TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 08/18 pp82–92
DOI: 10.26599/TST.2021.9010075
Volume 28, Number 1, February 2023

Thermal-Aware on-Device Inference Using Single-Layer Parallelization with Heterogeneous Processors

Jinghui Zhang*, Yuchen Wang, Tianyu Huang, Fang Dong*, Wei Zhao, and Dian Shen

Abstract: Numerous neural network (NN) applications are now being deployed to mobile devices. These applications usually have large amounts of calculation and data while requiring low inference latency, which poses challenges to the computing ability of mobile devices. Moreover, devices' life and performance depend on temperature. Hence, in many scenarios, such as industrial production and automotive systems, where the environmental temperatures are usually high, it is important to control devices' temperatures to maintain steady operations. In this paper, we propose a thermal-aware channel-wise heterogeneous NN inference algorithm. It contains two parts, the thermal-aware dynamic frequency (TADF) algorithm and the heterogeneous-processor single-layer workload distribution (HSWD) algorithm. Depending on a mobile device's architecture characteristics and environmental temperature, TADF can adjust the appropriate running speed of the central processing unit and graphics processing unit , and then the workload of each layer in the NN model is distributed by HSWD in line with each processor's running speed and the characteristics of the layers as well as heterogeneous processors. The experimental results, where representative NNs and mobile devices were used, show that the proposed method can considerably improve the speed of the on-device inference by 21%–43% over the traditional inference method.

Key words: neural network inference; mobile device; temperature adjustment; channel-wise parallelization

1 Introduction

Recent years have witnessed the rapid progress in neural networks (NNs)[1], which have made many new services become a reality. Meanwhile, with the massive use of mobile devices, more NN services are now deployed in mobile devices, such as mobile phones and smart cameras. With the help of virtual assistants, such as Google Assistant[2] and Apple Siri[3], users can convert words into commands. Computer vision[4] applications could recognize pictures and classify them into different categories. Furthermore, NN services are widely used in industrial production, with images and vibration signals collected by intelligent sensors, and functions, such as object surface detection and system fault diagnosis, can be deployed without manual operation[5]. Running NN inference efficiently on mobile devices has become significant with such widespread applications, and the diversity of application scenarios has raised various requirements to the service deployment[6, 7]. Traditionally, mobile devices use sensors, such as cameras and radars, to collect surrounding information, and then perform inference tasks. To be concrete, the computation-intensive parts could be sent to the cloud center (Fig. 1a) or partially processed on local device and then the cloud center process the intermediate data (Fig. 1b). It is possible that some mobile devices with

• Jinghui Zhang, Yuchen Wang, Tianyu Huang, and Fang Dong are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. E-mail: {jhzhang, yuchen_seu, tyhuang, fdong}@seu.edu.cn.
• Wei Zhao is with the Hefei Comprehensive National Science Center, Hefei 231299, China, and also with the School of Computer Science and Technology, Anhui University of Technology, Hefei 230026, China. E-mail: zhaoweistuart@gmail.com.
• Dian Shen is with the Department of Computer Science & Engineering, The Chinese University of Hong Kong, Hong Kong 999077, China, and also with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. E-mail: dianshen@cuhk.edu.hk.
* To whom correspondence should be addressed.
Manuscript received: 2021-10-01; accepted: 2021-10-13

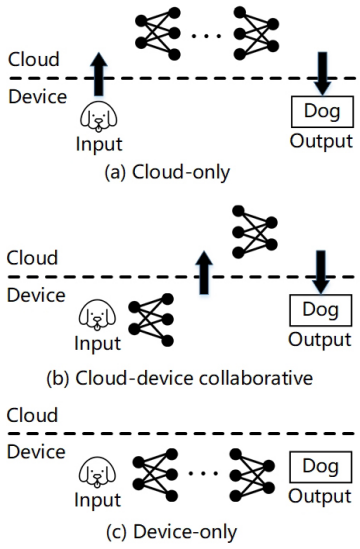


Fig. 1 An illustration of how the computation-intensive inference tasks are performed in three existing inference methods.

accelerators available for the application to perform the complete inference tasks locally (Fig. 1c). To meet the high responsiveness requirement of mobile services, the cloud center with abundant computing resources runs all or part of the inference task and transmits the results to the mobile device. However, sending these messages to the cloud side might lead to privacy disclosure, which would affect users' safety. Although the cloud centers could quickly perform computing tasks, the performance of such a mode is largely affected by the bandwidth. When the wireless network delay is high, it will take a long time to transmit data between the mobile device and the cloud computing center, which could lead to catastrophic consequences. In the case of high-performance mobile system-on-chips (SoCs), current mobile devices are usually equipped with multiprocessors, such as the central processing unit (CPU) and graphics processing unit (GPU), which allows employing on-device inference and executes the entire NN model on mobile devices with their builtin processors.

However, as current mobile devices are more energy efficient, the single processor on them is usually not powerful enough and cannot meet the delay requirement. Therefore, using multiprocessors on mobile devices to cooperatively accelerate the NN inference is a potential option. Particularly, although numerous parallelism algorithms, such as data parallelism[8] and model parallelism[9], have conducted considerable work on parallelism for NN models, little work has focused on parallelizing NN models with heterogeneous processors in one device, especially when there is only single NN layer.

In addition, as NN services are deployed in a wide variety of scenes, many constraints besides latency should be considered. Taking the industrial production scenario as an example, image recognition applications deployed on smart cameras are often used in these scenarios to check product quality. However, as the ambient temperature is usually high in such an environment, devices can be easily overheated, consequently causing degraded performance and even shortened device life. Hence, it is important to control the service power consumption and reduce the device temperature. In this study, considering a scenario where the working temperature of the device has a limit, we proposed an algorithm to accelerate the NN inference with heterogeneous processors on one mobile device. First, we proposed a dynamic power model to capture the effect of the inference execution and the ambient temperature on the device's temperature. Second, we classified the layers in the NNs according to the characteristics of the layers. Then, we distributed the workload of the layer to heterogeneous processors according to the computing characteristics of the layers and the computing ability of different processors on the device. Finally, we built our thermal-aware heterogeneous inference model to set the workload of each layer and the running speed of the heterogeneous processors.

