symmetry

MDPI

Article

An Energy-Efficient Method for Recurrent Neural Network Inference in Edge Cloud Computing

Chao Chen [1,†], Weiyu Guo [2,†], Zheng Wang [1,*], Yongkui Yang [1,*], Zhuoyu Wu [3] and Guannan Li [4,5,6]

[1] Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China
[2] Artificial Intelligence Thrust, Information Hub, Hong Kong University of Science and Technology, Guangzhou 511458, China
[3] Department of Engineering, Durham University, Lower Mountjoy, South Rd., Durham DH1 3LE, UK
[4] Marine Robot Engineering Research Center, Huzhou Institute of Zhejiang University, Huzhou 313000, China
[5] State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences (CAS), Shenyang 110016, China
[6] Hangzhou Innovation Institute, Beihang University, Hangzhou 310051, China
* Correspondence: zheng.wang@siat.ac.cn (Z.W.); yk.yang@siat.ac.cn (Y.Y.)
† These authors contributed equally to this work.

Abstract: Recurrent neural networks (RNNs) are widely used to process sequence-related tasks such as natural language processing. Edge cloud computing systems are in an asymmetric structure, where task managers allocate tasks to the asymmetric edge and cloud computing systems based on computation requirements. In such a computing system, cloud servers have no energy limitations, since they have unlimited energy resources. Edge computing systems, however, are resource-constrained, and the energy consumption is thus expensive, which requires an energy-efficient method for RNN job processing. In this paper, we propose a low-overhead, energy-aware runtime manager to process tasks in edge cloud computing. The RNN task latency is defined as the quality of service (QoS) requirement. Based on the QoS requirements, the runtime manager dynamically assigns RNN inference tasks to edge and cloud computing systems and performs energy optimization on edge systems using dynamic voltage and frequency scaling (DVFS) techniques. Experimental results on a real edge cloud system indicate that in edge systems, our method can reduce the energy up to 45% compared with the state-of-the-art approach.

Keywords: recurrent neural network; energy optimization; machine learning; edge computing; deep learning; artificial neural network; LSTM neural network; GRU neural network

Citation: Chen, C.; Guo, W.; Wang, Z.; Yang, Y.; Wu, Z.; Li, G. An Energy-Efficient Method for Recurrent Neural Network Inference in Edge Cloud Computing. *Symmetry* **2022**, *14*, 2524. https://doi.org/10.3390/sym14122524

Academic Editors: Liangmin Wang, Keyang Cheng and Haiqin Wu

Received: 22 October 2022
Accepted: 25 November 2022
Published: 29 November 2022

1. Introduction

With the rapid growth of big data and artificial intelligence (AI), researchers have developed artificial neural networks (ANN) for fast and effective data mining [1–4]. The recurrent neural network (RNN) is a particularly important ANN, and it is widely used to process temporal sequence tasks such as speech recognition, text generation and biometric authentication [5–8].

Edge cloud computing systems are in an asymmetric structure which consists of cloud servers and edge devices, as shown in Figure 1. RNN tasks are processed in edge cloud computing systems, where the training of RNNs is performed on symmetric cloud servers with various accelerators [9,10] and the inference of short tasks can be performed on edge nodes, taking into account the factors of latency, security, etc.

However, it is worth mentioning that cloud servers and edge devices have different energy requirements. On the one hand, cloud servers have no energy limitations because they are connected to the electrical grid, which provides unlimited energy resources. On the other hand, edge devices have low energy consumption requirements because they use batteries for computing and are thus constrained in energy resources.

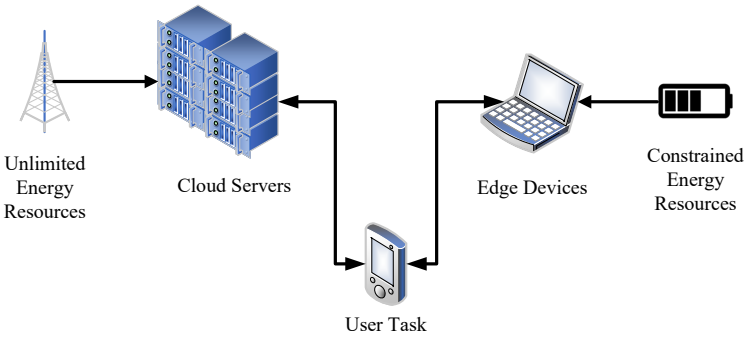

Figure 1. The topology of cloud edge computing systems.

