

Energy-Efficient Federated Training on Mobile Device

Qiyang Zhang, Zuo Zhu, Ao Zhou, *Member, IEEE*, Qibo Sun, *Member, IEEE*, Schahram Dustdar, *Fellow, IEEE*, Shangguang Wang, *Senior Member, IEEE*

***Abstract*—On-device deep learning technology has attracted increasing interest recently. CPUs are the most common commercial hardware on devices and many training libraries have been developed and optimized for them. However, CPUs still suffer from poor training performance (i.e., training time) due to the specific asymmetric multiprocessor. Moreover, the energy constraint imposes restrictions on battery-powered devices. With federated training, we expect the local training to be completed rapidly therefore the global model converges fast. At the same time, energy consumption should be minimized to avoid compromising the user experience. To this end, we consider energy and training time and propose a novel framework with a machine learning-based adaptive configuration allocation strategy, which chooses optimal configuration combinations for efficient on-device training. We carry out experiments on the popular library MNN and the experimental results show that the adaptive allocation algorithm reduces substantial energy consumption, compared to all batches with fixed configurations on off-the-shelf CPUs.**

***Index Terms*—Ubiquitous intelligence, Deep learning, Asymmetric multiprocessor, Energy efficiency**

I. Introduction

Deep learning (DL) technology is widely used by mobile devices (smartphones, IoTs, wearables, etc) in real-world applications [1], such as input methods and virtual assistants. Meanwhile, we are witnessing the emergence of a novel paradigm that directly leverages mobile devices for model training/inference, referred to as ubiquitous intelligence [2]. DL inference is known to happen on devices due to the advantages of network resilience and quick response without cloud offloading [3]. Fueled by the increasingly powerful processors, it becomes possible to train models on devices apart from inference. This breaks the paradigm that the training stage of DL is commonly placed on data centers with massive data and computing resources.

Though various System of Chips (SoCs) have been developed recently, CPUs remain the dominant hardware because mobile devices are equipped with them. CPUs have advantages in general availability and the mature programming environment while other AI accelerators lack a uniform interface to facilitate their development [4]. For example, GPUs are also widely available and have better performance on the majority of devices, but cannot support many DL models. The training time of GPU is even longer than that of CPU since most training libraries are incomplete [5]. So CPUs still play a vital role in on-device DL. Thus, we mainly focus on CPUs in this paper.

As an emerging on-device computing paradigm, Federated Training (FT) is an algorithm to enable DL training across devices and has gained huge attention in both academia and industry [6]. The key idea of FT is to employ a set of mobile devices to train a model collaboratively under the orchestration of a central server. The training process of FT takes place on mobile devices with uploading model parameters instead of user personal data. Multiple mobile devices collaboratively train local models until the global model converges. In this paper, we focus on efficient local training on mobile devices with multiprocessors to obtain an optimal model, as shown in Fig. 1. For each device with diverse computing capacities, DL developers expect on-device training to be finished as soon as possible to keep a synchronized pace. Meanwhile, the energy consumption shall be minimized to not compromise user experience due to limited battery. This explains why training time and energy are critical for mobile devices. Inspired by the CPU architecture, we conduct the preliminary measurement to explore the training performance and the results shed light on the future tradeoff between training time and energy.

In this paper, aiming to select the optimal configuration combinations for CPUs in the training stage, we propose an overall system framework including the profile and execution module to reduce energy consumption. In the execution module, we utilize machine learning-based techniques to obtain the energy estimation and configuration switching models to better estimate the energy loss and switching overhead. In the execution module, we present an adaptive scheduling algorithm to identify the most efficient configuration combinations, considering the asymmetric CPU architecture. In a nutshell, we propose an adaptive configuration combinations allocation strategy by adjusting system parameters (e.g., CPU cores, frequency) to reduce the energy for training models under the training time constraint. The key contributions of our work are as follows:

- We investigate the impact of system parameters on training performance. We also derive some interesting observations, which help guide us in the promising direction to enable ubiquitous intelligence efficiently.
- We propose a machine learning-based adaptive configuration combinations allocation strategy that greatly im-

