sensors

MDPI

Article

Multi-Model Running Latency Optimization in an Edge Computing Paradigm

Peisong Li [1], Xinheng Wang [1,*], Kaizhu Huang [2], Yi Huang [3], Shancang Li [4] and Muddesar Iqbal [5]

[1] School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou 215123, China
[2] Data Science Research Center, Division of Natural and Applied Sciences, Duke Kunshan University, Suzhou 215316, China
[3] Department of Electrical and Computer Engineering, University of Liverpool, Liverpool L69 3BX, UK
[4] School of Computer Science and Informatics, Cardiff University, Cardiff CF10 3AT, UK
[5] Renewable Energy Laboratory, Communications and Networks Engineering Department, College of Engineering, Prince Sultan University, Riyadh 11586, Saudi Arabia
* Correspondence: xinheng.wang@xjtlu.edu.cn

Abstract: Recent advances in both lightweight deep learning algorithms and edge computing increasingly enable multiple model inference tasks to be conducted concurrently on resource-constrained edge devices, allowing us to achieve one goal collaboratively rather than getting high quality in each standalone task. However, the high overall running latency for performing multi-model inferences always negatively affects the real-time applications. To combat latency, the algorithms should be optimized to minimize the latency for multi-model deployment without compromising the safety-critical situation. This work focuses on the real-time task scheduling strategy for multi-model deployment and investigating the model inference using an open neural network exchange (ONNX) runtime engine. Then, an application deployment strategy is proposed based on the container technology and inference tasks are scheduled to different containers based on the scheduling strategies. Experimental results show that the proposed solution is able to significantly reduce the overall running latency in real-time applications.

Keywords: edge computing; latency optimization; multi-model; task scheduling; autonomous driving; AI

check for updates

Citation: Li, P.; Wang, X.; Huang, K.; Huang, Y.; Li, S.; Iqbal, M. Multi-Model Running Latency Optimization in an Edge Computing Paradigm. *Sensors* **2022**, *22*, 6097. https://doi.org/10.3390/s22166097

Academic Editor: Arturo de la Escalera Hueso

Received: 22 July 2022
Accepted: 11 August 2022
Published: 15 August 2022

1. Introduction

The bandwidth congestion and heavy load on the core network always make traditional cloud computing unable to process data instantly and cause extra energy consumption. In order to combat latency in a real-time computing environment, a new dubbed edge computing paradigm is proposed, which is able to relocate the services originally hosted in the cloud server to the proximity of end devices [1].

Nowadays, Internet of Things (IoT) devices are usually equipped with abundant sensors and generate a large volume of data at the network edge [2]. However, it is often infeasible to transfer these massive data to the cloud because of the restricted network bandwidth and constraint reaction time [3]. As a result, there is a demand for artificial intelligence (AI) services to be deployed at the network edge, near where the data is generated. This demand has resulted in the convergence of edge computing and AI, culminating in a new paradigm—AI at the edge, also known as edge AI [4].

Edge AI is widely employed in the automotive industry [5]. For example, in Autonomous Driving (AD), AI in automotive can recognize dangerous situations by monitoring different sensors such as camera, light detection and ranging (LiDAR), and radio detection and ranging (RADAR) [6]. In this process, there are multiple tasks from different sensors that need different AI algorithms to run concurrently [7]. However, some problems hindering the development of autonomous driving must be addressed:

Firstly, much of the current autonomous driving technology development is focusing on improving target detection capabilities while ignoring considering how to reduce the overall system latency [8]. Autonomous driving systems must meet strict safety requirements and the vehicles should have the ability to control themselves autonomously as soon as possible [9]. So, one of the main challenges is that the task completion time must be low enough.

Secondly, the vehicles are typically equipped with heterogeneous computing hardware in order to achieve more processing capabilities and offer increasingly computation-intensive services at a lower cost [10]. For example, a system with traditional CPU elements can execute general computations; Graphics Processing Unit (GPU) and Field Programmable Gate Array (FPGA) can provide much more powerful computing capabilities for certain kinds of workloads in a cost-efficient way [11]. In this context, how to be compatible with heterogeneous platforms when deploying diverse AI models is challenging.

Thirdly, recent advances in deep learning techniques allowed significant improvements in the detection accuracy and running time of computer vision algorithms, accelerating their deployment in autonomous driving and industrial embedded systems [12]. Compared to cloud servers, embedded edge devices provide advantages such as low latency, low power consumption, low price, ease of deployment, etc. [13]. However, how to run multiple models concurrently on a single embedded device is still a new paradigm that is being explored.

In summary, the following challenges must be addressed:

(1) *Running latency*. In many cases, the tasks are related and aim to collaboratively reach one goal instead of obtaining high quality in each standalone task. Therefore, one critical question in autonomous systems is how to reduce the overall running latency to meet the safety requirement.
(2) *Hardware heterogeneity*. The computing devices are heterogeneous, consisting of CPUs, GPUs, FPGAs, and dedicated accelerators [14]. Therefore, the portability of AI models across different platforms is crucial.
(3) *AI on embedded edge*. The DL model inference requires high memory and computational requirements. Fitting these algorithms onto embedded devices is a challenge in itself.

In order to address the aforementioned challenges, in this paper, we propose a scheduling strategy-based multi-model task execution system. Firstly, a runtime optimizer is designed to reduce the overall running latency by running multiple model inferences in a collaborative way. Secondly, the models trained on cloud servers are compressed and converted to ONNX, a cross-platform, high-performance inference engine for AI models trained from a variety of frameworks. Finally, a set of separately trained methods are held on embedded devices. Multiple apps are concurrently deployed on an embedded device using container technology and select an appropriate model to execute, respectively, based on the proposed runtime optimizer.

The main contributions are summarised as follows:

(1) A real-time tasks scheduling strategy is proposed, in which multi-model tasks can be scheduled using a collaborative decision-making algorithm aiming to reduce the overall running latency without compromising the performance.
(2) A DL model convert solution is proposed that can convert trained models from Tensorflow/Pytorch to ONNX to make an edge device able to concurrently run multiple DL workloads.
(3) To address the heterogeneity of the edge computing system, a concurrent containerization scheme over the ONNX architecture is introduced for application.

In the next Section, we will introduce the related work.

2. Related Work

In this section, we reviewed the pioneer works on the development of running AI models on edge devices.

2.1. AI Inference from Cloud to Edge

Recently, in order to fully realize the potential of big data, there has been an urgent need to deploy AI services to the network edge [15]. To address this demand, edge AI has emerged as a promising solution, appearing in an increasing number of application scenarios [16–18].

Chen et al. [19] designed a Deep Learning-based Distributed Intelligent Video Surveillance (DIVS) system using an edge computing architecture. Gong et al. [20] proposed an intelligent cooperative edge (ICE) computing that combines AI and edge computing to build a smart IoT. In this paper, the generated data and AI model are distributed among IoT devices, edge and cloud.

However, in order to further improve the accuracy, Deep Neural Networks (DNNs) become deeper and with huge parameters to be processed [21]. The existing works focus mostly on higher accuracy, however, do not consider the hardware requirements. This causes a schism between software and hardware designs, resulting in algorithms that are highly accurate yet impossible to be implemented on embedded edge platforms with limited resources.