The remainder of this paper is organized as follows. Section 2 discusses related works. Section 3 reviews the NN backgrounds and the integrated architecture in mobile devices. Section 4 describes the dynamic channel-wise parallelizing inference algorithm for mobile devices. Section 5 shows the evaluation results. Section 6 concludes this paper.

2 Related Work

2.1 Accelerating the inference on mobile devices

As recent mobile SoCs are equipped with diverse computing resources, such as CPUs and GPUs, running the NN inference on mobile devices has become a promising option. However, as those processors on mobile devices usually have weak computing power, some existing mobile NN frameworks utilize multiprocessors to execute the NN models. The multicolumn deep neural network[10] distributes multiple

inputs to different processors and lets them run the inference separately. Therefore, it could meet the overall latency requirement of multiple tasks, but the single-input latency gets bounded by the single-processor performance as each input is processed by a single processor. In Ref. [11], each layer was executed on different processors to achieve low latency. By distributing layers, not inputs, the single-input latency can decrease. However, as each layer is still processed by a single processor, the single-input latency is still bounded. By simultaneously utilizing diverse heterogeneous processors on a mobile device and by performing computations using processor-friendly quantization, the μLayer[12] accelerates each NN layer, but it does not consider the different computing powers of processors on mobile devices and their occupancy rate.

2.2 Heterogeneous multiprocessor computing

FinePar[13] considers the architectural differences of the CPU and GPU on an integrated architecture and leverages fine-grained collaboration to accelerate matrix computation. It considers the different computing characteristics of those processors and separates the computing task. However, it mainly considers traditional computing tasks, such as graph calculation, but lacks research on the NN computing.

2.3 Thermal-aware resource management frameworks

Many studies have focused on temperature control in devices during task execution. In Ref. [14], the impact of the CPU frequency on the device temperature was observed, and a federated learning model training framework was proposed. C2RM[15] paid attention to the energy efficiency in training NN models with heterogeneous computing. Furthermore, RTTRM[16] realized thermal and timing constraints under dynamic temperature variations in an automotive microcontroller. However, these works mainly focused on the temperature consumption in large clusters and ignored the case of mobile devices. In addition, the methods mentioned above aimed at reducing the power consumption, even at the cost of accuracy loss, which is not reasonable in practical applications.

3 Background

3.1 NNs

NNs can learn how to perform tasks without task-specific procedures or rules[1]. Each neuron multiplies signals by the weights of the associated connections, applies some nonlinear functions on the sum of the multiplication outputs, and then transmits the output to other neurons. NNs can perform artificial intelligence applications, such as face recognition and natural language processing. In practice, training an NN refers to the process of adjusting the weights to improve the accuracy of the network. As mobile devices usually use a pretrained model to perform the inference task, in this study, we focused on the inference process. Among various types of NNs, we mainly focused on convolutional neural networks (CNNs)[17] as they are widely used in mobile devices and have various applications.

3.2 CNN layers

A CNN comprises an input layer, an output layer, and multiple hidden layers. The hidden layers typically consist of a stack of convolution, pooling, and fully connected (FC) layers. The convolution (Conv) layer is the core building block and consumes most of the computation time in a CNN model (e.g., 73.8% for VGG-16[18] and 99.93% for YOLOv2[19]). The layer's parameters comprise a set of learnable filters (or kernels), which have a small receptive field, but they extend through the full depth of the input volume. Additionally, the pooling layer's function progressively reduces the spatial size of the representation to reduce the number of parameters and the computation in the network, and hence to control overfitting. In the FC layers, all outputs are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are the FC layers that compile the data extracted by previous layers to form the final output. They are the most time-consuming layers except the Conv layer.

3.3 Integrated architecture of mobile devices

The common architectures of present computing devices, e.g., computers and servers, usually contain powerful computing processors, such as CPUs and GPUs, which are separately equipped with memories. In this architecture, users can utilize different processors to perform complex computing tasks. Nevertheless, as these powerful processors have their own memories to speed up their computational efficiency, transmitting data between different memories while using multiprocessors takes a long time. However, the situation is quite different in mobile devices, such as smartphones or Raspberry Pi. Although the processors in

mobile devices are not as powerful as those in advanced servers, the transmission of data takes less time due to the sharing of the memory (Fig. 2), which makes using multiprocessors to run computing tasks closer to reality.

4 Thermal-Aware Channel-Wise Heterogeneous Inference Algorithm

4.1 Dynamic power model

For a mobile device D, we define f^C_{clock} and f^G_{clock} (in cycle/s) as the clock frequency of the CPU and GPU on D. The computing speeds of the CPU and GPU can be defined as $f^C = f^C_{clock} \times n^C$ and $f^G = f^G_{clock} \times n^G$, respectively, where n^C and n^G are the numbers of CPU and GPU floating point operations per second per cycle, respectively. According to Ref. [20], the power consumption of a processor can be modeled as a function of the clock frequency: $P^{processor} = \Psi(f_{clock})^3$ where the coefficient $\Psi(\text{in W}/(\text{cycle}\cdot\text{s}^{-1})^3)$ depends on the chip architecture and f_{clock} is the clock frequency. Using this model, the power consumptions of the CPU and GPU on D can be defined as follows:

$$P^C = \Psi^C(f^C_{clock})^3 = \Upsilon^C(f^C)^3 \quad (1)$$

$$P^G = \Psi^G(f^G_{clock})^3 = \Upsilon^G(f^G)^3 \quad (2)$$

where $\Upsilon^C = \Psi^C/(n^C)^3$ and $\Upsilon^G = \Psi^G/(n^G)^3$.

