

## RESEARCH ARTICLE

# TAS: A Temperature-Aware Scheduling for Heterogeneous Computing

XIANG GAO 

Research Institute of China Telecom Corporation Ltd., Guangzhou 510660, China

e-mail: gaiox15@chinatelecom.cn


**ABSTRACT** With the development of AI technology, the parameters and calculation overhead of advanced models have increased exponentially, resulting in the existing low-end GPU(Graphic Processing Unit) being unable to meet the computing power required for model operation. In order to speed up the inference speed in edge scenarios, various manufacturers have launched NPU(Neural Processor Unit), a special chip for neural networks, which can improve the overall inference efficiency and reduce energy consumption through a certain loss of precision. However, in the current common edge-side solutions, the problem of CPU+GPU+NPU co-processing is not well considered. At the same time, edge-side devices are more easily affected by the ambient temperature. In this paper, CPU+GPU+NPU is used to jointly process edge-side inference tasks, and we first established a heterogeneous device temperature perception model based on the ambient temperature of the edge device, then proposed a TAS(temperature-aware schedule) algorithm to control the running speed of the heterogeneous device, and then proposed a task scheduling algorithm for the heterogeneous device, namely TASTS(TAS-based task schedule). At the same time, we also use a hungarian matching algorithm to optimize the final result. This paper finally verified several models in real edge environment, found that it can improve the performance by 20-50% compared with conventional methods under temperature constraints.

**INDEX TERMS** Heterogeneous computing, schedule, temperature-aware.

## I. INTRODUCTION

With the rapid advancement of global science and technology, artificial intelligence plays a role in more and more fields, such as smart machines, smart homes, smart phones, etc. The operating speed, application scenarios and equipment configurations are gradually improving, and the edge devices must process a large amount of image and video information. With the increase of images, frame numbers, and resolutions, the demand for high-performance and high-speed hardware is also gradually increasing [1], [2], [3].

In the past ten years, convolutional neural network [4] has become an irreplaceable tool due to its excellent performance on some complex image problems, including image classification [5], object detection [6] and semantic segmentation [7], the influence of convolutional neural network has gone beyond academia, and it has been continuously applied

The associate editor coordinating the review of this manuscript and approving it for publication was Shih-Wei Lin .

in industry, such as some real-time processing tasks such as automatic driving, intelligent robot and intelligent camera technology [8].

However, cloud devices have problems such as poor interface flexibility, dependence on data networks, and time-consuming communication, which cannot meet the requirements of some systems for power consumption and portability. Terminals equipped with artificial intelligence chips are centered on engineering applications and have strict requirements on software and hardware modules and power consumption costs.

At the same time, the complexity of the environment where the inference task is located may affect the performance of the edge device, and it is necessary to control the power consumption of each processor in the edge device to ensure the smooth operation of the edge device.

At present, the edge device mainly adopts the CPU+GPU solution(CPU:Central Processing Unit, GPU:Graphics Processing Unit), and the NPU(Neural Processor Unit) has not

been widely used in edge scenarios. CPU+GPU heterogeneous computing has many limitations at the edge, including limited inference speed and high energy consumption. The NPU can not only effectively solve the above-mentioned problems and accelerate inference task, but also can reduce power consumption. At the same time, the environmental load on the edge side, especially the ambient temperature will seriously affect the operating efficiency of the device, and the existing research rarely considers the impact of this problem.

The contributions of this paper mainly include:

1. According to the ambient temperature and processor power, the temperature model of the edge device under temperature constraints is established.
2. Combined with the temperature model, a temperature-aware schedule algorithm to control the running speed of the heterogeneous device is proposed, namely TAS.
3. According to the TAS algorithm, we establish the heterogeneous task scheduling algorithm, namely TASTS (TAS-based task schedule).
4. Propose an overall heterogeneous scheduling framework that uses NPU to improve overall efficiency and reduce power consumption, uses GPU to reduce precision loss, and uses CPU to schedule tasks and optimize results
5. Propose a collaborative optimization algorithm to process the detection frame obtained by NPU inference, and match and merge the detection frame obtained by Hungarian matching algorithm and GPU, so as to achieve the optimal inference effect.

The organization structure of this paper is as Figure 1. The Section I is introduction, the Section II is related work, the Section III is modeling, the Section IV is system and algorithm design, the Section V is test and analysis, and finally, the Section VI is conclusions.

## II. RELATED WORK

### A. NEURAL NETWORK PARALLEL TECHNOLOGY

With the rapid development of neural network, the layers number of neural network continues to deepen, and the amount of parameters and calculations also continue to increase. The number of parameters in AlexNet [9] is about 60 million, and the number of parameters in VGG16 and VGG19 [10] are 138 million and 143 million respectively, while the number of parameters of ResNet [11] reaches 3.6 billion. Faced with such a huge amount of parameters and calculations, if only the original training and inferencing methods are used, the convergence time of neural network training and inference delay will be longer, which cannot meet the user's delay requirements.

Based on this situation, in recent years, researchers have proposed the concept of distributed machine learning [12]. Distributed machine learning refers to the division of large-scale models or parameters, and the training of neural networks by using multiple processors at the same time. At present, neural networks mainly have two parallel methods: data parallelism [13] and model parallelism [14].