In order to minimize energy consumption and task latency in asymmetric structures, some inference tasks are performed on edge systems, while others are processed on remote servers [11–16]. On the one hand, edge systems that are close to users can reduce task latency by avoiding task transmission to servers. On the other hand, remote cloud servers help process large and complex inference tasks.

At present, some researchers adopt input-independent task allocation strategies between asymmetric cloud and edge computing systems, while others take advantage of data characteristics and perform input-dependent optimizations such that tasks with short estimated processing times can be executed on edge systems [15–18]. It is worth mentioning that when dealing with tasks on edge systems, various techniques can be applied to minimize energy consumption while meeting quality of service (QoS) requirements, which refer to task latency in this paper.

In this paper, we propose a low-cost runtime manager to assign tasks between asymmetric edge and cloud computing systems. Our proposed manager dynamically allocates tasks based on their predicted running times using a regression task model and QoS requirements. It then leverages dynamic voltage and frequency scaling (DVFS) techniques to perform energy optimization on edge systems. The experimental results on a real edge cloud system reveal that our method can reduce the energy by up to 45% compared with existing approaches on edge systems.

The rest of this paper is organized as follows. Section 2 introduces the background of the work, and our motivation is demonstrated in Section 3. Section 4 explains the related works, the methodology is presented in Section 5, the evaluation is performed in Section 6, and Section 7 concludes the paper.

2. Background

The RNN is a type of artificial neural network that is widely adopted to tackle sequence-related tasks such as speech recognition and natural language processing. The node connections of RNNs form a directed or undirected graph, which provides the internal states to deal with variable-length sequences of inputs.

In traditional feedforward neural networks (e.g., CNNs), the current output is independent of previous outputs. The RNN leverages feedback structures. This way, the RNN saves the output of a particular layer and feeds this back to the input in order to predict the output of the layer; that is, prior outputs can be utilized by memory units (i.e., hidden layers) for the current output.

An example of an RNN architecture is illustrated in Figure 2. Figure 2a displays a rolled RNN, where x_i and y_i represent the input and output at time step i, respectively. Hidden layers are used to model memory units using a hidden state h_i such that data can be persisted into the system. This shows that hidden layers exhibit feedback structures. They send prior information to the current state, which affects the output y_i for a given input x_i.

Figure 2b unrolls the RNN from Figure 2a. We can see that the current hidden state of the hidden layers depends on previous information and the current input (i.e., h_i =

$f(h_{i-1}, x_i)$). The current output y_i is determined by the current input x_i and state h_i (i.e., $y_i = g(h_i) = g(f(h_{i-1}, x_i))$). It is clear that the prior state can affect the current output. For example, the final output y_2 depends on the hidden state h_2, which is affected by the prior state h_1. Therefore, to obtain the final result of an RNN model, the computation of previous states must be accomplished. Aside form that, the unrolled architecture indicates that the computational time increases linearly with the number of hidden states.

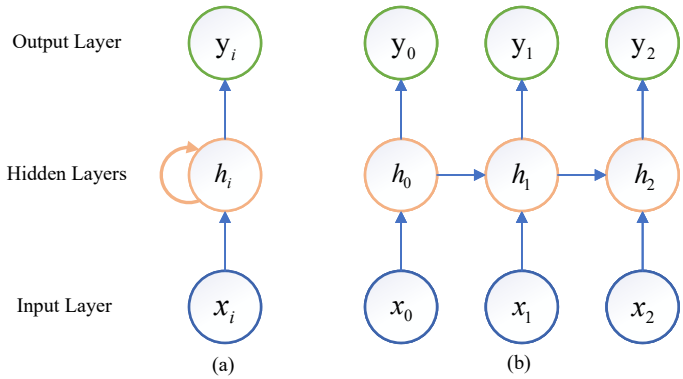

Figure 2. The architecture of an RNN (**a**) Rolled RNN, (**b**) Unrolled RNN.

