Skip2-LoRA: A Lightweight On-device DNN Fine-tuning Method for Low-cost Edge Devices

Hiroki Matsutani
Keio University
Yokohama, Japan
matutani@arc.ics.keio.ac.jp

Masaaki Kondo
Keio University
Yokohama, Japan
kondo@acsl.ics.keio.ac.jp

Kazuki Sunaga
Keio University
Yokohama, Japan
sunaga@arc.ics.keio.ac.jp

Radu Marculescu
The University of Texas at Austin
Austin, Texas, USA
radum@utexas.edu

ABSTRACT

This paper proposes Skip2-LoRA as a lightweight fine-tuning method for deep neural networks to address the gap between pre-trained and deployed models. In our approach, trainable LoRA (low-rank adaptation) adapters are inserted between the last layer and every other layer to enhance the network expressive power while keeping the backward computation cost low. This architecture is well-suited to cache intermediate computation results of the forward pass and then can skip the forward computation of seen samples as training epochs progress. We implemented the combination of the proposed architecture and cache, denoted as Skip2-LoRA, and tested it on a $15 single board computer. Our results show that Skip2-LoRA reduces the fine-tuning time by 90.0% on average compared to the counterpart that has the same number of trainable parameters while preserving the accuracy, while taking only a few seconds on the microcontroller board.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

On-device learning, Fine-tuning, DNN, Edge AI, LoRA

ACM Reference Format:
Hiroki Matsutani, Masaaki Kondo, Kazuki Sunaga, and Radu Marculescu. 2025. Skip2-LoRA: A Lightweight On-device DNN Fine-tuning Method for Low-cost Edge Devices. In *Proceedings of 30th Asia and South Pacific Design Automation Conference (ASPDAC '25).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3658617.3697589

1 INTRODUCTION

On-device learning is an emerging research direction in edge AI aiming to reduce the gap between pre-trained and deployed models. Since the available compute resources are limited in edge environments, full retraining of deep models is hardly feasible; thus, lightweight retraining methods of neural networks have been studied recently [8, 11, 15].

Such on-device learning methods can be broadly classified into 1) ELM (extreme learning machine) based retraining and 2) fine-tuning of some specific layers using a backpropagation algorithm. In the ELM-based on-device learning [11, 12], the OS-ELM (online sequential ELM) algorithm [7] is used for training the weight parameters of neural networks that have a single hidden layer. Thus, the ELM-based approach cannot be applied to DNNs (deep neural networks) that have multiple or many hidden layers; instead, the backpropagation-based approach can be used for such DNNs. A well-known method based on backpropagation fine-tunes the last layer of DNNs [9]; in this case, the backward compute cost is very small compared to the full training, but the network expressive power remains limited since only the last-layer weights can be updated. Another method freezes the weight parameters while only updating the bias modules [2]. TinyTL introduces the lite residual module as a generalized bias module to be fine-tuned [2]. All these methods update parts of the pre-train model. In addition, fine-tuning methods have been widely studied in the context of LLMs (large language models). A popular approach in LLMs is to add trainable adapters to pre-trained networks. Trainable adapter layers can be inserted in series to pre-trained networks [4], or attached to pre-trained weight matrixes in parallel [5]. LoRA (low-rank adaptation) [5] employs the latter approach; that is, trainable rank decomposition matrixes are attached to each layer of a Transformer architecture [13]. This approach is portable, meaning that only the adapters are updated while the original weights are untouched.

In this paper, we extend the LoRA-based fine-tuning methods for resource-limited edge devices. Starting from adding LoRA adapters to each layer of DNNs, we propose a lightweight fine-tuning approach called Skip2-LoRA. Our contributions are summarized as follows:

- We propose a new architecture where trainable LoRA adapters are inserted between the last layer and every other layer to enhance the network expressive power while keeping the backward compute cost low.
- This new architecture enables us to cache the intermediate compute results during the forward pass and thus skip the forward computation of seen samples as training epochs progress.

ASPDAC '25, January 20–23, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-0635-6/25/01
https://doi.org/10.1145/3658617.3697589

- Our experimental results show that Skip2-LoRA reduces the fine-tuning time by 89.0% to 92.0% compared to the baseline while achieving comparable accuracies, while taking only a few seconds on a $15 single board computer.

This paper is organized as follows. Sections 2 and 3 review preliminaries and basic knowledge of fine-tuning methods. Section 4 proposes Skip2-LoRA, and Section 5 evaluates it in terms of accuracy, execution time, and power consumption. Section 6 summarizes our contributions.

2 PRELIMINARIES

Let N and M be the input and output dimensions of an FC (fully-connected) layer, respectively, in a DNN. A forward pass of the FC layer is computed as follows:

$$\boldsymbol{y} = G(\boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b}), \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^{B \times N}$, $\boldsymbol{y} \in \mathbb{R}^{B \times M}$, $\boldsymbol{W} \in \mathbb{R}^{N \times M}$, $\boldsymbol{b} \in \mathbb{R}^{M}$, and B represent the input feature map, output feature map, weight parameters, bias parameters, and batch size, respectively.

A backward pass of the FC layer is computed as follows:

$$\boldsymbol{gW} = \boldsymbol{x}^{\mathsf{T}} \cdot \boldsymbol{gy} \tag{2}$$

$$\boldsymbol{gb} = \sum^{B} \boldsymbol{gy} \tag{3}$$

$$\boldsymbol{gx} = \boldsymbol{gy} \cdot \boldsymbol{W}^{\mathsf{T}}, \tag{4}$$

where $\boldsymbol{gx} \in \mathbb{R}^{B \times N}$, $\boldsymbol{gy} \in \mathbb{R}^{B \times M}$, $\boldsymbol{gW} \in \mathbb{R}^{N \times M}$, and $\boldsymbol{gb} \in \mathbb{R}^{M}$ represent the gradients of $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{W}$, and $\boldsymbol{b}$, respectively.

