

RESEARCH ARTICLE

Dual-Core-Based Microcontrollers Inference Design and Performance Analysis

DONGCHAN LEE¹, JEONG-SI KIM², AND SEUNGTAE HONG^{1,2}¹Artificial Intelligence, University of Science and Technology (UST), Daejeon 34113, South Korea²Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, South Korea

Corresponding author: Seungtae Hong (sthong@etri.re.kr)

This work was supported by the Challengeable Future Defense Technology Research and Development Program through the Agency for Defense Development (ADD) funded by the Defense Acquisition Program Administration (DAPA), in 2022, under Grant 915062201.

ABSTRACT As the frontier of Artificial Intelligence (AI) expands, embedding AI models into compact devices, particularly in microcontroller units (MCUs), becomes increasingly critical. This study introduces an innovative approach leveraging a dual-core architecture, notably the STM32H747 MCU, to significantly elevate inference efficiency in MCUs beyond the traditional single-core configurations. We detail the adaptation of lightweight AI models to this architecture, emphasizing preprocessing, memory sharing techniques, and the independent operation of dual cores. Our comparative analysis reveals substantial enhancements: data processing speeds in MCUs increase by up to 6 times, and overall processing efficiency improves by up to 30% compared to conventional single-core systems. These advancements not only underscore the dual-core system's capability to surpass previous benchmarks but also its potential to drive significant performance and speed enhancements for real-time AI applications in embedded systems. This work sets a new precedent for optimizing AI deployment on resource-constrained devices, opening avenues for more sophisticated AI applications in the realm of embedded computing.

INDEX TERMS MCU, dual-core, AI, TensorFlow lite micro, embedded system.

I. INTRODUCTION

Recent rapid advancements in AI technology have spurred efforts to integrate AI functionalities into portable devices [1]. However, portable devices are resource-constrained and have many limitations in performing training and inference, which require many resources. One common approach involves offloading input data from the device to a large server for computation, followed by sending the results back to the device. However, this approach requires additional communication and power in the personal device environment, which increases the cost and size. In addition, the more communication the server needs, the more stressful the server becomes, and there are limitations in transmitting sensitive data, which makes it difficult to apply machine learning on embedded devices.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Khalil Afzal¹.

TABLE 1. STM32H747 disco board and peripherals.

CORE	ARM-CortexM7 & ARM-CortexM4
RAM	1Mbyte
FLASH	2Mbyte
LCD	NT35510
SDRAM	is42s32800j
CAMERA	OV5640

Recently, many companies have adopted on-device AI that recognizes these limitations and executes AI services within a small device [2]. Such on-device AI can provide price competitiveness and ultra-personalized services using personal information obtained by applying AI to remove fewer parts and intermediate servers within the device, emphasizing the role of personal assistant in the hand. To this end, a lot of research has been conducted on how to obtain a model with high accuracy as the model becomes smaller.

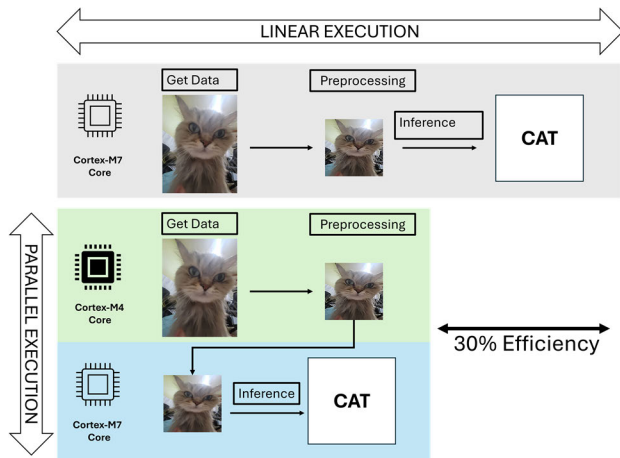


FIGURE 1. Single-core and dual-core differences.

MCUs are typically smaller than cell phones, with limited system resources of tens to hundreds of kB. However, due to their competitive price and low power consumption, MCUs are used in a wide variety of applications, from automotive parts to TV remote controls. The integration of AI into MCUs addresses a crucial aspect of AI deployment, particularly for mitigating disparities. While integrating AI into MCUs is appealing, it presents several limitations. Due to their limited memory, MCUs inherently restrict the size of AI models. Furthermore, their low computing power may hinder the execution of complex calculations or lead to significant delays, thereby failing to achieve real-time processing performance. Thus, the successful implementation of AI models in embedded systems depends on how efficiently and quickly these systems can process computations to achieve real-time execution. Therefore, successfully implementing AI models in the embedded systems depend on how efficiently and quickly these systems can process computations to achieve real-time execution. [3].

The integration of AI capabilities directly into MCUs presents a unique set of challenges and opportunities. The limited computational resources and power availability of traditional single-core MCUs often impose memory constraints on the performance of AI models, restricting their complexity and the speed at which they can operate. To address these challenges, our research investigates the potential of utilizing dual-core MCUs for AI applications. Utilizing dual-core MCUs allows for the separation of tasks by core, enabling fast and efficient operations. By dividing the tasks of data preprocessing and AI inference between two cores, MCUs can achieve higher efficiency and faster processing times. This dual-core functionality can reduce the burden on a single core without additional power consumption. This dual-core approach not only enhances the performance of AI models on MCUs but also opens up new possibilities for real-time, energy-efficient AI applications in embedded systems.