3. Motivation

When dealing with RNN inference task allocations in edge cloud computing systems, existing approaches [15,16] leverage input-dependent methods to dynamically allocate tasks among different computing devices.

It is worthwhile to mention that the running time of an RNN inference task is almost linearly proportional to that of the input length. Thus, state-of-the-art approaches allocate the tasks to edge and cloud devices based on the input length. The tasks of short lengths are executed on edge systems, while others are on cloud servers such that the overall energy consumption can be reduced by utilizing the energy-efficient task processing capabilities of edge systems.

Nonetheless, we note that existing input-dependent approaches [15,16] do not take into account task latency, which we define as the QoS requirements. Modern CPUs allow for dynamic voltage and frequency scaling, which make it feasible to save system energy using DVFS techniques without QoS violations.

We adopt an edge device (as described in Section 6.1) and measure its energy consumption using a multimeter for a fixed input, as listed in Table 1. From the table, we can see that when lowering the frequency, the energy consumption decreases, and the running time increases accordingly. The energy consumption reduces by 41% when varying the frequency from 1.5 GHz to 0.6 GHz. Therefore, we conclude that DVFS can help save energy in an RNN inference task.

Table 1. RNN task energy consumption using DVFS techniques.

Frequency (GHz)	Power (W)	Time (ms)	Energy (mJ)
1.5	1.43	48	68.64
1.0	0.91	65	59.15
0.6	0.40	101	40.40

4. Related Work

Numerous methods have been proposed to deal with inference tasks in edge and cloud computing systems [19–23]. To efficiently run deep learning (DL) tasks, researchers have proposed various hardware accelerators and interconnections in edge devices. Belabed et al. [24] developed an energy-efficient DL accelerator using FPGAs. Xia et al. [25]

leverage FPGAs to process lightweight CNN models. Liu et al. [26] designed FPGA acceleration with a systolic array, matrix tiling, fixed-point precision and parallelism for compact MobileNet models. Xu et al. [27] implement FPGA acceleration for computer vision DL tasks that effectively reduced the response time and energy consumption.

Another approach is to optimize DL models for inference acceleration. Zhou et al. [28] developed a lightweight CNN model for edge computing. Kim et al. [29] optimized DL models for hardware-specific characteristics using TVM. Li et al. [30] proposed dynamic filter pruning together with a model transformation method to reduce the computational complexity. Matsubara et al. [31] used knowledge distillation to build compressed models for edge devices. Kim and Deka [32] optimized configurations for DL models to work on tensor processing units. Li et al. [33] took advantage of a greedy-based filter pruning technique to optimize DL models.

In addition, researchers utilized collaborative edge cloud systems for DL tasks. Zhou et al. [34] explored various accelerators and guided accelerator selection for tasks. Gong et al. [35] dealt with private and public data on edge and cloud devices, respectively. Feng et al. [36] proposed a blockchain-enabled technique that optimizes offloading tasks between edge devices and servers. Liu et al. [37] presented a framework that offloads computation from unmanned aerial vehicles (UAVs) to devices based on optimal decisions and resource management policies. Kennedy et al. [38] proposed a framework to offload DL tasks to virtualized GPUs. Kuang et al. [39] minimized task latency while meeting the requirements of power and energy.

Note that current methods mainly focus on feedforward networks such as CNNs, and few collaborative methods study recurrent networks. Pagliari et al. [15] managed RNN inference task mappings between edge and cloud systems using an input-dependent method, and they extended the method to a collaborative mapping engine to support an arbitrary number of devices and interconnections [16]. Nonetheless, for RNN inference tasks, existing approaches map tasks based on the length of an input sequence. We propose a technique combining the input sequence and DVFS that minimizes the energy consumption of edge devices while meeting the QoS requirements of RNN tasks.