$\boldsymbol{W}$ and $\boldsymbol{b}$ are updated as follows:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \cdot \boldsymbol{gW} \tag{5}$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \eta \cdot \boldsymbol{gb}, \tag{6}$$

where η represents a learning rate.

A forward pass of a LoRA adapter of rank R for the FC layer is computed as follows:

$$\boldsymbol{y_A} = \boldsymbol{x} \cdot \boldsymbol{W_A} \tag{7}$$

$$\boldsymbol{y_B} = \boldsymbol{y_A} \cdot \boldsymbol{W_B} \tag{8}$$

$$\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{y_B}, \tag{9}$$

where $\boldsymbol{y_A} \in \mathbb{R}^{B \times R}$ and $\boldsymbol{y_B} \in \mathbb{R}^{B \times M}$ are intermediate outputs to update $\boldsymbol{y}$. $\boldsymbol{W_A} \in \mathbb{R}^{N \times R}$ and $\boldsymbol{W_B} \in \mathbb{R}^{R \times M}$ are the weight parameters of the adapter.

A backward pass of the LoRA adapter is computed as follows:

$$\boldsymbol{gW_B} = \boldsymbol{y_A}^{\mathsf{T}} \cdot \boldsymbol{gy} \tag{10}$$

$$\boldsymbol{gx_B} = \boldsymbol{gy} \cdot \boldsymbol{W_B}^{\mathsf{T}} \tag{11}$$

$$\boldsymbol{gW_A} = \boldsymbol{x}^{\mathsf{T}} \cdot \boldsymbol{gx_B} \tag{12}$$

$$\boldsymbol{gx_A} = \boldsymbol{gx_B} \cdot \boldsymbol{W_A}^{\mathsf{T}} \tag{13}$$

$$\boldsymbol{gx} \leftarrow \boldsymbol{gx} + \boldsymbol{gx_A}, \tag{14}$$

where $\boldsymbol{gx_B} \in \mathbb{R}^{B \times R}$ and $\boldsymbol{gx_A} \in \mathbb{R}^{B \times N}$ are intermediate gradients. $\boldsymbol{gW_B} \in \mathbb{R}^{R \times M}$ and $\boldsymbol{gW_A} \in \mathbb{R}^{N \times R}$ represent the gradients of $\boldsymbol{W_B}$ and $\boldsymbol{W_A}$, respectively.

$\boldsymbol{W_A}$ and $\boldsymbol{W_B}$ are updated as follows:

$$\boldsymbol{W_A} \leftarrow \boldsymbol{W_A} - \eta \cdot \boldsymbol{gW_A} \tag{15}$$

$$\boldsymbol{W_B} \leftarrow \boldsymbol{W_B} - \eta \cdot \boldsymbol{gW_B}. \tag{16}$$

Table 1: Compute types of FC layers and LoRA adapters.

FC_y	Compute $\boldsymbol{y}$
FC_{ywbx}	Compute $\boldsymbol{y}$, $\boldsymbol{gW}$, $\boldsymbol{gb}$, and $\boldsymbol{gx}$
FC_{ywb}	Compute $\boldsymbol{y}$, $\boldsymbol{gW}$, and $\boldsymbol{gb}$
FC_{ybx}	Compute $\boldsymbol{y}$, $\boldsymbol{gb}$, and $\boldsymbol{gx}$
FC_{yb}	Compute $\boldsymbol{y}$ and $\boldsymbol{gb}$
FC_{yx}	Compute $\boldsymbol{y}$ and $\boldsymbol{gx}$
$LoRA_{ywx}$	Compute $\boldsymbol{y_A}$, $\boldsymbol{y_B}$, $\boldsymbol{gW_B}$, $\boldsymbol{gW_A}$, $\boldsymbol{gx_B}$, and $\boldsymbol{gx_A}$
$LoRA_{yw}$	Compute $\boldsymbol{y_A}$, $\boldsymbol{y_B}$, $\boldsymbol{gW_B}$, $\boldsymbol{gW_A}$, and $\boldsymbol{gx_B}$

3 BASELINE FINE-TUNING METHODS

A forward pass of an FC layer computes $\boldsymbol{y}$, while the backward pass computes $\boldsymbol{gW}$, $\boldsymbol{gb}$, and $\boldsymbol{gx}$. In fine-tuning scenarios, not all are necessary; for example, $\boldsymbol{gW}$ and $\boldsymbol{gb}$ of an FC layer are not necessary when weight and bias parameters of the layer are not updated. Compute types of FC layers are classified as listed in the upper half of Table 1. The number of floating-point operations and memory size can be modeled for each compute type, but they are omitted due to the page limitation.

As basic fine-tuning methods, in this paper, FT-All, FT-Last, and FT-Bias are defined as follows:

- FT-All: Weight and bias parameters of all layers are updated.
- FT-Last: Weight and bias parameters of the last layer are updated.
- FT-Bias: Bias parameters of all layers are updated.