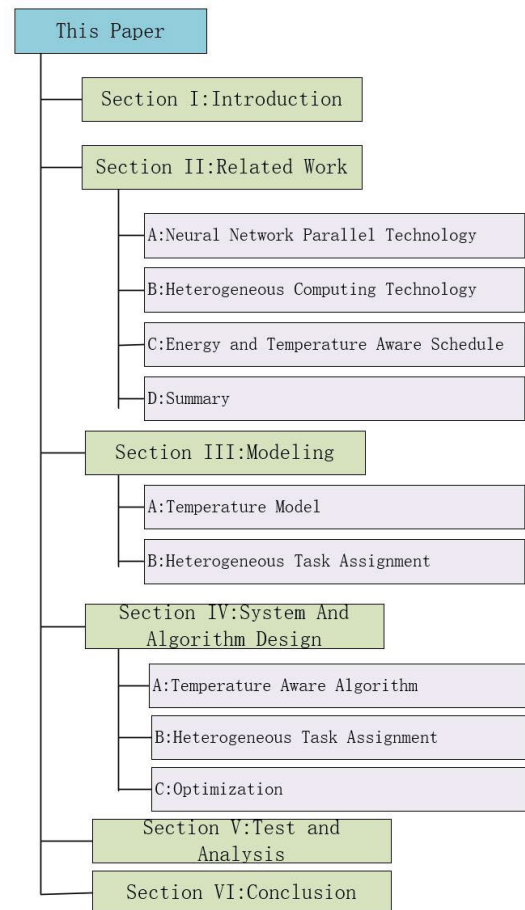


FIGURE 1. The structure of this paper.

Yadan et al. [15] used both data parallelism and model parallelism to train convolutional neural networks on multiple GPUs. The experimental results showed that the mixed parallelism achieved faster convergence than the single parallelism. Wu [16] et al. proposed GNMT, which uses the attention mechanism to connect the bottom layer of the decoder and the top layer of the encoder, and divides the encoder and decoder networks along the depth dimension and places them on multiple GPUs. Mirhoseini [17] and others proposed ColocRL, which achieved a 19.7% improvement in the training speed of deep neural networks compared to other traditional methods. Jia [18] and others proposed Opt CNN, which uses an effective algorithm to find the globally optimal parallelization strategy and realize support for hierarchical parallelism.

Existing distributed machine learning methods pay more attention to large server clusters, and pay less attention to parallel inferencing on small edge devices. Most of the parallel methods used in the above are homogeneous processors, and heterogeneous processors are not considered.

### B. HETEROGENEOUS COMPUTING TECHNOLOGY

Heterogeneous computing mainly refers to the computing method that uses a variety of computing units with different structures to jointly execute the same relatively complex

algorithm in the same system. The more common heterogeneous structure is the heterogeneity of the CPU and GPU. Due to the structure is different, and there are also significant differences in performance. The CPU is mainly used as a control unit, which plays a basic control and management role. The GPU is mainly used as a computing unit, and is more responsible for calculating complex floating-point operations or matrix operations. In addition to these two computing units, ASIC, FPGA, and NPU have also joined the ranks of heterogeneous computing in recent years, and their computing power has greatly improved compared to CPU computing power and performance parameters.

Lane et al. proposed DeepX [19], a collaborative inference method for multiple heterogeneous processors in mobile devices. Deep X decomposes the deep neural network into various types of unit blocks, and divides each unit block according the characteristics to CPU and GPU respectively, and the network is compressed according to the performance of the processor to adapt to the computing power of the processor. Kim [20] et al. proposed  $\mu$ layer, a parallel inference method for single-layer neural networks based on mobile devices. Zhang et al. [21] proposed Fine Par, a fine-grained calculation division method for matrix operations, and put most of the calculations with fewer non-zero bits on the GPU. Wang et al. [22] studied the use of heterogeneous mobile devices for distributed deep neural network training in federated learning [18], and used an efficient polynomial time algorithm to schedule different workloads on different mobile devices.

### C. ENERGY AND TEMPERATURE AWARE SCHEDULE

There are many papers address the challenge of energy-efficient and temperature-aware scheduling on heterogeneous platforms, with different approaches and trade-offs. Moulik et al. presents a low-overhead heterogeneous multi-core scheduler for real-time periodic tasks [23]. The proposed scheduler utilizes a novel task mapping algorithm, which minimizes the cache miss rate and reduces contention among cores while achieving high performance. Nair P P et al. proposes Fest, a fault-tolerant energy-aware scheduling approach for two-core heterogeneous platforms [24]. Fest maximizes energy efficiency by utilizing dynamic voltage and frequency scaling (DVFS) techniques while ensuring fault tolerance via replica-based scheduling. Moulik et al. introduces TARTS, a temperature-aware real-time deadline-partitioned fair scheduler that considers both real-time and thermal constraints [25]. The proposed scheduler partitions the available resources into multiple virtual CPUs and assigns priorities based on the deadline of each task and the temperature of the corresponding CPU.

Singh et al. proposes a semi-partitioned scheduler that partitions the cores into two groups and schedules tasks accordingly to reduce energy consumption [26]. Hussain et al. proposes a scheduler that considers both temperature and energy constraints while scheduling tasks on heterogeneous

platforms [27]. Kumar and Kumar eproposes a cluster-based approach that schedules tasks based on their criticality and their placement in different clusters [28]. M. Lim et al. proposes a hybrid approach that combines dynamic voltage frequency scaling and task migration to minimize energy consumption while meeting timing constraints and temperature limits [29]. Rahman et al. proposes a fault-tolerant scheduler that assigns tasks to two types of cores with different performance levels, while meeting energy and temperature constraints [30]. Li et al. proposes a cluster-based scheduler that considers both energy and temperature awareness along with task criticality [31].