In this paper, we propose a method to perform system control and inference based on dual cores to efficiently perform

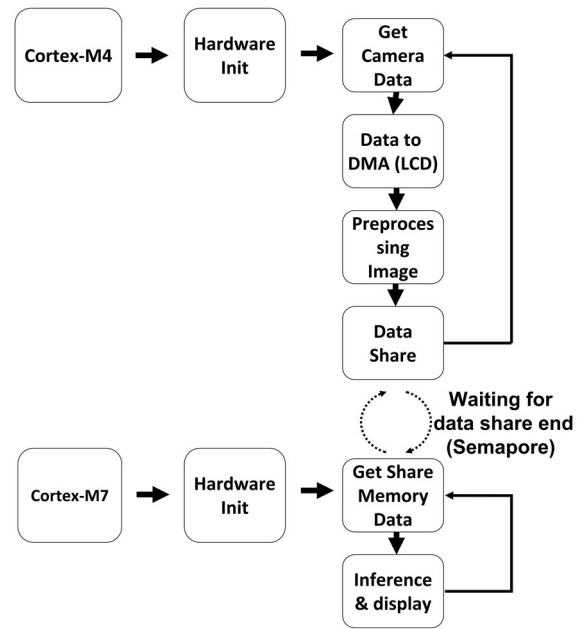


FIGURE 2. System processes.

artificial intelligence on MCUs with limited system resources as shown in Figure 1. This method enables more efficient execution through parallel execution of the two cores rather than sequential execution. To address this, we designed a system utilizing the STM32H747 Disco board equipped with an STM32H747 [4] MCU configured with Arm Cortex-M4 and Cortex-M7 dual cores. The system consisted of the components listed in Table 1. STM32H747 features Cortex-M4 and M7 on one chip, and the two cores can run independently and perform tasks in parallel for fast task processing. The Cortex-M series is a CPU that is a CPU family typically used in MCUs to MCUs, and the Cortex-M7 offers the highest performance within the current M series, while the Cortex-M4 is a mid-range performance product. In general, M7 has a double precision decimal point unit, which is a superior calculation performance than M4, so the M4, which had less computational power than the M7, performed communication and input/output. In this paper, we perform data input and processing on M4 as an offloading method, and execute deep learning inference on M7, which has superior computational power. The proposed design resembles an offloading strategy, where data input and initial processing occur on the M4 core, while the more powerful M7 core handles deep learning inference. Although this approach demands a sophisticated processing and memory architecture, it yields superior performance in processing speed.

In this study, we develop a dual-core MCU inference system designed to achieve both rapid and consistent data processing. By leveraging the parallel processing capabilities of a dual-core architecture, our system notably enhances inference speed, making it highly suitable for applications requiring swift data analysis. Furthermore, this paper focuses

on the ambitious goal of enabling real-time vision processing directly from cameras on MCUs, despite the inherent limitations in system resources typically associated with such devices. Through this work, we address the critical challenge of optimizing computational efficiency and processing speed, paving the way for advanced real-time applications on resource-constrained embedded systems. First, we improve the overall deep learning execution performance by using dual cores at the same time, where each core is responsible for system control and deep learning inference. Second, we propose a data sharing space that utilizes shared memory to allow different cores to access the same memory when utilizing dual cores. Finally, we propose an image preprocessing technique for efficient image preprocessing on MCUs. Section II introduces related work on applying inference in MCUs. Section III explains the methods and applications used in this paper. In Section IV, we compare and explain the difference between using dual cores and single cores in terms of execution time and power, conclude this work in Section V. In this paper, we utilize dual cores for higher frames than previous studies and perform the operation. The system configuration, as shown in Figure 2, maximizes the use of dual cores within the system to enhance performance. We propose a design that enables faster inference than the existing MCU-based deep learning process by dividing M4 into setting up and executing peripheral devices such as cameras, and M7 into inference that requires mathematical calculations. Furthermore, to effectively manage this dual-core system, we introduce an innovative design that incorporates shared memory processing. This design entails a hardware-oriented approach utilizing shared memory for efficient data management, coupled with an advanced method for image preprocessing. By conducting a comparative analysis of these strategies, we have identified a pathway to significantly optimize the overall inference time. Our findings demonstrate a marked improvement in performance, particularly in the realms of real-time processing and enhancement of systems with traditionally lower performance capabilities. This advancement is especially noteworthy in the context of MCUs, where such enhancements can be pivotal.

The main achievements of our research are highlighted as follows:

- Introduction of a pioneering method for enhancing AI model inference on dual-core microcontrollers, particularly using the STM32H747 MCU. This method leverages the dual-core architecture to improved inference speed and system responsiveness in resource-limited environments.
- Development of a system architecture that leverages shared memory for efficient data transfer between the Cortex-M4 and Cortex-M7 cores. This setup allows for optimized concurrent processing of data preprocessing and deep learning inference, leading to enhanced AI application performance on MCUs.
- Through our experiments, we have demonstrated the significant contributions of the dual-core method. We have

verified that the dual-core approach maintains comparable inference speed to that of a single core, while simultaneously achieving 6 times increase in data processing and 30% faster total processing time, all without additional power consumption. These findings underscore a substantial performance enhancement compared to existing single-core systems. Importantly, our study highlights the profound impact of MCUs not only on artificial intelligence but also on real-time processing systems.

II. RELATED WORK

Despite increasingly large models, there are many attempts to apply AI models on MCUs with limited system resources. Through the development of optimized inference engines and model lightweighting, AI is being applied to MCUs with limited system resources. The performance of MCUs is also increasing with the advent of dual-core MCUs. The advantage of dual-core microcontrollers is that the two cores operate independently, which is superior to previous MCUs, and the expectation of applying AI in MCUs is growing. Most existing studies focus on single-core MCUs, which is different from our research.

A. SOFTWARE

The Embedded Learning Library (ELL), developed by Microsoft, was developed to move away from cloud-based MCU inference and compiles models into binary [5]. MicroNet is a project led by ARM that released and distributed models with increased performance by optimizing the ARM core, which is also used in this paper [6]. In this paper, we also conducted experiments with the models distributed by MicroNet. Tensorflow, a well-known name among deep learning engineers, has also released Tensorflow Lite Micro (TFLM) for running models on MCUs [7]. This framework is a lightweight development of Tensorflow Lite for use on existing mobile and edge devices and uses an interpreter approach to store all expressions and call functions when needed. This approach is easy to adapt for MCUs and allows the model to be changed freely.

