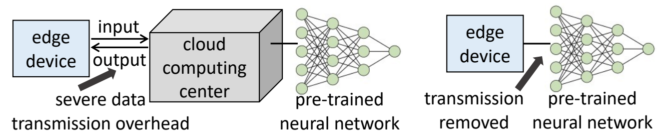
EdgeNN: Efficient Neural Network Inference for CPU-GPU Integrated Edge Devices

Chenyang Zhang[◇], Feng Zhang[◇], Kuangyu Chen[◇], Mingjun Chen[◇], Bingsheng He[+], Xiaoyong Du[◇]
[◇]Key Laboratory of Data Engineering and Knowledge Engineering (MOE), and School of Information,
Renmin University of China
[+]National University of Singapore
{chenyangzhang,fengzhang,kuangyuchen,mingjunchen}@ruc.edu.cn, hebs@comp.nus.edu.sg, duyong@ruc.edu.cn

***Abstract*—With the development of the architectures and the growth of AIoT application requirements, data processing on edge becomes popular. Neural network inference is widely employed for data analytics on edge devices. This paper extensively explores neural network inference on integrated edge devices and proposes EdgeNN, the first neural network inference solution on CPU-GPU integrated edge devices. EdgeNN has three novel characteristics. First, EdgeNN can adaptively utilize the unified physical memory and conduct the *zero-copy* optimization. Second, EdgeNN involves a novel inference-targeted inter- and intra-kernel CPU-GPU hybrid execution approach, which co-runs the CPU with the GPU to fully utilize the edge device's computing resources. Third, EdgeNN adopts a fine-grained adaptive inference tuning approach, which can divide the complicated inference structure into sub-tasks mapped to the CPU and the GPU. Experiments show that on six popular neural network inference tasks, EdgeNN brings an average of 3.97×, 3.12× and 8.80× speedups to inference on the CPU of the integrated device, inference on a mobile phone CPU, and inference on an edge CPU device. Additionally, it achieves 22.02% time benefits to the direct execution of the original programs. Specifically, 9.93% comes from better utilization of unified memory, and 10.76% comes from CPU-GPU hybrid execution. Besides, EdgeNN can deliver 29.14× and 5.70× higher energy efficiency than the edge CPU and the discrete GPU respectively. We have made EdgeNN available at https://github.com/ChenyangZhang-cs/EdgeNN.**

I. Introduction

With the rapid development and great benefits of the artificial intelligence of things (AIoT) [23], [38], [45], [47], [48], [50], [53], [61], [70], [76], [77], [81], [89], [100], [101], neural networks have been applied to different scenarios in data management and data engineering domains. Unlike the training process, neural network inference is widely employed on IoT edge devices as an important data analytics service, whose applications involve recommendation [28], [73], [92], image recognition [31], autonomous driving [33], [40], natural language processing [19], health care [13], [85], *etc*. Due to edge devices' low processing power, data are usually uploaded to the cloud for processing; then, the results are sent back to the device as shown in Figure 1 (a). Accordingly, this process involves transmission overhead. Fortunately, hardware developers like Intel, NVIDIA, and AMD have launched several power-efficient integrated edge processors [2], [6], [7]. Employing and exploring integrated processors is a trend in academic and industry approaches [8], [44], [62], [68], [82], [94]. As a result of this integration, the inference task can be executed directly on the edge device as shown in Figure 1 (b).



(a) Inference on cloud. (b) Inference on edge.

Fig. 1. Neural network inference for AIoT applications on edge.

The application scope of edge devices is substantially broadened, allowing them to retain original benefits, such as low power consumption while achieving improved computing performance. Therefore, it's essential to explore the performance of inference directly on such integrated edge processors.

Applying neural network inference on CPU-GPU integrated edge devices is a great advantage compared with inference on edge CPU devices and the cloud. First, because CPU-GPU integrated edge devices usually have higher computing capacity compared with edge CPUs, inference on integrated devices achieves lower latency. Second, the risk of data leakage in the cloud is avoided, and user privacy can be better protected. Third, inference on edge is more stable and practical because it does not rely on the network connection and cloud servers. Fourth, inference directly on edge devices eliminates overhead brought by cloud computing, resulting in possibly lower latency. To the best of our knowledge, the inference behavior directly on integrated edge devices is still unclear, and a thorough exploration is of great help.

Currently, there are many studies executing machine learning workloads on CPU-GPU integrated architectures, primarily for cloud environments [18], [35], [52], [65], [94], [97]. iMLBench [94] explores the training process of classical neural networks on the CPU-GPU integrated architecture, but the inference process remains to be investigated for edge devices. Gu *et al.* [34] analyzed the energy consumption of two neural networks on integrated and discrete platforms and concluded that the integrated architecture is more energy-efficient. DART [88] is a framework arranging CPUs and GPUs to reduce the response time of neural network inference. There is also research about inference on other edge devices [37], [43], [83] and inference on GPUs [42], [91], [95]. However, no research explores the performance and optimization of inference on CPU-GPU integrated edge devices.

Performing efficient inference on CPU-GPU integrated edge devices requires handling the following three challenges. First, different from discrete GPUs, the integrated edge devices use a unified DRAM for both CPU and GPU, losing the benefits of high memory bandwidth of discrete GPUs. Hence, specific optimizations need to be proposed for this memory hierarchy. Second, the computational capacity of edge devices is still lower than that of HPC servers with discrete GPUs. Therefore, all resources, including the CPU, need to be utilized to obtain high efficiency for neural network computations. Third, modern inference can involve complicated neural network structures, so fine-grained mapping of neural networks to processors needs to be carefully designed.

We propose EdgeNN, which is a solution for efficient inference on CPU-GPU integrated edge devices. Our proposal is a holistic and effective approach in improving the performance of neural network inferences with the aspect of memory efficiency as well as task scheduling. EdgeNN involves three novel designs to solve the above challenges. First, EdgeNN includes a semantic-aware memory management method for inference tasks. It involves two memory usage mechanisms and can choose one based on the data processing semantics. Second, EdgeNN includes an inference-targeted inter- and intra-kernel CPU-GPU hybrid execution approach. It can collaborate with the semantic-aware memory management method. Third, due to the novelties of our memory management and hybrid execution method, existing tuning models cannot optimize the computing tasks efficiently. Hence, we develop a fine-grained adaptive inference tuning approach, which divides the complicated inference structure into sub-tasks. The tuning technique determines the execution strategy by analyzing the dependency and features of these sub-tasks. Then, sub-tasks are mapped to the CPU and the GPU. The performance statistics are collected to adjust the execution strategy adaptively. To sum up, we optimize inference on integrated edge devices by utilizing both architectural features of edge devices and characteristics of inference tasks.

We use NVIDIA Jetson AGX Xavier [7] as the CPU-GPU integrated edge platform, the CPU on Jetson, Raspberry Pi 4 Model B [10], and MediaTek Dimensity 8100 [9] as CPU edge platforms, along with NVIDIA GeForce RTX 2080 Ti graphics card [1] as a cloud discrete GPU platform. EdgeNN is evaluated on six popular neural networks, including Fully Connected Neural Network [30], LeNet [49], AlexNet [46], VGG [80], SqueezeNet [41], and ResNet [36]. Experiments show that EdgeNN brings an average of 3.97×, 3.12× and 8.80× speedups to inference on the three edge CPUs. Besides, EdgeNN achieves 22.02% time benefits compared to the execution on the integrated GPU. Specifically, 9.93% comes from better utilization of unified memory, and 10.76% comes from CPU-GPU collaboration. Compared with inference on the edge CPU and the discrete GPU, EdgeNN achieves 29.14× and 5.70× higher energy efficiency. Also, edge devices are usually substantially cheaper than cloud HPC servers.

To summarize, we make the following contributions:

- We find that the CPU-GPU integrated edge device is more suitable for inference on edge compared to edge CPUs and discrete GPUs, but special optimizations are needed.
- We propose EdgeNN, the first solution for executing inference on integrated edge devices. EdgeNN can perform inter- and intra-kernel CPU-GPU hybrid execution with semantic-aware memory management for inference.
- We conduct a comprehensive evaluation to measure the performance of EdgeNN on the CPU-GPU integrated edge device, and conclude that the three main designs of our approach accumulatively bring significant improvements for inference on edge.

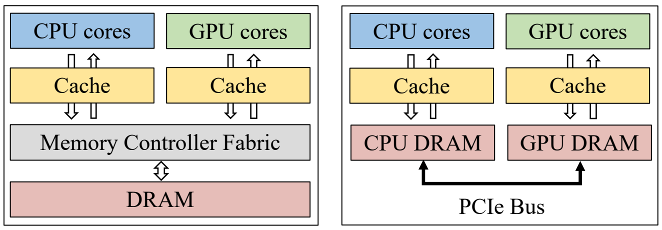
II. Background

In data management domain, edge computing is a new processing paradigm that moves data storage and processing closer to the point of use. The closest device is the edge device, which is usually used in intelligent cameras, robots, cars, *etc* [33], [40], [77], [89]. Recently, the quantity of edge devices has been increasing rapidly, but the network bandwidth does not expand as much. This becomes a bottleneck and facilitates a wide range of edge research.

Data management on edge devices. Data management and analytics are required for edge devices in many real application scenarios, such as intelligent cars [71], Unmanned Aerial Vehicles (UAV) [17], Mobile Service Robot (MSR) [14], *etc*. Processing data directly on edge processors can alleviate network transmission pressure and reduce latency from cloud. Besides, computing on edge attains higher security compared to computing on cloud. For applications such as Tesla's Autopilot for intelligent cars [5], Arduino Robot [4], and some UAVs [29], [39], data is directly managed and processed on edge devices due to the above advantages. However, data analytic tasks can require edge processors with high computing capacity to achieve low latency. Fortunately, the CPU-GPU integrated edge architecture can meet most applications' requirements due to its relatively higher computing capacity.