Overall, these papers address the challenge of energy-efficient and temperature-aware scheduling on heterogeneous platforms, with different approaches and trade-offs. However, they did not consider edge heterogeneous acceleration device scenarios such as NPU in their researchs.

### D. SUMMARY

Problems existing in the current edge heterogeneous solution:

1) The environment of the edge scene is complex, and the stability of the device is very high, especially the ambient temperature. Many existing scheduling algorithms do not take this issue into consideration.

2) GPU is a special engine for image processing, which has higher computing density and parallel computing characteristics. However, AI computing only requires the parallel computing capability of GPU, not its graphics rendering capability. Moreover, the cost of GPU is high, and it generates a lot of heat. For the problem of high power consumption, the resources at the edge end are already limited, and its power consumption needs to be further reduced.

3) Compared with GPU, NPU is more suitable for neural network inference and has higher energy efficiency ratio. However, the existing NPU uses quantization technology to convert float models into data such as int8 or int16 to reduce inference time. The cost of computing power and energy consumption is a certain loss of precision. Existing work does not consider how to reduce or avoid these losses.

## III. MODELING

In addition to being widely used in daily life, tasks such as neural network inference are also widely used in industrial production, such as product quality inspection, equipment flaw detection and other fields, and these application fields often have the characteristics of high ambient temperature. Therefore, it is necessary to set the power consumption of the device processor according to the ambient temperature, so as to ensure the stable operation of the device. For a edge device D equipped with CPU, GPU and NPU, the relevant attributes in the device are shown in the table 1.

### A. TEMPERATURE MODEL

This section focuses on the establishment of the temperature model for a edge device D, which shows the relationship between temperature and speed.

TABLE 1. Math symbols.

Math Symbol	Paraphrase
CPU	Central Processing Unit
GPU	Graphics Processing Unit
NPU	Neural Processor Unit
$F_C$	CPU frequency
$F_G$	GPU frequency
$F_N$	NPU frequency
$OP_C$	CPU operations per cycle
$OP_G$	GPU operations per cycle
$OP_N$	NPU operations per cycle
$Speed_C$	CPU operation speed per second
$Speed_G$	GPU operation speed per second
$Speed_N$	NPU operation speed per second
$Power_C$	CPU Power
$Power_G$	GPU Power
$Power_N$	NPU Power
$Power_{idle}$	Idle Power
$T_{env}$	Environment Temperature
$T_{max}$	Max Working Temperature
G	neural network
L	Collection of layers of the model
$L_i$	The i-th layer of L
C	Collection of calculations of the model
$C_i$	The i-th layer calculations
$C^C$	Collection of CPU calculations of the model
$C^G$	Collection of GPU calculations of the model
$C^N$	Collection of NPU calculations of the model
$C_i^C$	The i-th layer calculations with CPU
$C_i^G$	The i-th layer calculations with GPU
$C_i^N$	The i-th layer calculations with NPU
$t_i^C$	Latency of layer $L_i$ when in parallel
$t_i^G$	Latency of layer $L_i$ when all use CPU
$t_i^N$	Latency of layer $L_i$ when all use GPU
$t_i^N$	Latency of layer $L_i$ when all use NPU

The clock frequencies (cycles per second) of the CPU, GPU and NPU in device D are  $F_C$ ,  $F_G$ ,  $F_N$ . The number of floating-point operations per cycle is  $O_C$ ,  $O_G$  and  $O_N$ . The floating-point operation speed (times/second) can be respectively:

$$Speed_C = F_C O_C \tag{1}$$

$$Speed_G = F_G O_G \tag{2}$$

$$Speed_N = F_N O_N \tag{3}$$

The power consumption of the edge device is composed of the standby power consumption of the device itself and the power consumption generated by the processor in the device when it is running. The power consumption of a processor is related to the clock frequency of the processor, and the correspondence between them can be expressed as:

$$Power_p = \mu F^3 \tag{4}$$

$Power_p$  represents the power consumption of the processor,  $\mu$  is a processor architecture-determined. The power consumption of CPU, GPU and NPU in device D is:

$$Power_C = \mu^C \left(\frac{Speed_C}{O_C}\right)^3 = \chi^C Speed_C^3 \tag{5}$$

$$Power_G = \mu^G \left(\frac{Speed_G}{O_G}\right)^3 = \chi^G Speed_G^3 \tag{6}$$

$$Power_N = \mu^N \left(\frac{Speed_N}{O_N}\right)^3 = \chi^N Speed_N^3 \tag{7}$$

Base on the equations 5, 6, 7, the power consumption of the processor is positively correlated with the floating-point operation speed of the processor. Therefore, the power consumption of the processor can be adjusted by changing the floating-point operation speed of the processor.