2.2. The Deployment of AI Model on Embedded System

Techniques for reducing the size and connectivity of AI model network architecture are attractive for deploying these models on embedded systems. Pruning [22] and quantization [23] are the two basic ways for reducing model memory footprint.

Minakova et al. [24] proposed using both task-level and data-level parallelism simultaneously for running Convolutional Neural Networks (CNN) inference on embedded devices. In this way, a high-throughput execution of CNN inference can be ensured and the restricted processing capabilities of embedded SoCs (Systems on Chip) can be fully used. Dey et al. [25] proposed using a partial execution strategy and partitioning algorithm for embedded systems to support scenarios that need to deal with heavy input data and huge model inferences in a short period of time.

Furthermore, TensorFlow Lite and TensorRT are considered cutting-edge inference compilers that incorporate most of the compiler optimization approaches offered for embedded devices. In [26], a detailed performance comparison of two recent compilers, TensorFlow Lite (TFLite) and TensorRT, is performed using commonly-used AI models on various edge devices.

However, the computing platform is heterogeneous and composed of various hardware components. In an autonomous vehicle, for example, the computing platform typically includes GPUs, CPUs, FPGAs and other dedicated deep learning accelerators [27]. Different frameworks for deploying AI models on hardware platforms can result in significantly different results.

2.3. Multi-Model Data Fusion

Nowadays, the fusion of multiple AI models has become a new paradigm. In [28], Mujica et al. introduced a novel messaging system-based edge computing platform to address the problem of how to efficiently fuse various data generated on heterogeneous hardware platforms. In [29], Fu et al. proposed fusing multiple roadside sensors, such as cameras and radar, on the edge of IoT networks to provide environment perception services for autonomous driving. In [30], an effective fusing approach for fusing the LiDAR and camera features is described, which is then processed for target detection and trajectory prediction. Mendez et al. [31] also designed a sensor fusion method integrating LiDAR and camera sensors for object detection that takes into account the vehicle's limited computing capabilities while simultaneously reducing running latency by using edge computing architecture.

However, these methods only focus on precision while they did not consider how to reduce the processing latency.

2.4. Optimization of Latency

Edgent, proposed by Li et al. [4], employs edge computing for DNN model inference by collaborating device and edge. Model partitioning is utilized in this framework to separate computing tasks, and model right-sizing is used to further reduce latency by exiting inference at an appropriate intermediate DNN layer. In order to eliminate the time difference between data collection of sensors and approximately synchronize the data, Warakagoda et al. [32] only gather the data, including steering angle, LiDAR reading, and camera image, which are generated almost simultaneously. This approach solves the problem of the sensors operating at different frequencies. In [8], a multi-task environment detection framework is applied to autonomous driving with reduced time and power consumption. In this study, the vehicle detection model and lane detection model are combined to sense the vehicles' surroundings and the weight pruning technique based on the Alternate Direction Method of Multipliers (ADMM) is utilized to decrease the running latency.

However, these studies focus on optimizing the inference time of a single AI model; time optimization for running multiple models on a single edge device is still a new paradigm that needs to be investigated.

This paper provides an effective solution to a difficult industrial challenge: Optimizing the edge computing architecture to reduce the latency when executing multiple model inferences on one edge device.

3. Problem Formulation

3.1. The Scenario

In the case of autonomous driving, for example, in order to better understand its surrounding environment, a vehicle must have multiple cameras mounted [33]. Depending on its function, each camera generates images at different frequencies, which can subsequently be processed and analyzed simultaneously by the vehicle's object detection and recognition application. The architecture of the multi-model-based object detection system for autonomous driving is shown in Figure 1.

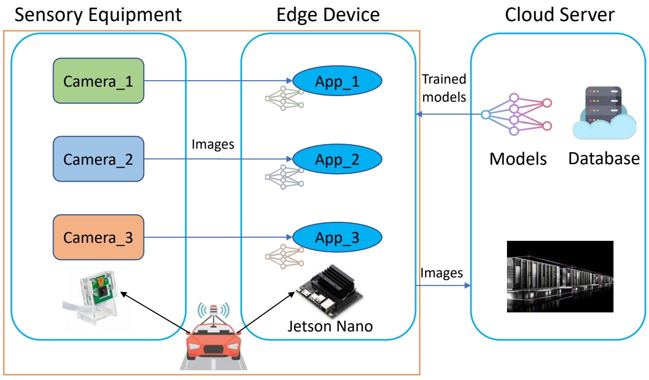


Figure 1. The architecture of the multi-model-based object detection system for autonomous driving.

As indicated in Figure 1, appropriate deep learning models are selected for training in the Cloud server based on the vehicle's object detection requirements as the cloud server has adequate computing capacity. The models trained in various frameworks are then converted to ONNX format. Each model is packaged into a container and deployed in the vehicle once it has been trained and converted into the cloud.

The procedure of object detection, as shown in Figure 2, can be divided into three stages: data collecting, data processing and information fusion. Firstly, three sensors (cameras) installed on the vehicle take pictures at a fixed frequency to acquire image data; then three different instances of the application (App)–one process for each camera–perform image-based object detection tasks independently with different AI models to detect vehicles, traffic lights and pedestrians in the images; finally, the system fuses the detection results to obtain the final execution instructions, which we call the collaborative decision making.

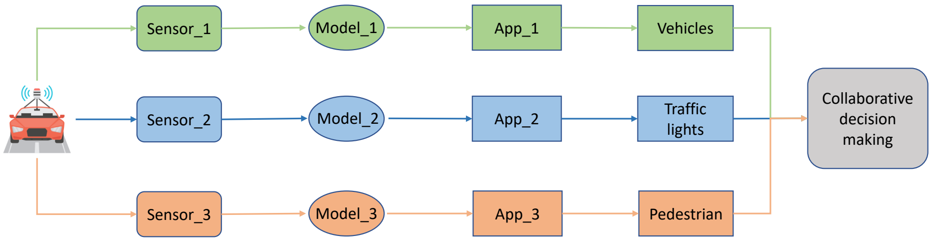


Figure 2. Data processing procedure of object detection and surrounding environment perception in autonomous driving.

3.2. Problem Formulation

The executive duration of the task in each application (App) can be formulated as Equation (1).

$$T_{task} = T_{in} + T_{pro} + T_{inf} + T_{out} \tag{1}$$

where T_{in}, T_{pro}, T_{inf}, and T_{out} denote the time for inputting images, image processing, model inference and outputting result, respectively. Among them, the model inference accounts for the main part.

The running time of each App after the n rounds of tasks allocation can be formulated as Equation (2).

$$T_A = \sum_{i=1}^{n} T_{task_i} \tag{2}$$

In this paper, we aim to reduce the accumulated overall multi-model running time. Because the collaborative decision-making occurs only after all the three tasks in three Apps in each batch have been completed, the overall running time is determined by the App that is the last to complete the n rounds of the task. In this case, the overall running time T can be calculated by Equation (3). The parameters are illustrated in Figure 3, the arrow with different colors represents different object detection tasks, and the length of the arrow denotes the task's executive duration. In Figure 3, after the first three rounds of tasks allocation, the overall running latency T is T_{A_3}.

$$T = \max\{T_{A_1}, T_{A_2}, T_{A_3}\} \tag{3}$$

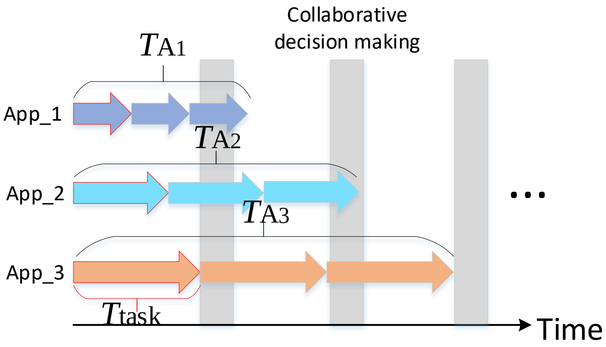


Figure 3. Schematic of tasks execution on the embedded edge device.