Widespread use of AIoT in data management at the edge. AIoT, especially neural network inference, has been widely employed in data management to perform diverse types of data analytics tasks on edge, including recommendation, image recognition, autonomous driving, natural language processing, and health care [13], [21], [24], [31], [33], [40], [45], [53], [54], [56], [58], [61], [73], [77], [81], [100]–[102]. Compared to the previous works, we investigate the deployment of AIoT on high-performance CPU-GPU integrated edge devices to further enable efficient near-end data management and processing.

Characteristics of the CPU-GPU integrated edge devices. We show the architecture of the CPU-GPU integrated edge device and compare it to the discrete GPU architecture in Figure 2. The integrated edge device involves three characteristics. First, the edge device integrates both CPU and GPU, bringing more possibilities and opportunities for edge programming. Second, the integrated edge device does not use discrete memory for GPU but uses unified DRAM memory shared with CPU. This design eliminates data transfer and enables a zero-copy mechanism between CPU and GPU. Third, the edge device is usually compact and power-efficient. The computing capacity of edge devices is lower than discrete architectures, and the integrated edge devices are more powerful than edge devices without GPU. The integrated edge device poses new opportunity for data management applications.



(a) Jetson edge architecture. (b) Discrete architecture.

Fig. 2. Comparison between integrated edge and discrete architectures.

III. Motivation

A. Revisiting AIoT Inference at the Edge

In data management and engineering, there are three strategies to perform inference for edge data analytics. The three strategies include using cloud servers for computing, computing directly with discrete GPUs, and computing directly on the CPU-only edge device. Since neural network inference is more suitable to be processed by GPU than CPU, we explore inference at CPU-GPU integrated edge devices which embedded GPU. The integration of GPU makes the edge devices provide higher computing capacity and have two processors with different features. We demonstrate the comparison between inference on CPU-GPU integrated edge devices and the above-mentioned three common methods.

- **Comparison with inference on cloud.** Inference on cloud has unstable latency and privacy problems, even incurring a delay of seconds [12], [63], [79]. The cloud overhead consists of the network round trip time through the cloud, cloud waiting time, failure processing time, etc, which is unpredictable. In contrast, inference on edge devices eliminates latency brought by cloud transmission and is a compelling option that prioritize privacy and security.
- **Comparison with inference on discrete GPUs.** Inference on discrete GPUs can result in severe PCIe transmission overhead between CPU and GPU, even reaching 36% as detailed in Section V-C2, amortizing the performance benefits brought by the GPU. Fortunately, the unified memory design of CPU-GPU integrated devices can solve this problem. In addition, integrated edge devices consume less power than discrete GPUs, which is essential for AIoT.
- **Comparison with inference on CPU-only edge devices.** CPU-only edge devices often lack computing capacity. The integrated edge device can compensate for insufficient computing power with an embedded GPU while keeping energy consumption low.

B. Challenges

Although neural network inference at the edge has high potential, to enable efficient inference on CPU-GPU edge devices, we need to consider architecture features of integrated edge devices as well as features of inference tasks, which brings three challenges.

Challenge 1: neural network inference utilizing unified memory of edge devices. As stated in Section II, limited by the energy support, cost, space, and heat dissipation, the integrated edge device utilizes a unified DRAM instead of the high-bandwidth memory used in discrete GPUs. For example, the memory bandwidth of NVIDIA Jetson is only 137 GB/s, while that of NVIDIA 2080 Ti reaches 616 GB/s. Moreover, both CPU and GPU share the same memory, which means that CPU and GPU contend for the same memory controller and bandwidth, so efficient utilization of the unified memory from both CPU and GPU is critical for inference on integrated edge devices. Besides, CPU and GPU have their private caches and registers. Neural network inference needs to be redesigned targeting this unique memory hierarchy to achieve better performance.

Challenge 2: the insufficient computing power of low-performance heterogeneous edge devices. Low-power designs are common in edge devices. The NVIDIA Jetson platform, for example, is powered by ARM CPUs, which have basic circuitry, low power consumption, and poor performance. Insufficient computing capacity limits the applicability of edge devices. Popular neural networks in real applications usually contain many network layers and adjustable parameters for computation. Although embedded GPU has been integrated into edge devices, such as the Jetson platform, the edge devices still have a huge performance gap compared to discrete GPU servers. For instance, the NVIDIA Jetson AGX Xavier contains 512 GPU cores, while the NVIDIA 2080 Ti contains 4352 GPU cores. Accordingly, multi-processor resources need to be utilized on edge devices, but the different architectural features of the CPU and the GPU make programming challenging.

Challenge 3: mapping of complicated network inference structure to integrated edge devices. Current popular neural networks involve many layers and are more diverse, making the mapping from complex networks to edge devices complicated. For example, AlexNet [46] has 25 layers, VGG [80] has 40 layers, and SqueezeNet [41] has more than 60 layers. There can be multiple layers with different structures in a neural network, such as a fully connected layer, convolution layer, recurrent layer, pooling layer, activation layer, dropout layer, etc. Allocating inference tasks on CPU and GPU can have a large solution space because the dependencies between layers and the characteristics of different layers should be carefully considered. For instance, SqueezeNet consists of five types of layers. Besides, there are layers with no direct dependency (as shown in Figure 5), and thus they can be computed simultaneously by different processors.

IV. EdgeNN

We propose a neural network inference solution, called EdgeNN, to address the above challenges on CPU-GPU integrated edge devices. In this section, we first provide an overview of EdgeNN and then show its detailed designs.

A. Overview

We show the overview of EdgeNN in Figure 3. EdgeNN consists of three major designs: 1) semantic-aware memory management for inference tasks, 2) inference-targeted inter- and intra-kernel CPU-GPU hybrid execution, and 3) fine-grained adaptive inference tuning approach. These three designs guarantee high performance of inference on edges with fine-grained cooperation. Next, we show the workflow, offer solutions to the challenges, outline the novelties, and discuss the difference to previous works.

Workflow. As Figure 3 shows, firstly the fine-grained adaptive inference tuning approach (detailed in Section IV-D) determines the distribution of sub-tasks between two processors and memory usage strategies. It divides the network into layers and builds a directed acyclic graph (DAG) whose nodes represent layers and edges represent the execution sequences of layers. Then, the dependencies between layers are analyzed and used to distribute tasks between the CPU and the GPU. Based on the performance statistics of previous executions, the distribution is adjusted adaptively. The tuning approach's results include sub-task assignments for processors and memory usage strategies. Then, the edge device begins to compute under the guidance. The semantic-aware memory management technique involves two mechanisms (detailed in Section IV-B). One uses *zero-copy* technique, and the other is regular memory allocation. These two memory management mechanisms are employed for different arrays to obtain better performance. After that, the hybrid execution technique (detailed in Section IV-C) enables the CPU and the GPU to collaborate on processing the sub-tasks assigned to them. During execution, the performance statistics are recorded to guide the tuning approach.

Solutions to challenges. EdgeNN can deal with the three challenges discussed in Section III-B. The semantic-aware memory management technique utilizes two memory usage mechanisms to solve the first challenge (Section IV-B). To address the second challenge, the CPU and the GPU co-run to accelerate the inference tasks (Section IV-C). The proposed fine-grained adaptive inference tuning approach helps map the complicated computing task to the integrated edge device, which handles the third challenge (Section IV-D).

Novelties. EdgeNN is the first solution for efficiently executing inference on integrated edge devices. It consists of three novelties. First, the semantic-aware memory management technique supports two memory usage mechanisms and can adaptively choose one between them based on the data processing semantics. Second, the CPU-GPU hybrid execution method targets neural networks with dozens of kernels and involves inter-kernel and intra-kernel co-running. Besides, it is specific to the integrated edge architecture and utilizes the unified memory for better hybrid execution. Third, the fine-grained adaptive tuning method analyzes the complicated structure of neural networks and maps sub-tasks to the CPU and the GPU on edge. The mapping strategy of sub-tasks is fine-grained and adaptive according to the collected performance statistics.

Difference from existing memory usage methods. We observe that the usage of *zero-copy* technique cannot always bring benefits during neural network inference, and the advantage is based on the data processing semantics. Therefore, we apply two different memory usage methods for better performance. Although existing studies [59], [86], [94], [96], [99] have explored the memory usage for integrated architectures, none of them explore applying different memory management mechanisms simultaneously according to the data processing semantics.

Difference from existing CPU-GPU hybrid execution methods. Existing works [15], [16], [51], [60], [74], [93] explored the hybrid execution of CPU and GPU in discrete platforms. However, they cannot perform the fine-grained cooperation between CPU and GPU due to PCIe overhead. For the works exploring hybrid execution on integrated architectures [55], [97], they co-run the two processors to perform only intra-kernel co-running. The number of kernels within a program is very small, or even equal to one. Besides, they do not cooperate with the utilization of unified memory. Other works [20], [87] execute only inter-kernel co-running, which means that a part of the computing kernels is assigned to the GPU, while the others are assigned to the CPU. We consider the features of neural network inference tasks and find that they contain many dependent and independent kernels. If we conduct only inter-kernel co-running, the dependent kernels cannot be accelerated. Moreover, the independent kernels cannot be accelerated efficiently if we consider only intra-kernel co-running. Therefore, based on inference tasks' unique features, we conduct inter- and intra-kernel hybrid execution. We deliver a detailed discussion in Section VI.

Difference from existing tuning methods. Due to the specialties of the semantic-aware memory management technique and CPU-GPU hybrid execution method, existing tuning techniques [15], [16], [51], [60], [93] cannot optimize the computing tasks efficiently. Therefore, we invent a tunning method that targets our proposed memory management technique and hybrid execution method.

Relevance to data management. As we have discussed in Section II, data management at the edge is very common in current application scenarios. Besides, with the development of AI, AIoT at the edge for data management and data analysis is becoming prevalent. Inference at the edge is the most significant workload for AIoT. EdgeNN explores the opportunities of executing inference efficiently at the integrated edge devices, which can shed light on future research on data management and AIoT applications at the edge.