5. Methodology

Motivated by the energy reduction observations with DVFS in Section 3, we propose an energy-aware runtime manager that takes advantage of DVFS techniques to consume less energy and meet the QoS requirements of RNN tasks. The workflow of the runtime manager is shown in Figure 3. In the Task Model phase, we first collect the running time at various operating frequencies for different input lengths. Then, we perform data training to build the regression model snd predict the running time in various operating conditions.

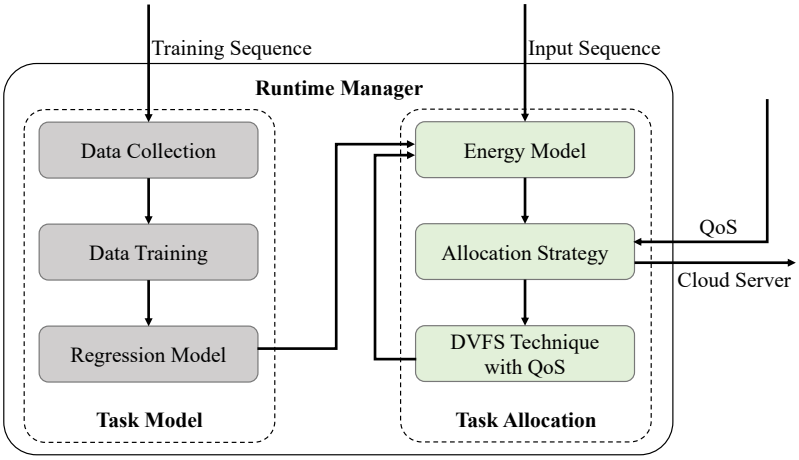

Figure 3. The workflow of the proposed energy-aware runtime manager.

In the Task Allocation phase, we establish the energy model using the predicted running time together with the power consumption. The allocation strategy is then applied to perform task allocations between edge and cloud devices. It takes into account the QoS requirement to make sure that tasks are properly assigned to cloud servers and edge devices. When operating on edge devices, DVFS techniques are applied to edge computing systems to minimize energy consumption while meeting QoS requirements.

5.1. Task Model

When dealing with different RNN tasks, we need to build a performance model to estimate the running time of RNN tasks. With the task model, one can predict the running time of each task, which helps determine task allocation strategies afterward.

To build a task model, we first collected data from the training sequences. We adopted n different lengths of input sequences, and each length of input sequences was repeated m times to measure the variations. Note that due to DVFS, there existed different CPU operating frequencies. Suppose that there were p frequency modes. Thus, we obtained $n \times m \times p$ sets of input sequences, and the jth time repetition of the ith length of input sequences for the kth frequency is denoted by $d_{i,j,k}$, which yields the running time $t_{i,j,k}$. Therefore, we obtained data tuples $< d_{i,j,k}, t_{i,j,k} >, 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq k \leq p$ in the data collection.

We used the collected data for training to obtain the task model. Since the running time of an RNN inference task is almost linearly proportional to that of the the input length, we built a linear performance model as follows:

$$T_k(d) = a_k \cdot d + b_k + c_k \tag{1}$$

where T_k is the predicted running time of the RNN task, d is the length of the input sequence, a_k is the proportional coefficient between the input length and running time, b_k is the bias coefficient, c_k is the overhead time to switch frequencies and subscript $_k$ denotes the kth operating frequency, as described in Section 6.2.

To obtain the coefficients of Equation (1), we employed the linear least squares regression method using data collection $< d_{i,j,k}, t_{i,j,k} >$. Then, we obtained

$$a_k = \frac{m \cdot n \cdot \Sigma_{i,j}(d_{i,j,k} \cdot t_{i,j,k}) - \Sigma_{i,j} d_{i,j,k} \cdot \Sigma_{i,j} t_{i,j,k}}{m \cdot n \cdot \Sigma_{i,j} d_{i,j,k}^2 - (\Sigma_{i,j} d_{i,j,k})^2} \tag{2}$$

$$b_k = \frac{\Sigma_{i,j} t_{i,j,k} - a_k \cdot \Sigma_{i,j} d_{i,j,k}}{m \cdot n} \tag{3}$$