Q. Zhang, Z. Zhu, A. Zhou, Q.Sun (corresponding author), and S. Wang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. E-mail:{qyzhang,zhuzuo,aozhou,qbsun,sgwang}@bupt.edu.cn

Schahram Dustdar is Full Professor of Computer Science heading the Research Division of Distributed Systems at the TU Wien, 1040 Wien, Austria. E-mail: dustdar@dsg.tuwien.ac.at

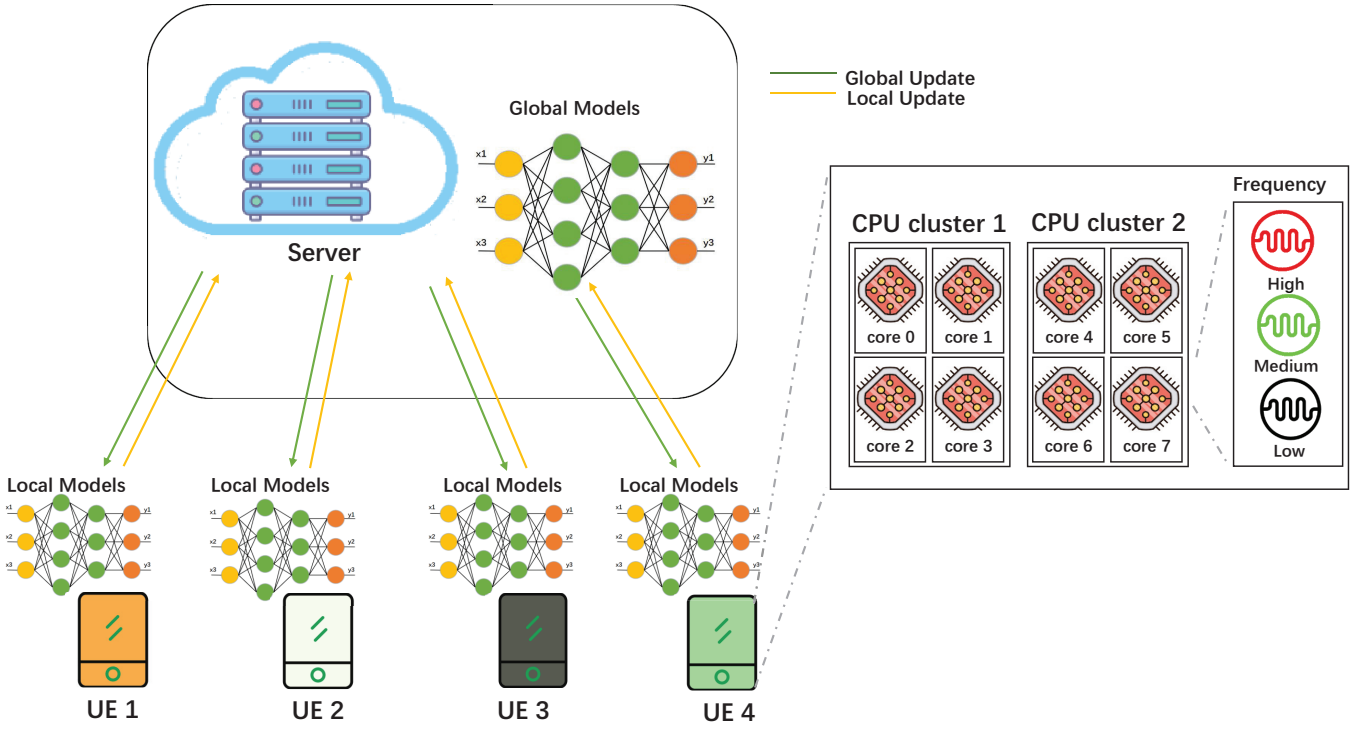


Fig. 1. The FT System Architecture. The work focuses on mobile devices with diverse computing capacities to complete the local training quickly.

proves the training performance (i.e., training time and energy).

- We deploy a simplified case study of on-device training on Meizu 16T. The experimental results demonstrate that the algorithm performs better than all batches with fixed configurations in terms of training performance.