In Ref. [21], the relationship between the device's leakage power consumption (P^{idle}) and the temperature (T_{eno}) can be expressed by a linear function:

$$P^{idle} = V(\beta_1 T_{eno} + \beta_0) \quad (3)$$

where β_1 and β_0 are device-dependent constants and V is the voltage of the device. Therefore, the power dissipation of the device can be expressed as:

$$P = P^{idle} + P^C + P^G = V(\beta_1 T_{eno} + \beta_0) + \Upsilon^C(f^C)^3 + \Upsilon^G(f^G)^3 \quad (4)$$

Suppose the initial temperature of the device is $T(0)$, at time t, the temperature of the device is as follows:

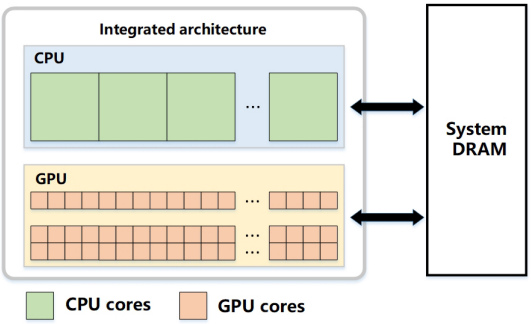


Fig. 2 Integrated architecture of mobile devices and data exchange with dynamic random-access memory (DRAM).

$$T(t) = T(0)\mathrm{e}^{-\frac{t}{RC}} + (T_{eno}(t) + PR)\left(1 - \mathrm{e}^{-\frac{t}{RC}}\right) \quad (5)$$

where R and C are the thermal resistance and capacitance, respectively, and $T_{eno}(t)$ is the environmental temperature at time t. The steady temperature of the device is defined as follows:

$$\begin{aligned} T(\infty) &= T_{eno}(\infty) + PR = \\ &T_{eno}(\infty) + (P^{idle} + P^C + P^G)R = \\ &(1 + VR\beta_1)T_{eno}(\infty) + R\Upsilon^C(f^C)^3 + \\ &\quad R\Upsilon^G(f^G)^3 + VR\beta_0 = \\ &\alpha_1 \cdot T_{eno}(\infty) + \alpha_2 \cdot (f^C)^3 + \alpha_3 \cdot (f^G)^3 + \alpha_0 \end{aligned} \quad (6)$$

where coefficients $\alpha_0 = VR\beta_0, \alpha_1 = 1 + VR\beta_1, \alpha_2 = R\Upsilon^C$, and $\alpha_3 = R\Upsilon^G$, respectively.

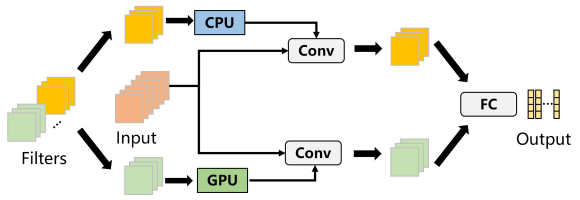
To ensure that the device works properly, $T(\infty)$ should always be kept lower than T_{max}, which is the maximum normal operating temperature. In consequence, the computing speeds of the CPU (f^C) and GPU (f^G) should comply with the following constraint:

$$\alpha_2(f^C)^3 + \alpha_3(f^G)^3 \leqslant T_{max} - \alpha_1 T_{eno}(\infty) - \alpha_0 \quad (7)$$

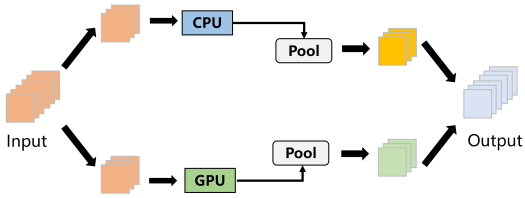
4.2 Channel-wise parallelism

Now that the cooperative single-layer acceleration has a high potential, mobile devices could distribute the computation of a single CNN layer to the CPU and GPU in a way that maximizes the performance gains. Because the Conv layer and the FC layers perform a similar pattern of computation, we divided our channel-wise parallelism in CNN models into two categories and mainly focused on the parallelism of these layers. As shown in Fig. 3a, for the Conv and FC layers, the filters are distributed to the CPU and GPU, while the input data are shared as the filters extend through all the input channels. Using the distributed filters and shared input data, the CPU and GPU generate their portions of the output channels. The generated output channels are then merged to form complete output data. As the filters are distributed with no overlaps, no redundant calculations would be generated.

For pooling layers (Fig. 3b), because the global function is spatially applied, the input data are distributed across channels. Then, the CPU and the GPU perform their tasks on their portions and generate their parts of the output data. Later, the output data are merged. Similar to the Conv and FC layers, no redundant calculations would be generated for the pooling layer.



(a) Conv and FC layers



(b) Pool parallelism

Fig. 3 An illustration of channel-wise parallelism for Conv, FC, and pooling layers.

4.3 Layer workload distribution to heterogeneous processors

Given a CNN model G with n layers, $L = \{l_1, l_2, ..., l_n\}$ is the set of the layers in G and $l_i \in L$ is the i-th layer. $W = \{W_1, W_2, \ldots, W_n\}$ is the workload of the inference, and $W_i \in W$ denotes the workload of the Layer l_i. While running l_i with one processor, the latency of this layer is $\frac{W_i}{f^C}$ for the CPU and $\frac{W_i}{f^G}$ for the GPU. By contrast, in the parallelization method, we divided the workload W into two parts: $W^C = \{W_1^C, W_2^C, \ldots, W_n^C\}$ and $W^G = \{W_1^G, W_2^G, \ldots, W_n^G\}$, for the workload W_i in Layer l_i, i.e., $W_i = W_i^C + W_i^G, i \in \{1, 2, \ldots, n\}$.