The relationship between the standby power consumption of the device and the environment and the device voltage can be approximated by the performance model as:

$$T = \lambda_1 T_{env} + \lambda_0 \tag{8}$$

$$Power_{idle} = VT = V(\lambda_1 T_{env} + \lambda_0) \tag{9}$$

The  $Power_{idle}$  standby the power consumption of device D, V is the voltage of the device,  $T_{env}$  is the ambient temperature, the coefficients  $\lambda_1$  and  $\lambda_0$  are related to the performance of the device. It can be seen from equation 9 that the standby power consumption of the device is related to the ambient temperature of the device and the voltage of the device. When the ambient temperature of the device is high, the standby power consumption of the device will also be high. Then, the total power consumption of device D can be expressed as:

$$\begin{aligned} Power &= Power_{idle} + Power_C + Power_G + Power_N \\ &= V(\lambda_1 T_{env} + \lambda_0) + \chi^C (Speed_C)^3 \\ &\quad + \chi^G (Speed_G)^3 + \chi^N (Speed_N)^3 \end{aligned} \tag{10}$$

So, for device D, its power consumption is affected by the floating-point speed of the CPU, GPU and NPU in the device, as well as the ambient temperature. According to the thermal circuit model, the temperature of the device can be expressed as a function related to the power consumption of the device. The temperature of the device at time t can be expressed as:

$$\begin{aligned} T(t) &= T_{env}(t) + PowerR \\ &= (1 + VR\lambda_1)T_{env}(t) + R\chi^C (Speed_C)^3 \\ &\quad + R\chi^G (Speed_G)^3 + R\chi^N (Speed_N)^3 + VR\lambda_0 \\ &= \kappa_1 T_{env}(t) + \kappa_2 (Speed_C)^3 + \kappa_3 (Speed_G)^3 \\ &\quad + \kappa_4 (Speed_N)^3 + \kappa_0 \end{aligned} \tag{11}$$

For the equation 11, the R represent thermal resistance. To make the device work stably for a long time, the temperature of the device should always be lower than its maximum stable working temperature  $T_{MAX}$ . Correspondingly, the floating-point operation speed of the CPU, GPU and NPU in device D should obey the constraint:

$$\begin{aligned} \kappa_2 (Speed_C)^3 + \kappa_3 (Speed_G)^3 + \kappa_4 (Speed_N)^3 \\ \leq T_{max} - \kappa_1 T_{env}(t) - \kappa_0 \end{aligned} \tag{12}$$

### B. HETEROGENEOUS TASK ASSIGNMENT

This section mainly divides computing tasks according to the performance of heterogeneous processors in edge devices and the structural characteristics of each layer in neural networks. For a neural network G, the parameters in the device are shown in the table 1.

For a neural network G with n layers,  $L = \{L_1, L_2, \dots, L_n\}$  is the set of layers in G, where  $L_i \in L$  is the i-th layer

in G.  $C = \{C_1, C_2, \dots, C_n\}$  is the calculation amount of each layer in G, where  $C_i \in C$  is the calculation amount of the  $i$ -th layer in G.  $C^C = \{C_1^C, C_2^C, \dots, C_n^C\}$ ,  $C^G = \{C_1^G, C_2^G, \dots, C_n^G\}$  and  $C^N = \{C_1^N, C_2^N, \dots, C_n^N\}$  represent the calculation amount of each layer of CPU, GPU and NPU when executing the inference task of neural network G, where:

$$C_i = C_i^C + C_i^G + C_i^N, i \in \{1, 2, 3, \dots, n\} \quad (13)$$

For layer  $L_i$ , when both CPU, GPU and NPU are used to perform calculations on this layer, if the layer is a convolutional layer or a fully connected layer with the number of convolution kernel channels  $K$ , the CPU, GPU and NPU will perform convolution calculations on the input data using convolution kernels of  $K^C$ ,  $K^G$  and  $K^N$  channels respectively, where  $K = K^C + K^G + K^N$ . Then, the calculation amount of CPU, GPU and NPU at this layer can be expressed as:

$$C_i^C = \frac{K^C}{K} C_i \quad (14)$$

$$C_i^G = \frac{K^G}{K} C_i \quad (15)$$

$$C_i^N = \frac{K^N}{K} C_i \quad (16)$$

Therefore, for layer  $L_i$ , using CPU, GPU and NPU to perform the calculation of this layer at the same time, the time required to perform calculations are  $\frac{C_i^C}{Speed_C}$ ,  $\frac{C_i^G}{Speed_G}$  and  $\frac{C_i^N}{Speed_N}$ , respectively. Then, the latency of executing layer L in parallel can be expressed as:

$$t_i^P = \max\left\{\frac{C_i^C}{Speed_C}, \frac{C_i^G}{Speed_G}, \frac{C_i^N}{Speed_N}\right\} \quad (17)$$

The latency is the lowest when the  $\frac{C_i^C}{Speed_C} = \frac{C_i^G}{Speed_G} = \frac{C_i^N}{Speed_N}$ . So, the lowest execution time of layer  $L_i$  is:

$$t_i = \sum_{i=1}^n \min\{t_i^P, t_i^C, t_i^G, t_i^N\} \quad (18)$$

The  $t_i^C = \frac{C_i}{Speed_C}$ ,  $t_i^G = \frac{C_i}{Speed_G}$  and  $t_i^N = \frac{C_i}{Speed_N}$  is the time required to execute layer  $L_i$  using only CPU, GPU and NPU, respectively. Then, in order to obtain the lowest inference delay of the neural network G, it is necessary to load-distribute the computing tasks of each layer in G, so that the inference delay of each layer in G is the lowest.