II. Related Work

Recently, various emerging approaches have been designed to optimize FT, including privacy, fairness guarantee, and communication cost between servers and devices [7]. Existing FT studies typically make simulation methods because real-world deployment is expensive. As a result, it's assumed that all devices are always equipped with homogeneous hardware specifications (e.g., the same CPU). However, the assumption could be too ideal in real-world applications. More specifically, FT usually requires a substantial number of devices with hardware specifications to collaboratively accomplish a learning task. On-device learning works in the existing literature can be divided into two categories summarized in Table I:

The first category is neural network-aware implementation. General training techniques are to train models with quantized weights, activations, and gradients since the model size arises to be the major bottleneck [8]. That's because the key difference between training and inference is training needs to store intermediate activations for back-propagation while inference doesn't. Therefore, reducing the activation size is critical for training. TinyTL [9] leverages a hardware-friendly module to refine the feature extractor and reduce the training memory footprint. Neural Projections [10] further extends neural networks with computationally efficient operations to generate compact representations. Low-bit Neural Network Training [11] and Melon [12] provide a memory-friendly framework that enables training tasks by quantization. Unfortunately, these approaches also have a strong dependency on low-bit training. Especially for a large-scale distributed training environment, the quantization techniques are no longer enough as the gap between high-end networking and the normal one is large [8]. The second category is to train DL models by optimizing resource scheduling. Mandheling [13] enables highly resource-efficient on-device training by orchestrating mixed-precision training with DSP offloading, which leverages the available computing capability of the on-chip DSP to improve training performance. The authors propose a deep reinforcement learning-based frequency scaling technique to maximize application performance [14]. These existing recent works use Raspberry Pi and seldom use smartphones which is the killer use-case for ubiquitous intelligence. Unlike them, our work is to reduce the energy of single model training from the system-level perspective, which is orthogonal and compatible with those existing works.

III. Background

In this section, to investigate the impact of system parameters on training, we conduct experiments on smartphones based on several typical system configurations. Meanwhile, we also obtain some interesting observations.

As we all know, mobile devices are energy-constrained because users are sensitive to battery consumption by nature [15]. For instance, mobile devices may experience delay or even failure in sending updates because the battery is dead in the process of FT. Meanwhile, the asymmetric big.LITTLE

TABLE I. Comparisons between our work and on-device training in the existing literature.

Categories	Typical Literature	Features
Neural-network aware	TinyTL [9]	1) Transfer learning method to reduce the training memory footprint; 2) A memory-efficient bias module to improve the model capacity.
	Neural Projections [10]	1) Locality-sensitive projections to generate compact binary representations; 2) Neural nerworks with computationally efficient operations.
	Low-bit Neural Network Training [11]	1) Software optimizations by low-bit quantization; 2) Hardware design of a bit-flexible multiply-and-accumulate array sharing the same sources.
	Melon [12]	A memory-friendly framework that enables the training tasks with large batch size.
Resource scheduling	Mandheling [13]	1) CPU-DSP mixed-precision training; 2) Self-adaptive rescaling.
	zTT [14]	A deep reinforcement learning based technique to achieve maximum performance while ensuring zero thermal throttling.
Our work	Adaptive scheduling	1) Choosing the optimal configuration combinations at the training stage; 2) Reducing the energy consumption without compromising the user experience.

TABLE II. Training performance with different CPU configurations on Meizu 16T. "H": highest frequency (2.4GHz/1.8GHz for big/little core); "M": medium frequency (1.6GHz/1.2GHz for big/little core); "L": lowest frequency (0.8GHz/0.6GHz for big/little core).

CPU Conf.	Time(s)			Energy(J)		
	H	M	L	H	M	L
Big 1×	4.2	5.4	10.8	10.6	8.0	6.9
Big 2×	2.6	3.2	6.4	8.9	7.7	7.0
Big 4×	2.0	3.3	8.4	7.1	8.7	8.2
Little 1×	25.0	33.9	57.8	10.4	7.2	3.1
Little 2×	13.3	18.0	31.8	10.1	8.4	4.8
Little 4×	8.0	11.0	52.3	11.4	9.6	8.2
Hybrid 8×	3.8	6.5	50.4	13.4	13.9	14.4