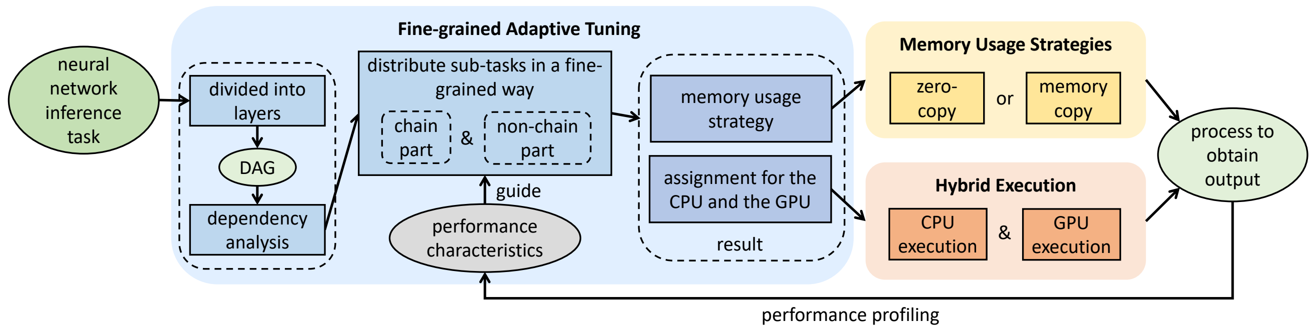


Fig. 3. Main designs of EdgeNN.

B. Semantic-Aware Memory Management for Inference Tasks

Guideline: The effect of applying *zero-copy* technique is not always positive and is determined by data processing semantics. The memory should be managed according to the semantics.

The edge device provides a unified memory that is accessible for both the CPU and the GPU. This unique memory design provides new opportunities for CPU-GPU communication on the neural network inference process. It helps reduce latency by decreasing memory transmission overhead, which is especially beneficial for inference with strict latency requirements. Besides, the unified memory offers chances for CPU-GPU cooperation. Therefore, we develop a semantic-aware memory management for inference tasks to better utilize the unified memory.

Design. The semantic-aware memory management technique employs two memory usage strategies. One is the *zero-copy* technique utilizing the unified physical memory between the CPU and the GPU. We apply CUDA Unified Memory [64] provided by NVIDIA to achieve *zero-copy*. CUDA [75] is a parallel computing platform and programming model for using GPUs. CUDA unified memory is a managed memory address space accessible to any processor in the system. Arrays in the unified memory are designed with hardware and software technologies to provide fine-grained accessbility to different processors without explicit memory copies. Arrays are automatically migrated to processors on-demand so that all data are accessible. The unified memory technique is very suitable for the edge architecture because its CPU and GPU share the same system memory and can access the same physical address. Therefore, using unified memory helps avoid excessive memory copies between processors for integrated architectures. On the contrary, the usage of CUDA unified memory brings no benefit for the discrete architecture due to the PCIe transmission overhead. Since applying unified memory on integrated architectures can decrease transmission overhead between processors, we have new opportunities for CPU-GPU collaboration.

The other memory usage strategy employed by the semantic-aware memory management technique is the standard memory allocation strategy used in discrete GPU architectures. With this strategy, the array is a regular CUDA array with two copies for both the CPU and the GPU, respectively. The reason for applying this method is that, though using *zero-copy* has many advantages, *zero-copy* incurs consistency overhead. Suppose a array in unified memory is frequently modified by the CPU and the GPU, or it is read by one device and modified by another device simultaneously. In these cases, there can be race conditions on the array. To solve this problem, the *zero-copy* technique incurs massive page faults and memory copies to guarantee fine-grained memory consistency. The output array of each layer on the neural network inference process strictly conforms to the above situation. In that case, the array should be a regular CUDA array with two copies for the CPU and the GPU separately. In this way, each processor modifies its copy, and then an explicit merge operation is required to combine the two duplicates for the final results after all modifications. Though overhead is produced by copying a array twice, which is substantially smaller than the cost of utilizing zero-copy.

Existing commercial and research works [59], [86], [94], [96], [99] use only one memory management mechanism. Instead, we first analyze the features of these two memory management mechanisms with a combination of the characteristics of the CPU-GPU integrated edge architecture. We find that applying one memory management mechanism to all data cannot achieve high performance on CPU-GPU integrated edge devices. Accordingly, we propose a semantic-aware memory management technique to support both mechanisms and select one of them for particular data according to the data processing semantics and features of the two mechanisms.

Implementation details. In this part, we briefly introduce how to implement these two memory usage mechanisms. The first one with *zero-copy* technique is implemented with CUDA unified memory. The CUDA API *cudaMallocManaged()* is used to allocate a array in CUDA unified memory. This array can be accessed directly by the CPU and the GPU, so there is no need to copy data between processors explicitly. If a GPU kernel uses the array long after the CPU has modified the array, an explicit memory prefetching by the CUDA API *cudaMemPrefetchAsync()* can help prepare for the upcoming kernel and improve its performance. After the GPU kernel is launched, the API *cudaDeviceSynchronize()* is called to wait for the GPU to complete. This ensures that the CPU reads data from the unified memory after the kernel completes; otherwise, the CPU can read unpredictable data or get a segmentation fault. For the other usual memory usage strategy, the array is allocated by *cudaMalloc()* and can be accessed by GPU but not CPU. Therefore, explicit memory copies by *cudaMemcpy()* are required to transfer data between CPU and GPU.

C. Inference-Targeted CPU-GPU Hybrid Execution

Guideline: The neural network inference task involves many dependent and independent kernels. Applying only inter-kernel hybrid execution or only intra-kernel hybrid execution cannot process inference efficiently on the integrated edge platform. Therefore, both inter- and intra-kernel co-running should be used.

In a normal neural network inference process on discrete architectures, the CPU is used to assign tasks and initialize the input for the GPU kernel. Then, the GPU computes to obtain the result. During the GPU execution, the CPU is idle in the majority of the time. This is not a big concern for discrete GPU architectures because the discrete GPU is of high computing capacity and is suitable for most neural network tasks. However, the integrated GPU on edge architectures is not as powerful as the discrete GPU, and the idle resource of the CPU is non-trivial. For example, the NVIDIA Jetson AGX Xavier has an 8-core CPU with a maximum 2.26GHz. Hence, if the CPU can assist the GPU for computing tasks, the overall performance can be improved. We propose an inference-targeted inter- and intra-kernel CPU-GPU hybrid execution method, which enables the CPU and the GPU to compute collaboratively with this basic idea.

Analysis. The difficulty of designing the fine-grained CPU-GPU cooperation is that there can be more memory communications between processors. Fortunately, it can be solved on edge architectures with *zero-copy*. The *zero-copy* strategy supports fine-grained memory sharing with high speed from both hardware and software perspectives. Note that the fine-grained collaboration is inefficient for discrete CPU-GPU architectures: the overhead of memory communication between processors through PCIe or NVLink is huge when performing *CPU-GPU collaboration*. In EdgeNN design, we treat *CPU-GPU collaboration* as a unique design targeting CPU-GPU integrated edge architecture with unified system memory.

Design. The inference-targeted inter- and intra-kernel CPU-GPU hybrid execution method directs the CPU and the GPU to compute their sub-tasks simultaneously, based on the sub-task and execution sequence from the fine-grained adaptive inference tuning approach. We show an example in Figure 4. The GPU tasks (1, 2, and 3) and the CPU tasks (4 and 5) are independent. Therefore, they can be processed separately. Assume that the GPU finishes task 3 before the CPU finishes task 5. The GPU needs to wait for the completion of task 5. If two processors compute the result of one task, like task 7 in Figure 4, there are synchronization and result merging after they finish processing. Note that the architectures of the CPU and the GPU on edge are different, and specific designs towards each processor should be considered to obtain better performance of inference on edge devices.

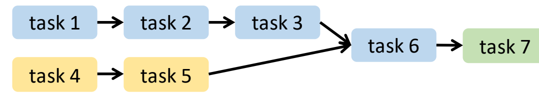


Fig. 4. An example of *CPU-GPU collaboration*. The tasks in blue are assigned to the GPU, and tasks in yellow are assigned to the CPU. The green one is calculated by the CPU and the GPU together.

Implementation details. We briefly introduce how the CPU-GPU hybrid execution works for inference on edge architectures. We use GPU kernels to process the compute-intensive tasks. To support CPU computing, we add CPU kernels for these tasks with OpenMP [22] to enable efficient CPU multi-thread computing. Specifically, the CPU initializes data and invokes the GPU kernel. Then, the GPU begin to execute the GPU kernel. Instead of waiting for the completion of the GPU kernel, as in discrete architectures, the hybrid execution method assigns the idle CPU to execute its tasks assigned by the fine-grained adaptive inference tuning approach. During CPU-GPU cooperation, kernel synchronization is handled based on data dependency. For computing resource utilization, we apply *lazy synchronization* strategy, which means synchronizing the results only when the data processing involves dependency. In this way, we reduce the synchronization overhead and better utilize the computing resources on the edge device.

D. Fine-Grained Adaptive Tuning Approach

We propose a fine-grained adaptive tunning approach to guide the memory management and CPU-GPU hybrid execution for inference on edge architectures. The proposed *zero-copy* and *CPU-GPU collaboration* strategies support the CPU and the GPU on edge architectures to collaborate for inference in a fine-grained strategy. Besides, the inference structure and multiple layers of neural networks provide opportunities for multi-processor co-running. EdgeNN partitions a neural network inference process into several sub-tasks by layers, builds a DAG for the sub-tasks, and then provides a sub-task set and execution sequence for each processor.

Design. We show an example of partial DAG of SqueezeNet [41] in Figure 5 to explain our design in detail. The first part of the DAG from "input" to "squeeze" in Figure 5 is a chain, which must be processed in sequence order. Taking the first convolutional layer as an example, the GPU calculates the convolution results of the first k input channels, and the CPU calculates the results of the remaining input channels. After the two processors finish their tasks, the combined results are copied through the memory to obtain the final output of the first layer. The rest of the DAG in Figure 5 is not a chain and has two independent execution chains, which can be assigned to two processors directly. CPU and GPU need to synchronize before going on to the concatenation layer "concat".

