we have built with different sizes, which are introduced in Table 3.

On WMT14 En-De, our MobileNMT-10MB requires only 4.6% of the parameters to maintain 88.4% performance of Transformer-big, while it achieves 87.2× compression ratio and 47.0× speed up. Our MobileNMT-20MB can maintain 95.5% performance of Transformer-big with only 9.2% parameters, while it achieves 43.6× compression ratio and 27.7× speed up. Experiments on En-Fr show similar results. In addition, thanks to the memory optimization strategies adopted in our engine, MobileNMT requires significantly less running memory than other models (0.5%∼0.9% of Transformer-big). All these experiments demonstrate that MobileNMT is efficient in terms of parameters, computation, and memory, and can be easily deployed on mobile devices.

6 Conclusion

We propose MobileNMT, a Transformer-based machine translation system that can translate in 15MB and 30ms. It uses existing resources efficiently and can be easily deployed in real-world scenarios. We develop a mobile inference engine with GEMM and memory optimization, hoping that it can bridge the gap between theoretical research and real-world applications on efficient machine translation.

Acknowledgments

This work was supported in part by the National Science Foundation of China (No. 62276056), the National Key R&D Program of China, the China HTRD Center Project (No. 2020AAA0107904), the Natural Science Foundation of Liaoning Province of China (2022-KF-16-01), the Yunnan Provincial Major Science and Technology Special Plan Projects (No. 202103AA080015), the Fundamental Research Funds for the Central Universities (Nos. N2216016, N2216001, and N2216002), and the Program of Introducing Talents of Discipline to Universities, Plan 111 (No. B16009).

Limitations

Multilingual Translation. Here we mainly discuss the design principles of efficient architectures for bilingual machine translation. Compared with bilingual translation, multilingual translation tasks require significantly more parameters and computations to perform well, and different model scales may lead to different design considerations. We will leave this for future exploration.

Knowledge Distillation. As a small model that requires only 10MB/20MB of storage, MobileNMT will inevitably suffer from performance loss compared to other Transformer-based models. To reduce performance loss, here we adopt knowledge distillation and choose the Transformer-base model as the teacher. From a training efficiency perspective, although the teacher model can help MobileNMT improve performance, it also introduces additional training costs.

Compatibility. Here our inference engine only provides implementation for the ARM CPU. We will make it available for other AI accelerator (such as NPU) on mobile devices in the future.

References

Farhad Akhbardeh, Arkady Arkhangorodsky, Magdalena Biesialska, Ondrej Bojar, Rajen Chatterjee, Vishrav Chaudhary, Marta R. Costa-jussà, 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, WMT@EMNLP 2021, Online Event, November 10-11, 2021*, pages 1–88. Association for Computational Linguistics.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7948–7956.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. 2019. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF*

International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*, pages 3009–3018. IEEE.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3123–3131.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Shuoyang Ding, Adithya Renduchintala, and Kevin Duh. 2019. A call for prudent choice of subword merge operations in neural machine translation. In *Proceedings of Machine Translation Summit XVII Volume 1: Research Track, MTSummit 2019, Dublin, Ireland, August 19-23, 2019*, pages 204–213. European Association for Machine Translation.

Julian Faraone, Nicholas J. Fraser, Michaela Blott, and Philip H. W. Leong. 2018. SYQ: learning symmetric quantization for efficient deep neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4300–4309. Computer Vision Foundation / IEEE Computer Society.

Tao Ge, Si-Qing Chen, and Furu Wei. 2022. Edgeformer: A parameter-efficient transformer for on-device seq2seq generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 10786–10798. Association for Computational Linguistics.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1302–1310. PMLR.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4107–4115.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2704–2713.

Jangho Kim, Yash Bhalgat, Jinwon Lee, Chirag Patel, and Nojun Kwak. 2019. QKD: quantization-aware knowledge distillation. *CoRR*, abs/1911.12491.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Junghyup Lee, Dohyung Kim, and Bumsub Ham. 2021. Network quantization with element-wise gradient scaling. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 6448–6457. Computer Vision Foundation / IEEE.

Ye Lin, Yanyang Li, Tong Xiao, and Jingbo Zhu. 2021. Bag of tricks for optimizing transformer efficiency. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 4227–4233. Association for Computational Linguistics.

Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. 2020. Very deep transformers for neural machine translation. *CoRR*, abs/2008.07772.

Zechun Liu, Kwang-Ting Cheng, Dong Huang, Eric P. Xing, and Zhiqiang Shen. 2022. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 4932–4942. IEEE.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3431–3440. IEEE Computer Society.

Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Delight: Deep and light-weight transformer. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Asit K. Mishra and Debbie Marr. 2018. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. *CoRR*, abs/1910.05895.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 1–9. 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, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 186–191. Association for Computational Linguistics.