technology is widely adopted in popular mobile devices [7]. Specifically, Android smartphones have a substantial number of adept configurations (core number × frequency). The big and little processors often have isolated domains and thus can set frequency separately. Each SoC can be independently managed (e.g., turn on/off and configuration control) for the dynamic workloads. Compared to the big processors, the little processors have a lower CPU frequency and lower power cost [4]. The training tasks are always improperly assigned to the big processors much more often than the little ones by the *Schedutil* governor [14]. Unfortunately, we find that the training time with the only big processor is even shorter than both processors. Meanwhile, the high frequency of CPUs is only suitable for training time rather than energy cost. Taking the low frequency as an example, the training time using both processors is slower than only big processors, although the little processors provide additional computing capability. Similarly, it's challenging to find the optimal energy consumption for the execution of both processors. So we conclude that DL training barely gains speedup by using both big and little processors compared to just utilizing big processors. As such, it is critical to set CPU configuration directly for DL training on CPUs.

To have a better understanding of how on-device training performs, we set the fixed configuration for all batches of the overall training. We combine the CPU configurations between core numbers (1×, 2×, 4× big/little cores, 8 hybrid cores) and the frequency of each core (highest, medium, lowest). Meizu 16T trained AlexNet model based on MNN [3] for 20 epochs and the results are summarized in Table II. Overall, the results are consistent with the fact that CPU architecture is asymmetric as we stated above. In terms of training time, 4× big cores with the highest frequency lead to the best performance. In terms of energy consumption, 1× small core with the lowest frequency achieves the lowest energy consumption. The cost of the best energy configuration is only 43.7% of the best training time, despite it running 28.9× slower. The results show a big potential for energy saving given that developer set a proper CPU frequency. In other words, the results shed light on the tradeoff between training time and energy consumption to efficiently enable ubiquitous intelligence. To visually show the performance differences among these CPU configurations, Fig. 2 exposes the unbalanced performance with the most representative configurations (1× and 4× big/little cores with high/medium/low frequency). Intuitively, we summarize that big processors are good for training time, while small processors have smaller energy consumption. In fact, such a configuration that meets both training time and energy consumption requirements is significant for ubiquitous intelligence. Thus, we should adopt those suitable parameters to enable efficient local training.

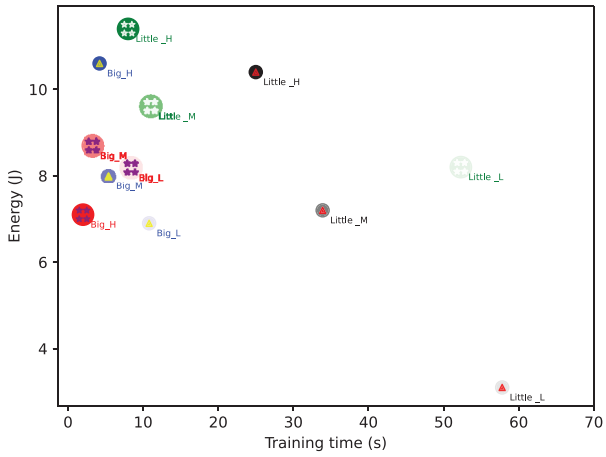


Fig. 2. The unbalanced performance with the most representative configurations. The color of the circles represents the CPU frequency (the color from light to dark denotes low, medium, and high frequency, respectively). The number of stars and triangles inside circles denotes the core number.

There are still two key challenges in selecting the suitable configuration for on-device training. Firstly, the optimal sys-

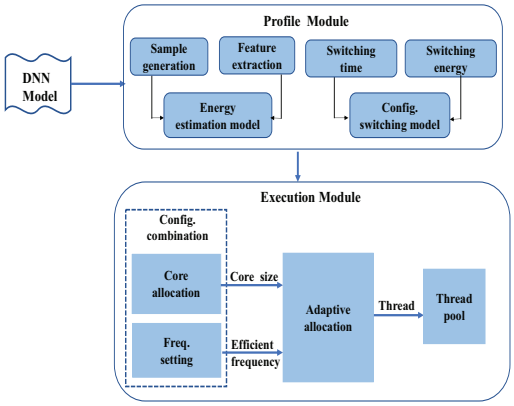


Fig. 3. The overview of the proposed framework.