In this paper, we also used TFLM to conduct experiments. However, the practice of storing all operations presents an optimization challenge, so MCUNet, a study conducted by MIT, stores, and executes only the necessary operations [8]. This optimization resulted in higher accuracy and optimization, following this study, researchers introduced MCUNet V2 [9], which uses patch-based inference to reduce peak memory usage by dividing images into smaller parts. This approach demonstrated impressive AI performance on MCUs by achieving lower peak memory usage, higher accuracy, and the ability to perform multiple recognitions. This study high resolution of the optimization resulted in real-time inference at 7.3 frames per second (FPS), which is faster than previous studies. As a follow-up study, the researchers conducted learning on the MCU and published a paper that enables learning and inference on the MCU itself [10]. These studies

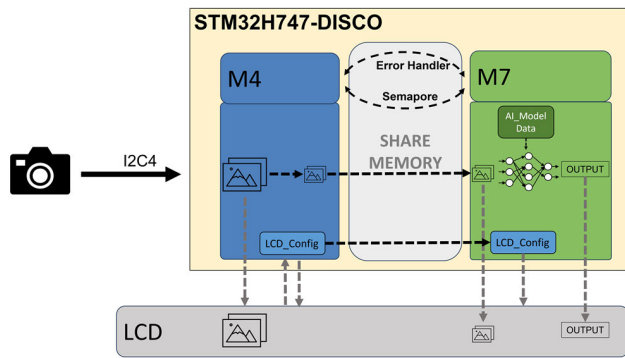


FIGURE 3. System configuration and operation.

highlight the remarkable advancements in applying AI within traditional MCUs.

STMicroelectronics, the developer of the board used in this paper, has also developed a framework for optimization [11]. In previous work, the problem of adapting models to different MCUs has been a challenge, which STM32CubeAI [11] has solved by using their STM32CubeIDE [12] on STM32 MCUs or in the cloud [13]. A comparative study was conducted to evaluate different frameworks for inferring time-sequenced data. Specifically, this study compared CubeAI and TFLM frameworks [14]. The research demonstrated that the process of converting models into C language did not result in any data loss. Moreover, it provided an in-depth comparison between STM32CubeAI and TFLM, highlighting their respective strengths and weaknesses. STM32CubeAI optimizes models for better performance in terms of memory usage and accuracy. However, unlike the open-source TFLM, STM32CubeAI presents licensing issues.

STM32 Model Zoo [15], designed to offer a collection of optimized machine learning models for execution on STM32 microcontrollers, includes models specifically for image classification. Among these, it was demonstrated [16] that the image classification models achieve a performance of 9 FPS, or approximately 111.11 milliseconds (ms) per frame, on the STM32H747. A paper comparing these various frameworks [17] allows for a performance comparison between TFLM and STM32CubeAI. Additionally, this paper introduces a new framework called microAI, demonstrating that it can achieve performance that of STM32CubeAI on other MCUs.

B. HARDWARE

Based on the above research, there have been many attempts to apply it to Arduino, the most popular MCU [18]. However, although Arduino is easy to program, its performance is limited. Therefore many researchers have primarily applied it to high-performance MCUs such as STM32 and NXP series [19]. Dual-core based MCUs have been developed by several companies in addition to the STM32H7 series used in this paper. In response to this demand, an article [20] provides a detailed overview of the technology, benefits,

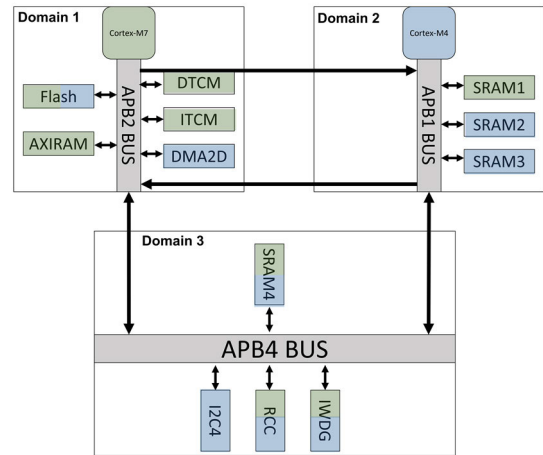


FIGURE 4. STM32H747 domain block diagram.

and implementation methods of dual-core MCUs. This article explains the advantages and applications of dual-core MCUs. The Raspberry Pi Pico is an MCU developed by the Raspberry Pi Foundation, which has two Cortex-M0+ cores with low performance and few IO pins, but it is a popular MCU due to its low chip price and high clock. The LPC4300 series is capable of running complex algorithms with the high performance of dual cores of Cortex-M0 and Cortex-M4. The i.MX RT1050 series, developed by the same company, is a Cortex-M4 and M7 with similar performance to the one used in our study. A study utilizing dual-core MCUs to efficiently use renewable energy exists [21]. The responsiveness and efficiency of dual-core MCUs were leveraged to maximize the performance of the inverter system.

III. STRUCTURE DESCRIPTION AND DESIGN METHOD

The STM32H747 adopts a dual-core system with two cores, Cortex-M7 and Cortex-M4. Each core can run independently of the other by connecting them with individual buses, and information can be shared between the buses to share information. In addition, data processed by one core can be stored in a shared memory area to share information between cores. However, a strategy for memory storage is essential. We outline these rules in Section III-B.

The Cortex-M4 is a single-precision floating-point unit and is strong at calculating real numbers. However, the Cortex-M7 is a step up from the Cortex-M4 and has more computational power and is currently the highest performing Cortex-M. Its double-precision floating-point unit allows it to calculate real numbers better than the Cortex-M4, which has a single-precision unit. With these features, in this paper, Cortex-M4 is used for sensor processing and data processing, and Cortex-M7 is used for artificial intelligence that requires a lot of computation. Figure 3 shows the effective utilization of the dual-core architecture by separating processing roles so that each core operates autonomously, similar to using two separate MCUs within a single MCU. However, such complexity requires sophisticated design and processing techniques, as detailed in Section III-A.