The heterogeneous inference problem of edge equipment under temperature constraints can be transformed into an optimization problem subject to certain constraints:

$$\begin{aligned} & \sum_{i=1}^n \min\{(t_i^P + t_i^m), t_i^C, t_i^G, t_i^N\}, i \in \{1, 2, \dots, n\} \\ & \text{s.t. } \kappa_2(Speed_C)^3 + \kappa_3(Speed_G)^3 \\ & \quad + \kappa_4(Speed_N)^3 \leq A \\ & A = T_{max} - \kappa_1 T_{env}(t) - \kappa_0 \\ & Speed_C \leq Speed_{C_{max}} \end{aligned}$$

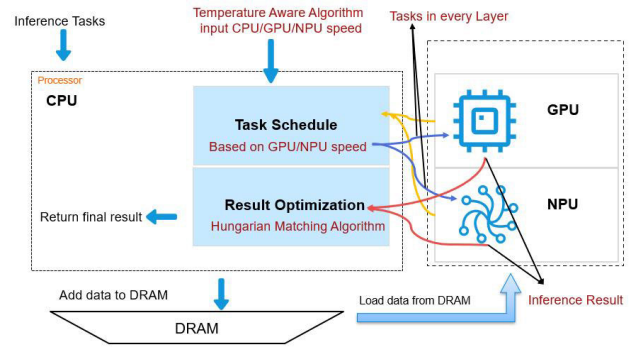


FIGURE 2. Overall architecture.

$$\begin{aligned} Speed_G & \leq Speed_{max}^G \\ Speed_N & \leq Speed_{max}^N \\ t_i^P & = \frac{C_i^C}{Speed_C} = \frac{C_i^G}{Speed_G} = \frac{C_i^N}{Speed_N} \end{aligned} \quad (19)$$

The  $Speed_{max}^C$ ,  $Speed_{max}^G$ ,  $Speed_{max}^N$  represents the maximum floating-point operation speed of CPU, GPU and NPU in D respectively,  $P_{max}$  represents the maximum power consumption. It is necessary to maximize the computing power of the device by setting the floating-point operation speed of the CPU, GPU and NPU under the power consumption and temperature constraints of the device, and minimize the single-layer inference time of each layer in the neural network, so as to obtain the lowest delay inferred by the neural network of the terminal device, and ensure the long-term stable operation of the device.

#### IV. SYSTEM AND ALGORITHM DESIGN

The overall system architecture is shown in the Figure 2.

From the architecture diagram, we can see that the overall architecture can be divided into three parts:

1. TAS module: the main function is to calculate the operating speed of CPU, GPU and NPU that are most suitable for the current ambient temperature according to the TAS model established previously
2. Scheduling module: according to the chip running speed determined by the TAS module, the tasks of each layer can be further scheduled to GPU and NPU for execution according to the task scheduling algorithm
3. Optimization module: according to the inference result of GPU, lower the confidence threshold of NPU and match it with the inference result of GPU combined with Hungarian matching algorithm. Through this joint optimization, reduce the precision loss caused by NPU inference

##### A. TEMPERATURE AWARE ALGORITHM

In order to achieve the optimization goals, this paper proposes a temperature-aware scheduling algorithm, TAS, and the main algorithm flow is shown in the algorithm 1. When device D receives an inference task from a neural network G, it first judges according to the ambient

temperature. If the device can perform the task with maximum performance under this temperature, the floating-point operation speed of the CPU, GPU and NPU is set to  $Speed_{max}^C, Speed_{max}^G, Speed_{max}^N$ ; otherwise, the algorithm will adjust the floating-point speed of the CPU, GPU and NPU to meet the temperature constraints. This paper uses a fast multi-parameter search strategy based on binary search to quickly set the floating-point operation speed of CPU, GPU and NPU.

The complexity of judging whether the maximum floating-point operation speed conforms to the constraint is  $O(1)$ , and the complexity of adjusting the CPU and GPU floating-point operation speed is  $O(\log(Speed_{max}^C + Speed_{max}^G + Speed_{max}^N))$ , therefore, the complexity of algorithm 1 is  $O(\log(n))$ .

---

### Algorithm 1 TAS Algorithm

---

**Data:**  $Speed_{max}^C, Speed_{max}^G, Speed_{max}^N, Power^{idle}, T_{max}, T_{env}$   
**Result:**  $Speed_C, Speed_G, Speed_N$   
 # If the current system temperature meets the threshold, the GPU and NPU can be at the maximum speed if  $\kappa_2(Speed_C)^3 + \kappa_3(Speed_G)^3 + \kappa_4(Speed_N)^3 \leq T_{max} - \kappa_1 T_{env}(t) - \alpha_0$   
 then  
      $Speed_C = Speed_{max}^C$   
      $Speed_G = Speed_{max}^G$   
      $Speed_N = Speed_{max}^N$   
 else  
 # If the current temperature exceeds the threshold, find the maximum CPU, GPU and NPU speed under the temperature threshold according to the dichotomy  
      $Speed_{low}^C = 0, Speed_{low}^G = 0, Speed_{low}^N = 0$   
      $Speed_{high}^C = Speed_{max}^C, Speed_{high}^G = Speed_{max}^G$   
      $Speed_{high}^N = Speed_{max}^N$   
      $Speed_C = \frac{Speed_{max}^C}{2}, Speed_G = \frac{Speed_{max}^G}{2}$   
      $Speed_N = \frac{Speed_{max}^N}{2}$   
 while  $\alpha_2(Speed_C)^3 + \alpha_3(Speed_G)^3 + \alpha_3(Speed_N)^3$  not in  $[F - \sigma, F]$   
 do  
     Using the binary search to get the result  
 return  $Speed_C, Speed_G, Speed_N$

---

## B. HETEROGENEOUS TASK ASSIGNMENT

When we get the GPU and NPU floating-point operation speed base on the temperature, we need to allocate computing load according to the computing performance of CPU and GPU, as well as the computing volume and structural characteristics of each layer in the neural network. The detail algorithm is shown as algorithm 2.