tem configuration for batches of training is difficult to select in real-world applications due to several configurations and batch numbers. Specifically, there may be hundreds or even thousands of batches for one training and each batch can also select any configuration from the candidate configurations (CPU core number × frequency). For this issue, we propose an adaptive configuration combination algorithm to select system configuration for each batch instead of manual selection. Secondly, switching different configurations between two adjacent batches results in extra overhead, when selecting the optimal configuration for batches in the stage of training. We deem that measuring the cost of real-world training is too costly, or even infeasible. Considering the switching cost, we propose a machine learning-based model to obtain the overhead closer to real applications. Overall, to achieve efficient on-device training, we propose a framework to seek the optimal configuration for minimizing the overall energy consumption without compromising the user experience. Therefore, we transform this issue into the *Min-Energy* problem.

IV. On-device Training with Adaptive Processor Configurations

In this section, we introduce the *Min-Energy* problem where the goal is to minimize energy while satisfying a certain training time threshold as our objective. More specifically, we expect to seek optimal configuration combinations for local training on each device. We first model this problem by formula.

Let $C = (x_1, x_2, ..., x_n)$ denote a configuration combination. It is also a list of tuples x_i, which indicates that the ith batch selects the CPU configuration. n is the number of all batches in training. Let $E(C)$ denote the energy of the training local model with a configuration combination C and the training time constraint is θ_t. We also have $T(C) = T(x_1, x_2, ..., x_n)$ as the real training time. Therefore, the *Min-Energy* problem can be formulated as follows.

$$\begin{aligned} \min \ & E(x_1, x_2, ..., x_n) \\ \text{s.t. } & T(x_1, x_2, ..., x_n) \leq \theta_t \end{aligned} \tag{1}$$

A basic idea for solving the *Min-Energy* problem is to try all possible configuration combinations by brute force. However, the brute force method is impossible to find the optimal solution due to a large number of combinations. Therefore, determining the best configuration combination efficiently is a very challenging task. In addition, it is also challenging to estimate computing overhead when there are configuration switchings, since switching different configurations between two adjacent batches inevitably leads to extra overhead.

Today there is no existing tool to measure the overhead directly, an efficient method shall be proposed to estimate the extra energy and switching time. So we propose a framework that considers energy consumption and training time to obtain energy-efficient training, as shown in Fig. 3. More specifically, based on time and energy estimation models derived from historical data, all batches with the least-energy configuration work together to complete local training tasks during execution. After a DNN model is loaded in the initialization, the adaptive configuration combinations bind threads to CPU cores and schedule tasks to threads, because each processor has isolated a power domain and can set frequency separately by the *Userspace* governor [14].

The proposed framework consists of two main modules: the profile and execution modules. In the profiling module, we train the energy estimation and configuration switching models. We collect samples for training and testing the energy loss estimation model. 300 configurations are randomly generated and the energy loss is measured on smartphones. Among these samples, the ratio is 5:1 for model training and testing, respectively. More specifically, we generate the configuration combinations by scheduling each configuration and repeat the above steps until 300 different configuration combinations have been generated.

To train the model, we need to exact features from the samples. The feature vector contains three parts: configuration, memory, and thermal features. The configuration features include the binary variables $x_i, (i \in [1, n])$ which indicate whether the ith batch is selected as the optimal configuration and the average of their energy loss. The memory features include the memory space occupied by the configuration combinations parameters and input\output data. The thermal features include the vectors that affect energy consumption. Since the exacted feature vectors have different ranges, normalizing their ranges can improve the performance of machine learning models. *Min-Max normalization* is applied to scale the range of features to [0, 1]. With the normalized features, we train the model to estimate energy loss. The biggest challenge is to decide which model should be used. It is also not the best choice in the problem due to overfitting, where the model can be learned from the training samples well but not generalized to new data. The overfitting problem may be caused by the small number of samples and low-dimensional features. Therefore, we train some models (i.e., Linear Regression, GBRT, etc.) to estimate the energy loss and choose the best-performing model.