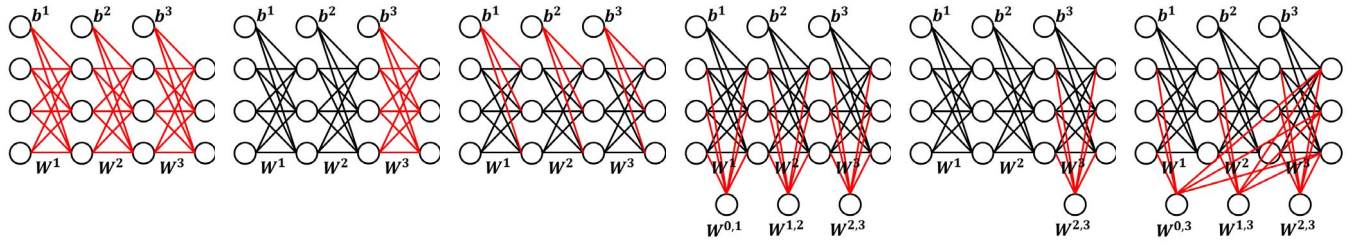
Figures 1(a), 1(b), and 1(c) illustrate FT-All, FT-Last, and FT-Bias methods, respectively, for DNNs consisting of three layers. In these figures, the parameters to be updated are colored in red. The compute types of the first, second, and third FC layers in FT-All are $\{FC_{ywb}, FC_{ywbx}, FC_{ywbx}\}$. Those in FT-Last are $\{FC_y, FC_y, FC_{ywb}\}$, and those in FT-Bias are $\{FC_{yb}, FC_{ybx}, FC_{ybx}\}$. In the first layer, $\boldsymbol{gx}$ is not propagated any more and thus can be omitted.

A forward pass of a LoRA adapter computes $\boldsymbol{y_A}$ and $\boldsymbol{y_B}$, while the backward pass computes $\boldsymbol{gW_B}$, $\boldsymbol{gW_A}$, $\boldsymbol{gx_B}$, and $\boldsymbol{gx_A}$. Compute types of LoRA adapters are classified as listed in the lower half of Table 1. The compute and memory cost model for each compute type is omitted in this paper.

As fine-tuning methods of LoRA, LoRA-All and LoRA-Last are defined as follows:

- LoRA-All: LoRA adapters are added to all layers.
- LoRA-Last: A LoRA adapter is added to the last layer.

Figures 1(d) and 1(e) illustrate LoRA-All and LoRA-Last methods, respectively, for DNNs consisting of three layers, where the parameters to be updated are colored in red. The compute types of the first, second, and third LoRA adapters in LoRA-All are $\{LoRA_{yw}, LoRA_{ywx}, LoRA_{ywx}\}$, and those of the FC layers are $\{FC_y, FC_{yx}, FC_{yx}\}$. Similarly, the compute types of the first, second, and third LoRA adapters in LoRA-Last are $\{\phi, \phi, LoRA_{yw}\}$, and those of the FC layers are $\{FC_y, FC_y, FC_y\}$. The backward compute cost of LoRA-Last is thus much smaller than that of LoRA-All. On the other hand, LoRA-All introduces LoRA adapters to all the layers, while LoRA-Last introduces only a single LoRA adapter to the last layer; thus, LoRA-All has a higher expressive power than LoRA-Last.



(a) FT-All (b) FT-Last (c) FT-Bias (d) LoRA-All (e) LoRA-Last (f) Skip-LoRA

Figure 1: Fine-tuning methods of DNNs consisting of n FC layers, where $n = 3$. W^k and b^k denote weights and biases for k-th layer. In LoRA-All and LoRA-Last, $W^{k-1,k}$ denotes weights for k-th LoRA adapter, where rank $R = 1$. Parameters to be updated are colored in red.

Table 2: Execution times breakdown of FT-All-LoRA on two datasets.

Forward	Fan	HAR	Backward	Fan	HAR
FC1	71.80	88.58	FC3	1.28	1.22
LoRA1	2.75	1.72	LoRA3	1.93	1.05
BN1	2.22	0.81	Act2	0.29	0.16
Act1	0.30	0.11	BN2	2.81	1.55
FC2	17.52	6.63	FC2	34.03	18.29
LoRA2	1.69	0.61	LoRA2	3.30	1.78
BN2	2.23	0.81	Act1	0.29	0.15
Act2	0.30	0.11	BN1	2.84	1.53
FC3	0.50	0.36	FC1	49.47	70.46
LoRA3	0.68	0.25	LoRA1	3.76	3.80
Total (%)	100.00	100.00	Total (%)	100.00	100.00

3.1 Performance Analysis

In this section, the compute costs of fine-tuning methods are analyzed. To analyze the compute costs, "FT-All-LoRA" is defined as a full fine-tuning method that combines FT-All and LoRA-All. We assume a simple 3-layer DNN that consists of FC layer (FC1), LoRA adapter (LoRA1), batch normalization [6] (BN1), ReLU (Act1), FC2, LoRA2, BN2, Act2, FC3, LoRA3, and cross entropy loss (CEL) function. Table 2 shows the execution times breakdown of the forward and backward passes without CEL. Two datasets, Fan and HAR which will be explained in Section 5.1, are examined. As shown, the first and second FC layers consume most of the compute costs. To reduce these compute costs, Skip-LoRA and Skip2-LoRA are proposed in the next section.

4 DESIGN AND IMPLEMENTATION OF SKIP2-LORA

4.1 Proposed Architecture: Skip-LoRA

Our first proposal is "Skip-LoRA" which aims to achieve a comparable expressive power to LoRA-All, yet with a comparable backward compute cost to LoRA-Last. Skip-LoRA is defined as follows:

- Skip-LoRA: LoRA adapters are added between output nodes of the last layer and input nodes of the other layers.