However, this process does not easily allow for fast synchronization across multiple cameras. In order to make a final decision fast, we introduced the following scheduling strategies in Section 4.

4. System Design

This section introduces a typical scenario to apply the proposed method. Then, the designed two kinds of scheduling strategies are presented. The target is to minimize the average data fusion latency when multiple model inference workloads are running simultaneously on one edge device.

4.1. Optimal Selection Method

The first allocation strategy we have designed is the Optimal Solution Selection Method (OSSM). By the method, the optimal allocation solution is chosen each time, so that the overall latency can be guaranteed to be minimal. The method is described in detail as follows:

Firstly, there are n sensor inputs connected to one embedded edge device.

$$S = \{s_1, s_2, \cdots, s_i, \cdots, s_n\} \tag{4}$$

where s_i represents the ith sensor input, n denotes the number of sensors.

Moreover, there are n Apps deployed at each edge device.

$$A = \{a_1, a_2, \cdots, a_i, \cdots, a_n\} \tag{5}$$

where a_i represents the ith App.

For n inputs, each input task requires different processing and inference latency.

$$L = \{l_1, l_2, \cdots, l_i, \cdots, l_n\} \tag{6}$$

where l_i represents the inference latency of s_i. The latency to complete a single inference task on a specific device is a basically fixed value.

Since the n tasks need to be assigned to n Apps, so there are $n!$ assigning choices. the assigning matric is shown in Equation (8).

$$C = \{c_1, c_2, \cdots, c_i, \cdots, c_{n!}\} \tag{7}$$

where c_i represents the ith allocation policy.

$$\begin{array}{ccc} & \begin{array}{cccc} a_1 & a_2 & \cdots & a_n \end{array} & \\ \begin{array}{c} c_1 \\ c_2 \\ \vdots \\ c_{n!} \end{array} & \begin{bmatrix} s_1 & s_2 & \cdots & s_n \\ s_2 & s_1 & \cdots & s_n \\ \cdots & \cdots & \cdots & \cdots \\ s_{n!} & s_{n-1} & \cdots & s_1 \end{bmatrix} & \begin{array}{c} T_{c_1} \\ T_{c_2} \\ \vdots \\ T_{c_{n!}} \end{array} \end{array} \tag{8}$$

As shown in Equation (8), in each round of requests the n tasks will be assigned to different apps. Firstly, we can try each assigning choice and obtain the T_{a_i,c_k} by Equation (9).

$$T_{a_i,c_k} = T_{a_i}(r-1) + l_j, \quad i,j = 1,2,\cdots,n; \; k = 1,2,\cdots,n! \tag{9}$$

where r represents the rth round of tasks assigning, T_{a_i,c_k} is the accumulated running time of App a_i under the allocation policy c_k.

Then, we can obtain the accumulated latency of each app under one specific choice by Equation (9). The data fusion latency depends on the app with maximum latency, so the data fusion latency of each assigning choice can be calculated using Equation (10).

$$T_{c_k} = \max\{T_{a_1}, T_{a_2}, \cdots, T_{a_n} \mid c_k\} \tag{10}$$

in which T_{c_k} represents the accumulated running time under the kth allocation policy in the rth round of request.

In order to minimize the running latency, we choose the assigning c_x with the minimum latency by Equation (11).

$$c_x = \min\{T_{c_1}, T_{c_2}, \cdots, T_{c_{n!}}\} \tag{11}$$

All n tasks can be allocated to n different apps based on the assigning choice c_x, and the accumulated latency of each app can be calculated by Equation (12).

$$T_{a_i}(r) = T_{a_i}(r-1) + l_{k \mid c_x, a_i} \tag{12}$$

$$T_{a_i}(r) = \sum_{m=1}^{r} l_{k|m} \tag{13}$$

in which $T_{a_i}(r)$ represents the accumulated running time of ith application at the rth round of request.

The pseudocode of this process is shown in Algorithm 1:

Algorithm 1 Scheduling strategy 1

```
Input: Initialization: $L$, $A$, $T_{a_i}$
Output: Selection of scheduling combination $c_x$
  $r \leftarrow 1$
  for $i \leftarrow 1$ to $n$ do
    $T_{a_i}(r) \leftarrow l_i$
  end for
  repeat
    $r \leftarrow r + 1$
    for $k \leftarrow 1$ to $n!$ do
      for $i \leftarrow 1$ to $n$ do
        Update $T_{a_i,c_k}$ based on Equation (9)
      end for
      Update $T_{c_k}$ based on Equation (10)
    end for
    Update $c_x$ based on Equation (11)
    for $j \leftarrow 1$ to $n$ do
      Update $T_{a_j}(r)$ based on Equation (12)
    end for
  until Scheduling finished
```

As shown in Algorithm 1, in this case, the time complexity is:

$$T(n) = O(n!) \tag{14}$$

The space complexity is:

$$S(n) = O(1) \tag{15}$$

At each allocation, this strategy compares all the allocation solutions and selects the one that results in the minimal running latency; therefore, this strategy can definitely obtain the optimal solution and minimize the running latency of multi-model inference tasks. However, the disadvantage of this strategy is that it takes a long execution time for allocation strategy-making when running a large number of model inferences simultaneously. In order to tradeoff between the execution time of the allocation strategy and the execution time of multi-model inference tasks, a second strategy is proposed in Section 4.2.

4.2. Simplest Allocation Method

The second allocation strategy we have devised is the Simplest Allocation Method (SAM). This method ensures that both the overall latency and the complexity of the algorithm can be reduced. The method is described in detail as follows:

Firstly, the n sensor inputs are ranked from the maximum to the minimum:

$$S_{rank} = Rank\{s_1, s_2, \cdots, s_i, \cdots, s_n \mid descending\} \tag{16}$$

At each round, the accumulated running latency of n apps is ranked from the minimum to the maximum:

$$T_{rank} = Rank\{T_{a_1}, T_{a_2}, \cdots, T_{a_n} \mid ascending\} \tag{17}$$

At last, the ith task in the S_{rank} is assigned to the ith app in the T_{rank}:

$$T_{rank}(r) = T_{rank}(r-1) + S_{rank} \tag{18}$$

The pseudocode of this process is shown in Algorithm 2:

Algorithm 2 Scheduling strategy 2

```
Input: Initialization: $L$, $A$, $T_{a_i}$
Output: Selection of scheduling combination $c_x$
  $r \leftarrow 1$
  for $i \leftarrow 1$ to $n$ do
    $T_{a_i}(r) \leftarrow l_i$
  end for
  Update $S_{rank}$ based on Equation (16)
  repeat
    $r \leftarrow r + 1$
    Update $T_{rank}$ based on Equation (17)
    for $i \leftarrow 1$ to $n$ do
      $T_{a_i}(r) \leftarrow T_{a_i}(r-1) + S_{rank}\{i\}$
    end for
  until Scheduling finished
```

As shown in Algorithm 2, in this case, the time complexity is:

$$T(n) = O(n \log n) \tag{19}$$

The space complexity is:

$$S(n) = O(1) \tag{20}$$

The mathematical modeling of the strategy is shown in Figure 4. As shown in the figure, at each allocation, the task requiring the longest execution time is assigned to the currently idle application, until the task requiring the least execution time is assigned to the currently occupied application. This strategy can effectively reduce the overall running delay.