Jerry Quinn and Miguel Ballesteros. 2018. Pieces of eight: 8-bit neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 3 (Industry Papers)*, pages 114–120.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. 2015. Resiliency of deep neural networks under quantization. *CoRR*, abs/1511.06488.

Sho Takase and Shun Kiyono. 2021. Lessons on parameter sharing across layers in transformers. *CoRR*, abs/2104.06022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. 2022. Deepnet: Scaling transformers to 1, 000 layers. *CoRR*, abs/2203.00555.

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 Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 1810–1822. Association for Computational Linguistics.

Ephrem Wu. 2020. Learning accurate integer transformer machine-translation models. *CoRR*, abs/2001.00926.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long-short range attention. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. 2020. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR.

Jingjing Xu, Hao Zhou, Chun Gan, Zaixiang Zheng, and Lei Li. 2021. Vocabulary learning via optimal transport for 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, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 7361–7373. Association for Computational Linguistics.

A Transformer Architecture

We chose Transformer for study because it is one of the most successful neural models for machine translation. It consists of a N-layer encoder and a M-layer decoder, where N=M=6 in the original Transformer-base and Transformer-big. Each encoder layer consists of two sublayers, including the self-attention and feed-forward network (FFN). Each decoder layer has an additional cross-attention sublayer to bridge the encoder and decoder.

The self-attention takes the output X of the previous sublayer as its input. The cross-attention is similar to the self-attention, except that it takes the encoder output as an additional input. Both types of attention first compute the attention distribution A_x and then average X by A_x. We denote the transformation matrices of Q, K, V as W_q, W_k, W_v, the subsequent transformation matrices as W_o, and the attention as $Y_a = \mathrm{Attn}(X)$, then:

$$A_x = \mathrm{SoftMax}(\frac{XW_qW_k^TX^T}{\sqrt{d}}) \quad (1)$$

$$Y_a = A_xXW_vW_o \quad (2)$$

The FFN applies non-linear transformation to its input X. We denote the FFN as $Y_f = \mathrm{FFN}(X)$:

$$Y_f = \mathrm{ReLU}(XW_1 + b_1)W_2 + b_2 \quad (3)$$

where W_1 and b_1 denote the weight and bias of the first linear transformation, W_2 and b_2 are parameters of the second transformation.

Here we preprocess each sublayer input by the layer normalization (Ba et al., 2016). All sublayers are coupled with the residual connection (He et al., 2016a).

B PTQ and QAT

As an appealing solution to model compression, quantization enables the model to use lower-bit values (such as 8-bit integer) to compute faster and consume less storage space (Hubara et al., 2016; Micikevicius et al., 2018; Quinn and Ballesteros, 2018; Jacob et al., 2018).

Post-Training Quantization (PTQ) can be seen as the basis for Quantization Aware Training (QAT), it adds quantization nodes to a well-trained floating-point model. To quantize a floating-point tensor r to a tensor with n bits, a scale s is introduced to map these two types of values (Wu, 2020):

$$s = \frac{\max(r) - \min(r)}{2^n - 1} \quad (4)$$

System	Params (M)	Size (MB)	BLEU
Transformer-base	65	260	27.40
+ Reducing Vocab	48	192	26.20
+ Reducing Width	10	40	22.01
+ Other Dimensions	10	40	22.54
+ Distillation	10	40	23.77
+ Quantization	10	10	23.76
+ Hyperparameters	10	10	25.48
+ Greedy Search	10	10	25.08

Table 6: Ablation study on MobileNMT-10MB. The colors refer to Model Architecture in Section 2, Training Strategies in Section 3 and Greedy Search.

To get a faster computation speed, both weights and activations will be quantized to n-bit. Suppose $r_m = \min(r)$, the quantization function is:

$$Q(r) = \lfloor (r - r_m)/s \rceil \times s + r_m \quad (5)$$

where $\lfloor \cdot \rceil$ represents rounding to the nearest integer.

However, in PTQ, applying quantization directly to the floating-point network will result in significant performance losses. Based on PTQ, QAT simulates the behavior of n-bit computation by minimizing quantization errors during training, which helps the model achieve higher accuracy. In addition to the learnable weights of the model itself, s is also learnable.

C Operations except GEMM

Since general matrix multiplication (GEMM) accounts for 80.44% of the overall decoding time, we have concluded that optimizing GEMM is the key to decoding speed up in Section 4. As for operations except GEMM, Fig. 10 shows the proportion of running time in the decoding process. The corresponding data is measured in 32-bit floating point format on the ONNX Runtime.

D Setups

All sentences were segmented into sequences of sub-word units (Sennrich et al., 2016). In the implementation, we adopt the normalization before layers (Baevski and Auli, 2019; Xiong et al., 2020; Nguyen and Salazar, 2019). Most previous work only shared source and target vocabularies on the En-De task. In our MobileNMT, both En-De and En-Fr adopt shared vocabularies for efficiency reasons, which leads to a larger compression gain at the expense of performance. We test on the model ensemble by averaging the last 5 checkpoints and report SacreBLEU scores (Post, 2018).