The task partitioning strategy of EdgeNN can better utilize the idle CPU but incurs overhead of CPU-GPU consistency. Hence, the adaptive tuning approach should choose between utilizing idle CPU and avoiding consistency cost. For the chain part of a DAG, the specific partitioning ratio of computing one layer between processors should be determined. For the

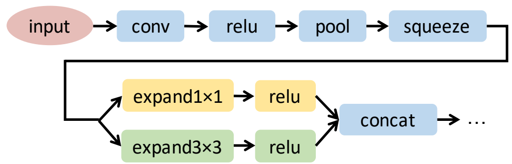


Fig. 5. A DAG example of partial SqueezeNet.

DAG part that is not a chain, the processors' and tasks' features should be considered to assign tasks to processors. All the decisions mentioned above are determined adaptively in the computing process for a specific neural network by EdgeNN. The fine-grained adaptive inference tuning approach applies different strategies each time and discovers the optimal partitioning strategy.

Implementation details. To achieve efficient fine-grained partitioning for the chain part of the DAG, such as the first four layers in Figure 5, we need to partition each layer for the CPU and the GPU in the chain. In detail, we first use the CPU and the GPU to calculate the whole layer separately and record their execution time. Assume that the recorded CPU time is t_{cpu}, and the GPU time is t_{gpu}. Because the CPU and the GPU can compute simultaneously, the CPU-GPU collaboration time t_{co} is the maximum value between the CPU running time and the GPU running time. t_{co} is calculated based on Equation 1 given the CPU computing proportion as p_{cpu}, where $0 \leq p_{cpu} \leq 1$.

$$t_{co} = max(t_{cpu}p_{cpu}, t_{gpu}(1 - p_{cpu})) \tag{1}$$

Note that t_{co} contains only computation time but no data transmission time. We calculate data transmission time t_{data} in Equation 2 given the data volume of this layer's output v_o and the memory copy rate s between the CPU and the GPU.

$$t_{data} = \frac{p_{cpu}v_o}{s} \tag{2}$$

The total execution time is the sum of CPU-GPU collaboration time and data transmission time, and we can obtain the total time t_{total} of a layer according to Equation 3.

$$\begin{aligned} t_{total} &= t_{co} + t_{data} \\ &= max(t_{cpu}p_{cpu}, t_{gpu}(1 - p_{cpu})) + \frac{p_{cpu}v_o}{s} \\ &= \begin{cases} t_{cpu}p_{cpu} + \frac{p_{cpu}v_o}{s} & \frac{t_{gpu}}{t_{cpu}+t_{gpu}} < p_{cpu} \leq 1 \\ t_{gpu}(1 - p_{cpu}) + \frac{p_{cpu}v_o}{s} & 0 \leq p_{cpu} \leq \frac{t_{gpu}}{t_{cpu}+t_{gpu}} \end{cases} \end{aligned} \tag{3}$$

Our goal is to obtain an optimal p_{cpu} to minimize t_{total}. We set the optimal p_{cpu} as p_{op}, and get p_{op} shown in Equation 4 according to Equation 3.

$$p_{op} = \begin{cases} 0 & \frac{v_o}{s} \geq t_{gpu} \\ \frac{t_{gpu}}{t_{cpu}+t_{gpu}} & \frac{v_o}{s} < t_{gpu} \end{cases} \tag{4}$$

According to Equation 4, the fine-grained adaptive inference tuning approach can determine the CPU proportion when the CPU and the GPU collaborate to calculate one layer.

For task assignments for the non-chain part of a DAG, the tunning approach explores different assignment strategies. Taking Figure 5 as an example, the non-chain part consists of the yellow and green layers. The approach sets the CPU execution time of the two yellow layers as t_{c1}, and the time of the two green layers as t_{c2}. The GPU execution time of yellow layers and green layers is recorded as t_{g1} and t_{g2}, respectively. The output data volumes of the yellow and green ReLU layers are v_1 and v_2. One assignment strategy is to assign the yellow layers to the CPU and the green layers to the GPU. The total time is " $max(t_{c1}, t_{g2}) + v_1/s$". The second method is to assign green layers to the CPU and yellow layers to the GPU, and the total time is "$max(t_{c2}, t_{g1}) + v_2/s$". The third method is to assign them all to the GPU, and the total time is "$t_{g1}+t_{g2}$". Based on these options, the approach selects an assignment strategy with a minimum total time.

V. Evaluation

A. Experimental Setup

Evaluated methods. To obtain a comprehensive understanding of EdgeNN, we conduct four sets of comparative experiments. The metrics include execution time, power efficiency, and cost efficiency, which are common concerns for deep learning and architecture research [25], [72], [78], [84]. The first one compares EdgeNN on an integrated edge device with inference on edge CPUs. The second is to compare EdgeNN with inference on the edge GPU alone in the same edge device. This experiment shows the benefits of EdgeNN on integrated edge devices. The third one makes a comparison with inference on cloud servers with discrete GPUs. The forth is to compare EdgeNN with a state-of-the-art CPU-GPU hybrid execution methods [96]. To measure the actual power consumption when running the benchmark, we use jetson-stats, a package for monitoring NVIDIA Jetson devices, to measure the actual power of the Jetson platform. An electric power meter is employed to obtain the actual power of Raspberry Pi. The actual power consumption of the mobile platform cannot be measured. We employ the NVIDIA System Management Interface (nvidia-smi) to get the power of the discrete GPU device.

Platforms. We use four platforms in evaluation, including a CPU-GPU integrated edge device, an edge CPU device, a mobile processor, and a cloud GPU device. They are introduced in detail as follows.

- The CPU-GPU integrated edge device used in evaluation is NVIDIA Jetson AGX Xavier [7]. It is an edge device with a small size, large compute density and high energy efficiency. It consists of an embedded 512-core Volta GPU and an 8-core ARM v8.2 CPU. The memory is a 32GB LPDDR4x with a bandwidth of 137 Gbps. Its size is just 100 × 87 mm, and its price is \$699 while it provides high performance. Jetson AGX Xavier provides three power options of 10W, 15W, and 30W. The operating system we use is Ubuntu 18.04.4.
- The CPU edge device used in evaluation is a Raspberry Pi 4 Model B [10]. It is a 86 × 56 mm small computer

equipped with Quad core ARM Cortex-A72 64-bit SoC, a shared 1MB L2 cache, and a 8GB LPDDR4 SDRAM. Its maximum power consumption is only 6.4W [11], and its price is $75. The operating system is Raspberry Pi OS 5.10.

- The mobile phone processor is MediaTek Dimensity 8100 [9] released in 2022. It contains four Arm Cortex-A78 cores up to 2.85GHz and four Arm Cortex-A55 cores up to 2.0GHz. The memory is LPDDR5, and the bandwidth is 6400 Mbps. The operating system is Android 12. We execute the benchmarks with C++ based on Termux, which enables working on the real phone without rooting.
- The cloud GPU device employed is NVIDIA GeForce RTX 2080 Ti graphics card [1]. GeForce RTX 2080 Ti is powered by the Turing GPU architecture, and this architecture is widely used as cloud GPU servers. The 2080 Ti GPU has more GPU cores and is much more powerful than the Jetson edge device, but with higher price and more power consumption. The TDP of 2080 Ti is 260W, almost nine times that of Jetson, resulting in that 2080 Ti cannot be employed in edge application scenarios with limited power supply. The operating system is Ubuntu 20.04.2 LTS.

Benchmarks. We use six popular neural network inference tasks to evaluate the performance of EdgeNN. **Fully connected neural network (FCNN)** [32] is a fundamental network and is commonly used as part of prevalent neural networks. A FCNN consists of at least three layers: an input layer, at least one hidden layer, and an output layer. The FCNN in this work has three hidden layers. **LeNet** [49] is a simple and classical convolutional neural network (CNN). LeNet consists of seven layers, including convolutional layers, pooling layers, and fully connected layers. **AlexNet** [46] is a CNN to do image classification. With the same layers as LeNet, AlexNet is deeper and achieves higher accuracy. **VGG** [80] is a deep CNN for image classification tasks, and the VGG explored in this work is VGG-16, which consists of 16 weight layers. VGG has been widely applied in real-world applications [57], [69]. **SqueezeNet** [41] is a small CNN achieving the same accuracy with AlexNet on ImageNet dataset [26] with 50× fewer parameters. SqueezeNet has relatively low requirements for processors' computing capacity and memory size, so it is suitable for edge architectures. **Residual Neural Network (ResNet)** [36] is a CNN with a special residual technique. It has many variants and we select ResNet-18 for evaluation.

B. Comparison with Edge CPUs

The neural network inference tasks directly on edge are usually executed on the CPUs of the edge device. To demonstrate the advantages of the integrated CPU-GPU edge architecture, we conduct comparison experiments between EdgeNN on the integrated edge device and inference on three edge CPUs, including Jetson's CPU, the mobile phone CPU, and Raspberry Pi's CPU.

1) Time Benefits: Figure 6 shows the speedups of EdgeNN over inference on three edge CPUs. The average speedups to these three edge CPUs are 3.97×, 3.12×, and 8.80×. The computing capacities of Jetson's CPU and mobile phone CPU are higher than Raspberry Pi. We conclude that EdgeNN on the integrated device outperforms inference at edge CPUs with different computing capacities.

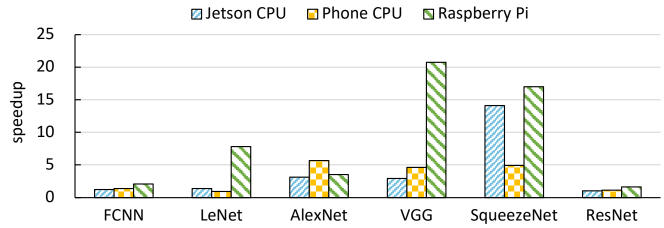


Fig. 6. Performance speedups of EdgeNN on the integrated device to inference on edge CPUs.