For Conv and FC layers, suppose the number of the filters in this layer is N, in the channel-wise parallelism method, the CPU and GPU will execute the computing task with N^C and N^G filters, respectively. Then, the workloads of the CPU and GPU are as follows:

$$W_i^C = \frac{N^C}{N} \cdot W_i \tag{8}$$

$$W_i^G = \frac{N^G}{N} \cdot W_i \tag{9}$$

For pooling layers, supposing that the number of the input data's channels is M, the CPU and GPU will execute the computing task with M^C and M^G channels, respectively. Then, the workloads of the CPU and GPU are as follows:

$$W_i^C = \frac{M^C}{M} \cdot W_i \tag{10}$$

$$W_i^G = \frac{M^G}{M} \cdot W_i \tag{11}$$

Therefore, we could get the inference latency with the parallelization method for Layer l_i:

$$t_i^P = \max\left\{\frac{W_i^C}{f^C}, \frac{W_i^G}{f^G}\right\} + t_i^m \tag{12}$$

where $t_i^m \in t^m$, $t^m = \{t_1^m, t_2^m, \ldots, t_n^m\}$ is the time for merging the output of each processor in Layer l_i. Note that a maximum value is taken here since the overall inference latency of heterogeneous processors is determined by the slower one.

Clearly, when $\frac{W_i^C}{f^C} = \frac{W_i^G}{f^G}$, the inference latency of Layer l_i with the parallelization method would be minimized. Therefore, the shortest inference time for Layer l_i is

$$t_i = \min\left\{t_i^P, t_i^C, t_i^G\right\} \tag{13}$$

where $t_i^C = \frac{W_i}{f^C}$ and $t_i^G = \frac{W_i}{f^G}$ are the inference latencies using only the CPU and GPU, respectively.

4.4 Thermal-aware workload distribution and dynamic adjustment

Although we had determined the theoretical minimum inference latency of each layer in the previous section, we had to consider other constraints, such as the temperature of the devices and the maximum running speed of the processors. Considering the constraints in a real environment, we formulated the workload distribution as an optimization problem subject to the following constraints:

$$\begin{aligned}
&\arg\min_{W^C, W^G, f^C, f^G} \sum_{i=1}^{n} t_i \\
\text{s.t.}\quad & t_i = \min\{t_i^P, t_i^C, t_i^G\}, \quad \forall i \in \{1, 2, \ldots, n\}; \\
& \alpha_2 (f^C)^3 + \alpha_3 (f^G)^3 \leqslant F; \\
& t_i^p = \frac{W_i^C}{f^C} = \frac{W_i^G}{f^G}, \ \forall W_i^C \in W^C, \ \forall W_i^G \in W^G; \\
& t_i^C = \frac{W_i}{f^C}, \ \forall W_i \in W; \\
& t_i^G = \frac{W_i}{f^G}, \ \forall W_i \in W; \\
& F = T_{max} - \alpha_1 T_{eno}(\infty) - \alpha_0; \\
& W_i = W_i^C + W_i^G, W_i^C \geqslant 0, W_i^G \geqslant 0; \\
& f^C \leqslant f_{max}^C; \\
& f^G \leqslant f_{max}^G
\end{aligned} \tag{14}$$

where f_{max}^C and f_{max}^G are the maximum running speeds of the CPU and GPU in device D, respectively.

To achieve the optimization objective in Formula (14), the running speeds of the CPU and GPU should be set first based on the maximum operating temperature limit of the device and the maximum floating point operation speed limit of the CPU and GPU, to maximize the computing power of the device and keep the device work steadily. We proposed an algorithm called the thermal-aware dynamic frequency (TADF) algorithm to derive f^C and f^G.

Algorithm 1 presents our TADF method. With an NN model G to device D, we first set the running speeds of the CPU and GPU in D as the maximum speeds, f^C_{max} and f^G_{max}; under the environments temperature T_{eno}, we could derive the power consumption of P. The initial P is calculated by Eq. (6) to examine if the device temperature would exceed the maximum working temperature (T_{max}), while the device could work steadily with the maximum working speed. Accordingly, f^C and f^G will be set as f^C_{max} and f^G_{max}, respectively. Otherwise, f^C and f^G will be adjusted until the device can work steadily.

Algorithm 1 Thermal-aware dynamic frequency (TADF) algorithm

Input: Maximum computing speeds of CPU f^C_{max} and GPU f^G_{max}, stand-by power consumption P^{idle}, the maximum operating temperature T_{max}, environmental temperature T_{eno}, coefficients α_0, α_1, α_2, and α_3;
Output: Running speeds of CPU (f^C) and GPU (f^G);

if $\alpha_2(f^C_{max})^3 + \alpha_3(f^G_{max})^3 \leqslant T_{max} - \alpha_1 T_{eno} - \alpha_0$ **then**
  $f^C \leftarrow f^C_{max}$, $f^G \leftarrow f^G_{max}$
else
  $f^C_{low} \leftarrow 0$, $f^G_{low} \leftarrow 0$
  $f^C_{high} \leftarrow f^C_{max}$, $f^G_{high} \leftarrow f^G_{max}$
  $f^C \leftarrow \frac{f^C_{max}}{2}$, $f^G \leftarrow \frac{f^G_{max}}{2}$
  while $\alpha_2(f^C_{max})^3 + \alpha_3(f^G_{max})^3 \notin [F-\theta, F]$ **do**
    if $\alpha_2(f^C_{max})^3 + \alpha_3(f^G_{max})^3 > F$ **then**
      if $\sqrt[3]{\alpha_2} f^C \geqslant \sqrt[3]{\alpha_3} f^G$ **then**
        $f^C_{high} \leftarrow f^C$, $f^C \leftarrow \frac{f^C_{low} + f^C_{high}}{2}$
      else
        $f^G_{high} \leftarrow f^G$, $f^G \leftarrow \frac{f^G_{low} + f^G_{high}}{2}$
    else if $\sqrt[3]{\alpha_2} f^C \geqslant \sqrt[3]{\alpha_3} f^G$ **then**
      $f^G_{low} \leftarrow f^C$, $f^G \leftarrow \frac{f^G_{low} + f^G_{high}}{2}$
    else
      $f^C_{low} \leftarrow f^C$, $f^C \leftarrow \frac{f^C_{low} + f^C_{high}}{2}$
return f^C, f^G