With Equations (2) and (3), we obtained the regression model at the kth operating frequency, which was used for task allocation. We obtained a task model $\mathbb{T}$ of all frequencies as follows:

$$\mathbb{T} = \{T_k | 1 \leq k \leq p\} \tag{4}$$

5.2. Task Allocation

From the observations in Section 3, we found that the frequency affects the task energy. To reduce the energy consumption of edge cloud computing systems, we first built an energy model at the kth operating frequency:

$$E_k = T_k \cdot P_k \tag{5}$$

where e_k is the task energy, T_k is the running time of the inference task and P_k is the power at the kth operating frequency. Then, we obtained the energy model of all frequencies as follows:

$$\mathbb{E} = \{E_k | 1 \leq k \leq p\} \tag{6}$$

Then, we designed a task allocation strategy as follows:

$$S(d,q) = \begin{cases} \text{Edge} & \text{if } \exists T_k \in \mathbb{T}, T_k(d) \leq q \\ \text{Cloud} & \text{otherwise} \end{cases} \tag{7}$$

where S is the task allocation strategy function, d is the input data sequence and q is the QoS requirement (i.e., the maximum allowable task latency). If the QoS requirement can be met on edge devices, the task allocation strategy executes the task on the edge. Otherwise, it sends the task to the cloud server for execution.

When executing tasks on edge devices, we applied DVFS techniques to dynamically adjust the processor frequency to minimize energy consumption as follows:

$$\underset{E_k}{\operatorname{argmin}} f(E_k) = \{E_k | E_k \in \mathbb{E}, T_k \leq q\} \tag{8}$$

We selected the lowest operating frequency that met the QoS requirements to save energy. Note that when we varied the CPU frequencies from the ith operating frequency to the kth operating frequency, the overhead time in Equation (1) was computed as follows:

$$c_k = \begin{cases} 0 & \text{if } i = k \\ \text{C} & \text{otherwise} \end{cases} \tag{9}$$

where C is a constant, as described in Section 6.2.2.

After frequency variation using DVFS techniques, we updated the energy model using the latest data. We updated the coefficients in Equation (1) using online data such that the model always reflected the current characteristics of the edge device.

6. Experimental Results

6.1. Experimental Set-Up

To evaluate the effectiveness of our proposed method, without loss of generality, we used a server with Intel processors and a Raspberry Pi board device to emulate the cloud server and the edge device of an edge cloud computing system, respectively. Note that our method can be applied to a system with multiple cloud servers and edge devices, where each RNN task is allocated accordingly. In this paper, our experimental system set-up was as follows:

- An ARM Cortex-A72@1.5 GHz, 4 GB RAM to denote an edge computing device;
- A Dual Intel Xeon E5-2630@2.40 GHz, 128 GB RAM plus an NVIDIA 3080Ti GPU to represent a cloud computing server.

We used Python 3.6 to build an RNN model and implement the proposed task allocation strategy. In addition, we applied a multimeter to measure the power throughout the experiments.

Figure 4 displays the experimental set-up of the edge computing device, where the Raspberry Pi development board was used as the edge computing device, a multimeter was adopted for power measurement, and the screen was used to show the working status of the edge device, such as the task latency and system status.

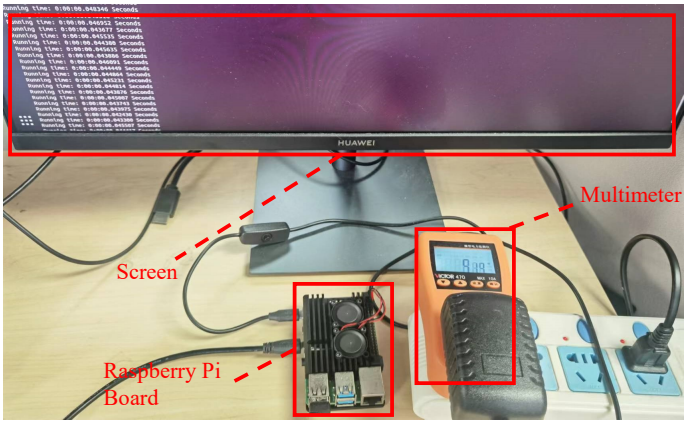

Figure 4. Experimental set-up of the edge computing device.