As shown in Figures 1(d) and 1(e), weight parameters of a LoRA adapter for the k-th layer are denoted as $\boldsymbol{W}^{\boldsymbol{k-1,k}}$. For a DNN consisting of n layers, additional weight parameters of LoRA-Last and LoRA-All are denoted as $\boldsymbol{W}^{\boldsymbol{n-1,n}}$ and $\sum_{k=1}^{n} \boldsymbol{W}^{\boldsymbol{k-1,k}}$, respectively. On the other hand, those of Skip-LoRA are denoted as $\sum_{k=1}^{n} \boldsymbol{W}^{\boldsymbol{k-1,n}}$. In this case, the forward pass of all the FC layers is computed normally with Equation 1. Then, the forward pass of n LoRA adapters is computed, and the results are added to the output feature map of the n-th FC layer as follows:

$$\boldsymbol{y}^{\boldsymbol{n}} \leftarrow \boldsymbol{y}^{\boldsymbol{n}} + \sum_{k=1}^{n} \boldsymbol{x}^{\boldsymbol{k}} \cdot \boldsymbol{W}_{\boldsymbol{A}}^{\boldsymbol{k-1,n}} \cdot \boldsymbol{W}_{\boldsymbol{B}}^{\boldsymbol{k-1,n}}, \tag{17}$$

where $\boldsymbol{x}^{\boldsymbol{k}}$ and $\boldsymbol{y}^{\boldsymbol{k}}$ are the input and output feature maps of the k-th FC layer.

The compute types of the first, second, and third LoRA adapters in Skip-LoRA are $\{LoRA_{yw}, LoRA_{yw}, LoRA_{yw}\}$, and those of the FC layers are $\{FC_y, FC_y, FC_y\}$. The compute types of FC layers of Skip-LoRA are identical to those of LoRA-Last, while the compute types of LoRA adapters are more complicated in Skip-LoRA. Please note that a LoRA adapter is a low-rank approximation of an FC layer; thus, we can expect $R << N, M$. In this case, the computation cost of the FC layers is dominant compared to that of LoRA adapters, as demonstrated in Table 2. We can thus expect that the backward compute cost of Skip-LoRA is close to that of LoRA-Last while Skip-LoRA has n LoRA adapters to enhance the expressive power compared to LoRA-Last.

4.2 Proposed Cache: Skip-Cache

Skip-LoRA can reduce the backward compute cost, as well as LoRA-Last. The next bottleneck is the forward compute cost. Here, we aim to reduce the forward compute cost by reusing the forward compute results which have been already computed.

Let E be the number of fine-tuning epochs. In the stochastic gradient descent, it is expected that the same training sample appears E times on average during a fine-tuning process. Assume we have a set of training samples $\boldsymbol{T}$ for the fine-tuning. Let $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}} \in \mathbb{R}^N$ and $\boldsymbol{y}_{\boldsymbol{i}}^{\boldsymbol{k}} \in \mathbb{R}^M$ be the input and output feature maps of the k-th FC layer for the i-th training sample, where $0 \leq i < |\boldsymbol{T}|$. $\boldsymbol{y}_{\boldsymbol{i}}^{\boldsymbol{k}}$ is computed and the result is cached as $\boldsymbol{c}_{\boldsymbol{i}}^{\boldsymbol{k}}$ when the i-th sample appears

Algorithm 1 Fine-tuning with Skip2-LoRA

```
1:  function FT_SKIP2_LORA
2:      C_skip ← φ                                  ▷ Initialize C_skip
3:      for e = 0 to E − 1 do
4:          for b = 0 to |T|/B − 1 do
5:              load_train_batch(B)
6:              forward_fc(C_skip)                  ▷ Forward with C_skip
7:              add_cache(C_skip)                   ▷ Add results to C_skip
8:              forward_lora()
9:              backward_lora()
10:             update_lora_weight()
```

Algorithm 2 FC forward with Skip-Cache

```
1:  function FORWARD_SINGLE_FC(C_skip)
2:      for i = 0 to B − 1 do
3:          if x_i ∈ C_skip then                    ▷ If result y_i is cached
4:              continue
5:          for m = 0 to M − 1 do
6:              y_{i,m} ← b_m
7:              for n = 0 to N − 1 do
8:                  y_{i,m} ← y_{i,m} + x_{i,n} · W_{n,m}   ▷ Scalar MAC
9:      return y
```

at the first time, while $\boldsymbol{c_i^k}$ is reused when the i-th sample appears again during the fine-tuning process. This approach is denoted as "Skip-Cache" in this paper.

Skip-Cache works well if the cached result $\boldsymbol{c_i^k}$ is valid throughout the fine-tuning process over E epochs. Conversely, Skip-Cache does not work well for FT-All, FT-Bias, and LoRA-All as illustrated below:

- FT-All: $\boldsymbol{W^k}$ and $\boldsymbol{b^k}$ are updated every fine-tuning batch, where $1 \leq k \leq n$, obsoleting the cached results frequently.
- FT-Bias: $\boldsymbol{b^k}$ are updated every fine-tuning batch, where $1 \leq k \leq n$.
- LoRA-All: $\boldsymbol{W_A^{k-1,k}}$ and $\boldsymbol{W_B^{k-1,k}}$ are updated every fine-tuning batch, where $1 \leq k \leq n$.

Obviously, Skip-Cache works well for FT-Last, LoRA-Last, and Skip-LoRA, except for the last FC layer because:

(1) Their output feature maps except for the last layer can be computed normally with Equation 1, and
(2) Their parameters (e.g., $\boldsymbol{W^k}$ and $\boldsymbol{b^k}$) are not changed throughout the fine-tuning process over E epochs, where $1 \leq k < n$.

