Efficient Deep Learning Inference on IoT Devices

Zhizhuo Liu[1], Min Liu[1], Chaonong Xu[1] ✉, Chao Li[2]
[1]China University of Petroleum-Beijing, [2]Zhejiang Lab
Beijing, China
xuchaonong@cup.edu.cn

Abstract

Deploying deep learning models on resource-constrained IoT devices based on microcontroller units (MCU) is appealing, but faces significant challenges due to constrained computational capabilities and memory resources. To squeeze deep learning models into MCU, we propose MCUQ, a novel framework that combines an efficient quantization approach (RLQuant) with a lightweight low-precision inference engine (LiteEngine). RLQuant finds the optimal quantization policy by solving a bi-level optimization problem, where the upper optimization problem utilizes reinforcement learning to search for the optimal bitwidth and the lower optimization problem utilizes SGD for quantization step. After each iteration of RL, LiteEngine is executed by employing generic operand packing technique, which reduces memory requirements and accelerates model inference. LiteEngine is closely coupled with RLQuant due to feedback on its inference latency, memory usage and accuracy, wherein such feedback is used in determining the optimal quantization policy of RLQuant through reinforcement learning. MCUQ results in reduced memory usage by $1.5-2.8\times$ and accelerated inference by $1.6-3.0\times$ compared to CMix-NN, CMSIS-NN, and TinyEngine.

CCS Concepts: • **Computer systems organization → Embedded software**; **Neural networks**.

Keywords: efficient deep learning, quantization, MCU, efficient inference

1 Introduction

Deep neural networks (DNNs) have yielded impressive results in various fields. Due to their high memory and computing requirements, DNNs are typically deployed on data-parallel accelerators. Recently, some researchers deployed DNNs on resource-constrained IoT devices. IoT devices often have low computing performances and small memory sizes, which highlights a contradiction between the high resource demands of model deployment and the limited resources of IoT devices. To resolve this contradiction, quantization is considered as an appropriate scheme. However, existing quantization research has only achieved a balance between accuracy and either inference latency or memory usage, and the resulting models are not efficiently deployable on MCU.

On the other side, DNN libraries are widely used in IoT devices to improve the throughput and computational efficiency of DNN inference. CMix-NN [1] and CMSIS-NN [3] are low-precision linear algebra kernels designed for deployment of quantized neural networks on MCU targets. Using these kernels can efficiently parallelize the computation of 8-bit operands, but is not optimal for sub-8-bit operands. TVM [2] uses bit-serial computation to solve the problem of computing sub-8-bit operands, but it cannot be extended to the ARM Cortex-M based MCU.

In this paper, we propose MCUQ, a novel framework that combines an efficient quantization approach (RLQuant) with a lightweight low-precision inference engine (LiteEngine). RLQuant, which finds the optimal bitwidth and step size for every layer of networks simultaneously, enhances the accuracy after quantization. LiteEngine, which utilizes a generic operand packing method, results in higher utilization of computing and memory resources. Last but not least, based on the inference accuracy evaluation, the bit width and step size output by RLQuant affect the depth of the operand packing, while the depth has an impact on the inference accuracy. It is their cooperation that lays the foundation for high accuracy, low memory usage, and fast inference on IoT devices.

RLQuant finds the optimal quantization policy by solving a bi-level optimization problem, employing reinforcement learning at the upper level to search for optimal bitwidth, and SGD at the lower level for quantization step. After each iteration of RL, LiteEngine is executed by employing generic operand packing technique, which packs multiple operands into register lanes. Because of its close integration with multiply instructions, especially the Multiply-Reduce Instructions, LiteEngine ensures efficient parallel computation. It also reduces memory usage for most operands through bit-level memory scheduling. LiteEngine is closely coupled with RLQuant due to feedback on its inference latency, memory usage and accuracy, wherein such feedback is used in determining the optimal quantization policy of RLQuant through RL.

We have evaluated the performance of MCUQ on the ARM Cortex-M MCU, and the results indicate a significant improvement compared to other quantization methods. Specifically, it achieves the acceleration ratio of $1.6-3.0\times$ compared to CMix-NN [1], CMSIS-NN [3], and TinyEngine [4]. Additionally, the memory usage is reduced by a factor of $1.5-2.8\times$, while its inference accuracy remains almost unaffected. In summary, this paper makes the following contributions:

- Introducing a **generic operand packing** method for IoT device instructions that packs multiple operands into register lanes, ensuring close integration with