6.2. Task Model

6.2.1. Linear Coefficients

We first performed experiments to determine the task model's linear coefficients in Equation (1) (i.e., a_k and b_k). The input length and running time results at different operating frequencies are scattered in Figure 5, where each input length was measured five times to evaluate the time variation. From Figure 5, we applied the linear least squares regression method in Equations (2) and (3) to obtain the coefficients of a_k and b_k, respectively. The computed coefficients are displayed in Table 2.

Table 2. RNN task energy consumption using DVFS techniques.

k	Frequency (GHz)	a_k	b_k
1	0.6	0.4629	8.133
2	0.7	0.3966	8.2
3	0.8	0.3498	8.4133
4	0.9	0.3113	8.1733
5	1.0	0.2854	7.4533
6	1.1	0.2591	7.8
7	1.2	0.2434	6.5333
8	1.3	0.2297	5.12
9	1.4	0.2175	5.3467
10	1.5	0.1998	6.88

In Figure 5, the dotted linear line is displayed for each frequency. We observe that for a fixed frequency, the running time was linearly proportional to the input length. There were some points that exhibited offsets from the linear line, such as the case of an input length of 300 in Figure 5j, but those points are not far away from the linear line. There was a linear relationship between the input length and its running time, because for each input data point, all hidden states were calculated, which resulted in a fixed processing time. As the input length increased linearly, the running time accumulated linearly as well.

As the frequency increased from 0.6 GHz to 1.5 GHz, the running time decreased accordingly, and this makes sense since higher frequencies lead to higher computing capabilities. The line slope a_k decreased from 0.4629 to 0.1998 from 0.6 GHz to 1.5 GHz, and this indicates that when the input length increased by 100, the running time increment reduced from 46.29 ms to 19.98 ms, and the system performance was improved at higher frequencies.