Please note that a special treatment is needed only for the last layer (i.e., $k = n$) as illustrated below:

- FT-Last: The output feature map of the last layer (i.e., $\boldsymbol{y_i^n}$) cannot be reused because $\boldsymbol{W^n}$ and $\boldsymbol{b^n}$ are updated every fine-tuning batch.
- LoRA-Last: The result of $G(\boldsymbol{x_i^n} \cdot \boldsymbol{W^n} + \boldsymbol{b^n})$ can be reused as $\boldsymbol{c_i^n}$; then, $\boldsymbol{y_i^n} \leftarrow \boldsymbol{c_i^n} + \boldsymbol{x_i^n} \cdot \boldsymbol{W_A^{n-1,n}} \cdot \boldsymbol{W_B^{n-1,n}}$ is recomputed because $\boldsymbol{W_A^{n-1,n}}$ and $\boldsymbol{W_B^{n-1,n}}$ are updated every fine-tuning batch.
- Skip-LoRA: $\boldsymbol{c_i^n}$ can be reused as well as LoRA-Last; then, $\boldsymbol{y_i^n} \leftarrow \boldsymbol{c_i^n} + \sum_{k=1}^{n} \boldsymbol{x_i^k} \cdot \boldsymbol{W_A^{k-1,n}} \cdot \boldsymbol{W_B^{k-1,n}}$ is recomputed because weight parameters of all the n LoRA adapters (i.e., $\forall \boldsymbol{k}, \boldsymbol{W_A^{k-1,n}}$ and $\forall \boldsymbol{k}, \boldsymbol{W_B^{k-1,n}}$ where $1 \leq k \leq n$) are updated every fine-tuning batch.

In this paper, the combination of Skip-LoRA and Skip-Cache is denoted as "Skip2-LoRA".

4.3 Implementation of Skip2-LoRA

Skip2-LoRA is implemented with the C language without any external libraries except for libm ("-lm" option). Algorithm 1 shows the fine-tuning with Skip2-LoRA algorithm. $\boldsymbol{T}$, $|\boldsymbol{T}|$, E, and B are the training samples for fine-tuning, the number of training samples, the number of epochs, and the batch size, respectively.

In line 2, Skip-Cache $\boldsymbol{C_{skip}}$ is initialized. In line 5, a batch of training samples is randomly selected from $\boldsymbol{T}$. In line 6, the forward pass of all the FC layers is computed for the batch. During this computation, $\boldsymbol{C_{skip}}$ is examined and unnecessary computation is skipped. Algorithm 2 shows the forward pass of a single FC layer with $\boldsymbol{C_{skip}}$. This is a typical matrix multiplication algorithm that computes Equation 1 except that $\boldsymbol{C_{skip}}$ is introduced. Let $\boldsymbol{x_i} \in \mathbb{R}^N$ be a training sample in the batch. If its compute result $\boldsymbol{y_i} \in \mathbb{R}^M$ has been cached in $\boldsymbol{C_{skip}}$, the computation is skipped so that we can reduce the compute cost of the forward pass (lines 3-4 of Algorithm 2). After completing the forward pass of all the FC layers, newly computed results are added to $\boldsymbol{C_{skip}}$ as shown in line 7 of Algorithm 1. In line 8, Equation 17 is computed. In lines 9-10, weight parameters of n LoRA adapters are updated.

As shown in Algorithm 1, $\boldsymbol{C_{skip}}$ is initialized at the beginning of the fine-tuning (line 2), and then the compute results are continuously added to $\boldsymbol{C_{skip}}$ (line 7) throughout the fine-tuning process over E epochs. Since each training sample appears E times on average during a fine-tuning process, it is expected that the forward compute cost is reduced to $1/E$.

Data structure of $\boldsymbol{C_{skip}}$ affects the cache hit rate, cache manipulation overhead, and storage size. The forward computation for the i-th training sample can be skipped when the compute results $\forall \boldsymbol{k}, \boldsymbol{y_i^k}$ where $1 \leq k \leq n$ are cached in $\boldsymbol{C_{skip}}$ [1]. In the Fan dataset which will be explained in Section 5.1, for example, the number of fine-tuning samples is 470, each containing 256 features in float32; in this case, the fine-tuning data size is 470KiB. Assuming a DNN consisting of three layers (e.g., 256-96-96-3), the size of $\boldsymbol{C_{skip}}$ to store $\forall \boldsymbol{i} \forall \boldsymbol{k}, \boldsymbol{y_i^k}$ where $1 \leq k \leq n$ is only 358KiB, which is smaller than the input data storage. Thus, in this paper, $\forall \boldsymbol{i} \forall \boldsymbol{k}, \boldsymbol{y_i^k}$ is fully stored in $\boldsymbol{C_{skip}}$. Since $\forall \boldsymbol{k}, \boldsymbol{y_i^k}$ is stored exclusively in the i-th element of $\boldsymbol{C_{skip}}$, the time complexity to find the cached results of the i-th training sample is $O(1)$. Alternatively, if the storage size is strictly limited, a key-value cache with a limited number of cache entries can be used. In any cases, there is a trade-off between the cache size and performance.

[1] In reality, the activation function and batch normalization are typically executed after each layer as in Table 2. In this case, the outputs after these functions should be cached except for the last layer (i.e., the outputs just after the FC layer should be cached in the case of the last layer).