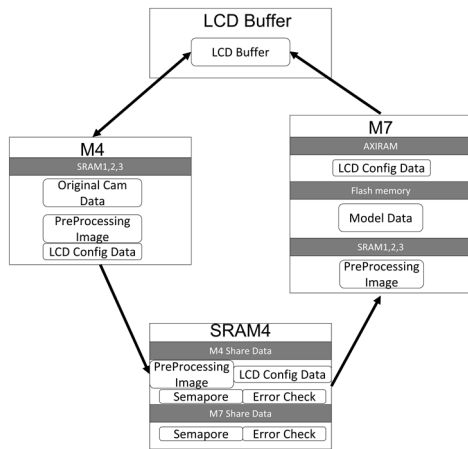


FIGURE 5. Memory configuration and structure.

Since it is a dual-core system, each core has a different allowable clock. Therefore, each core has its own power domain, and there is one domain that contains the other devices, for a total of three power domains. Figure 4 explains these relationships. Because of these features, the memory is distributed, resulting in a complex memory area with memory divided across multiple domains. A bus is connected between these domains to allow information to be shared between them. However, the speed of these buses is limited, so a feature called Direct Memory Access (DMA) is used to support fast data movement between cores. This method enables fast memory access by transfers data without accessing the internal CPU. So, DMA is used for LCD graphics processing [22].

There are many different formats for storing images. The devices employed in this study utilize a fixed image storage method, necessitating image preprocessing to address data transmission constraints. Within this system, data is received from the camera in RGB565 format, displayed on the LCD in ARGB888 format, and subsequently converted to RGB888 format for model inference. Given the importance of minimizing latency in MCU operations, it is recommended to apply only the necessary computations. To optimize this intricate conversion process on the MCU, we detail the methodology in Section III-C.

A. OPTIMIZING AI INFERENCE: THE DUAL-CORE APPROACH

This step explains the system design applied. The STM32H747 is a dual-core system with distributed memory. Unlike other MCUs, the memory address is complex and consists of multiple SRAM regions [23]. As shown in Figure 4, each core can access different memory regions, so we need to allocate memory areas and devices accordingly. In this paper, we specified the memory area as shown in figure 5 and conducted experiments. The feature of this system is that it can be configured to execute different tasks on a dual-core system to achieve efficient and fast operation. The Cortex-M4 performed hardware initialization and data

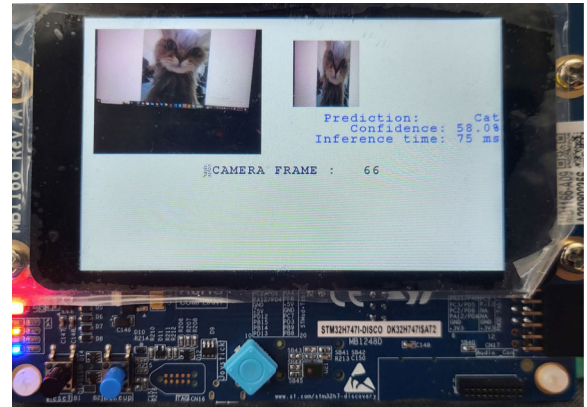


FIGURE 6. System operation and design.

input, and the Cortex-M7 fetched data and performed inference.

1) CORTEX-M4

First, we configured the hardware. To do this, the memory to be used in the hardware was allocated according to each hardware characteristic such as SDRAM and SRAM, and the hardware was configured using the Board Support Package (BSP) [24]. The BSP provided by default shows a slow speed of 7.5 FPS, so we modified the BSP to modify the clock register value of the camera in I2C4 to output a maximum of 40 FPS [25], and in this paper, we adjusted the clock to 30 FPS, which is a standard frame. Next, we proceed to set up the shared memory. When setting up BSP, hardware initialization is performed to allocate power and memory. However, if two cores initialize, they will crash each other, so it is not possible to use the same hardware on two cores at the same time in a normal way. To solve this problem, we share the hardware settings with shared memory so that two cores can use each other. In this paper, the LCD settings are stored in the shared memory area to use the LCD at the same time, and error control variables are stored to detect hardware errors. After the initialization, image acquisition is executed. The camera image data is displayed on the LCD using DMA2D [26]. Using the DMA2D function, the latency is reduced to 33ms from the existing 100ms processing speed. After that, preprocessing is performed. Preprocessing is performed to reduce the size of the input data in the AI model, thereby reducing the model size and to store it in a small shared data area. The preprocessing process is explained in more detail in Section III-C. The camera updates every 33 ms, which is 30 FPS, but due to the preprocessing, it processes the camera data every 66 ms, which is 15 FPS. Every 66 ms, the preprocessed image is stored in a shared memory area in domain 3.

2) CORTEX-M7

After setting up the shared memory area, proceed with the hardware setup. The shared memory area is stored at a

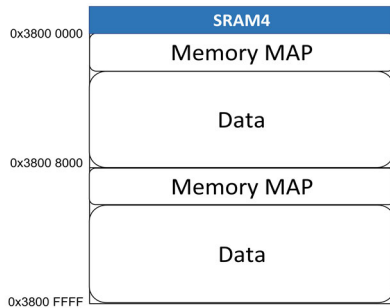


FIGURE 7. Shared memory partitioning and components.

different address in the same way as M4. The shared memory area stores the LCD setting data saved from the Cortex-M4 area, so the Cortex-M7 configures the hardware without the hardware initialization. After that, we configure the tasks required for inference. In this paper, we use the MicroNet model using TFLM to inference CIFAR-10-based data. After setting the tensor region and the operations to be used, we finish the initialization. The input data is processed on the Cortex-M4, and the Cortex-M7 inferences based on the input data stored in shared memory. It receives the data using the error control function that will be described in Section III-B because data crashes may occur while reading data or writing data. We infer it and measure the inferred value, accuracy, and time.

3) RESULT

The system operated as depicted in Figure 6, with memory utilization reaching 88% on the Cortex-M4 and 40% on the Cortex-M7. Impressively, inference speeds of 66 ms for image input and preprocessing (Cortex-M4) and 74 ms for inference were attained, ensuring swift camera frame processing and inference speeds.