Second, with the fixed f^C and f^G, each layer's inference latency in three methods, namely, CPU-only, GPU-only, and CPU-GPU parallelization method, will be calculated. Consequently, the best inference method for each layer and the workload distribution will be determined. Algorithm 2 shows the workload allocation method.

5 Experiment

We implemented and evaluated our algorithm on a mobile device. In this section, we present our results. Our evaluation focuses on whether the algorithm can reduce the inference latency on-device while guaranteeing the thermal constraints. We use three benchmarks: Inference with the CPU only, inference with the GPU only, and a coarse-grained workload partitioning method called CGP[22]. In the third method, the workload is distributed to the CPU and GPU with a fixed ratio without considering the processor's computing ability.

5.1 Experimental setup

NNs: We use AlexNet and VGG16 to test the performance of the four kinds of inference methods, because they have a small number of parameters and computations and are suitable for mobile device inference. AlexNet has five Conv layers, three pooling layers, three FC (linear) layers, and a softmax layer to determine the probabilities of each category. Moreover,

Algorithm 2 Heterogeneous-processors single-layer workload distribution (HSWD) algorithm

Input: The set of the layers L, the workload set of each layer W, running speeds of CPU (f^C) and GPU (f^G);
Output: The set of the each layer compution in CPU W^C, the set of the each layer compution in GPU W^G;

for $i = 1, 2, \ldots, n$ **do**
  $W^G_i \leftarrow \frac{W_i f^C}{f^G + f^C}$
  $W^C_i \leftarrow \frac{W_i f^G}{f^G + f^C}$
  if $\min\{t^P_i, t^C_i, t^G_i\} = t^P_i$ **then**
    break
  else if $\min\{t^P_i, t^C_i, t^G_i\} = t^C_i$ **then**
    $W^C_i \leftarrow W_i$
    $W^G_i \leftarrow 0$
  else
    $W^C_i \leftarrow 0$
    $W^G_i \leftarrow W_i$
return W^C, W^G

a rectified linear unit (ReLU)[23] layer was placed after each Conv layer and FC layer to increase the nonlinearity. As the cost of the ReLU layer and softmax layer is almost negligible, we mainly parallelized the inference in the Conv layer, FC layer, and pooling layer. VGG16 has 13 Conv layers, five pooling layers, three FC layers, and a softmax layer.

Platforms: We measured the performance of the parallelization methods on Nvidia Jetson Nano[24] (see Fig. 4), as it has an integrated CPU/GPU architecture and is suitable for our experiments. Nano has an ARM Cortex-A57 MPCore CPU with four cores, a GPU equipped with NVIDIA Maxwell framework and 128 NVIDIA CUDA cores, a memory size of 4 GB, and an Ubuntu 18.04 system. Nano supports two power modes: MaxN (10 W) and 5W (5 W). The parameters of each mode are shown in Table 1. We used PyTorch[25] to build the network model.