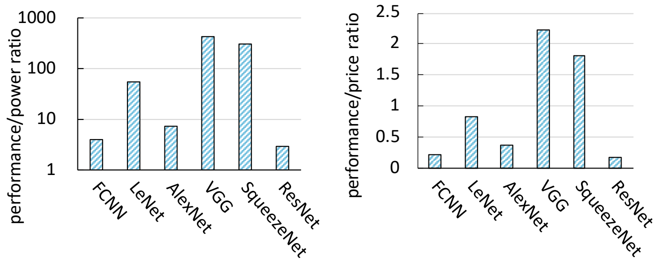
2) Energy and Cost Benefits: Energy consumption and price of edge devices are common concerns in edge application scenarios. Therefore, we compare the power utilization and cost-effectiveness of EdgeNN on the integrated edge device and inference on the edge CPU device. Since we cannot measure the actual power consumption of the mobile phone CPU, the edge CPU device employed here is Raspberry Pi. We measure the utilization of processors while the workloads are running. The average utilization of Raspberry Pi is 52%, and the average utilization of the CPU and the GPU at Jetson is 75% and 62%. Besides, we observe that the processors' utilization is positively related to power consumption. For example, when the CPU and GPU utilization at Jetson is 72% and 42% for ResNet, the power consumption is 5.5W. The CPU and the GPU utilization is all 100% for SqueezeNet, and the corresponding power is 7.9W. From this observation, although the utilization of the edge CPU is less than the integrated device, it consumes less power too. Hence, the performance/power metric is reasonable. We show the comparison results of EdgeNN and inference on the edge CPU in Figure 7. The values in Figure 7 (a) are calculated by Equation 5, and the values in Figure 7 (b) are calculated by Equation 6. Note that the power is the actual power consumption when each workload is running.

$$performance/power\ ratio = \frac{performance/power\ of\ EdgeNN}{performance/power\ of\ inference\ on\ edge\ CPU} \tag{5}$$

$$performance/price\ ratio = \frac{performance/price\ of\ EdgeNN}{performance/price\ of\ inference\ on\ edge\ CPU} \tag{6}$$

Performance/power ratio. We show the power efficiency comparison in Figure 7 (a). The geometric mean values of the ratios are 29.14. We observe that the power efficiency of EdgeNN exceeds that of inference on the edge CPU significantly for tasks including LeNet, VGG, and SqueezeNet. The experimental results expound that using integrated edge devices to perform inference tasks helps save energy compared with edge CPU devices.

Performance/price ratio. We show the cost-effectiveness comparison in Figure 7 (b). The arithmetic and geometric mean values of the ratios are 0.94 and 0.61, indicating that the edge CPU device is more cost-effective. Raspberry Pi



(a) Power efficiency. (b) Cost-effectiveness.

Fig. 7. The ratio of the power efficiency and cost-effectiveness of EdgeNN on the integrated edge device and inference on edge CPU.

computers are designed to be inexpensive and cost-effective. Even so, EdgeNN achieves high performance/price values for VGG and SqueezeNet, indicating that integrated edge devices can obtain high cost-effectiveness.

C. Improvement on Integrated Devices

1) General Improvement: We study the general performance improvements that EdgeNN achieves for inference on edge architectures. The baseline is inference on the edge GPU alone. We explore the benefits of each design in EdgeNN and show the results in Figure 8. Compared with the baseline, EdgeNN achieves performance improvements from 16.29% (VGG) to 27.22% (AlexNet), with an average improvement of 22.02%. This promising result indicates the effectiveness of EdgeNN for various neural network inferences on CPU-GPU integrated edge architectures. EdgeNN achieves the highest improvement of 27.22% for AlexNet, which contains fully connected layers and convolutional layers. These two types of layers exist commonly in neural networks, and it can be inferred that EdgeNN can bring improvement to other neural networks containing these types of layers.

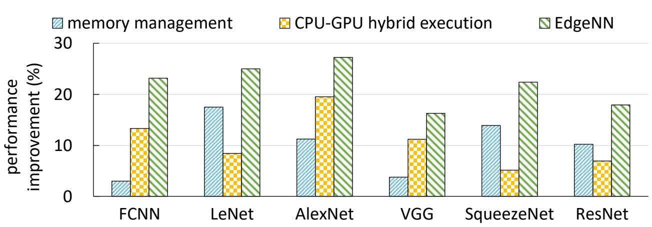


Fig. 8. Overall performance improvement of EdgeNN.

The improvement brought by semantic-aware memory management technique is from 2.97% (FCNN) to 17.50% (LeNet) with an average value of 9.93%. The improvement of LeNet is almost six times that of FCNN, indicating the effectiveness of the memory management technique varies significantly amongst neural networks. More details are discussed in Section V-C2.

The improvement brought by the inter- and intra-kernel CPU-GPU hybrid execution and the fine-grained adaptive tuning approach is denoted as "CPU-GPU hybrid execution" in Figure 8. It achieves improvement from 5.15% (SqueezeNet) to 19.53% (AlexNet), with an average value of 10.76%. The variety in performance improvements is due to the diverse structures of neural networks.

Moreover, from Figure 8, we observe that the improvements brought by the memory management technique and CPU-GPU hybrid execution are not completely relevant. For example, the former is low and the latter is relatively high for FCNN, while the scenario is the opposite of what SqueezeNet exhibits.

We find that CPU-GPU hybrid execution with memory management technique (denoted as "EdgeNN") achieves more benefits than CPU-GPU hybrid execution without memory management. This observation indicates that the combination of them is important for neural network inference on edges.

We observe that the total improvement of EdgeNN is not exactly the sum of the two parts. The improvement of EdgeNN can be greater than the sum of the two parts for neural networks including FCNN, SqueezeNet, and ResNet. The reason is that hybrid execution without memory management technique introduces more memory copies between the CPU and the GPU. Taking Figure 4 as an example, the output of task 5 needs to be transferred from the CPU to the GPU so that the GPU can process task 6. This data transmission does not exist without *CPU-GPU collaboration*. *Zero-copy* helps eliminate the newly introduced memory copy overhead. For the other neural networks, the improvements of EdgeNN are less than the sum of the two. The reason is that *Zero-copy* is not used for partial arrays when hybrid execution is applied, as discussed in Section IV-B. Whether using *Zero-copy* or not is decided by the fine-grained adaptive inference tuning approach. More details are discussed in Section V-C3.

2) Improvement from Semantic-Aware Memory Management: In this part, we analyze the performance benefits from the semantic-aware memory management technique. From Figure 8, we can see that the semantic-aware memory management technique achieves various improvements on different neural networks. To better understand the variety of the benefits, we study the memory usage of each neural network and show the result in Figure 9. In detail, we measure the time proportion of memory copy between the CPU and GPU without zero-copy on the integrated edge device, denoted as "integrated architecture" in Figure 9. To compare, we measure the CPU-GPU transmission overhead on the discrete CPU-GPU architecture, denoted as "discrete architecture". The average ratio on the integrated architecture is 11.46%, and the one on discrete architecture is 23.34%. Fortunately, all these overheads can be avoided with EdgeNN on the integrated edge devices. Moreover, we have the following observations.

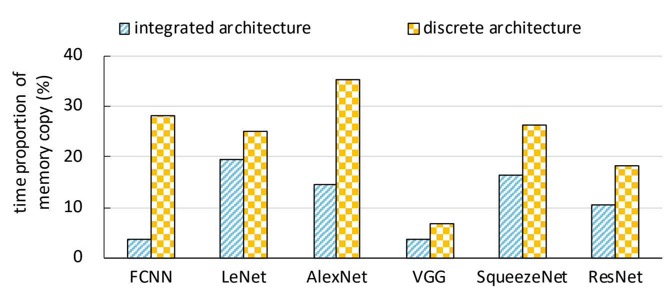


Fig. 9. Time proportion of memory copy between host and device of the integrated edge device and the discrete GPU.

First, we observe that the memory copy ratio in inference is high. The reason is that inference needs numerous input parameters and computes forward propagation only once. This is a distinct feature of neural network inference and such data transmission overhead can be alleviated by EdgeNN, which explains the benefits of our proposed *zero-copy* strategy.

Second, we can see that even executing the original inference benchmark directly on the edge device, the memory transfer time between the CPU and GPU is still faster than that on the discrete CPU-GPU architecture. There are two reasons. One is the different memory copy mechanisms. On the edge device, memory copy between CPU and GPU does not involve PCIe transmission, since the CPU and the integrated GPU share the unified DRAM. In contrast, the copy on discrete CPU-GPU architecture is transferring data through PCIe, which connects the separate CPU memory and GPU memory. The other reason is that the discrete GPU has higher computing capacity, resulting in less execution time. Accordingly, the time proportion of memory copy on discrete architectures becomes significant.

Third, comparing Figure 8 and Figure 9, we find that the performance improvement brought by *zero-copy* shown in Figure 8 is always less than the memory copy proportion shown in Figure 9. The reason is that the benefit of *zero-copy* comes from the elimination of memory copy between the host and the device by using CUDA unified memory. Because not all buffers are allocated in the CUDA unified memory with zero-copy, as we have discussed in Section IV-B, the overall improvement of *zero-copy* does not reach the memory copy ratio in Figure 9. Another reason is that the execution time of partial layers increases with *zero-copy*. We use AlexNet for illustration and show the influence of zero-copy on different layers in Figure 10. We can see that the execution time of the pooling layers increases with the use of *zero-copy*.

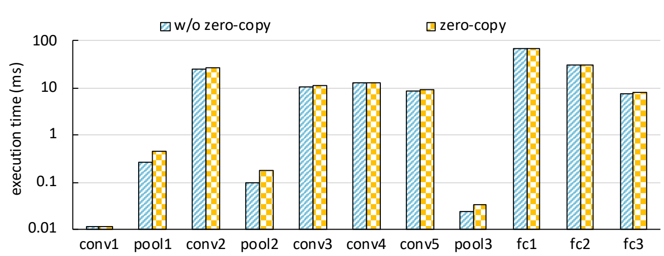


Fig. 10. Execution time of each layer of AlexNet with and without *zero-copy*. "conv": convolutional layer. "pool": pooling layer. "fc": fully connected layer. Note that the vertical axis is a logarithmic axis.

3) Improvement from CPU-GPU Hybrid Execution: To deal with the low computing capacity of edge devices and fully utilize the computing resource, we propose a inference-targeted inter- and intra-kernel hybrid execution strategy.