B. EFFICIENT DATA EXCHANGE THROUGH SHARED MEMORY ARCHITECTURE

The STM32H747’s architecture includes two cores, necessitating a shared memory area (SRAM4) that is accessible by both cores for efficient data exchange and processing. In this paper, the SRAM4 region is selected to be accessible by both cores. The SRAM4 region belongs to the D3 domain and can transfer data to the D1 and D2 regions [4].

As shown in Figure 7, the SRAM4 region is 64 KB in total, which requires a rule because two cores use the same region. Divide the shared memory in half or as much memory as one core needs (more if the preprocessing image is large) into two regions. Each region is given permission to create variables on each core but read and modify them freely on all cores.

Store data without specifying the data format so that various data can be input and output when entering data. Since the smallest unit of internal data management is byte (8bit), data is converted into one byte and inserted into memory one byte at a time. This storage method has the advantage of being able to store and output a wide variety of data, although

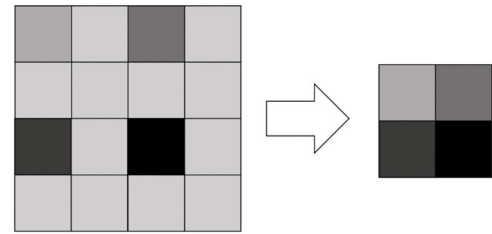


FIGURE 8. Image resize method.

the data format must be remembered by the developer when configuring the program because it is not specified inside the program. Due to the dual-core system, there may be a crash when reading data at the same time as data is written, so a memory lock variable is declared to make the data unreadable when modifying the data. In addition, each core can detect the internal state of the other core by creating a variable that can detect when an error such as hard fault error occurs on the other core, so that each core can detect the internal state of the other core. This is designed to be efficient for fault management and maintenance, as it can be restarted or stopped from running.

This setup allows the cores to share information and status among themselves, while increasing the reliability of their behavior. In this paper, the LCD is used to share information and check the status of each core.

C. ENHANCING INFERENCE SPEED WITH ADVANCED IMAGE PREPROCESSING TECHNIQUES

In general, the purpose of image preprocessing when performing inference on MCUs is to change the image format and reduce the input size of the inference model to create a lightweight model. In this paper, image preprocessing was performed for the same reason and to reduce the amount of data transfer in M4, the core that performs data acquisition, with limited shared memory capacity. While it is possible to use image processing libraries for image processing methods, doing so can introduce latency issues due to function calls and make it difficult to optimize memory usage because of unused functions and variables. Therefore, in typical edge embedded environments, it is recommended to implement and apply only the necessary computations. In our study, we also compared optimized computation methods and applied only the

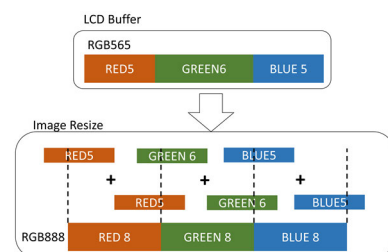


FIGURE 9. Change RGB565 image format.

Algorithm 1 Image Resizing Algorithm

```

1: Input: Camera buffer, originX, originY, resizeX, resizeY
2: Output: resize image
3: typedef struct {
4:   red8, blue8, green8
5: } RGB888
6: RGB888 resize image
7:
8: select_x = originX/resizeX
9: select_y = originY /resizeY
10:
11: for y to resizeY do
12:   for x to resizeX do
13:     pixel = camera_buffer[ y × select_y × cam width
+ x × select x ]
14:     red = ((pixel&0xf 800u) >> 11)
15:     red = red << 3|red >> 2
16:     green = ((pixel&0 × 07.0u) >> 5)
17:     green = green << 2|green >> 4
18:     blue = ((pixel&0 × 001f u) >> 0)
19:     blue = blue << 3 | blue >> 2
20:
21:     resize image.blue = blue
22:     resize image.green = green
23:     resize image.red = red
24:   end for
25: end for

```

necessary ones. Image preprocessing is a simple task, but it takes up a lot of CPU resources. There is a DMA2D method for processing images, but we could not apply it because we were already using the LCD display. This preprocessing slowed down the camera frame from 30 FPS to 15 FPS (66ms).

Since the MCU is difficult to process complex images, a simple reduction method was applied. The method described in Figure 8 divides the original image into several regions and selects a representative pixel as one of the regions, which allows for simple image processing. However, distortion may occur, and the image may be cropped depending on the image size due to processing speed issues. To solve this problem, adjusting the input size according to each device can not only improve the camera frame but also reduce distortion.



FIGURE 10. Changing the ARGB8888 format.

Algorithm 2 Image Resizing Algorithm

```

1: Input: lcd buffer, lcd_xpos, lcd_ypos, originX, originY,
resizeX,resizeY
2: Output: resize image
3: typedef struct {
4:   red8, blue8, green8
5: } RGB888
6: RGB888 resize image
7:
8: lcd_buffer = get_LCD_Memory()
9: Start address = lcd_ypos × LCD_Width + lcd_xpos
10:
11: RGB888 resize image
12:
13: Select x = originX/resizeX
14: Select y = originY/resizeY
15:
16: Buffer rgb.type = Resize image
17:
18: for y to resizeY do
19:   for x to resizeX do
20:     resize_image++ = lcd_buffer[Start_address + y ×
select y × LCD_Width +x ×select_x]
21:   end for
22: end for

```

The image format processing method is to pre-process the image in the camera buffer and transfer it from SRAM to SRAM, as shown in Figure 9. The camera image format is changed from RGB565 to RGB888. Since it is changing from 16bit format to 24bit format, the processing method is quite additional calculations and consumes a lot of processing time. Specifically, the 5-bit red, 6-bit green, and 5-bit blue components in RGB565 are expanded to 8-bit components in RGB888. This method is shown in Algorithm 1 requires some calculations.