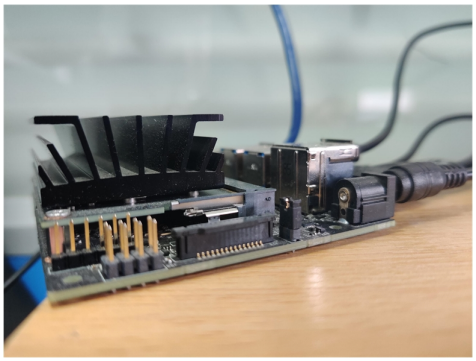


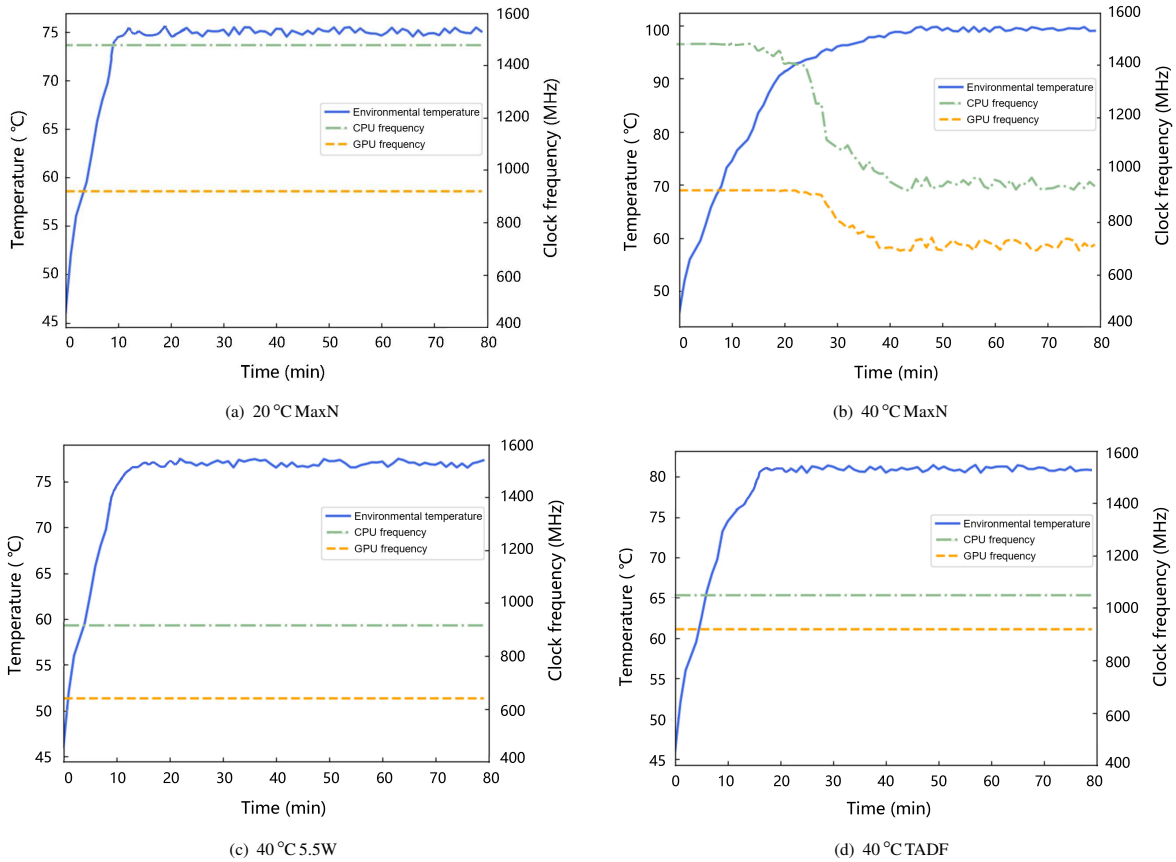
Fig. 4 Photo of Nvidia Jeston Nano.

5.2 Impact of temperature on the device's frequency

We first observed the impact of the environment on the device's frequency. We ran stress-ng[26], a stress test tool to test the CPU and GPU frequencies after a long-time operation. As shown in Fig. 5a, under a room

Table 1 Predefined parameters of Jeston Nano under two power modes.

Mode name	Power budget (W)	Model ID	Number of online CPUs	CPU max frequency (MHz)	GPU max frequency (MHz)	Number of GPU texture processing clusters
MaxN	10	0	4	1479	921	1
5W	5	1	2	918	640	1



(a) 20 °C MaxN (b) 40 °C MaxN

(c) 40 °C 5.5W (d) 40 °C TADF

Fig. 5 CPU and GPU clock frequencies under different environmental temperatures.

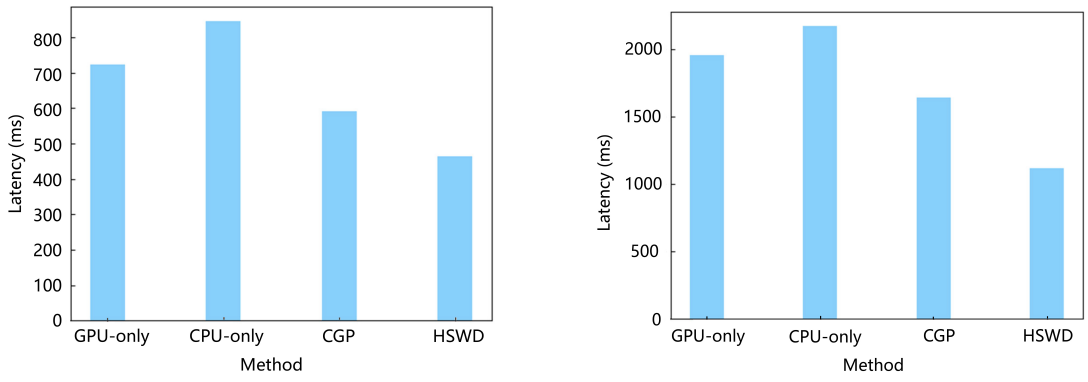
temperature of approximately 20 °C and device mode of MaxN, the device's temperature increases as the tool runs. The CPU can steadily run with a max frequency of 1.5 GHz when the device's temperature goes up to approximately 80 °C. The same outcome happens with the GPU, whose max frequency is 921 MHz. However, when the environmental temperature turns to 40 °C, the device temperature would increase to over 90 °C; under such temperatures, the CPU and GPU frequencies declined, as shown in Fig. 5b. We then turned the device mode to 5 W, as shown in Fig. 5c; under such a mode, the CPU and GPU can run steadily, and the device temperature can be kept below 80 °C. Under 5W mode, the device can work without frequency decline, but it cannot reach the maximum performance under the environmental temperature. Figure 5d shows the CPU and GPU frequencies set by the TADF algorithm. The CPU and GPU frequencies were set as 1050 and 921 MHz, respectively, after a long-time operation. The device temperature was kept below 85 °C, and the device ran steadily, which proves the effectiveness of our method.

5.3 Comparison of the inference latencies

After setting the CPU and GPU frequencies with TADF, we then compared the inference latencies of AlexNet and VGG16 with our heterogeneous-processor single-layer workload distribution (HSWD) algorithm and three benchmarks to verify the effectiveness of our method. The CPU and GPU frequencies were 1050 and 921 MHz, respectively, which was set by the TADF algorithm, as mentioned in Section 4. We first ran the inference multiple times with four methods to derive the average inference latency of each method. As shown in Fig. 6, because the architecture of the GPU is highly suitable for matrix computation, the latency of the GPU inference is lower than that of the CPU. In addition, as the CGP method uses the CPU and GPU to accelerate the inference, its latency is lower than that of the single-processor inference methods. However, as the workload was not well distributed to processors according to their computing ability and the layers' characteristics, its performance shows only a marginal advantage compared with the GPU-only method.