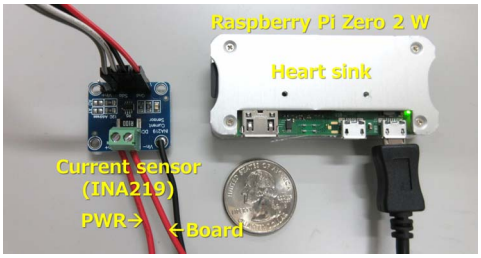


Figure 2: Evaluation environment consisting of Raspberry Pi Zero 2 W.

Table 3: Accuracy of before and after data drift on 3-layer DNN (%).

	Before	After
Damage1	60.61±13.73	98.99±2.81
Damage2	51.86±8.04	90.88±5.65
HAR	79.97±5.62	86.09±4.40

5 EVALUATIONS

The proposed Skip2-LoRA is compared with the counterparts in terms of accuracy and execution time using drifted datasets. It is also compared with the state-of-the-art method.

5.1 Evaluation Setup

FT-All, FT-Last, FT-Bias, FT-All-LoRA, LoRA-All, LoRA-Last, Skip-LoRA, and Skip2-LoRA are executed on a Raspberry Pi Zero 2 W board [1], which is known as a $15 computer (Figure 2). The clock frequency is fixed at 1GHz to measure the execution time stably. Skip2-LoRA and its counterparts (except for TinyTL [2]) are implemented with the C language and compiled with gcc version 8.3.0 with "-O3" option on the platform. They are further optimized with SIMD (Neon) instructions with "-mfpu=neon -ffast-math" option.

To evaluate the fine-tuning methods, we use three datasets each containing pre-train samples, fine-tune samples, and test samples as follows:

- Damage1 is a 3-class classification task of vibration patterns of cooling fans (i.e., stop, normal fan, and damaged fan with holes on a blade). Both the normal and damaged fans rotate at 1,500, 2,000, and 2,500 rpm. The numbers of input features and output classes are 256 and 3. The original dataset [11] contains vibration patterns in a silent office and those near a ventilation fan (they are denoted as "silent dataset" and "noisy dataset"). The model is pre-trained with the silent dataset. We assume the model is deployed in a "real" noisy environment. Thus, the model is fine-tuned with a half of the noisy dataset and then tested with the remaining half of the noisy dataset. The numbers of pre-train, fine-tune, and test samples are 470, 470, and 470.
- Damage2 is similar to the Damage1 dataset, but using a damaged fan with a chipped blade.
- HAR is a 6-class classification task of human activity recognition (i.e., walking, walking upstairs, walking downstairs, sitting, standing, and laying). The numbers of input features and output classes are 561 and 6. The original dataset [10] contains sensor data from 30 human subjects. We manually removed those of subjects 9, 14, 16, 19, and 25 from the original dataset and saved as "initial dataset". Those of subjects 9, 14, 16, 19, and 25 were saved as "drifted dataset". The model is pre-trained with the initial dataset. We assume the model is used for a group of different subjects. Thus, the model is fine-tuned with a half of the drifted dataset and then tested with the remaining half of the drifted dataset. The numbers of pre-train, fine-tune, and test samples are 5,894, 1,050, and 694.

We use a simple 3-layer DNN shown in Figure 1. The numbers of input and output nodes are 256 and 3 for the Damage1 and Damage2 datasets. They are 561 and 6 for the HAR dataset. The number of hidden nodes is 96 in all the hidden layers. The LoRA rank is set to 4. Batch normalization and ReLU are also executed as in Table 2.

5.2 Accuracy

Table 3 shows the baseline accuracy of the three datasets without fine-tuning on the 3-layer DNNs. In the "Before" case, the model is trained with the pre-train dataset and then tested with the test dataset. In the "After" case, the model is trained only with the fine-tune dataset and then tested with the test dataset. In each case, the number of training epochs is set to a large enough value (i.e., $E = 400$ and 900 for the Damage1/Damage2 and HAR datasets). Table 3 shows mean accuracy values over 20 trials. The accuracy is quite low in the Before case while it is significantly better in the After case. There is a significant accuracy gap between before and after the data drift; in this case, we can fill out the gap by the on-device fine-tuning as demonstrated below.

Table 4 shows the accuracies of FT-All, FT-Last, FT-Bias, FT-All-LoRA, LoRA-All, Skip-LoRA, and Skip2-LoRA on the 3-layer DNNs. The test is conducted with the following three steps. In each case, the number of training epochs is set to a large enough value.

(1) The model is trained with the pre-train samples ($E = 100$ and 300 for the Damage1/Damage2 and HAR datasets).
(2) The model is fine-tuned with the fine-tune samples ($E = 300$ and 600 for the Damage1/Damage2 and HAR datasets).
(3) The model is tested with the test samples.

Mean accuracies over 20 trials are reported in this table. In all cases, Skip2-LoRA shows almost the same accuracy as Skip-LoRA. Note the difference between FT-All and the "After" case in Table 3 is that FT-All is trained with both the pre-train and fine-tune datasets. For the Damage1 dataset, Skip2-LoRA achieves a higher accuracy than FT-Last, FT-Bias, and LoRA-Last, demonstrating a higher expressive power than these counterparts. However, its accuracy is lower than FT-All, FT-All-LoRA, and LoRA-All. For the Damage2 dataset, on the other hand, Skip2-LoRA shows a higher accuracy than FT-All, FT-All-LoRA, and LoRA-All. We expect that these counterparts cause overfitting to the "after drift" fine-tune dataset and thus they show a lower accuracy than Skip2-LoRA. For the HAR dataset, Skip-LoRA and Skip2-LoRA show the highest accuracies followed by LoRA-All, FT-All, FT-All-LoRA, LoRA-Last, FT-Last, and FT-Bias.