Algorithm 1 output the floating-point operation speed of GPU and NPU. Then for the neural network G, which may contain the convolutional layer, fully connected layer and pooling layer, it needs to determine how to distribute the workload between GPU and NPU. In algorithm 2, we first

distribute workload for each layer based on the floating-point operation speed of GPU and NPU. Then according the distributed load of GPU and NPU, it can get the inference delay of each layer in the neural network using parallel computing (GPU + NPU),  $t_i^P$ . Then we can calculate the latency when only using the GPU or NPU,  $t_i^G, t_i^N$ . Then we compare the latency, when the  $t_i^P$  is the lowest, we will use the parallel computing (GPU + NPU), otherwise we will only use GPU or NPU. The complexity of Algorithm 2 is  $O(n)$ .

---

### Algorithm 2 TAS-Based Task Schedule(TASTS)

---

**Data:**  $Speed_C, Speed_G, Speed_N$ , each layer load  $C_i$  for neural network G  
**Result:**  $C^G, C^N$   
 for  $i = 1$  to  $n$  do  
     # Initialize the task allocation ratio of each layer of GPU and NPU according to the running speed of NPU and GPU set in algorithm 1  
      $C_i^G \leftarrow \frac{C_i Speed_G}{Speed_G + Speed_N}$   
      $C_i^N \leftarrow \frac{C_i Speed_N}{Speed_G + Speed_N}$   
     # If the current GPU processing time delay is minimum  
     if  $\min\{t_i^P, t_i^G, t_i^N\} = t_i^G$  then  
          $C_i^G \leftarrow C_i, C_i^N \leftarrow 0$   
     # If the current NPU processing time delay is minimum  
     else if  
          $C_i^N \leftarrow C_i, C_i^G \leftarrow 0$   
     # If the current GPU and NPU work together to minimize the time delay  
     else  
         break  
 return  $C^G, C^N$

---

## C. OPTIMIZATION

For the inference results obtained by NPU and GPU, the Hungarian algorithm is used to match the inference results on the CPU side. The main matching principles are as follows:

1. GPU and NPU speculate different frames respectively
2. The threshold of GPU is set higher to ensure the detection accuracy
3. The threshold of NPU is set low to get more detection frames
4. Match the detection frames of the two and filter out inaccurate frames

## V. TEST AND ANALYSIS

In the previous algorithm 1 and 2, we set the floating-point operation speed of the processor under the temperature constraint, and at the same time assign tasks according to the speed and the structure of each layer of the neural network. In our experiment, we use the XiaoMi 11 mobile phone, which has Qualcomm Kryo 680 CPU with 1.8GHz, Qualcomm Adreno 660 GPU with 1.8GHz,

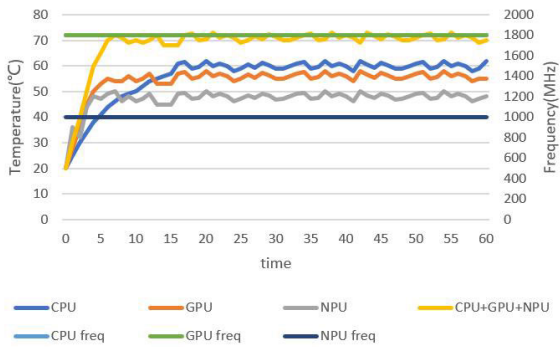


FIGURE 3. Device temperature and frequency under 20°C.

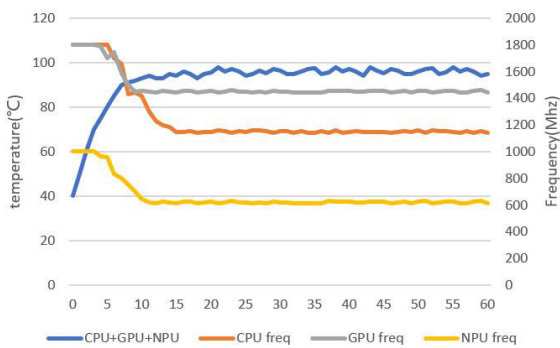


FIGURE 4. Device temperature and frequency under 40°C.

Qualcomm Hexagon 780 NPU with 1GHz. The model we use is YOLOV5sVGG16 and AlexNet.

In this paper, we first conduct experiments at room temperature of 20°C, and observe the temperature changes of the device after using the CPU, GPU, NPU, and CPU+GPU+NPU simultaneously for a period of time. As shown in Figure 3, with the operation of the device, the temperature gradually increased and gradually stabilized to around 70°C. During the entire running process, the CPU, GPU, NPU, and when using CPU+GPU+NPU at the same time, its clock frequency can always maintain a stable maximum value, mainly because the device temperature has not reached the maximum operating temperature.