We explore layer-wise performance improvement brought by hybrid execution. The bars denoted as "CPU-GPU hybrid execution" in Figure 8 show the improvement brought by hybrid execution without the semantic-aware memory management technique. We observe that the benefits from hybrid execution vary significantly for different neural networks. The reason is that those neural networks have various structures and layers. To explore the benefits in detail, we evaluate the execution time of each layer of AlexNet and show the results in Figure 11. Note that we do not show the layers whose time proportions are less than 1%. Given that AlexNet's DAG is a chain, distributing tasks entails using both the CPU and GPU to calculate one layer concurrently. We observe that the hybrid execution achieves significant improvement for fully connected layers in AlexNet, with an average of 31.71% and 53.80% without and with *zero-copy*, respectively. This indicates that the CPU on edge can compute fully connected layers of AlexNet efficiently to facilitate the GPU. On the contrary, hybrid execution cannot improve the performance of the convolutional layers of AlexNet. The results illustrate that using only the GPU can obtain the optimal execution time for convolutional layers of AlexNet.

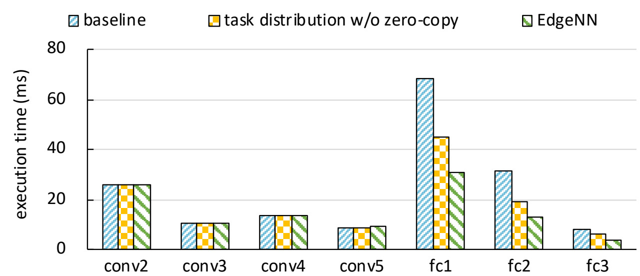


Fig. 11. Execution time of each layer of AlexNet with hybrid execution.

To explore whether the above observations hold in all cases, we further validate it in LeNet and VGG and conclude the results in Table I. The result expounds that hybrid execution improves the performance of most fully connected layers. For convolutional layers, the larger the computation scale, the greater the GPU's advantage. The CPU cannot help when the convolutional layer has considerable input and output scales. The convolutional layers in AlexNet all have large data scales, while the layers in LeNet are much smaller. VGG has both small and large convolutional layers. We conclude that our proposed hybrid execution strategy does bring benefits for partial convolutional layers. Fortunately, the fine-grained adaptive tuning approach can adjust to obtain better performance according to the performance feedback.

TABLE I
IMPROVEMENT BROUGHT BY CPU-GPU HYBRID EXECUTION WITH ZERO-COPY.

improvement (%)	LeNet		AlexNet		VGG	
	conv	fc	conv	fc	conv	fc
min	4.95	31.56	0	48.43	0	16.07
max	36.25	41.24	0	58.32	19.15	43.09
average	20.60	36.40	0	53.81	4.12	31.43

D. Comparison with Cloud Computing

There are two methods for generating inference results on edge, as discussed in Section I. One is to compute directly on edge devices, and the other is to upload the input to the cloud for processing. To show the advantages of our work, we compare EdgeNN with inference on cloud with a NVIDIA GeForce RTX 2080 Ti discrete GPU. The results are shown in Figure 12. The "EdgeNN" bars show the execution time of EdgeNN on the integrated edge device. The "on-cloud (computing only)" bars include only the computing time of inference on cloud. The bars denoted as "on-cloud" are the sum of "on-cloud computing" time, network transmission time, and cloud delays. We calculate the network transmission time by $t_{net} = v_{in}/b$, where v_{in} denotes the input data volume, and b denotes network bandwidth. The input data is a compressed image whose size is about 400 KB. We rent a server with 1 to 10 MB/s network bandwidth on Alibaba Cloud [3] to measure the cloud overhead. We test the network bandwidth between the edge device and the cloud with both stable and unstable network conditions. According to the experiments, we find the network bandwidth is about 1 MB/s on average. Studies [12], [79] show that the cloud latency is not trivial and is affected by many factors, including physical distance from the cloud service station, the occupancy of the cloud computing platform, the scheduling method, etc. Therefore, we also evaluate the cloud latency and find the average latency to be around 100ms. Based on the average network bandwidth and cloud delay, we obtain the total time of inference on the cloud, denoted as "on-cloud" in Figure 12.

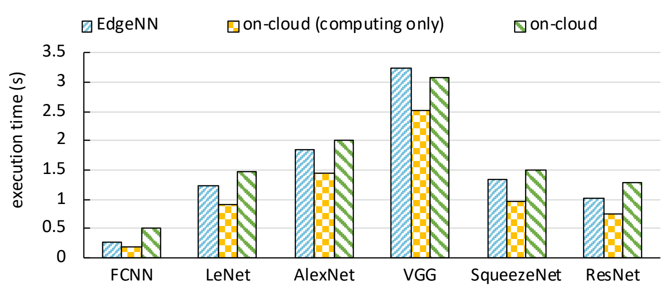
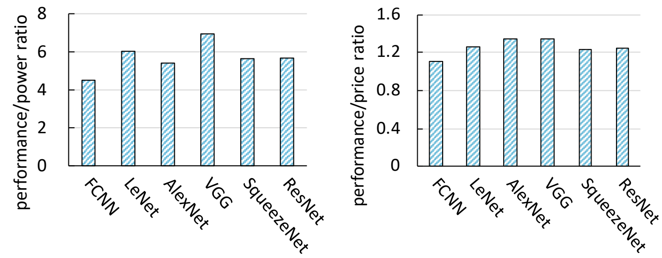


Fig. 12. Execution time of inference on EdgeNN and on cloud.

We observe from Figure 12 that for most tasks, EdgeNN achieves a shorter execution time than the cloud-based solution with an average improvement of 20.28%. For VGG, the inference process is very compute-intensive, so inference on the discrete architecture outperforms EdgeNN. This indicates that, compared with the cloud-based solution, EdgeNN is not suitable for inference of complex neural networks. However, limited by budget, equipment capacity, network condition, etc., not all edge devices have efficient access to cloud computing resources. For those scenarios, EdgeNN is still suitable.

E. Comparison with Discrete GPU

One remarkable feature of the edge device is its substantially lower energy consumption than servers. Besides, the low price of the edge device is another advantage. To better explore these two advantages of edge devices, we compare EdgeNN on integrated edge device and inference on the discrete GPU from a performance/power perspective shown in Figure 13 (a) and performance/price perspective shown in Figure 13 (b). These metrics have been introduced in Section 5.2.2. Note that the two metrics cannot indicate the energy and cost efficiency of cloud computing since a cloud server is shared with an unknown number of users. Instead, they demonstrate the energy and cost efficiency of the discrete GPU.



(a) Performance/power.

(b) Performance/price.

Fig. 13. The ratio of the performance/power and performance/price of EdgeNN on the integrated edge device and inference on the cloud GPU.

Performance/power ratio. From Figure 13 (a), we observe that the energy efficiency of EdgeNN exceeds inference at discrete GPUs significantly and achieves an average of 5.70× improvement. It demonstrates that the CPU-GPU integrated edge device is suitable for high-performance scenarios with limited power support. Although the discrete GPU is more powerful, it's not suitable for edge situations due to its higher power requirement. Besides, edge architectures' high performance/power ratio helps dramatically save the electric power budget.

Performance/price ratio. Figure 13 (b) shows that the cost-effectiveness of EdgeNN is higher than that of the discrete GPU for different neural networks. It achieves a 1.25× cost-effectiveness advantage on average. The results show that CPU-GPU integrated edge architecture with EdgeNN is more cost-effective than the discrete architecture.

F. Comparison with Current Hybrid Execution Approach

To demonstrate the advantages of EdgeNN's inference-targeted CPU-GPU hybrid execution method, we compare EdgeNN with a state-of-the-art hybrid execution approach [96], which utilizes the shared memory on the CPU-GPU integrated architectures for fine-grained hybrid execution. However, it supports only inter-kernel co-running. We employ their idea to perform hybrid execution for the six inference benchmarks on the integrated edge platform. Experiments show that applying inter-kernel co-running for inference can obtain only 8.27% performance improvement for SqueezeNet, while no improvement for the other neural networks. The reason is that this method does not support intra-kernel co-running so it can accelerate only the independent part during the inference process (e.g., the yellow part and the green part in Figure 5 are independent) by assigning them to different processors. In the benchmarks, only SqueezeNet and ResNet have independent parts. Therefore, we conclude that the previous hybrid execution method is inefficient for current inference on the integrated edge device, and EdgeNN is required.

G. Summary of Insights

EdgeNN provides an efficient approach for inference on integrated edge devices. The principles in EdgeNN can be employed in many other compute-intensive and data-intensive applications, such as graph computing, stream processing, query processing, etc. Since EdgeNN demonstrates that inference on integrated edge devices is a success in time, cost, applicability, and robustness perspectives, we believe integrated edge devices can be good solutions for other edge applications due to good energy efficiency.

EdgeNN aims to provide an efficient approach for inference at integrated edge devices. There are a bunch of hybrid platforms, and the idea behind EdgeNN is applicable to similar platforms, such as AMD's APU and Apple Silicon. EdgeNN has been demonstrated to be successful on NVIDIA integrated edge platforms (one of the most popular GPU products worldwide), and we have open-sourced it. We believe that EdgeNN can shed light on hardware and software designs in AIoT and other data applications in the edge. For example, according to our research, Internet Service Providers for edge devices can use integrated edge devices to provide AIoT inference services. Besides, hardware developers may consider the CPU-GPU co-running as a promising trend for current data engineering tasks and develop more hardware suitable for heterogeneous hybrid computing.

VI. Related Work

Data management and data processing on edge. Edge computing poses new opportunities for data management and data processing [27], [63], [66], [67], [90], [98]. Yang *et al.* [90] proposed a time-series edge database, called EdgeDB, utilizing the capacity of the edge devices to achieve much higher performance than the state-of-the-art time-series database. This work concludes that massive time-series data are better processed on edge to alleviate significant network overhead. DPaxos [63] is a Paxos-based protocol for distributed data management in edge nodes. DPaxos utilizes edge computing to achieve real-time response with low latency, quick recovering, and fast reacting. Paparrizos *et al.* [67] presented VergeDB, a database on edge devices. VergeDB supports complicated data analytics, machine learning tasks, and data compression mechanism. Dong *et al.* [27] proposed a decentralized distributed edge database, called SardineDB. SardineDB optimizes data storage on edge flash and achieves high write performance, low garbage collection burden, and low write amplification.