Table 4: Accuracy of proposed and counterpart fine-tuning methods on 3-layer DNN (%).

	FT-All	FT-Last	FT-Bias	FT-All-LoRA	LoRA-All	LoRA-Last	Skip-LoRA	Skip2-LoRA
Damage1	98.73±2.11	94.19±2.24	79.42±7.50	98.63±2.14	98.26±1.32	94.67±2.92	96.07±2.14	96.19±2.29
Damage2	88.12±6.13	92.43±3.67	79.56±6.47	88.88±5.73	86.45±4.90	93.55±3.50	93.24±3.86	93.46±3.21
HAR	90.99±1.86	89.31±1.06	82.21±1.27	90.40±2.49	91.09±1.26	89.79±1.46	92.10±1.05	91.99±1.00

Table 5: Accuracy of state-of-the-art fine-tuning methods [2] on ProxylessNAS [3] (%).

	TinyTL (GN)	TinyTL (BN)
Damage1	98.66±0.76	99.49±0.32
Damage2	92.09±3.17	96.01±2.74
HAR	88.76±0.91	89.27±1.13

Table 5 shows the accuracies of TinyTL [2] as a state-of-the-art fine-tuning method, where "GN" and "BN" mean the group normalization [14] and batch normalization. TinyTL uses GN [2], while its BN version is also tested. The number of trials is 20 in each case. The accuracy of Skip2-LoRA is not higher than that of TinyTL in the Damage1 dataset, while Skip2-LoRA outperforms TinyTL in the HAR dataset. Please note that the backbone network of TinyTL is ProxylessNAS [3] while ours use much simpler 3-layer DNNs.

5.3 Execution Time

Tables 6 and 7 show the execution times of FT-All, FT-Last, FT-Bias, FT-All-LoRA, LoRA-All, LoRA-Last, Skip-LoRA, and Skip2-LoRA on a Raspberry Pi Zero 2 W with Neon instructions. The results on the Damage1 and Damage2 datasets are the same and thus reported as "Fan" dataset in Table 6. The training batch size B is set to 20.

As shown in these tables, the training time of a batch consists of the forward pass, backward pass, and weight update. The execution times are mean values over the entire fine-tuning process, where the number of epochs E is the same as that in Section 5.2. Although Skip-LoRA and LoRA-All have the same number of trainable parameters, Skip-LoRA reduces the execution time of backward pass by 82.5% to 88.3% compared to LoRA-All, demonstrating benefits of the proposed Skip-LoRA architecture. In addition, Skip2-LoRA reduces the execution time of forward pass by 89.0% to 93.5% compared to Skip-LoRA, demonstrating benefits of the proposed Skip-Cache. As a result, Skip2-LoRA reduces the training time by 89.0% to 92.0% (90.0% on average) compared to LoRA-All that has the same number of trainable parameters. The training times are only 0.450msec and 0.595msec per batch in the Fan and HAR datasets, respectively.

In Skip2-LoRA, the training time is affected the number of epochs E, because a larger E can skip more forward pass computations. As mentioned in Section 5.2, E was set to a large enough value. Here, we estimate actual training time based on practical E. Figure 3 shows the training curves of Skip2-LoRA with the three datasets. X-axis shows the number of trained epochs, and Y-axis shows the test accuracy. Mean accuracies over 10 trials are reported in these graphs. Here, the number of required epochs in which the test accuracy first reaches within a 1% range of the reported accuracies in Table 4 is denoted as "required epochs". The required number of epochs are 100, 60, and 200 in the Damage1, Damage2, and HAR datasets, respectively. The number of their fine-tuning samples are 470, 470, and 1050; thus, the total fine-tuning times of Skip2-LoRA on a Raspberry Pi Zero 2 W are only 1.06sec, 0.64sec, and 2.79sec in the Damage1, Damage2, and HAR datasets, respectively.

5.4 Power Consumption

Skip2-LoRA with the HAR dataset ($E = 200$) is executed on a Raspberry Pi Zero 2 W and the power consumption is measured with a current sensor INA219 (Figure 2). Figure 4 shows the variation of power consumption and temperature, where the fine-tuning starts at 9sec. Once the fine-tuning starts, the clock frequency increases from 600MHz to 1GHz and the power consumption increases. Although the net compute time for the forward and backward passes is 2.79sec as mentioned above, the results in Figure 4 include overheads for reading the dataset and loading the pre-trained weight parameters. The power consumption is at most 1,455mW for a short duration and the temperature does not exceed 44.5 deg C.

6 SUMMARY

In this paper, we extended LoRA adapters as a new lightweight on-device fine-tuning mehtod for resource-limited edge devices. The proposed Skip2-LoRA synergistically combines Skip-LoRA architecture to reduce the backward compute cost and Skip-Cache to reduce the forward compute cost. Experimental results using three drifted datasets demonstrated that Skip2-LoRA reduces the fine-tuning time by 90.0% on average compared to LoRA-All that has the same number of trainable parameters while achieving comparable accuracies to the state-of-the-art method. The order of magnitude reduction of the compute cost enables a few seconds "quick" fine-tuning of DNNs on a Raspberry Pi Zero 2 W board with modest power and temperature.

Acknowledgements H.M. was supported in part by JST AIP Acceleration Research JPMJCR23U3, Japan.