In addition, to better estimate the switching time between two adjacent batches, especially when many configurations are selected. A configuration switching model can be implemented to obtain switching overhead. Similar to the energy estimation model, we collect several adjacent configurations to train the prediction model. Note that, although the energy

estimation model is related to the current configuration while the configuration switching model is involved in the two configurations, we can still accurately estimate the models. Therefore, it's necessary to consider these models separately for these configuration combinations. After all the preparation in the profile module, an adaptive configuration allocation algorithm in the execution module is proposed to solve the *Min-Energy* problem. Here we apply a dynamic programming-based algorithm to adaptively seek the optimal configuration combination with the least energy on asymmetric CPUs for batches.

V. A Case study of On-device Training

On-device training performance is related to many metrics including training time, energy consumption, memory footprint, and thermal dynamics [5]. As reported in Section III, these metrics guide us to seek an appropriate configuration combination for training. In reality, a developer or the OS might control the CPU core and frequency to harness such a tradeoff between training time and energy consumption. To ensure fast convergence of the global model in FT, each device should complete the learning task as fast as possible and minimize energy consumption. Hence, we focus on the *Min-Energy* problem to choose the optimal configuration combination for the batches.

Algorithm 1 shows the pseudocode. More specifically, a set $S(i)(i \in [0,n])$ is maintained for the optimal configuration combination. $S(i)$ is a set of triples, and each triple is donated as (C,T,E), which represents the training time T and the energy E of training the model with configuration combination C. Initially, we sort the configurations in descending order based on their energy and assign the configuration with minimal energy for the batches. Note that, for the same model and device, the difference in training time and energy of each batch is negligible because a large number of testing results show little bias ($< 5\%$) for training time and energy, respectively. A triple (C,T,E) is said to dominate another triple $(C^{'},T^{'},E^{'})$ if and only if $T \leq T^{'}$ and $E \leq E^{'}$.

We first investigated the DL libraries that support training on typical mobile devices. We observe that MNN achieves great performance on most devices such as IoTs and smartphones than its competitors [5]. MNN is already been adopted widely in the productions of Alibaba Inc. Therefore, this study focuses on MNN as the training framework. We also utilize Android APIs to read the current battery, USB power, and voltage supply on mobile devices. We thus obtain the real power at the stage of training. Energy is integral of power during training over time. We calculate it by multiplying the measured output power (USB and the power of the battery) and the training duration. The reported energy and training time are the arithmetic means of many runs when no background application is running.

A. Experimental Settings

We used two different datasets in the study: MNIST dataset (70,000 images, 10 classes, 28×28×1 Input Size) and a subset of ImageNet (3,200 images, 4 classes, 224×224×3 Input Size).

Algorithm 1: Solving Min-Energy

Input: the initial configuration combinations C_0, prefined threshold θ_t

Output: the configuration combinations E_{best} which satisfies the time requirement

```
S(0) ⟵ (C_0, T(C_0), E(C_0)),
E_best ⟵ E(C_0), C_best ⟵ C_0
for i = 1 to n do
    for each (C, T, E) in S(i − 1) do
        for each configuration in candidate set do
            Generate C' by adjusting the configuration in C
            if T(C') ≤ θ_t then
                Add (C', T(C'), E(C')) to S(i)
            if E_best > E_C' then
                E_best ⟵ E(C'), C_best ⟵ C'
        Remove the dominated triples from S(i)
        Update S(i)
Return E_best
```

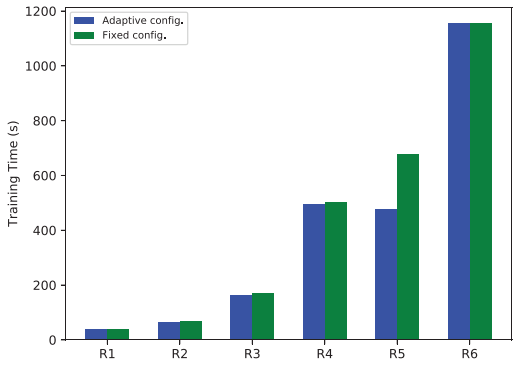
The larger batch size could be selected since a large bath size benefits the intra-operator parallelism to gain more energy-efficient. Fortunately, model training is known to be memory hungry and 16 is the maximal bath size that mobile phones can support common models [5]. That is different from the inference stage, because, during inference, the input is fed into the model one by one and the intermediate results don't need to be stored for backward propagation.