AIoT for data management. AIoT is a popular topic in the data management and data engineering community [45], [53], [54], [61], [77], [81], [100], [101]. Koliousis *et al.* [45] scaled deep learning with small batch sizes for efficient data processing. Liu *et al.* [53] proposed a framework to improve the efficiency of GNN training. Shaowang *et al.* [77] considered machine learning inference on edge as a future trend for data processing. Zhang *et al.* [100] introduced that Model Hopper Parallelism is suitable for deep learning on data systems, and provided a tradeoff among the approach space. Zhou *et al.* [101] accelerated large scale real-time GNN inference by pruning the dimensions in each layer while achieving little accuracy loss. Different from these works, we explore AIoT on CPU-GPU integrated edge devices.

CPU-GPU hybrid execution for integrated architectures. Zhang *et al.* [97] evaluated co-running behaviors of 42 programs on integrated architectures. However, in their workload, each program contains only few computing kernels with simple co-running, and they did not utilize the unified memory of integrated devices. Zhang *et al.* [94] developed iMLBench, a benchmark with general machine learning training workloads. However, the workloads are simple with only several kernels. Besides, the architecture they explored is AMD's powerful PC processor, which is different from low-power edge devices. Cho *et al.* [20] proposed an on-the-fly workload partitioning method for irregular workloads on integrated architectures. The proposed method divides the workload into similar and irregular loads and then assigns them to GPU and CPU. Lupescu *et al.* [55] used the integrated GPU to accelerate the sort algorithm coupled with the CPU. The GPU is assigned to sort a part of input data and transfer the result to the CPU. Then the CPU sorts the whole dataset based on the partially sorted data. Wen *et al.* [87] proposed a task scheduler, which assigns multiple kernels from many programs. The schedule is based on the data size and predicted speedup, which is predicted by a support vector machine. However, they just assigned the whole kernel to one processor and did not consider co-running a kernel in two processors. In conclusion, none of the existing works employs both inter-kernel and intra-kernel CPU-GPU co-running, together with utilizing a semantic-aware memory support for complicated neural network inference.

VII. Conclusion

Neural network inference is commonly used for data analytics in many edge applications. Executing inference directly on edge processors has several benefits compared to executing on the cloud. The CPU-GPU integrated edge architecture has relatively high computing capacity, so it is suitable for doing inference. Currently, there is no study focusing on inference on the CPU-GPU integrated edge devices. To fill this gap, we propose EdgeNN, which can perform efficient neural network inference on integrated edge devices. EdgeNN can utilize the zero-copy feature of the unified memory and consider CPU-GPU architecture differences in the inference process. Experiments show that EdgeNN can bring 22.02% performance improvement on average over the direct execution of the original programs. Compared to the discrete GPU, EdgeNN can provide 5.70× higher energy efficiency.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (62172419, 62072458, and 61732014) and CCF-Tencent Open Research Fund. Bingsheng's work is in part supported by a Singapore MoE Tier 2 grant (T2EP20121-0030). Feng Zhang is the corresponding author of this paper.

REFERENCES

[1] "NVIDIA GeForce RTX 2080 Ti Graphics Card," https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080-ti/, 2016.

[2] "2nd Generation AMD Embedded R-Series APU," https://www.amd.com/en/products/embedded-r-series-2nd-gen-apu, 2022.

[3] "Alibaba Cloud," https://www.aliyun.com/, 2022.

[4] "Arduino Robot," https://www.arduino.cc/en/Main.Robot, 2022.

[5] "Autopilot — Tesla," https://www.tesla.com/autopilot, 2022.

[6] "Innovative Platforms with Performance Hybrid Architecture, AI, and Media at the Edge," https://www.intel.com/content/www/us/en/products/docs/processors/embedded/12th-gen-iot-mobile-processors-brief.html, 2022.

[7] "JETSON AGX XAVIER," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/, 2022.

[8] "Jetson Community Projects," https://developer.nvidia.com/embedded/community/jetson-projects#hello_ai_world, 2022.

[9] "MediaTek Dimensity 8100," https://www.mediatek.com/products/smartphones-2/mediatek-dimensity-8100?token=lXg6jPDP3Ij1FpE_d-rGVr2aBtNuEV_j, 2022.

[10] "Raspberry Pi 4 Model B," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, 2022.

[11] "Raspberry Pi Dramble: Power Consumption Benchmarks," https://www.pidramble.com/wiki/benchmarks/power-consumption, 2022.

[12] M. Abdallah, C. Griwodz, K.-T. Chen, G. Simon, P.-C. Wang, and C.-H. Hsu, "Delay-sensitive video computing in the cloud: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 14, no. 3s, pp. 1–29, 2018.

[13] T. Al Hanai, M. M. Ghassemi, and J. R. Glass, "Detecting Depression with Audio/Text Sequence Modeling of Interviews," in *Interspeech*, 2018, pp. 1716–1720.

[14] K. Baraka, A. Paiva, and M. Veloso, "Expressive lights for revealing mobile service robot state," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 107–119.

[15] N. Boeschen and C. Binnig, "Gacco - a gpu-accelerated oltp dbms," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1003–1016. [Online]. Available: https://doi.org/10.1145/3514221.3517876

[16] S. Breß, B. Köcher, M. Heimel, V. Markl, M. Saecker, and G. Saake, "Ocelot/hype: Optimized data processing on heterogeneous hardware," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1609–1612, 2014.

[17] H. Chao, Y. Cao, and Y. Chen, "Autopilots for small unmanned aerial vehicles: a survey," *International Journal of Control, Automation and Systems*, vol. 8, no. 1, pp. 36–44, 2010.

[18] L. Chen, X. Huo, and G. Agrawal, "Accelerating MapReduce on a coupled CPU-GPU architecture," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.

[19] J. Cho, K. Sundaresan, R. Mahindra, J. Van der Merwe, and S. Rangarajan, "Acacia: context-aware edge computing for continuous interactive applications over mobile networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 375–389.

[20] Y. Cho, F. Negele, S. Park, B. Egger, and T. R. Gross, "On-the-fly workload partitioning for integrated cpu/gpu architectures," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 1–13.

[21] A. Dadashzadeh, A. T. Targhi, M. Tahmasbi, and M. Mirmehdi, "Hgr-net: a fusion network for hand gesture segmentation and recognition," *IET Computer Vision*, vol. 13, no. 8, pp. 700–707, 2019.

[22] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[23] O. Debauche, S. Mahmoudi, S. A. Mahmoudi, P. Manneback, and F. Lebeau, "A new edge architecture for ai-iot services deployment," *Procedia Computer Science*, vol. 175, pp. 10–19, 2020.

[24] J. Dehesa, A. Vidler, C. Lutteroth, and J. Padget, *Touché: Data-Driven Interactive Sword Fighting in Virtual Reality*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–14. [Online]. Available: https://doi.org/10.1145/3313831.3376714

[25] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "Tie: Energy-efficient tensor train-based inference engine for deep neural network," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. Association for Computing Machinery, 2019, p. 264–278.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[27] M. Dong, H. Zhong, B. Sun, S. Bi, and Y. Cai, "Sardinedb: A distributed database on the edge of the network," in *Web and Big Data*, L. H. U, M. Spaniol, Y. Sakurai, and J. Chen, Eds. Cham: Springer International Publishing, 2021, pp. 186–193.

[28] S. Duan, D. Zhang, Y. Wang, L. Li, and Y. Zhang, "Jointrec: A deep-learning-based joint cloud video recommendation framework for mobile iot," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1655–1666, 2019.

[29] S. Ehsan and K. D. McDonald-Maier, "On-board vision processing for small uavs: Time to rethink strategy," in *2009 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, 2009, pp. 75–81.

[30] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[31] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2015.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[33] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[34] J. Gu, M. Zhu, Z. Zhou, F. Zhang, Z. Lin, Q. Zhang, and M. Breternitz, "Implementation and Evaluation of Deep Neural Networks (DNN) on Mainstream Heterogeneous Systems," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, ser. APSys '14, 2014.

[35] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 243–254.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[37] C.-T. Huang, Y.-C. Ding, H.-C. Wang, C.-W. Weng, K.-P. Lin, L.-W. Wang, and L.-D. Chen, "ECNN: A Block-Based and Highly-Parallel CNN Accelerator for Edge Inference," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019, p. 182–195.

[38] S. Huh, S. Cho, and S. Kim, "Managing iot devices using blockchain platform," in *2017 19th international conference on advanced communication technology (ICACT)*. IEEE, 2017, pp. 464–467.

[39] D. Hulens, J. Verbeke, and T. Goedemé, "Choosing the best embedded processing platform for on-board uav image processing," in *International joint conference on computer vision, imaging and computer graphics*. Springer, 2015, pp. 455–472.

[40] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[41] 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¡ 0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.

[42] A. Jahanshahi, H. Z. Sabzi, C. Lau, and D. Wong, "GPU-NEST: Characterizing energy efficiency of multi-GPU inference servers," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 139–142, 2020.

[43] Z. Jiang, T. Chen, and M. Li, "Efficient deep learning inference on edge devices," *ACM SysML*, 2018.

[44] E. Jose, M. Greeshma, M. T. Haridas, and M. Supriya, "Face recognition based surveillance system using facenet and mtcnn on jetson tx2," in *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE, 2019, pp. 608–613.

[45] A. Koliousis, P. Watcharapichat, M. Weidlich, L. Mai, P. Costa, and P. Pietzuch, "Crossbow: Scaling deep learning with small batch sizes on multi-gpu servers," *Proc. VLDB Endow.*, vol. 12, no. 11, p. 1399–1412, jul 2019. [Online]. Available: https://doi.org/10.14778/3342263.3342276

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[47] N. M. Kumar, A. A. Chand, M. Malvoni, K. A. Prasad, K. A. Mamun, F. Islam, and S. S. Chopra, "Distributed energy resources and the application of ai, iot, and blockchain in smart grids," *Energies*, vol. 13, no. 21, p. 5739, 2020.