Compared with the benchmarks, the HSWD algorithm can choose the best method for each layer according to the architecture and data, hence it could achieve the lowest inference latency for each layer. As shown in Fig. 6, HSWD achieves the lowest latency in the two NNs. In AlexNet, the average inference latencies of the GPU-only, CPU-only, and CGP methods are 724, 845, and 592 ms, respectively, whereas the average inference delay of the HSWD method is 465 ms. Compared with the traditional method, the HSWD method can achieve a 21%–36% latency reduction. In VGG16, the average inference latencies of the GPU-only, CPU-only, and CGP methods are 1954, 2170, and 1638 ms, respectively, whereas the average inference latency of the HSWD method is 1117 ms. Compared with the traditional method, the HSWD method can achieve a 32%–43% latency reduction.

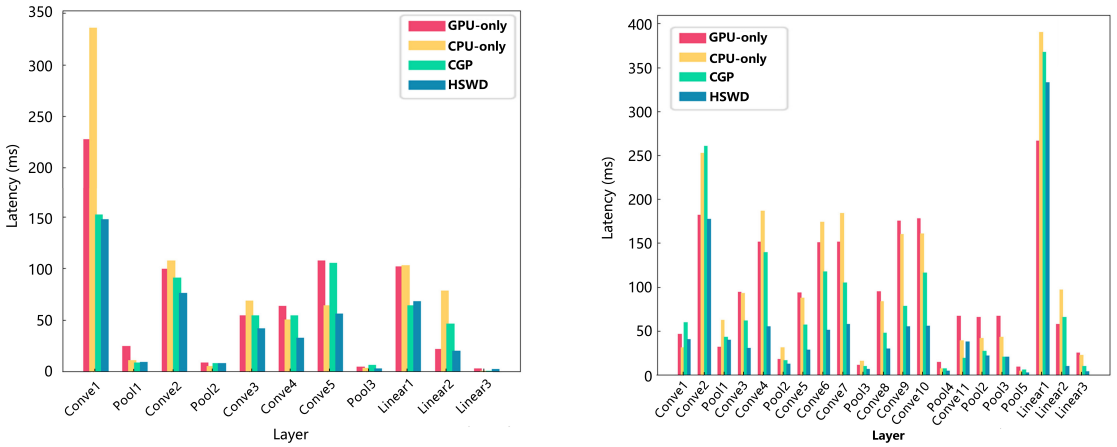
Figure 7 shows the inference latencies per layer of the four methods. In most layers, the GPU-only method runs faster than the CPU-only method, whereas, in a few layers, the situation is the opposite. The CGP method cannot optimally adjust the workload distribution of each layer so that it fails to achieve the best performance. Moreover, in some layers, because the workload distribution is not that reasonable, the additional overhead resulting from merging the data of the output from the two processors leads to a longer latency than that of the single processor-only



(a) AlexNet latency

(b) VGG16 latency

Fig. 6 Comparison on the inference latencies of different methods.



(a) AlexNet latency of each layer

(b) VGG16 latency of each layer

Fig. 7 Break-down of the inference latencies for each layer of the four methods.

methods. This situation indicates that only by selecting the appropriate inference method according to the characteristics of the single layer can the inference latency be minimized. In each layer, the HSWD method achieves the lowest latency. When the amount of computation is larger, the effect of the decrease of latency is more obvious. Especially in Conv4 and Conv7 in VGG16, the HSWD method achieves almost a 50% latency decrease, as compared with the large computation amount. The extra delay caused by the merging of data can be ignored.

6 Conclusion

In this study, we investigated the characteristics and requirements of mobile device inference. We identified the existing methods to determine whether the inference has disadvantages and cannot make full use of the computing resource on mobile devices. Existing methods have not paid attention to the influence of environmental implications, such as temperature. To solve this problem, we proposed a thermal-aware channel-wise heterogeneous inference algorithm, which contains two parts: The TADF algorithm and the HSWD algorithm. TADF is to set the CPU and GPU frequencies according to the environmental temperature and device performance, thereby ensuring the stable operation of devices in a high-temperature environment. HSWD can allocate the computing task to heterogeneous processors on mobile devices under the temperature constraint (Formula (7)) so as to minimize the inference latency. The experiment results verify that our method can significantly decrease the inference latency and maitain running stability.

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2018AAA0100500), the National Natural Science Foundation of China (Nos. 61972085, 61872079, and 61632008), the Jiangsu Provincial Key Laboratory of Network and Information Security (No. BM2003201), Key Laboratory of Computer Network and Information Integration of Ministry of Education of China (No. 93K-9), Southeast University China Mobile Research Institute Joint Innovation Center (No. R21701010102018), and the University Synergy Innovation Program of Anhui Province (No. GXXT-2020-012), and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization, the Fundamental Research Funds for the Central Universities, CCF-Baidu Open Fund (No. 2021PP15002000), and the Future Network Scientific Research Fund Project (No. FNSRFP-2021-YB-02). We also thank the Big Data Computing Center of Southeast University for providing the experiment environment and computing facility.

References

[1] R. Lippmann, An introduction to computing with neural nets, *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, 1987.

[2] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, Speech recognition using deep neural networks: A systematic review, *IEEE Access*, vol. 7, pp. 19143–19165, 2019.

[3] L. Coheur, From Eliza to Siri and beyond, in *Proc. 18th Int. Conf. Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Lisbon, Portugal, 2020, pp. 29–41.

[4] C. Szegedy, W. Liu, Y. Q. Jia, P. Sermanet, S. Reed, D.

Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Going deeper with convolutions, presented at the 2015 IEEE Conf. Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1–9.

[5] L. K. Zeng, E. Li, Z. Zhou, and X. Chen, Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things, *IEEE Network*, vol. 33, no. 5, pp. 96–103, 2019.

[6] R. Bi, R. Liu, J. K. Ren, and G. Z. Tan, Utility aware offloading for mobile-edge computing, *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 239–250, 2021.