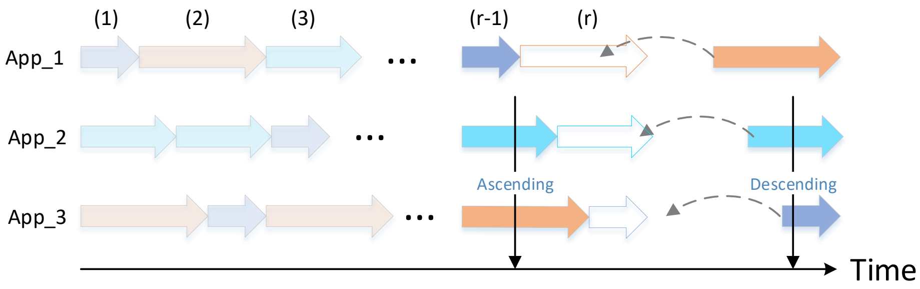


Figure 4. The mathematical modeling of the Simplest Allocation Method.

4.3. Overall System Description

The proposed optimized multi-model task scheduling and execution mechanism, as well as the schematic, are shown in Figure 5 and Figure 6, respectively.

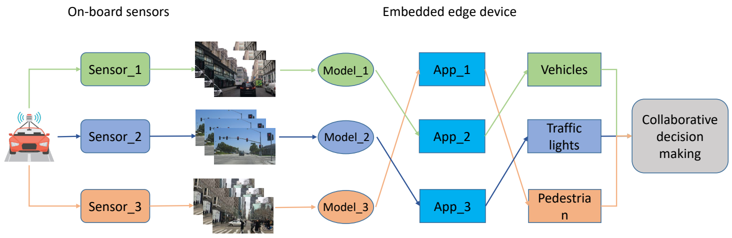


Figure 5. Optimized multi-model tasks scheduling-based object detection procedure.

As shown in Figure 5, images captured by various sensors are buffered on the edge device. Based on the proposed allocation strategy, each application (App) dynamically invokes a different AI model, which then conducts an object detection task on the input image and outputs the detection result. Finally, the collaborative decision is made based on the fusion of the detection results.

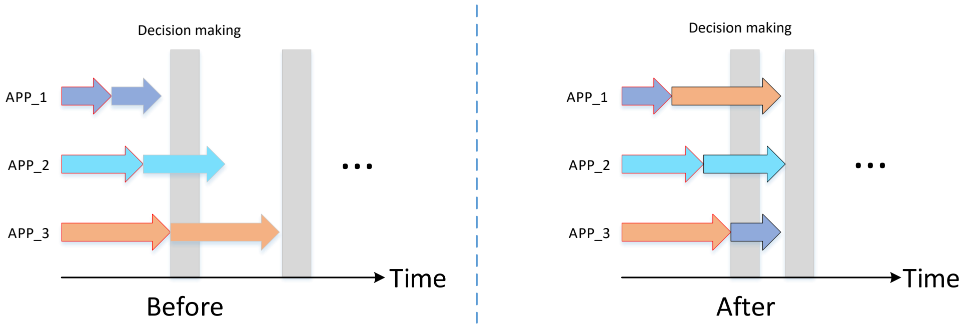


Figure 6. Comparison of the running time before and after the implementation of scheduling strategy.

Figure 6 compares the running time before and after the tasks scheduling strategy is implemented. Before adopting the scheduling strategy, each application continuously invokes a fixed model to perform a fixed object detection task. The collaborative decision-making (gray area in the figure) is performed only when all the tasks in three applications in each batch have been completed. Following the implementation of the scheduling strategy, each application dynamically performs different tasks based on the allocation decision. As shown in the figure, distributing multiple tasks to different applications at the same time based on the current state of each application efficiently reduces the time interval between two decision-making, achieving the goal of lowering the overall running time.

Figure 7 shows the optimized system architecture, which can be divided into four stages: *sensing*, *scheduling*, *detection* and *final decision-making*. At first, the cameras mounted on the vehicle capture images of the surrounding environment; Secondly, the images are assigned to different applications based on the scheduling strategy and the status of each application (App), this process is deployed on the *Runtime Optimizer*; Then, each application invokes a different AI model to process the image, perform the AI-based target detection task, and output the detection results; Finally, the detection results from each application are forwarded to the final decision-making block, where they are fused and being post-processed for decision making.

The role of the *Runtime Optimizer* is to receive requests, perform a scheduling strategy, and then allocate the requests to each application for inference. Since the inference time for each model differs, once all three models have been executed, the detection results are then fused. So the overall running latency (execution time) for final decision-making is determined by the model that has the longest inference time.

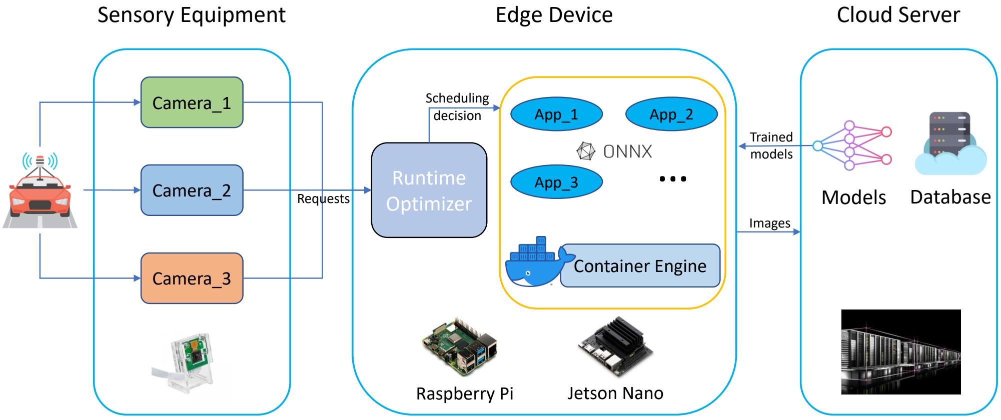


Figure 7. The edge computing architecture of the multi-model-based object detection system for autonomous driving.

5. Experiment

In this section, we describe the experimental setup in detail and demonstrate the latency advantages of our approaches.

5.1. Experimental Setup

In order to measure the performance of concurrent workloads execution at edge devices, in our test-bed, MobileNet, ShuffleNet and SqueezeNet are deployed on NVIDIA Jetson nano 2 GB.

5.1.1. Hardware

The *Nvidia* Jetson nano is chosen in the experiments as an embedded edge device. Technical Specifications (SPECS) of Jetson nano 2 GB is shown in Table 1.