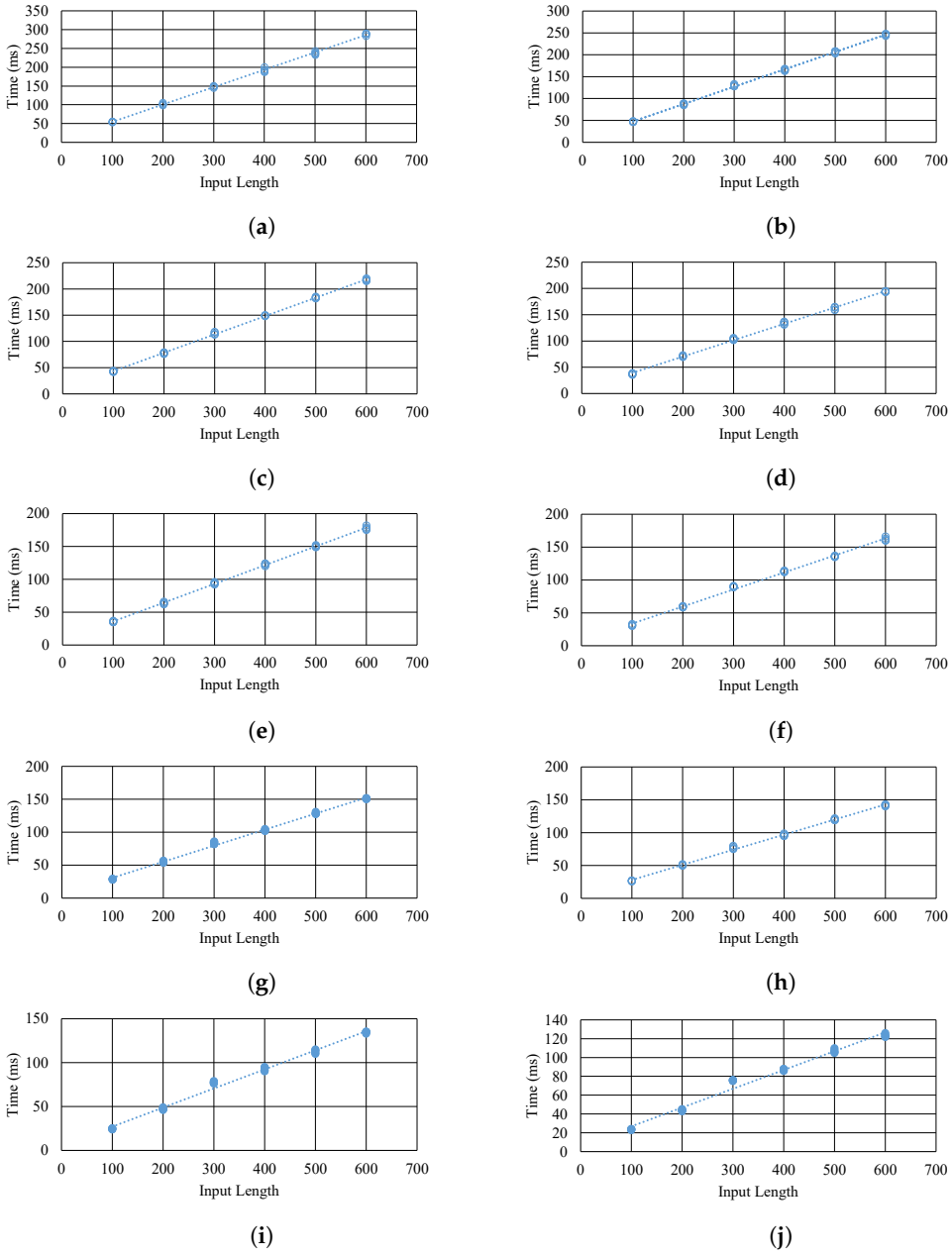

Figure 5. Input length and running time of RNN tasks at different operating frequencies. (**a**) Input length and running time at 0.6 GHz. (**b**) Input length and running time at 0.7 GHz. (**c**) Input length and running time at 0.8 GHz. (**d**) Input length and running time at 0.9 GHz. (**e**) Input length and running time at 1.0 GHz. (**f**) Input length and running time at 1.1 GHz. (**g**) Input length and running time at 1.2 GHz. (**h**) Input length and running time at 1.3 GHz. (**i**) Input length and running time at 1.4 GHz. (**j**) Input length and running time at 1.5 GHz.

6.2.2. Overhead Coefficient

In addition to linear coefficients, we also needed to determine the overhead coefficient c_k in Equation (1). We applied DVFS techniques and switched operating frequencies under different conditions, as shown in Table 3.

Table 3. The overhead of switching operating frequencies. The first column and the first row denote the target frequency and source frequency, respectively. The value of the table represents the switching time from the source frequency to the target frequency in ms.

Target \ Source	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**
0.6	0	9.43	8.59	8.03	7.69	7.37	7.14	7	6.82	6.67
0.7	10.01	0	8.12	7.76	7.3	6.91	6.63	6.35	6.2	6
0.8	10.03	8.83	0	7.56	7.14	6.92	6.54	6.26	6.08	5.81
0.9	10	8.69	8.01	0	7.03	6.76	6.43	6.19	5.99	5.79
1.0	9.89	8.65	8.06	7.39	0	6.77	6.44	6.15	5.92	5.69
1.1	9.92	8.58	7.94	7.4	6.99	0	6.35	6.05	5.82	5.61
1.2	9.79	8.51	7.95	7.43	7.01	6.73	0	6.07	5.85	5.74
1.3	9.93	8.68	8.05	7.46	7.02	6.67	6.5	0	5.85	6.02
1.4	9.77	8.57	7.87	7.31	6.82	6.46	6.16	5.95	0	5.62
1.5	9.87	8.62	7.94	7.4	6.88	6.54	6.23	5.99	5.71	0

The target frequency and source frequency are used as headers, and the switching from each source frequency to each target frequency is displayed. We can see that the switching overhead time was zero if the source and target frequency were the same because there was no need to change frequencies in this case. Otherwise, it took approximately 5–10 ms to finish the frequency switching. Therefore, we set up a lookup table for c_k such that for every instance of frequency switching, we could find the precise overhead value.