[7] Q. C. Cao, W. L. Zhang, and Y. H. Zhu, Deep learning-based classification of the polar emotions of "moe"-style cartoon pictures, *Tsinghua Science and Technology*, vol. 26, no. 3, pp. 275–286, 2021.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Proc. 25th Int. Conf. Neural Information Processing Systems*, Lake Tahoe, NV, USA, 2012, pp. 1097–1105.

[9] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, et al., Large scale distributed deep networks, in *Proc. 25th Int. Conf. Neural Information Processing Systems*, Lake Tahoe, NV, USA, 2012, pp. 1223–1231.

[10] S. Han, H. C. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, MCDNN: An approximation-based execution framework for deep stream processing under resource constraints, in *Proc. 14th Annu. Int. Conf. Mobile Systems, Applications, and Services*, Singapore, 2016, pp. 123–136.

[11] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, DeepX: A software accelerator for low-power deep learning inference on mobile devices, presented at the 2016 15th ACM/IEEE Int. Conf. Information Processing in Sensor Networks (IPSN), Vienna, Austria, 2016, pp. 1–12.

[12] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, μLayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization, in *Proc. 14th EuroSys Conf. 2019*, Dresden, Germany, pp. 1–15.

[13] F. Zhang, J. D. Zhai, B. Wu, B. S. He, W. G. Chen, and X. Y. Du, Automatic irregularity-aware fine-grained workload partitioning on integrated architectures, *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 867–881, 2021.

[14] C. Wang, Y. Y. Yang, and P. Z. Zhou, Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity, *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 394–410, 2021.

[15] Q. S. Zeng, Y. Q. Du, K. B. Huang, and K. K. Leung, Energy-efficient resource management for federated edge learning with CPU-GPU heterogeneous computing, *IEEE Trans. Wirel. Commun.*, doi: 10.1109/TWC.2021.3088910.

[16] Y. Lee, H. S. Chwa, K. G. Shin, and S. G. Wang, Thermal-aware resource management for embedded real-time systems, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2857–2868, 2018.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[18] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, presented at the 3rd Int. Conf. Learning Representations, San Diego, CA, USA, 2015, pp. 1–14.

[19] J. Redmon and A. Farhadi, YOLO9000: Better, faster, stronger, presented at the 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 6517–6525.

[20] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, and F. X. Lin, Power-efficient time-sensitive mapping in heterogeneous systems, presented at the 2012 21st Int. Conf. Parallel Architectures and Compilation Techniques (PACT), Minneapolis, MN, USA, 2012, pp. 23–32.

[21] Y. P. Liu, R. P. Dick, L. Shang, and H. Z. Yang, Accurate temperature-dependent integrated circuit leakage power estimation is easy, presented at the 2007 Design, Automation and Test in Europe Conference and Exposition, Nice, France, 2007, pp. 1526–1531.

[22] F. Zhang, J. D. Zhai, B. S. He, S. H. Zhang, and W. G. Chen, Understanding co-running behaviors on integrated CPU/GPU architectures, *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 905–918, 2017.

[23] A. F. Agarap, Deep learning using rectified linear units (ReLU), arXiv preprint arXiv: 1803.08375, 2019.

[24] S. Cass, Nvidia makes it easy to embed AI: The Jetson nano packs a lot of machine-learning power into DIY projects-[Hands on] , *IEEE Spectrum*, vol. 57, no. 7, pp. 14–16, 2020.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. M. Lin, N. Gimelshein, L. Antiga, et al., PyTorch: An imperative style, high-performance deep learning library, in *Proc. 33rd Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, pp. 8026–8037.

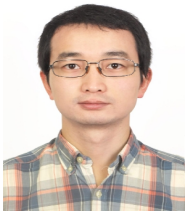
[26] K. Colin, Stress-ng, http://kernel.ubuntu.com/git/cking/stressng.

Jinghui Zhang received the BS degree from Southeast University, Nanjing, China, in 2005, and the PhD degree in computer science from Southeast University, Nanjing, China, in 2014. He is an associate professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. His current research interests include edge computing, distributed machine learning, and cloud computing.

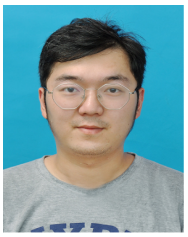
Yuchen Wang received the BS degree in computer science and technology from Jiangnan University, Wuxi, China, in 2020. Currently, he is pursuing the MS degree in computer science and engineering at Southeast University, Nanjing, China. His current research interests include distributed machine learning and federated graph learning.

Tianyu Huang received the MS degree in computer science from Southeast University, Nanjing, China, in 2021. Currently, he works in the general office of the People's Government of Jiangsu Province. His research interests include edge computing and distributed machine learning.

Fang Dong received the BS and MS degrees in computer science from Nanjing University of Science & Technology, Nanjing, China, in 2004 and 2006, respectively, and received the PhD degree in computer science from Southeast University, Nanjing, China, in 2011. He is currently a professor at School of Computer Science and Engineering, Southeast University, China. He is a member of both IEEE and ACM, he also served as the co-chair of ACM Nanjing Chapter and the general secretary of ACM SIGCOMM China. His current research interests include cloud computing, edge intelligence, and workflow scheduling.



Wei Zhao received the PhD degree in applied information from Tohoku University, Sendai, Japan, in 2015. He is an associate professor at the School of Computer Science and Technology, Anhui University of Technology, Hefei, China. His research interests include wireless network and cloud computing.



Dian Shen received the BS, MS, and PhD degrees from Southeast University, Nanjing, China, in 2010, 2012, and 2018, respectively. He is a postdoctoral researcher at The Chinese Univeristy of Hong Kong, Hong Kong, China. His research interests include edge computing, virtualization, and data center network.