Table 1. Technical Specifications.

Technical Specifications	
GPU	128-core NVIDIA Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	2 GB 64-bit LPDDR4 25.6 GB/s
Storage	64 GB

5.1.2. Software

Three deep learning models, including MobileNet, ShuffleNet and SqueezeNet, are used in the experiment to evaluate the performance of the proposed scheduling strategies.

(1) MobileNet: As a lightweight deep neural network, MobileNet models are very efficient in terms of speed and size and hence are ideal for embedded and mobile applications.
(2) ShuffleNet: An extremely computation-efficient CNN model designed specifically for mobile devices with very limited computing power.
(3) SqueezeNet: SqueezeNet models are highly efficient in terms of size and speed while providing good accuracy. This makes them ideal for platforms with strict constraints on size.

5.1.3. Model Deployment

(1) Model inference is executed based on *ONNX Runtime* Environment. ONNX Runtime is a cross-platform inference and training machine-learning accelerator, which supports models from various deep learning frameworks and is compatible with different hardware [34].

(2) Containerized app deployment. Each application served for model inference on edge devices is packaged and run as containers. Containerization technology can naturally shield hardware heterogeneity and bring great convenience to deployment and management.

(3) In this experiment, only the CPU is used to perform the model inference task. On the one hand, CPUs are ubiquitous and can be more cost-effective than GPUs for running AI-based tasks on resource-constrained embedded edge devices. On the other hand, we proposed using the ONNX runtime framework for model inference, which uses CPU and can speed up the model inference and result in lower costs, faster response times, and a more portable algorithm.

In addition, the image acquisition and analysis tasks were executed 1000 times. In order to validate the benefits of our proposed method, we implement and compare two proposed strategies against the system without scheduling.

5.2. Performance Evaluation

For this experiment, the main evaluation metric is overall running time, while the CPU usage and memory usage are also monitored during task execution.

5.2.1. Overall Running Time

Before applying the strategy, the inference time of each model is shown in Figure 8.

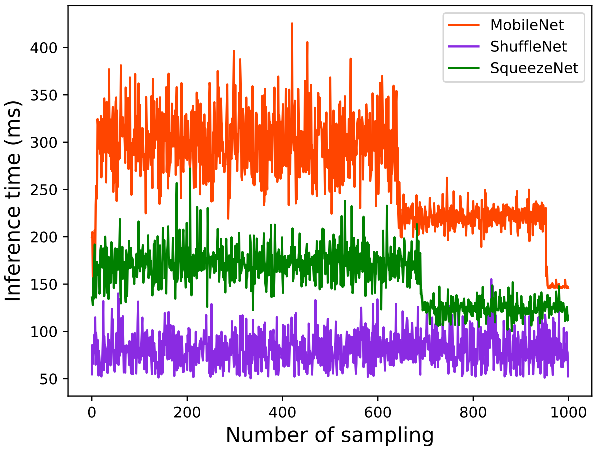


Figure 8. Inference time of each model without scheduling.

The overall running latency of each application is shown in Figure 9.

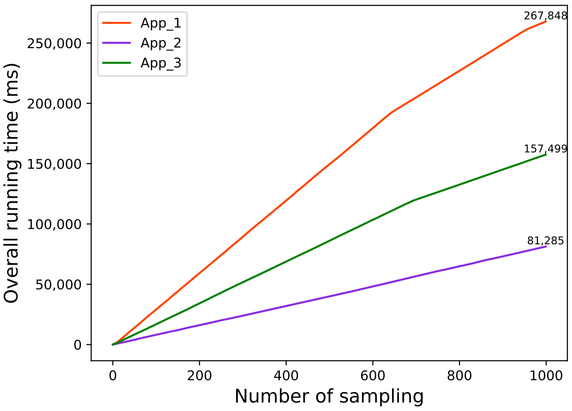


Figure 9. Overall running time of each App without scheduling.

The overall running latency of performing data fusion is determined by the App with the longest running time. As shown in Figure 8, MobileNet inference takes the longest running time, so the latency of data fusion is determined by MobileNet. According to Figure 9, MobileNet takes a total of 267,848 ms to conduct 1000-times inference tasks, hence the overall running latency after executing 1000 tasks is 267,848 ms.

The overall running time of the three Apps based on strategy 1 and strategy 2 are shown in Figures 10 and 11, respectively.

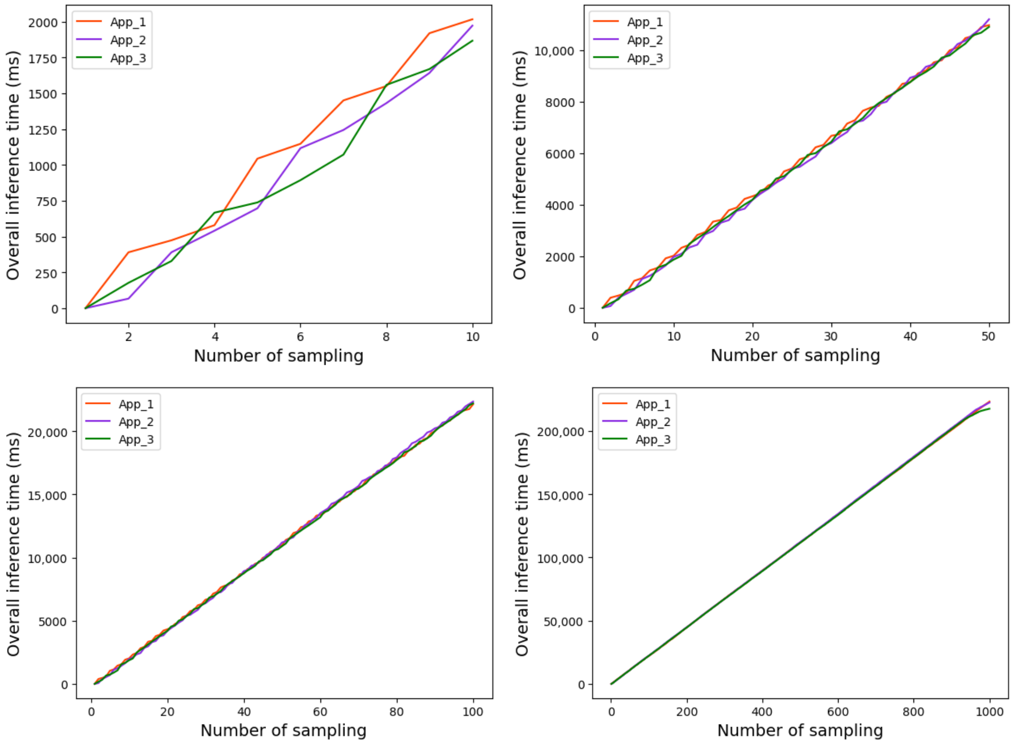


Figure 10. Overall running time of each App based on strategy 1.