Next, the effect of high-temperature environments on device performance was observed at higher ambient temperatures. The Figure 4 shows the calculation using all CPU, GPU, GPU at an ambient temperature of 40°C. It can be found that with the operation of the device, the temperature of the device rises rapidly, and after a period of time, it has exceeded 80°C. At this time, the clock frequency of the CPU, GPU and NPU also began to fluctuate, and the clock frequency of the CPU dropped from the initial 1800MHz to 1150MHz, the clock frequency of the GPU dropped from the initial 1800MHz to 1450MHz, the clock frequency of the NPU from the initial 1000MHz to 620MHz, indicating that the ambient temperature has a significant impact on the terminal equipment.

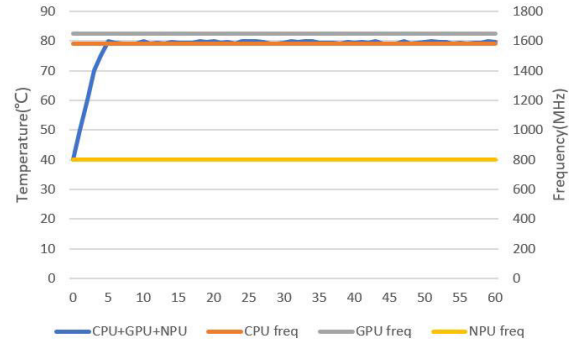


FIGURE 5. Device temperature and frequency with temperature-aware(TAS) under 40°C.

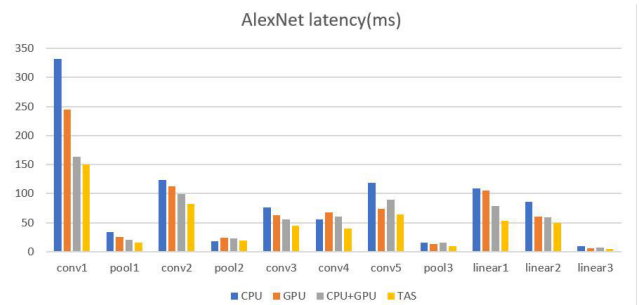


FIGURE 6. AlexNet performance.

Finally, through the TAS algorithm proposed in this paper, after setting the temperature threshold to [78,80], the optimal CPU, GPU, GPU floating-point operation speeds under this temperature threshold are calculated in an environment with a room temperature of 40°C. After setting the clock frequency of CPU, GPU and NPU as 1580MHz, 1650MHz and 800MHz respectively, Figure 5 shows the experimental results under this setting. Under this setting, as the device continues to operate, the temperature of the device continues to rise, and finally reaches a stable temperature around 78.5°C and fluctuates around this temperature. During the operation, the processor always maintains a stable clock frequency, no frequency drop occurred.

The above experiments show that by controlling the clock frequency of the processor of the edge device, the power consumption of the device can be adjusted, thereby adjusting the temperature of the device during operation. Therefore, in the neural network inference of intelligent terminal equipment, by setting the GPU and NPU floating point operation speed of the equipment according to the temperature constraints, the stable operation of the equipment can be guaranteed and the performance of the equipment can be improved.

The Figure 6, 7 and 8 shows the results of the above four methods of inference for the AlexNet, VGG16 and Yolov5s models. We can find that compared with a single processor, the GPU+NPU method has better performance in most scenarios, but due to the fixed task distribute ratio, each model layer is not well considered the characteristics and power consumption issues, so there is a scenario where the delay is

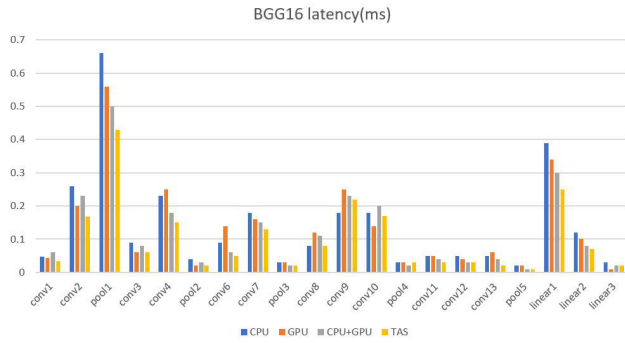


FIGURE 7. VGG16 performance.

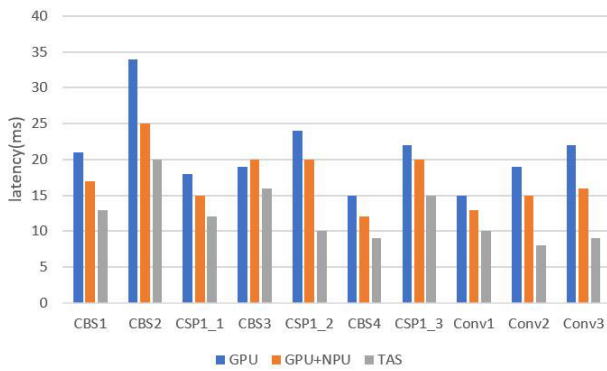


FIGURE 8. Yolov5s performance.

greater than that of a single processor. The TAS algorithm in this paper dynamically changes the running speed of the processor according to the ambient temperature, so as to ensure the optimal running speed without affecting the performance. At the same time, according to the task amount of each layer and the current processor Speed, dynamically adjust the task allocation of each layer, so as to achieve the optimal performance. From the data, we can also see that in the AlexNet, VGG16 and yolov5s scenarios, the average delay of the our temperature-aware CPU+GPU+NPU schedule(TAS) can get 20-40%, 16.6-29.9% and 29-50% performance benefit compared with traditional operation method.