We also carried out experiments with AlexNet (5 convs, 61M parameters) and LeNet (2convs, 3.2k parameters) on Meizu 16T (Snapdragon 855, 6GB RAM, 4 big cores + 4 little cores) produced in 2019. For convenience, we ran AlexNet for 20 runs (additional warmup runs are not excluded).

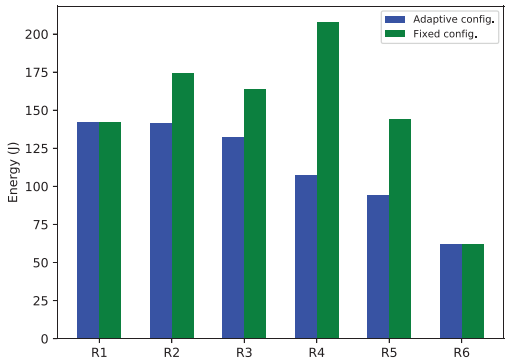
B. Numerical Results and Analysis

Based on the above experimental design, Fig. 4 discusses the performance of the proposed algorithm in terms of training time and energy. We choose the predefined training threshold including 4× big cores and 1× little core from the available options. These scenarios have a wide time range and are also universally represented configuration schemes. We thus can get the maximum and minimum time thresholds (i.e., including upper and lower bounds).

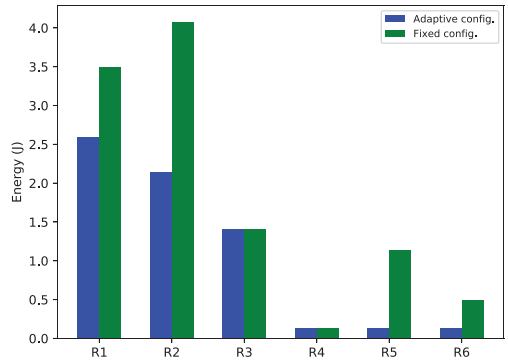
To compare the proposed algorithm with benchmark algorithms, we train the models on the smartphone and calculate the training time and energy consumption respectively. Fig. 4(a) describes the training time of training AlexNet. Overall, we observe that the proposed adaptive configuration algorithm takes less training time. When especially the predefined training time is small, the adaptive configuration takes the same time as the results of the fixed configuration (i.e., the lower bounds). The adaptive algorithm has the only configuration. Also, using 4× big cores with the highest frequency results in the smallest training time. Fig. 4(b) describes the energy



(a) Training time of AlexNet with different schemes.



(b) The energy of training AlexNet with different schemes.



(c) The energy of training LeNet with different schemes.

Fig. 4. Training performance of the proposed algorithm and benchmark algorithms. R1, R2, and R3 indicate high, medium, and low frequency in the case of Big 4×. R4, R5, and R6 indicate high, medium, and low frequency in the case of Little 1×.

consumption of training AlexNet and the proposed algorithm saves 30% energy consumption on average compared to fixed configurations. Extremely, using 1× little core with the lowest frequency leads to the smallest energy. Although this scheme has a larger threshold, to obtain the lowest energy consumption, the adaptive configuration is exactly the same as the fixed configuration.

This algorithm also allows training any other lightweight models on almost mobile devices. To evaluate the generalization performance of the algorithm, we trained LeNet as shown in Fig. 4(c), which shows the performance of training with 100 epochs. We also observe that the proposed algorithm still outperforms the fixed configuration allocation strategy. In general, employing the adaptive configuration can show significant advantages in training performance. We deem that a similar phenomenon also exists in other models.

C. Limitation and Discussion

In real-world applications, switching different configurations in adjacent batches is inevitably bound to incur overhead. Thus, we implement experiments with different configurations for training AlexNet for two adjacent batches on Meizu 16T. Surprisingly, the switching time generated by the configuration transformation is 2−5 ms. The energy consumption is also approximately the lowest one percent per batch. We conclude that the insignificant overhead has little impact on the adaptive configuration algorithm.