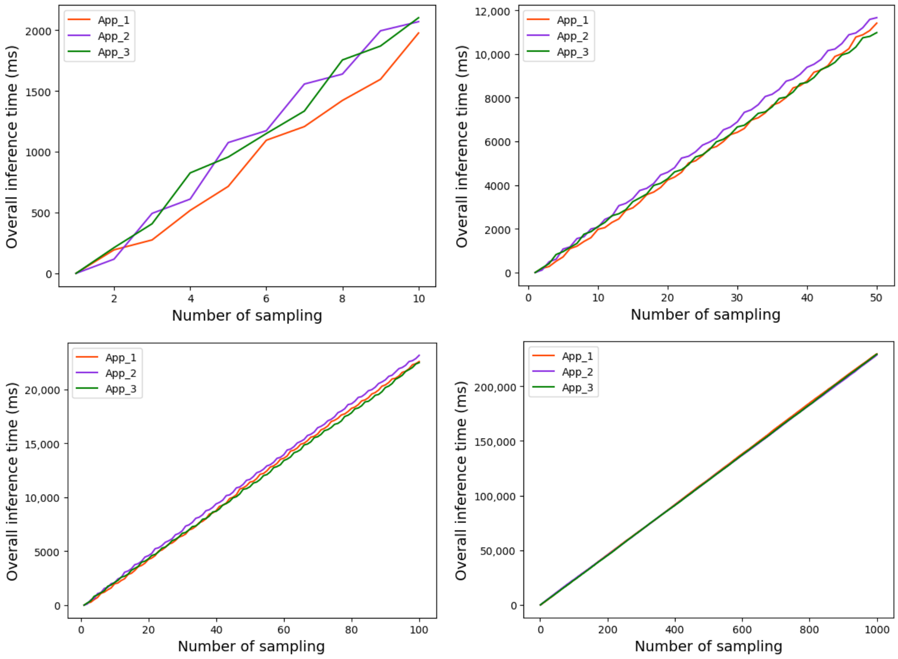


Figure 11. Overall running time of each App based on strategy 2.

As can be seen from Figure 9, before the implementation of the scheduling strategy, the time gap among the three applications becomes increasingly bigger as the tasks were executed, which resulted in time wasting. After the scheduling strategy is implemented, the time gap between the three Apps is basically not much different. In addition, as shown in Figures 10 and 11, the overall runtime latency is significantly reduced after the scheduling strategy is applied.

The comparison of the overall running time under the three conditions, without scheduling and two strategies, is shown in Figure 12.

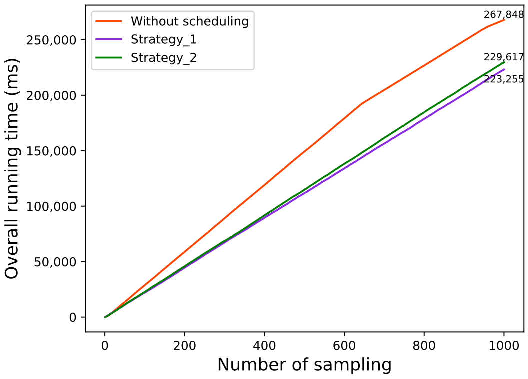
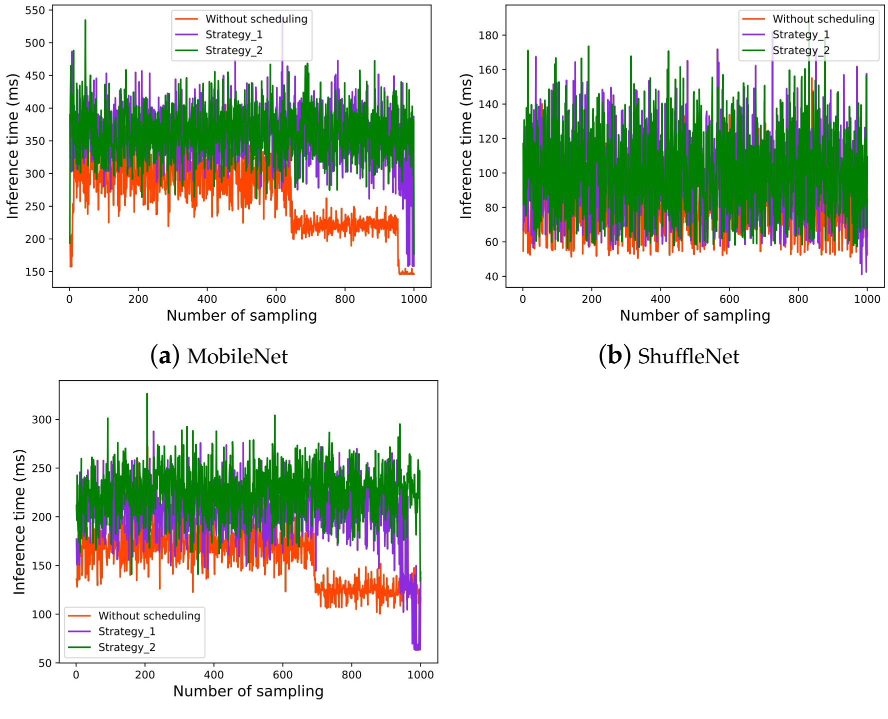


Figure 12. Comparison of overall running time under the three conditions.

As can be seen in Figure 12, compared to without scheduling strategy, Strategy 1 and Strategy 2 reduce the runtime latency by 16.7% and 14.3%, respectively. It can thus be demonstrated that adopting the scheduling mechanism efficiently reduces the overall running time of the system, resulting in a faster reaction time for vehicles.

5.2.2. Model Inference Time

The inference time is one of the most crucial metrics when deploying an AI model in a commercial setting. Most real-world applications necessitate lightning-fast inference time, ranging from a few milliseconds to one second. Inference time is measured as the time difference between the arrival time of an image and the completion time of object detection. The inference time of the three models in three situations is shown in Figure 13.



(**a**) MobileNet

(**b**) ShuffleNet

(**c**) SqueezeNet

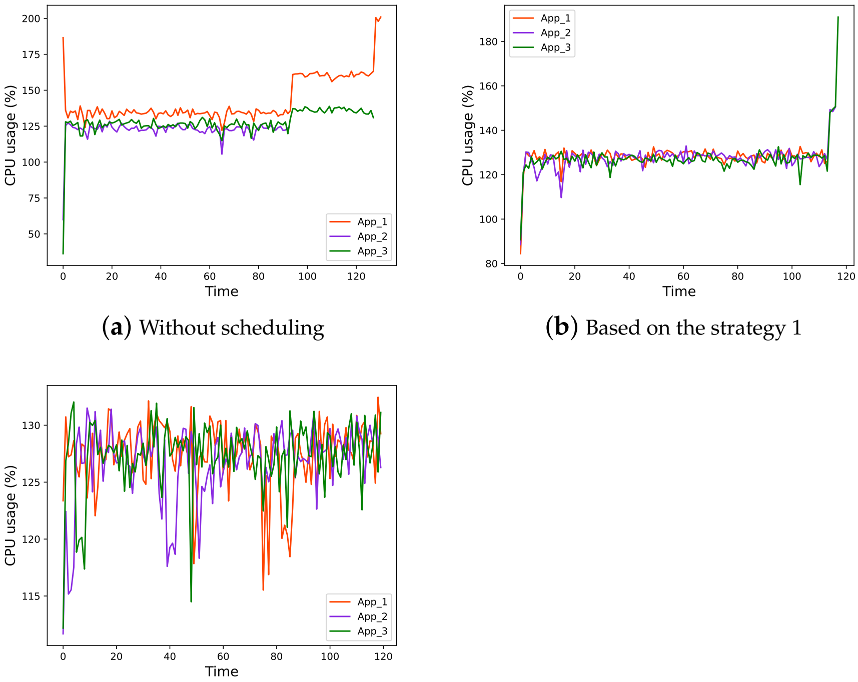
Figure 13. The inference time of the three models.