VI. CONCLUSION

In order to solve the problems of weak processor performance of terminal devices and high inference delay of neural network, heterogeneous collaborative inference acceleration devices such as CPU + GPU/NPU have been proposed. However, the architecture and technical implementation of heterogeneous devices are different, and it is necessary to combine the characteristics of the model to perform optimal scheduling of inference tasks. At the same time, the complexity of the environment where the task is located may affect the performance of the edge device. Therefore, this study conducts task modeling based on the heterogeneity of edge device, and constructs a temperature-based energy consumption constraint model. Finally, a task scheduling

algorithm TAS (Thermal Aware Schedule) based on heterogeneous devices is proposed. Through multiple sets of experiments, it is proved that the scheme in this study has higher performance under energy consumption constraints.

REFERENCES

- [1] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern Recognit.*, vol. 29, no. 1, pp. 51–59, Jan. 1996.
- [2] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, 1999, pp. 1150–1157.
- [3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 886–893.
- [4] 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 Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.
- [5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
- [6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.
- [7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.
- [8] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," 2016, *arXiv:1604.03168*.
- [9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
- [10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
- [11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, and Q. Le, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–9.
- [13] D. Tarditi, S. Puri, and J. Oglesby, "Accelerator: Using data parallelism to program GPUs for general-purpose uses," *ACM SIGPLAN Notices*, vol. 41, no. 11, pp. 325–335, Nov. 2006.
- [14] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014, *arXiv:1404.5997*.
- [15] O. Yadan, K. Adams, Y. Taigman, and M. Ranzato, "Multi-GPU training of ConvNets," 2013, *arXiv:1312.5853*.
- [16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, and J. Klingner, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.
- [17] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, N. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2430–2439.
- [18] Z. Jia, S. Lin, C. R. Qi, and A. Aiken, "Exploring hidden dimensions in accelerating convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2274–2283.
- [19] 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," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2016, pp. 1–12.
- [20] 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.*, Mar. 2019, pp. 1–15.
- [21] F. Zhang, J. Zhai, B. Wu, B. He, W. Chen, and X. Du, "Automatic irregularity-aware fine-grained workload partitioning on integrated architectures," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 867–881, Mar. 2021.



- [22] C. Wang, Y. Yang, and P. 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, Feb. 2021.
- [23] S. Moulik, R. Devaraj, and A. Sarkar, "HETERO-SCHED: A low-overhead heterogeneous multi-core scheduler for real-time periodic tasks," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun., IEEE 16th Int. Conf. Smart City, IEEE 4th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Jun. 2018, pp. 659–666.
- [24] P. P. Nair, R. Devaraj, and A. Sarkar, "FEST: Fault-tolerant energy-aware scheduling on two-core heterogeneous platform," in *Proc. 8th Int. Symp. Embedded Comput. Syst. Design (ISED)*, Dec. 2018, pp. 63–68.
- [25] S. Moulik, A. Sarkar, and H. K. Kapoor, "TARTS: A temperature-aware real-time deadline-partitioned fair scheduler," *J. Syst. Archit.*, vol. 112, Jan. 2021, Art. no. 101847.
- [26] S. Moulik, Z. Das, R. Devaraj, and S. Chakraborty, "SEAMERS: A semi-partitioned energy-aware scheduler for heterogeneous multicore real-time systems," *J. Syst. Archit.*, vol. 114, Mar. 2021, Art. no. 101953.
- [27] M. Hussain, M. I. Siddique, and F. Ahmed, "ETA-HP: An energy and temperature-aware real-time scheduler for heterogeneous platforms," in *Proc. 27th IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2022, pp. 561–564.
- [28] S. Moulik, Z. Das, and G. Saikia, "CEAT: A cluster based energy aware scheduler for real-time heterogeneous systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2020, pp. 1815–1821.
- [29] M. C. Lim, C. W. Tan, S. W. Ng, and Y. Zhang, "RT-SEAT: A hybrid approach based real-time scheduler for energy and temperature efficient heterogeneous multicore platforms," in *Proc. 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2022, pp. 457–462.
- [30] T. Rahman, M. I. Siddique, and F. Ahmed, "FATS-2TC: A fault tolerant real-time scheduler for energy and temperature aware heterogeneous platforms with two types of cores," in *Proc. 35th IEEE Int. Conf. Microelectron. (ICM)*, Dec. 2022, pp. 321–324.
- [31] J. Li, L. Jiao, G. Li, and Z. Li, "CETAS: A cluster based energy and temperature efficient real-time scheduler for heterogeneous platforms," in *Proc. 27th IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2022, pp. 1–13.



**XIANG GAO** was born in Dalian, China, in 1991. He received the master's degree from the National University of Defense and Technology, in 2017.

He is currently with Research Institute of China Telecom Corporation Ltd., engaged in the research of cloud computing, heterogeneous computing, and other advanced computing technologies.

• • •