What's more, choosing the optimal configuration for on-device learning is still challenging, as it depends on model structure, batch size, hardware specification, status, etc. Due to limitations such as resource-constrained data and memory of devices, on-device training can only be oriented to simple tasks. The implementation on devices shall be under root privileges. The ability of root permissions is also different due to the openness of the different smartphone manufacturers' systems. For instance, with the same root privilege, Meizu smartphones set the frequency directly through the writing configuration method while other smartphones such as Xiaomi through the command line. Notoriously, the overhead of syscalls shell commands is much greater than w/r files. Overall, the algorithm provides new attempts but has strict requirements for the devices.

ACKNOWLEDGMENT

This work was partly supported by National Key RD Program of China under grant number 2020YFB1805500, and National Natural Science Foundation of China under grant number 62032003, U21B2016. Qiyang Zhang was supported by BUPT Excellent Ph.D. Students Foundation(CX2021231).

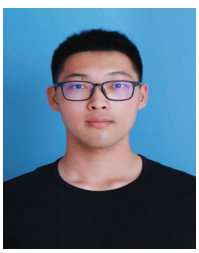
REFERENCES

[1] M. Xu, T. Xu, Y. Liu, and F. X. Lin, "Video analytics with zero-streaming cameras," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 459–472.

[2] M. Xu, X. Zhang, Y. Liu, G. Huang, X. Liu, and F. X. Lin, "Approximate query service on autonomous iot cameras," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 191–205.

[3] Q. Zhang, X. Li, X. Che, X. Ma, A. Zhou, M. Xu, S. Wang, Y. Ma, and X. Liu, "A comprehensive benchmark of deep learning libraries on mobile devices," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 3298–3307.
[4] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, "Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 215–228.
[5] D. Cai, Q. Wang, Y. Liu, Y. Liu, S. Wang, and M. Xu, "Towards ubiquitous learning: A first measurement of on-device training performance," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, 2021, pp. 31–36.
[6] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.
[7] C. Yang, Q. Wang, M. Xu, Z. Chen, K. Bian, Y. Liu, and X. Liu, "Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data," in *Proceedings of the Web Conference 2021*, 2021, pp. 935–946.
[8] H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han, "Enable deep learning on mobile devices: Methods, systems, and applications," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 3, pp. 1–50, 2022.
[9] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 285–11 297, 2020.
[10] S. Ravi, "Efficient on-device models using neural projections," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5370–5379.
[11] S. Choi, J. Shin, Y. Choi, and L.-S. Kim, "An optimized design technique of low-bit neural network training for personalization on iot devices," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.
[12] Q. Wang, M. Xu, C. Jin, X. Dong, J. Yuan, X. Jin, G. Huang, Y. Liu, and X. Liu, "Melon: Breaking the memory wall for resource-efficient on-device machine learning," 2022.
[13] D. Xu, M. Xu, Q. Wang, S. Wang, Y. Ma, K. Huang, G. Huang, X. Jin, and X. Liu, "Mandheling: Mixed-precision on-device dnn training with dsp offloading," *arXiv preprint arXiv:2206.07509*, 2022.
[14] S. Kim, K. Bin, S. Ha, K. Lee, and S. Chong, "ztt: learning-based dvfs with zero thermal throttling for mobile devices," *GetMobile: Mobile Computing and Communications*, vol. 25, no. 4, pp. 30–34, 2022.
[15] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 81–93.

Qiyang Zhang is a PhD candidate at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT). His research interests include mobile edge computing and ubiquitous intelligence.



Zuo Zhu received the Bachelor degree in North China University of Water Resources and Electric Power in 2022. Currently, he is a Master candidate at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT). His research interests include mobile edge computing.

Ao Zhou is an Associate Professor at the Beijing University of Posts and Telecommunication (BUPT). Her research interests include cloud computing and mobile edge computing.

Qibo Sun is an Associate Professor at the Beijing University of Posts and Telecommunication (BUPT). His research interests include cloud computing, mobile edge computing, and satellite communication technologies.

Schahram Dustdar is currently a professor of computer science with the Distributed Systems Group, Technische Universität Wien, Vienna, Austria. He was an elected member of the Academy of Europe, where he is the Chairman of the Informatics Section.

Shangguang Wang is a Professor at the School of Computing, Beijing University of Posts and Telecommunications (BUPT). He is a Vice-Director of the State Key Laboratory of Networking and Switching Technology. His research interests include service computing and mobile edge computing.