Another method is to preprocess the image in the LCD buffer processed by DMA2D and convert it as shown in Figure 10. It is a transfer method from SDRAM to SRAM and changes the LCD format from ARGB8888 to RGB888. This method is a simple preprocessing method because it only needs to remove the transparency 'A' value, but it may be slow because the speed of accessing SDRAM is slower than SRAM. This is shown in Algorithm 2.

Comparing the two methods, the conversion from SRAM (RGB565) remained constant at 66ms. However, the conversion from SDRAM (ARGB8888) was not as consistent, ranging from 64 to 70ms. Despite the time required for calculations, this is due to the inconsistent speed when accessing SDRAM. This result shows that image processing in SRAM is stable and has a high average speed. In further experiments, we also performed the conversion from SRAM

TABLE 2. System-specific inference engine.

System	Engine
Model Zoo Inference (Cortex-M7)	STM32Cube AI
Single Core Inference (Cortex-M7)	Tensorflow Lite Micro
Dual Core Inference(ours) (Cortex-M4, Cortex-M7)	Tensorflow Lite Micro

IV. EXPERIMENT

In this section, we compare the system performance of single-core inference using only Cortex-M7 and a dual-core inference using Cortex-M4 and Cortex-M7 using STM32H747 with TFLM inference engine [5]. We configure the system with the MicroNet model to perform CIFAR-10 based image classification.

In this paper, we conduct a comprehensive performance evaluation, benchmarking the MicroNet model alongside three reference models from the STM32 Model Zoo, which are specifically optimized for image classification tasks on STM32 microcontrollers. Table 2 outlines the inference engines employed across different configurations within our study. Notably, STM32CubeAI [11], STMicroelectronics proprietary inference engine, operates exclusively on the Cortex-M7 core for model inference. For our experiments, we selected FD_mobileNET [26], MobileNetV2 [27], and SqueezeNet [28] from the STM32 Model Zoo. These models, optimized for STM32 microcontrollers, were evaluated in both single-core and dual-core setups to facilitate a direct comparison of their performance.

Below are the detailed contents used for inference:

- Model: MicroNet, FdMobileNet, MobileNetV2, SqueezeNet
- Camera HardWare: B-CAMS-OMV
- Input resolution: 320×240 RGB565
- Preprocessing resolution: 32×32 RGB888

We conducted about 10 experiments and always got the same results. It is speculated that the same result was obtained due to the stable operation of the MCU internal clock. The robustness of the MCU's hardware features ensured minimal system errors across multiple performance evaluations, validating the consistency of our findings.

Our comparative analysis focuses on execution speed and power consumption, illustrating the dual-core approach's efficiency over traditional single-core inference. Details on the sizes of the models and their respective input dimensions are provided in Table 3. Various models were tested as described in Section III "Structure description and design

TABLE 3. Model size and input size comparison.

Model	MicroNet (ours)	FdMobileNet	MobileNetV2	Squeezenet
Model Size	2,995kB	1,186kB	3,788kB	4,958kB
Input size	32x32	128x128	128x128	128x128

method" by capturing real objects with a camera and measuring the latency for image preprocessing and inference, as shown in Figure 6.

A. INFERENCE AND PROCESSING LATENCY MEASUREMENT

The overarching latency within our system is a composite of both data input/processing latency and the latency stemming from the inference engine itself. Traditional systems handle these stages sequentially, culminating in inevitable delays for both data preprocessing and inference operations. The introduction of a dual-core inference system, as proposed in our study, significantly mitigates these waiting times by facilitating concurrent processing. As detailed in Table 4, the latency benchmarks for each model across different systems reveal that our dual-core approach, when paired with the same inference engine, reduces latency by up to 30% for image classification tasks. This efficiency gain is directly linked to the system's capacity to parallelize data processing and inference, effectively reducing the latency associated with inference.

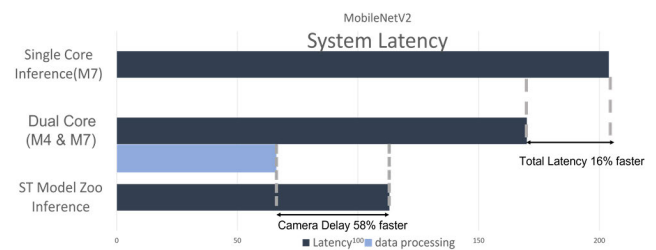
**FIGURE 11.** Compare latency differences for each system.

Figure 11 presents a detailed analysis of the latency performance of MobileNetV2 when operated under our dual-core system, particularly emphasizing data processing efficiency. Although initial observations might suggest that Model Zoo Inference offers superior inference speed, a deeper investigation reveals that its faster data processing advantage stems from a considerable waiting time attributed to data preprocessing in single-core configurations. In contrast, our dual-core inference architecture eliminates this waiting period altogether, facilitating immediate data processing. This unique attribute of the dual-core setup allows it to achieve an impressive by up to 6 times in processing speed compared to the conventional single-core system employing the same inference engine. The advantage of our dual-core system becomes even more distinct as the complexity and latency of the model's inference tasks escalate, highlighting its unmatched efficiency. Utilizing this dual-core architecture not only significantly boosts processing performance with identical inference engines but also paves the way for real-time processing applications. This is achieved by eliminating the latency commonly associated with inference tasks, thereby greatly enhancing the system's realism and responsiveness.

TABLE 4. Comparative analysis of total latency.

System	MicroNet(ours)	FD_MobileNet	MobileNetV2	Squeezenet1.1
Model Zoo Inference (Cortex-M7)	-	62ms	113ms	285ms
Single-Core Inference (Cortex-M7)	104ms	66ms	204ms	388ms
Dual-Core Inference (Ours) (Cortex-M4 & Cortex-M7)	74ms (-30%)	66ms (-0%)	170ms (-16%)	354ms (-10%)

B. POWER MEASUREMENT

The results of the current measurement are shown in Table 5. The current measurement was performed by connecting the board power via 5V (CN5 VIN pin) of the computer USB to the measurement device (oscilloscope) Since the power input current is measured, the current of various devices such as LCD and test LED are also included.