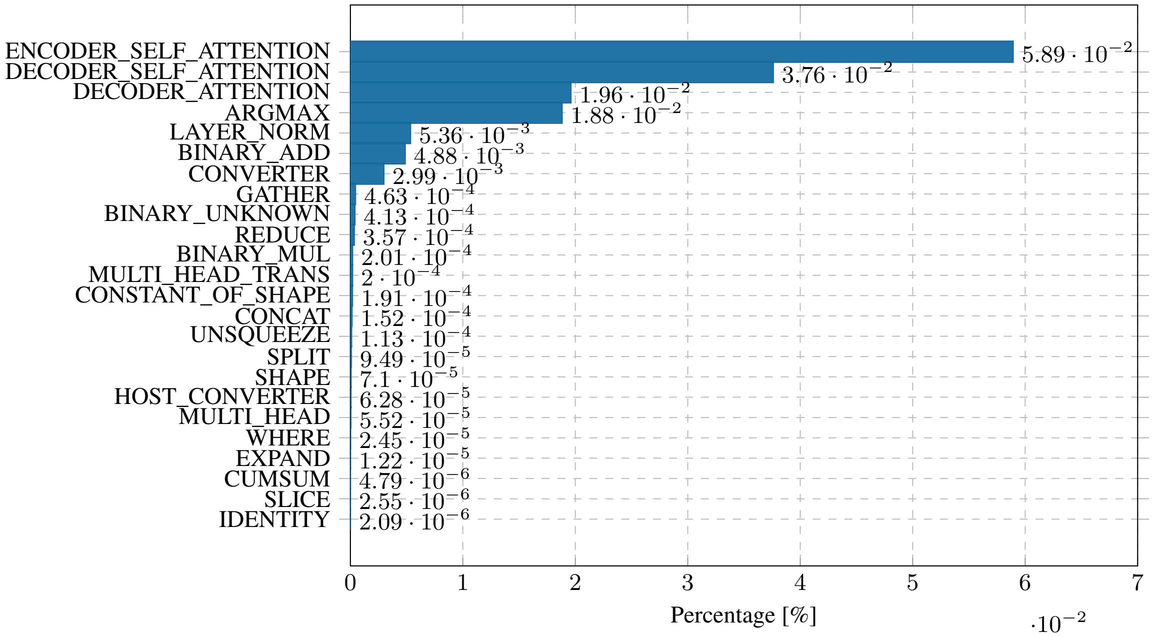


Figure 10: Proportions of different operations (except GEMM) on the Transformer-base model.

System	Params (M)	Bits (W-E-A)	Size (MB)	BLEU
Transformer-base	65	32-32-32	260	27.40
MobileNMT-10MB	10	32-32-32	40	25.79
	10	8-8-8	10	25.08
	10	4-8-8	5	25.43
	10	3-8-8	3.75	24.09
	10	2-8-8	2.5	21.25
MobileNMT-20MB	20	32-32-32	80	27.30
	20	8-8-8	20	27.09
	20	4-8-8	10	26.96
	20	3-8-8	7.5	26.23
	20	2-8-8	5	24.33

Table 7: Results of quantizing weights to lower bits.

For the experiments of MobileNMT in Table 4, we use the greedy search algorithm in our engine. Compared with beam search, greedy search can lead to more efficient decoding. For the experiments of TFLite in Table 4, since TFLite will expand all loop subgraphs, it is hard to support the entire decoding process (30 steps) of the Transformer-big/base model with limited memory (6GB in Google Pixel 4). For the memory of these two models, we only record the running memory of 1 step. For the corresponding latencies, we estimate the 30-step latency according to the 1-step and 5-step latencies. It is worth noting that except for the memory and latency on Transformer-big/base, all other data statistics are measured in real-world.

E Analysis

E.1 Ablation Study

Table 6 summarizes how each part of Section 2 and Section 3 affects the overall performance. Each row in Table 6 represents the result of applying the current part to the system obtained in the previous row.

To reduce the model parameters from 65M to 10M, the model performance decreased from 27.40 to 22.54, which illustrates the importance of network parameters on model capacity. We observe that both knowledge distillation and tuning hyper-parameters can bring significant performance improvements (from 22.54 to 25.48), which effectively compensate for the performance loss caused by parameter reduction.

E.2 Quantization Study

Table 7 studies how performance changes when quantizing the model to lower bits (i.e., 4-bit, 3-bit, and 2-bit). As introduced in Section 3.2, for 8-bit quantization, we use the element-wise quantization method (Lee et al., 2021). For lower-bit quantization, we use the row-wise quantization for accuracy preservation (Faraone et al., 2018).

As shown in Table 7, 8-bit and 4-bit quantization have almost no negative effect on model performance. When quantizing the model to lower bits, such as 3-bit and 2-bit integers, model performance will drop dramatically.