As can be seen from Figure 7, due to the implementation of *Runtime Optimizer*, tasks need to be allocated based on the state of the applications before being executed, and then the application will then invoke the corresponding AI model to perform model inference based on the task's requirements. Therefore, as shown in Figure 13, the inference time of a single task is slightly increased as compared to without scheduling.

5.2.3. Cpu Usage

We use *Docker stats* to display a live stream of container(s) resource usage statistics, including CPU and memory usage.

CPU usage in three situations is shown in Figure 14.



(**a**) Without scheduling

(**b**) Based on the strategy 1

(**c**) Based on the strategy 2

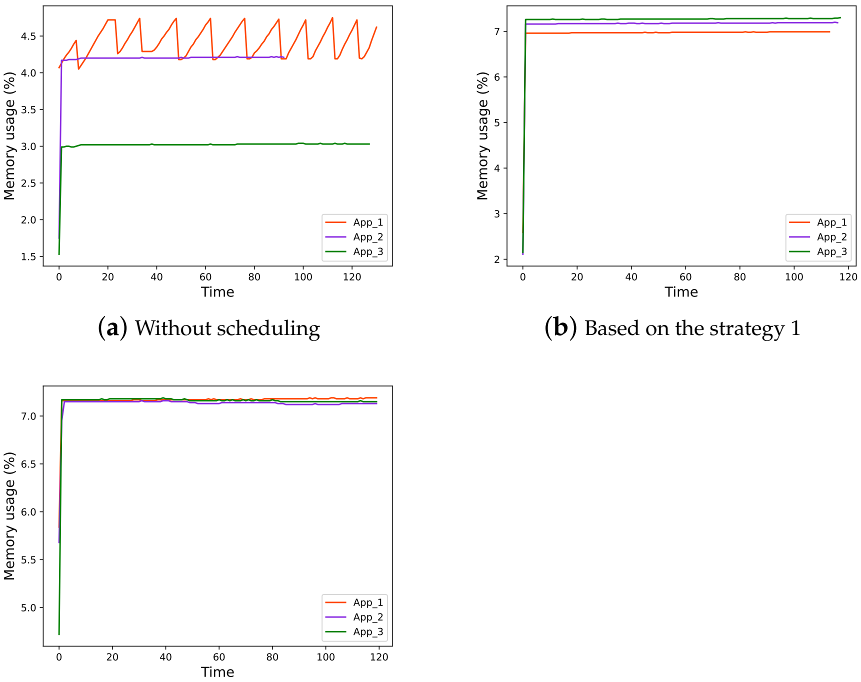
Figure 14. CPU usage of each App.

As shown from Figure 14a–c, each of the three applications takes up approximately one-third of the CPU resources, indicating that the proposed system can make full use of the resources of the embedded hardware system.

5.2.4. Memory Usage

Memory usage in three situations is shown in Figure 15.

As illustrated in Figure 15a–c, the memory usage varies between the different applications, which is related to the size of the model. As can be seen from Figure 15b,c, the memory footprint of all three applications is roughly the same at around 7%. We attribute these results to the fact that each application has essentially the same functionality and each application can call one of the three models. On the other hand, the memory footprint of the models has increased compared to before the scheduling strategy was adopted because each application has more functionality than before, and each application has to have the ability to perform reasoning for all three models.



(**a**) Without scheduling (**b**) Based on the strategy 1

(**c**) Based on the strategy 2

Figure 15. Memory usage of each App.

5.2.5. Inference Accuracy

The size and accuracy of models are crucial for deploying multiple models on resource-limited edge devices in an edge computing architecture. The model size and inference accuracy of the three models are shown in Table 2.

Table 2. Performance of the three models.

Model	Size	Top-1 Accuracy (%)	Top-5 Accuracy (%)
MobileNet	13.6 MB	70.94	89.99
ShuffleNet	9.2 MB	69.36	88.32
SqueezeNet	9 MB	56.34	79.12

5.2.6. Overhead Analysis

In summary, by adopting the scheduling strategy, the memory footprint is increased slightly but the runtime latency is reduced significantly.

6. Conclusions

This paper presents an overall running latency optimization solution for multi-model fusion. By adopting a task scheduling strategy, time-consuming tasks are assigned to applications with light workloads, thereby minimizing the overall latency of the system when handling multi-model data fusion. At the same time, the constrained platform resources for service deployment and the heterogeneity of hardware platforms can be better addressed by using the ONNX architecture and containerization technology. Experimental results also show that the approach can significantly reduce reaction time and hence improve system security. For future trends, the Deep Learning methods can be used to optimize the multiple model inference tasks allocation in Edge computing architecture. For future work, we plan to use the Reinforcement Learning (RL) method to optimize the multi-model tasks allocation strategy, which could dynamically allocate model inference tasks to different applications based on the application state and task characteristics through autonomous learning.