REFERENCES

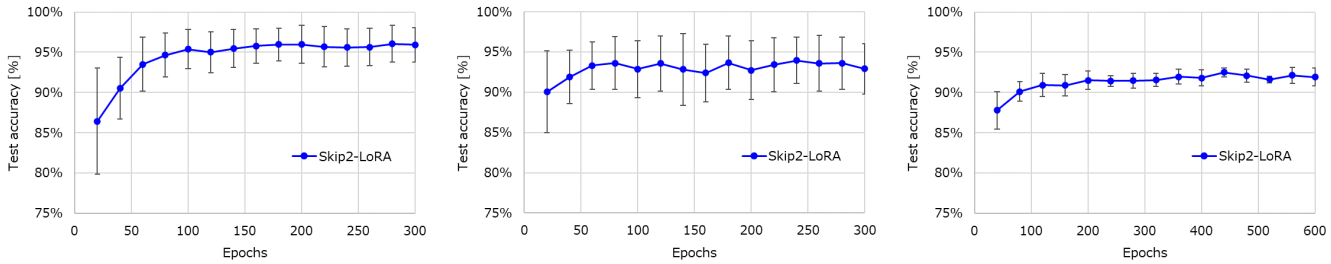
[1] [n. d.]. Raspberry Pi Zero 2 W. https://www.raspberrypi.com/products/raspberry-pi-zero-2-w.
[2] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. In *Proc. of the Annual Conference on Neural Information Processing Systems*. 11285–11297.
[3] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *Proc. of the International Conference on Learning Representations*.
[4] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proc. of the International Conference on Machine Learning*. 2790–2799.
[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.

Table 6: Execution time on Raspberry Pi Zero 2 W with Neon instructions for Fan dataset (msec).

	FT-All	FT-Last	FT-Bias	FT-All-LoRA	LoRA-All	LoRA-Last	Skip-LoRA	Skip2-LoRA
Train@batch	5.864	2.633	3.721	6.053	4.113	2.642	2.952	**0.450**
forward	2.812	2.601	2.832	2.868	2.942	2.613	2.807	0.309
backward	2.866	0.030	0.885	2.993	1.157	0.026	0.136	0.131
weight update	0.186	0.002	0.003	0.192	0.014	0.002	0.010	0.010
Predict@sample	0.142	0.144	0.148	0.150	0.155	0.143	0.151	0.154

Table 7: Execution time on Raspberry Pi Zero 2 W with Neon instructions for HAR dataset (msec).

	FT-All	FT-Last	FT-Bias	FT-All-LoRA	LoRA-All	LoRA-Last	Skip-LoRA	Skip2-LoRA
Train@batch	11.323	6.179	6.795	11.577	7.459	6.031	6.328	**0.595**
forward	6.569	6.129	6.050	6.660	6.390	6.005	6.130	0.396
backward	4.373	0.047	0.742	4.480	1.052	0.024	0.184	0.185
weight update	0.381	0.003	0.003	0.437	0.017	0.002	0.014	0.014
Predict@sample	0.308	0.307	0.304	0.317	0.314	0.309	0.314	0.317



(a) Damage1 (b) Damage2 (c) HAR

Figure 3: Training curves of Skip2-LoRA on three datasets. Required epochs are 100, 60, and 200 in Damage1, Damage2, and HAR datasets.

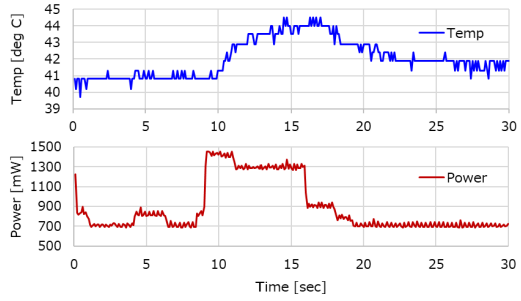


Figure 4: Power consumption and temperature of Skip2-LoRA with HAR dataset. Fine-tuning starts at 9sec.

[6] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. of the International Conference on Machine Learning*. 448–456.

[7] Nan-ying Liang, Guang-bin Huang, P. Saratchandran, and N. Sundararajan. 2006. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks* 17, 6 (2006), 1411–1423.

[8] Davide Nadalini, Manuele Rusci, Luca Benini, and Francesco Conti. 2023. Reduced Precision Floating-Point Optimization for Deep Neural Network On-Device Learning on Microcontrollers. *Future Generation Computer Systems* 149 (2023), 212–226.

[9] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. In *Proc. of the International Joint Conference on Neural Networks*. 1–8.

[10] Jorge Reyes-Ortiz, Davide Anguita, Alessandro Ghio, Luca Oneto, and Xavier Parra. 2012. Human Activity Recognition Using Smartphones. UCI Machine Learning Repository.

[11] Kazuki Sunaga, Masaaki Kondo, and Hiroki Matsutani. 2023. Addressing Gap between Training Data and Deployed Environment by On-Device Learning. *IEEE Micro* 43, 6 (2023), 66–73.

[12] Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. 2020. A Neural Network-Based On-device Learning Anomaly Detector for Edge Devices. *IEEE Trans. Comput.* 69, 7 (2020), 1027–1044.

[13] 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 *Proc. of the Annual Conference on Neural Information Processing Systems*. 6000–6010.

[14] Yuxin Wu and Kaiming He. 2018. Group Normalization. In *Proc. of the European Conference on Computer Vision*. 3–19.

[15] Ligeng Zhu, Lanxiang Hu, Ji Lin, Wei-Chen Wang, Wei-Ming Chen, and Song Han. 2023. PockEngine: Sparse and Efficient Fine-tuning in a Pocket. In *Proc. of the International Symposium on Microarchitecture*. 1381–1394.