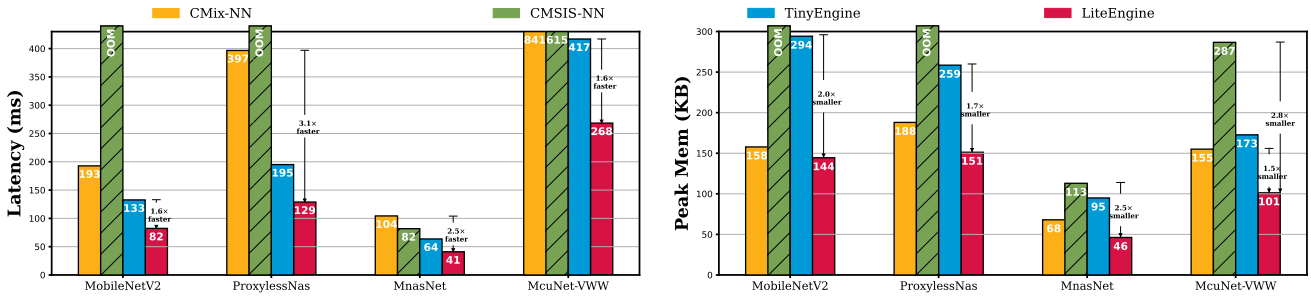

Figure 1. LiteEngine achieves higher inference efficiency than existing approachs while reducing memory usage. **Left**: LiteEngine is 3.0× faster than CMix-NN and 1.6× faster than TinyEngine. Note that if the required memory exceeds the constraint, it is marked as "OOM" (out of memory). **Right**: By reducing memory usage, LiteEngine can run various quantization policies with tiny memory, expanding the search space for RLQuant under resource-constrained MCU.

multiply series instructions, particularly the Multiply-Reduce Series Instructions.

- RLQuant finds the optimal bitwidth and step size for each layer of networks simultaneously by solving a bi-level optimization problem. LiteEngine is closely coupled with RLQuant due to feedback, which is used in determining the optimal quantization policy of RLQuant.

2 Generic operand packing approach and experiment results

The essence of operand packing lies in performing low-precision dot product computations by packing multiple sub-8-bit operands into a wider register and performing a regular multiplication using the two packed registers. Figure 2(a) illustrates the regular multiplication operation, which can only accommodate one pair of operands. In contrast, Figure 2(b) showcases the utilization of the operand packing technique, enabling the dot product of two pairs of 2-bit operands by performing an 8-bit multiplication. The result is then represented as $w_1a_0 \cdot 2^8 + (w_0a_0 + w_1a_1) \cdot 2^4 + w_0a_1$. However, since the result exceeds the maximum value of the destination register, the high-order term $w_1a_0 \cdot 2^8$ overflows, leaving the destination register with the value $(w_0a_0 + w_1a_1) \cdot 2^4 + w_0a_1$. To obtain the correct result, the destination register undergoes a masking operation (with 0xFF00) followed by a right shift operation.

Figure 1 demonstrates that LiteEngine achieves a remarkable acceleration of up to 3× compared to CMSIS-NN [3], CMix-NN [1], and TinyEngine [4]. This significant speedup is primarily attributed to the generic operand packing method employed by LiteEngine. Additionally, the co-design between the algorithm and system allows precise control over the executed models, ensuring optimized memory scheduling and code generation specifically for quantized models using RLQuant. LiteEngine eliminates runtime interpretation latency and avoids memory allocation for storing model metadata, enabling the processing of high-precision models. As a result, LiteEngine improves memory usage efficiency, utilizing 2.8× less memory compared to CMSIS-NN.

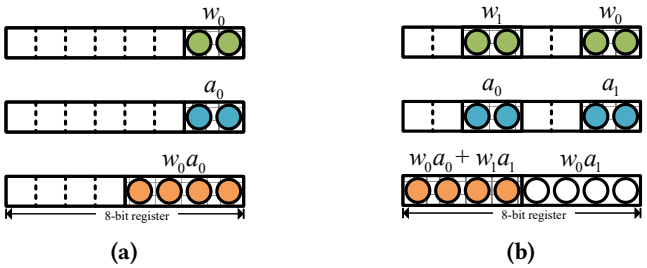

Figure 2. Left: Multiply computation, which can only perform operations on one pair of operands. **Right**: Dot product computation using operand packing, which can simultaneously perform operations on two pairs of operands.

Acknowledgement

This study is supported by National Key R&D Program of China (2022YFB4501600).

References

[1] Alessandro Capotondi, Manuele Rusci, Marco Fariselli, and Luca Benini. 2020. CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices. *IEEE Transactions on Circuits and Systems II: Express Briefs* 67, 5 (2020), 871–875. https://doi.org/10.1109/TCSII.2020.2983648

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.

[3] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601* (2018).

[4] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems* 33 (2020), 11711–11722.