[48] S. Kumar, R. D. Raut, and B. E. Narkhede, "A proposed collaborative framework by using artificial intelligence-internet of things (ai-iot) in covid-19 pandemic situation for healthcare workers," *International Journal of Healthcare Management*, vol. 13, no. 4, pp. 337–345, 2020.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[50] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[51] R. Lee, M. Zhou, C. Li, S. Hu, J. Teng, D. Li, and X. Zhang, "The art of balance: a rateupdb™ experience of building a cpu/gpu hybrid database product," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2999–3013, 2021.

[52] P. Li, Y. Luo, N. Zhang, and Y. Cao, "HeteroSpark: A heterogeneous CPU/GPU Spark platform for machine learning algorithms," in *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2015, pp. 347–348.

[53] H. Liu, S. Lu, X. Chen, and B. He, "G3: When graph neural networks meet parallel graph processing systems on gpus," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 2813–2816, aug 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415482

[54] R. Liu, S. Krishnan, A. J. Elmore, and M. J. Franklin, "Understanding and optimizing packed neural network training for hyper-parameter tuning," in *Proceedings of the Fifth Workshop on Data Management for End-To-End Machine Learning*, 2021, pp. 1–11.

[55] G. Lupescu, E.-I. Sluşanschi, and N. Tăpuş, "Using the integrated gpu to improve cpu sort performance," in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 2017, pp. 39–44.

[56] H. Luqman, E.-S. M. El-Alfy, and G. M. BinMakhashen, "Joint space representation and recognition of sign language fingerspelling using gabor filter and convolutional neural network," *Multimedia Tools and Applications*, vol. 80, no. 7, pp. 10 213–10 234, 2021.

[57] M. Mateen, J. Wen, S. Song, Z. Huang *et al.*, "Fundus image classification using VGG-19 architecture with PCA and SVD," *Symmetry*, vol. 11, no. 1, p. 1, 2019.

[58] O. Mazhar, B. Navarro, S. Ramdani, R. Passama, and A. Cherubini, "A real-time human-robot interaction framework with robust background invariant hand gesture detection," *Robotics and Computer-Integrated Manufacturing*, vol. 60, pp. 34–48, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584518302953

[59] G. Mencagli, D. Griebler, and M. Danelutto, "Towards parallel data stream processing on system-on-chip cpu+ gpu devices," in *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2022, pp. 34–38.

[60] S. Meraji, B. Schiefer, L. Pham, L. Chu, P. Kokosielis, A. Storm, W. Young, C. Ge, G. Ng, and K. Kanagaratnam, "Towards a hybrid design for fast query processing in db2 with blu acceleration using graphical processing units: A technology demonstration," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1951–1960.

[61] S. W. Min, K. Wu, S. Huang, M. Hidayetoğlu, J. Xiong, E. Ebrahimi, D. Chen, and W.-m. Hwu, "Large graph convolutional network training with gpu-oriented data communication architecture," *Proc. VLDB Endow.*, vol. 14, no. 11, p. 2087–2100, jul 2021. [Online]. Available: https://doi.org/10.14778/3476249.3476264

[62] S. Mittal, "A survey on optimized implementation of deep learning models on the nvidia jetson platform," *Journal of Systems Architecture*, vol. 97, pp. 428–442, 2019.

[63] F. Nawab, D. Agrawal, and A. El Abbadi, "Dpaxos: Managing data closer to users for low-latency and mobile applications," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1221–1236. [Online]. Available: https://doi.org/10.1145/3183713.3196928

[64] D. Negrut, R. Serban, A. Li, and A. Seidl, "Unified memory in cuda 6.0. a brief overview of related data access and transfer issues," *SBEL, Madison, WI, USA, Tech. Rep. TR-2014-09*, 2014.

[65] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 292–302, 2018.

[66] Z. Pan, F. Zhang, Y. Zhou, J. Zhai, X. Shen, O. Mutlu, and X. Du, "Exploring data analytics without decompression on embedded GPU systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1553–1568, 2021.

[67] J. Paparrizos, C. Liu, B. Barbarioli, J. Hwang, I. Edian, A. J. Elmore, M. J. Franklin, and S. Krishnan, "Vergedb: A database for iot analytics on edge devices." in *CIDR*, 2021.

[68] S. Puthoor and M. H. Lipasti, "Systems-on-chip with strong ordering," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 1, jan 2021. [Online]. Available: https://doi.org/10.1145/3428153

[69] Z. Qawaqneh, A. A. Mallouh, and B. D. Barkana, "Deep convolutional neural network for age estimation based on VGG-face model," *arXiv preprint arXiv:1709.01664*, 2017.

[70] K. Rabah, "Convergence of ai, iot, big data and blockchain: a review," *The lake institute Journal*, vol. 1, no. 1, pp. 1–18, 2018.

[71] R. Rajkumar, "Self-driving vehicles: The challenges and opportunities ahead," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1. [Online]. Available: https://doi.org/10.1145/2809695.2823464

[72] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, p. 267–278, jun 2016.

[73] F. Ricci, "Mobile recommender systems," *Information Technology & Tourism*, vol. 12, no. 3, pp. 205–231, 2010.

[74] V. Rosenfeld, S. Breß, and V. Markl, "Query processing on heterogeneous cpu/gpu systems," *ACM Comput. Surv.*, vol. 55, no. 1, jan 2022. [Online]. Available: https://doi.org/10.1145/3485126

[75] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[76] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, "Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios," *Ieee Access*, vol. 8, pp. 23 022–23 040, 2020.

[77] T. Shaowang, N. Jain, D. D. Matthews, and S. Krishnan, "Declarative data serving: The future of machine learning inference on the edge," *Proc. VLDB Endow.*, vol. 14, no. 11, p. 2555–2562, jul 2021. [Online]. Available: https://doi.org/10.14778/3476249.3476302

[78] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19, 2019, p. 304–317.

[79] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE network*, vol. 27, no. 4, pp. 16–21, 2013.

[80] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[81] J. Sun and G. Li, "An end-to-end learning-based cost estimator," *Proc. VLDB Endow.*, vol. 13, no. 3, p. 307–319, nov 2019. [Online]. Available: https://doi.org/10.14778/3368289.3368296

[82] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE, 2020, pp. 1–5.

[83] U. Thakker, J. Beu, D. Gope, G. Dasika, and M. Mattina, "Run-Time Efficient RNN Compression for Inference on Edge Devices," in *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, 2019, pp. 26–30.

[84] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghunathan, "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 13–26, jun 2017.

[85] S. Wan, Z. Gu, and Q. Ni, "Cognitive computing and wireless communications on the edge for healthcare service robots," *Computer Communications*, vol. 149, pp. 99–106, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419307960

[86] D. Wang, F. Zhang, W. Wan, H. Li, and X. Du, "Finequery: Fine-grained query processing on cpu-gpu integrated architectures," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 355–365.

[87] Y. Wen, Z. Wang, and M. F. O'boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *2014 21st International conference on high performance computing (HiPC)*. IEEE, 2014, pp. 1–10.

[88] Y. Xiang and H. Kim, "Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 392–405.

[89] W. Xu, O. Curé, and P. Calvez, "Succinctedge: A succinct rdf store for edge computing," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 2857–2860, aug 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415493

[90] Y. Yang, Q. Cao, and H. Jiang, "Edgedb: An efficient time-series database for edge computing," *IEEE Access*, vol. 7, pp. 142 295–142 307, 2019.

[91] Z. Yao, S. Cao, W. Xiao, C. Zhang, and L. Nie, "Balanced sparsity for efficient dnn inference on gpu," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 5676–5683.

[92] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "Qos prediction for service recommendation with deep feature learning in edge computing environment," *Mobile networks and applications*, vol. 25, no. 2, pp. 391–401, 2020.

[93] B. W. Yogatama, W. Gong, and X. Yu, "Orchestrating data placement and query execution in heterogeneous cpu-gpu dbms," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2491–2503, 2022.

[94] C. Zhang, F. Zhang, X. Guo, B. He, X. Zhang, and X. Du, "iMLBench: A Machine Learning Benchmark Suite for CPU-GPU Integrated Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1740–1752, 2020.

[95] F. Zhang, Z. Chen, C. Zhang, A. C. Zhou, J. Zhai, and X. Du, "An Efficient Parallel Secure Machine Learning Framework on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2262–2276, 2021.

[96] F. Zhang, L. Yang, S. Zhang, B. He, W. Lu, and X. Du, "{FineStream}:{Fine-Grained}{Window-Based} stream processing on {CPU-GPU} integrated architectures," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 633–647.

[97] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen, "Understanding co-running behaviors on integrated CPU/GPU architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 905–918, 2016.

[98] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and X. Du, "POCLib: A high-performance framework for enabling near orthogonal processing on compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 459–475, 2022.

[99] K. Zhang, J. Hu, B. He, and B. Hua, "Dido: Dynamic pipelines for in-memory key-value stores on coupled cpu-gpu architectures," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 671–682.

[100] Y. Zhang, F. McQuillan, N. Jayaram, N. Kak, E. Khanna, O. Kislal, D. Valdano, and A. Kumar, "Distributed deep learning on data systems: A comparative analysis of approaches," *Proc. VLDB Endow.*, vol. 14, no. 10, p. 1769–1782, jun 2021. [Online]. Available: https://doi.org/10.14778/3467861.3467867

[101] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, "Accelerating large scale real-time gnn inference using channel pruning," *Proc. VLDB Endow.*, vol. 14, no. 9, p. 1597–1605, may 2021. [Online]. Available: https://doi.org/10.14778/3461535.3461547

[102] Z. Zhou, K. Chen, X. Li, S. Zhang, Y. Wu, Y. Zhou, K. Meng, C. Sun, Q. He, W. Fan *et al.*, "Sign-to-speech translation using machine-learning-assisted stretchable sensor arrays," *Nature Electronics*, vol. 3, no. 9, pp. 571–578, 2020.