Author Contributions: Methodology, validation, formal analysis, investigation and writing—original draft preparation, P.L.; writing—review and editing, S.L. and M.I.; funding acquisition and supervision, X.W.; project administration, K.H. and Y.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by: Key Program Special Fund in XJTLU under project KSF-E-64; XJTLU Research Development Funding under projects RDF-19-01-14 and RDF-20-01-15; the National Natural Science Foundation of China (NSFC) under grant 52175030.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank Xi'an Jiaotong-Liverpool University for her financial support to conduct this research by Key Program Special Fund in XJTLU under project KSF-E-64, XJTLU Research Development Funding under projects RDF-19-01-14 and RDF-20-01-15, and the National Natural Science Foundation of China (NSFC) under grant 52175030.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Luo, Q.; Hu, S.; Li, C.; Li, G.; Shi, W. Resource scheduling in edge computing: A survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2131–2165. [CrossRef]
2. Holler, J.; Tsiatsis, V.; Mulligan, C.; Karnouskos, S.; Avesand, S.; Boyle, D. *Internet of Things*; Academic Press: Waltham, MA, USA, 2014.
3. Munir, A.; Blasch, E.; Kwon, J.; Kong, J.; Aved, A. Artificial intelligence and data fusion at the edge. *IEEE Aerosp. Electron. Syst. Mag.* **2021**, *36*, 62–78. [CrossRef]
4. Li, E.; Zeng, L.; Zhou, Z.; Chen, X. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Trans. Wirel. Commun.* **2019**, *19*, 447–457. [CrossRef]
5. Brandalero, M.; Ali, M.; Le Jeune, L.; Hernandez, H.G.M.; Veleski, M.; da Silva, B.; Lemeire, J.; Van Beeck, K.; Touhafi, A.; Goedemé, T.; et al. AITIA: Embedded AI Techniques for Embedded Industrial Applications. In Proceedings of the International Conference on Omni-layer Intelligent Systems (COINS), Barcelona, Spain, 23–25 August 2020; pp. 1–7.
6. Sleight, M. How Do Self-Driving Cars Work? Available online: https://www.bankrate.com/insurance/car/how-do-self-driving-cars-work/ (accessed on 22 June 2021).
7. Gupta, A. Machine Learning Algorithms in Autonomous Driving. Available online: https://www.iiot-world.com/artificial-intelligence-ml/machine-learning/machine-learning-algorithms-in-autonomous-driving/ (accessed on 4 April 2021).
8. Zhou, S.; Xie, M.; Jin, Y.; Miao, F.; Ding, C. An End-to-end Multi-task Object Detection using Embedded GPU in Autonomous Driving. In Proceedings of the 22nd International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 7–8 April 2021; pp. 122–128.
9. Liu, L.; Lu, S.; Zhong, R.; Wu, B.; Yao, Y.; Zhang, Q.; Shi, W. Computing Systems for Autonomous Driving: State of the Art and Challenges. *IEEE Internet Things J.* **2020**, *8*, 6469–6486. [CrossRef]
10. Collin, A.; Siddiqi, A.; Imanishi, Y.; Rebentisch, E.; Tanimichi, T.; de Weck, O.L. Autonomous driving systems hardware and software architecture exploration: Optimizing latency and cost under safety constraints. *Syst. Eng.* **2020**, *23*, 327–337. [CrossRef]
11. Dong, Z.; Shi, W.; Tong, G.; Yang, K. Collaborative autonomous driving: Vision and challenges. In Proceedings of the International Conference on Connected and Autonomous Driving (MetroCAD), Detroit, MI, USA, 27–28 February 2020; pp. 17–26.
12. Verucchi, M.; Brilli, G.; Sapienza, D.; Verasani, M.; Arena, M.; Gatti, F.; Capotondi, A.; Cavicchioli, R.; Bertogna, M.; Solieri, M. A systematic assessment of embedded neural networks for object detection. In Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 937–944.
13. Lin, C.; Zhang, Z.; Li, H.; Liu, J. ECSRL: A Learning-Based Scheduling Framework for AI Workloads in Heterogeneous Edge-Cloud Systems. In Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, Coimbra, Portugal, 15–17 November 2021; pp. 386–387.
14. Hao, C.; Chen, D. Software/Hardware Co-design for Multi-modal Multi-task Learning in Autonomous Systems. In Proceedings of the IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), Washington, DC, USA, 6–9 June 2021; pp. 1–5.
15. Wang, X.; Han, Y.; Leung, V.C.; Niyato, D.; Yan, X.; Chen, X. *Edge AI: Convergence of Edge Computing and Artificial Intelligence*; Springer Nature: Singapore, 2020.

16. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* **2019**, *107*, 1738–1762. [CrossRef]
17. Calo, S.B.; Touna, M.; Verma, D.C.; Cullen, A. Edge computing architecture for applying AI to IoT. In Proceedings of the IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 3012–3016.
18. Campolo, C.; Genovese, G.; Iera, A.; Molinaro, A. Virtualizing AI at the distributed edge towards intelligent IoT applications. *J. Sens. Actuator Netw.* **2021**, *10*, 13. [CrossRef]
19. Chen, J.; Li, K.; Deng, Q.; Li, K.; Philip, S.Y. Distributed deep learning model for intelligent video surveillance systems with edge computing. *IEEE Trans. Ind. Inform.* **2019**. [CrossRef]
20. Gong, C.; Lin, F.; Gong, X.; Lu, Y. Intelligent cooperative edge computing in internet of things. *IEEE Internet Things J.* **2020**, *7*, 9372–9382. [CrossRef]
21. Bi, J. Improving Training and Inference for Embedded Machine Learning. Ph.D. Thesis, University of Southampton, Southampton, UK, 2020.
22. Wu, R.T.; Singla, A.; Jahanshahi, M.R.; Bertino, E.; Ko, B.J.; Verma, D. Pruning deep convolutional neural networks for efficient edge computing in condition assessment of infrastructures. *Comput.-Aided Civ. Infrastruct. Eng.* **2019**, *34*, 774–789. [CrossRef]
23. Tonellotto, N.; Gotta, A.; Nardini, F.M.; Gadler, D.; Silvestri, F. Neural network quantization in federated learning at the edge. *Inf. Sci.* **2021**, *575*, 417–436. [CrossRef]
24. Minakova, S.; Tang, E.; Stefanov, T. Combining task-and data-level parallelism for high-throughput cnn inference on embedded cpus-gpus mpsocs. In Proceedings of the International Conference on Embedded Computer Systems, Samos, Greece, 5–9 October 2020; Springer: Cham, Switzerland, 2020; pp. 18–35.
25. Dey, S.; Mukherjee, A.; Pal, A. Embedded Deep Inference in Practice: Case for Model Partitioning. In Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems, 2019, New York, NY, USA, 10 November 2019; pp. 25–30.
26. Verma, G.; Gupta, Y.; Malik, A.M.; Chapman, B. Performance evaluation of deep learning compilers for edge inference. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, 17–21 June 2021; pp. 858–865.
27. Hao, C.; Sarwari, A.; Jin, Z.; Abu-Haimed, H.; Sew, D.; Li, Y.; Liu, X.; Wu, B.; Fu, D.; Gu, J.; et al. A hybrid GPU+ FPGA system design for autonomous driving cars. In Proceedings o the IEEE International Workshop on Signal Processing Systems (SiPS), Nanjing, China, 20–23 October 2019; pp. 121–126.
28. Mujica, G.; Rodriguez-Zurrunero, R.; Wilby, M.R.; Portilla, J.; Rodríguez González, A.B.; Araujo, A.; Riesgo, T.; Vinagre Diaz, J.J. Edge and fog computing platform for data fusion of complex heterogeneous sensors. *Sensors* **2018**, *18*, 3630. [CrossRef] [PubMed]
29. Fu, Y.; Tian, D.; Duan, X.; Zhou, J.; Lang, P.; Lin, C.; You, X. A Camera–Radar Fusion Method Based on Edge Computing. In Proceedings of the IEEE International Conference on Edge Computing (EDGE), Beijing, China, 19–23 October 2020; pp. 9–14.
30. Fadadu, S.; Pandey, S.; Hegde, D.; Shi, Y.; Chou, F.C.; Djuric, N.; Vallespi-Gonzalez, C. Multi-view fusion of sensor data for improved perception and prediction in autonomous driving. *arXiv* **2020**, arXiv:2008.11901.
31. Mendez, J.; Molina, M.; Rodriguez, N.; Cuellar, M.P.; Morales, D.P. Camera-LiDAR Multi-Level Sensor Fusion for Target Detection at the Network Edge. *Sensors* **2021**, *21*, 3992. [CrossRef]
32. Warakagoda, N.; Dirdal, J.; Faxvaag, E. Fusion of lidar and camera images in end-to-end deep learning for steering an off-road unmanned ground vehicle. In Proceedings of the 22th International Conference on Information Fusion (FUSION), Ottawa, ON, Canada, 2–5 July 2019; pp. 1–8.
33. Yang, M.; Wang, S.; Bakita, J.; Vu, T.; Smith, F.D.; Anderson, J.H.; Frahm, J.M. Re-thinking CNN frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge. In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Montreal, QC, Canada, 16–18 April 2019; pp. 305–317.
34. Microsoft. ONNX Runtime. Available online: https://microsoft.github.io/onnxruntime/ (accessed on 21 July 2022).