The formula for power is voltage current, so the greater the current, the greater the power. The measurement was performed for about 5 minutes.

The current measurements for each system showed that the power difference between the model and system on the STM32H747 was minimal. The power consumption was 288mA at startup and increased to 440mA when various peripherals such as LCD and camera were turned on, and the current increased up to 472mA.

When using STM32CubeIDE [12] to measure the current of the MCU alone in a power simulation measurement, the STM32H747XI used 224.56 mA, and the STM32H747, a dual-core inference MCU, operated for 12 hours on a single alkaline battery AA LR6. It can be assumed that the system described in this work will run for about 6 hours of continuous operation on a single AA battery. So, it is possible to use dual-core inference to improve performance at low power with no additional power consumption.

TABLE 5. System power comparison.

	Single Core Inference	Dual Core Inference	Model Zoo Inference
Max Current	472mA	472mA	476mA
Average Current	368mA	368mA	364mA

C. EXPERIMENT RESULT

The difference in latency between dual-core and single-core systems primarily stems from the system configuration.

Traditional single-core systems introduce additional latency by sequentially handling data processing and inference. Our proposed dual-core inference system significantly reduces this additional latency. By parallelizing data processing and inference, we achieved up to a 6-fold increase in data processing speed and up to a 30% reduction in inference time. This improvement is particularly significant for real-time processing on MCUs.

Energy efficiency was also notable, with no additional power consumption. The system operated for up to 6 hours on a single alkaline AA battery. Since no power-saving modes were used, the system's operational lifespan can be further extended with appropriate design.

V. DISCUSSION

A. SIGNIFICANCE OF THE PAPER

In traditional MCUs, the inference task was executed on a single core, which posed a significant problem for MCUs where real-time processing is crucial. In this paper, the tasks previously performed by a single core are separated into two processes, enabling fast and efficient inference operations. This separation increases real-time processing capability, allowing for smooth operation in various systems. It increased camera processing speed by up to 6 times and reduced total latency by 30%, enabling highly efficient real-time inference. In actual environments, sensor input and preprocessing can be performed on one core while inference is executed on another core, or different inferences can be executed on each core, resulting in more efficient and faster inference. This suggests that AI models will perform better on MCUs. Due to the limited performance of general MCUs, our research offers significant improvements. Our findings demonstrate how dual-core MCUs can significantly enhance the performance and efficiency of AI-powered systems.

B. COMPARISON WITH OTHER WORK

Based on our research results and the study [21] that improved processing speed using dual-core systems, we are confident that applying well-established AI models, such as those in the CubeAI framework from stm32modelzoo [15],

can achieve even shorter latency. Additionally, applying dual-core architecture to previous single-core research is expected to yield significant performance improvements. For example, MCUNetV2 proposed a patch-based inference method that divides images for processing, demonstrating high performance and accuracy with lower peak memory usage and multiple object detection. Despite these strengths, it encountered latency issues during the patch inference process. We are certain that combining the dual-core inference method with the patch inference technique will significantly reduce latency.

C. LIMITATIONS

By examining the efficient architecture within MCUs using existing models, this study aims to pave the way for significant improvements. Although we did not select models specifically tailored to the characteristics of MCUs and dual-core systems, the potential for enhancement is substantial. The methods employed in this paper utilize the Cifar-10 dataset and the MicroNet model. During the preprocessing stage, images are resized to 32×32 and transferred to another core via shared memory. This resizing process, however, faces several limitations due to maximum memory constraints and the size of the shared memory, resulting in several challenges:

1) ACCURACY WITH REDUCED RESOLUTION

Reducing image resolution has the advantage of using less memory, which is critical in resource-constrained environments like MCUs. However, it also makes the system more vulnerable to noise and reduces representational accuracy, affecting the overall accuracy of inference.

2) LATENCY IN PREPROCESSING

The MicroNet model based on the Cifar-10 dataset and the images from the camera use different formats and sizes, which introduced additional latency during preprocessing. Reducing this preprocessing time could lead to faster processing times overall.

Future research will address these issues with the following approaches:

3) MODEL CREATION

Developing models specifically tailored to this system can reduce preprocessing latency by minimizing the need for format conversion, thereby achieving faster processing times, and minimizing information loss.

4) SHARED MEMORY UTILIZATION

The limitation of low resolution due to shared memory size can be addressed by modifying the way shared memory is utilized and redesigning the model architecture, allowing for higher quality image inference.

Additionally, structural design changes needed to accommodate a dual-core system pose challenges, as dividing the same space requires more storage capacity. By redesigning

the model to reduce the burden of preprocessing and transmission while also separating the model, we aim to achieve efficient performance.

VI. CONCLUSION

In this study, we introduced a new dual-core system architecture using the STM32H747, designed to significantly improve deep learning inference capabilities in resource-limited environments. By integrating a shared memory system and developing an advanced image preprocessing method specifically for MCUs, our approach effectively uses parallel processing and careful resource optimization. The result is up to 6 times greater efficiency in data processing, such as with camera operations, and a 30% reduction in total processing time, all without increasing power consumption. This achievement demonstrates the significant benefits of parallel processing in dual-core systems and highlights the crucial role of artificial intelligence in enhancing computational processes for real-time applications that require fast and consistent performance.

There have been previous studies [21] that significantly improved system performance using dual-core technology, but we have demonstrated that even greater performance gains can be achieved when applied to AI systems. Although this study focused on overall system improvements with dual-core technology and did not achieve the highest possible performance, we believe that further optimization tailored to the characteristics of MCUs can lead to even better performance and faster processing speeds. In future research, we plan to reduce the burden of preprocessing and separate the model to highlight the features and advantages of dual-core systems through model redesign. We believe our research can enhance performance across the entire field of embedded system AI.

Our goal is to set new standards for using dual-core MCUs in the field of embedded AI, making it easier and more practical to integrate AI into everyday life. By doing so, we aim to expand the scope of AI, significantly improving the functionality, efficiency, and adaptability of embedded systems. This will not only advance the field of embedded AI but also contribute to the seamless integration of intelligent systems into our daily environments, transforming the potential applications of AI in many areas.

REFERENCES

- [1] J. Ching Yuen Siu, J. Chen, Y. Huang, Z. Xing, and C. Chen, "Towards real smart apps: Investigating human-AI interactions in smartphone on-device AI apps," 2023, *arXiv:2307.00756*.
- [2] R. Singh and S. S. Gill, "Edge AI: A survey," *Internet Things Cyber-Physical Syst.*, vol. 3, pp. 71–92, May 2023.
- [3] J. Kim, Y. W. Cho, and D.-H. Kim, "Anomaly detection of environmental sensor data using recurrent neural network at the edge device," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2020, pp. 1624–1628.
- [4] *STMicroelectronics AN5557 Application Note*, document STM32H747/757, 2022.
- [5] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, and R. Rhodes, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 800–811, Sep. 2021.

- [6] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. Whatmough, "MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 517–532, Mar. 2021.
- [7] (2024). *Microsoft*. Accessed: Feb. 27, 2024. [Online]. Available: <https://github.com/microsoft/ELL>
- [8] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 11711–11722.
- [9] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," 2021, *arXiv:2110.15352*.
- [10] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 22941–22954.
- [11] STMicroelectronics. (2024). *STM32Cube.AI, Your Software Tool To Optimize Artificial Neural Networks on STM32*. Accessed: May 4, 2024. [Online]. Available: https://www.st.com/content/st_com/en/campaigns/stm32cube-ai.html
- [12] STMicroelectronics. (2024). *Integrated Development Environment for STM32(STM32CubeIDE)*. Accessed: Apr. 4, 2024. [Online]. Available: <https://www.st.com/en/development-tools/stm32cubeide.html>
- [13] STMicroelectronics. (2024). *Stm32ai-modelzoo*. Accessed: Feb. 27, 2024. [Online]. Available: <https://github.com/STMicroelectronics/stm32ai-modelzoo>
- [14] G. Crocioni, D. Pau, J.-M. Delorme, and G. Gruosso, "Li-ion batteries parameter estimation with tiny neural networks embedded on intelligent IoT microcontrollers," *IEEE Access*, vol. 8, pp. 122135–122146, 2020.
- [15] STMicroelectronics. (2024). *STM32Cube.AI Developer Cloud*. Accessed: Feb. 27, 2024. [Online]. Available: <https://stm32ai-cs.st.com/home>
- [16] STMicroelectronics. (2023). *Update: STM32Cube.AI and NVIDIA TAO Toolkit, Download and Watch a 10x Jump in Performance on an STM32H7 Running Vision AI*. Accessed: Aug. 20, 2024. [Online]. Available: <https://blog.st.com/tao-toolkit/>
- [17] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, p. 2984, Apr. 2021.
- [18] P. Warden and D. Situnayake, *TinyML: Machine Learning With Tensorflow Lite on Arduino and Ultra-low-power Microcontrollers*. Sebastopol, CA, USA: O'Reilly Media, 2019.
- [19] S. Hong, G. Park, and J.-S. Kim, "Image classification on resource-constrained microcontrollers," in *Proc. 14th Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2023, pp. 1453–1455.
- [20] J. Beningo. (2024). *Why and How to Get Started With Multicore Microcontrollers for IoT Devices At the Edge*. [Online]. Available: <https://www.digikey.kr/en/articles/why-and-how-to-get-started-with-multicore-microcontrollers>
- [21] Z. Meng, Y.-F. Wang, L. Yang, and W. Li, "High frequency dual-buck full-bridge inverter utilizing a dual-core MCU and parallel algorithm for renewable energy applications," *Energies*, vol. 10, no. 3, p. 402, Mar. 2017.
- [22] *Using, the STM32 Chrom-ART Accelerator To Refresh an LCD-TFT Display*, document AN4943, 2021.
- [23] *Reference Manual*, document RM0433, STMicroelectronics, 2023.
- [24] STMicroelectronics. (2024). *Stm32h747i-disco-bsp*. Accessed: Feb. 27, 2024. [Online]. Available: <https://github.com/STMicroelectronics/stm32h747i-disco-bsp>
- [25] *OV5640 Datasheet*, I. OmniVision Technol., Santa Clara, CA, USA, 2011.
- [26] Z. Qin, Z. Zhang, X. Chen, C. Wang, and Y. Peng, "Fd-mobilenet: Improved mobilenet with a fast downsampling strategy," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2018, pp. 1363–1367.
- [27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
- [28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5MB model size," 2016, *arXiv:1602.07360*.



DONGCHAN LEE received the B.E. degree from the Department of Electronics and Computer and Telecommunication Technology, Hannam University, Daejeon, South Korea, in 2020. He is currently pursuing the M.S. degree with the Department of Artificial Intelligence, Electronics and Telecommunications Research Institute, University of Science and Technology. His research interests include embedded software and artificial intelligence.



JEONG-SI KIM received the B.S., M.S., and Ph.D. degrees in computer science from Gyeongsang National University, Jinju, South Korea, in 1992, 1995, and 1999, respectively. Since 2000, she has been with the Electronics and Telecommunications Research Institute, Daejeon, South Korea, where she is currently a Principal Member of the Engineering Staff. She has participated in "Intelligent Devices Simulation Labs." Her research interests include on-device artificial intelligence embedded systems and debugging race conditions.



SEUNGTAE HONG received the B.S., M.S., and Ph.D. degrees in computer science from Chonbuk National University, Jeonju, South Korea, in 2008, 2010, and 2015, respectively. He has been with the Electronics and Telecommunications Research Institute, Daejeon, South Korea, since 2015. He was a Professor with the University of Science and Technology, South Korea, in 2022. He participated in "Intelligent Devices Simulation Labs" with the Electronics and Telecommunications Research Institute and "Artificial Intelligence" with the University of Science and Technology. His research interests include on-device deep learning, resource-aware computing, and embedded systems.

...