It is worth mentioning that on the one hand, if the source frequency is higher, it takes less time to finish the switching. For example, when switching to a 0.7 GHz target frequency from a 0.6 GHz source frequency, it took 10.01 ms, while the switching time decreased to 6 ms from a 1.5 GHz source frequency. There was a 4.01 ms switching time difference in this case.

On the other hand, if the target frequency is higher, it also spends less time switching. When either the source frequency or target frequency is higher, it is faster to finish the frequency switching because the CPU operates in higher frequencies, and it accomplishes tasks more quickly.

However, when the target frequency is high, the time difference is not as large as when the source frequency is high. For example, when the source frequency was 1.4 GHz, the switching time difference was 1.11 ms between the target frequencies of 0.6 GHz and 1.5 GHz. This was much smaller than the time difference value of 4.01 ms when the source frequency was high.

6.3. Energy Consumption

We tested different lengths of inputs and compared the edge energy consumption of our proposed method with that of the state-of-the-art approach by Pagliari et al. [15]. Similarly, the method of Pagliari [15] maps RNN tasks among cloud and edge devices such that the execution time and energy consumption requirements can be met. The method comparison is summarized in Table 4.

Table 4. Methodology comparison.

Method	Input Dependence	Cloud Edge Allocation	Energy Optimization	QoS Requirement	DVFS
Pagliari [15]	Yes	Yes	Yes	No	No
Our Method	Yes	Yes	Yes	Yes	Yes

We used $q = 200$ ms as the QoS latency requirement. The results are shown in Figure 6.

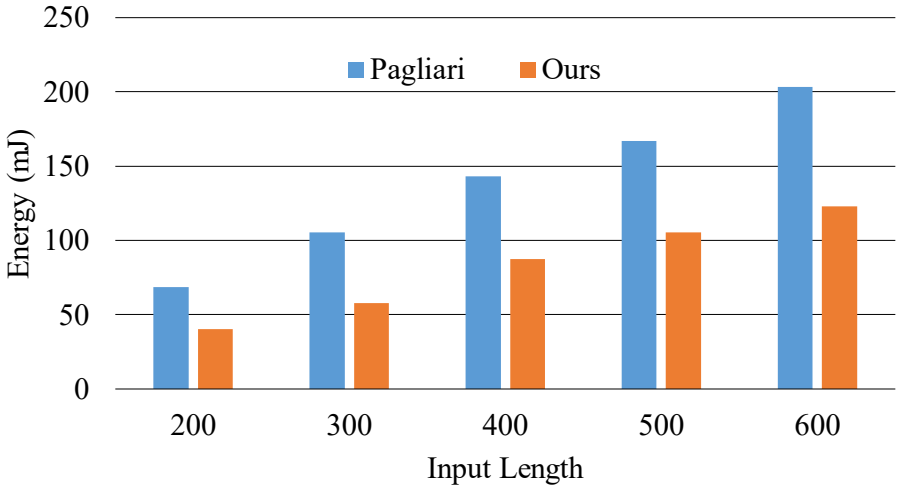

Figure 6. Energy consumption comparison on edge devices for separate input lengths.

In Figure 6, the x-axis represents separate input lengths from 200 to 600, and the y-axis denotes the energy consumption for each input length in mJ. We can see that compared with the method of Pagliari, our proposed method could effectively reduce the energy consumption of edge devices for each input length. It reduced the energy consumption from 36% to 45% for different input lengths and by up to 45% for the input length of 300.

Together with energy consumption, we plotted the task latency in Figure 7. We can see that for all input sequences, the task latency met the specified QoS requirement (i.e., $T_k \leq q$). When the input length was small (i.e., 200 and 300), the latency of such a task was very small, and the QoS could be readily met using the lowest frequency. However, as the input length increased, using the lowest frequency did not meet the QoS requirement. Therefore, the operating frequency varied accordingly to meet the QoS demands.

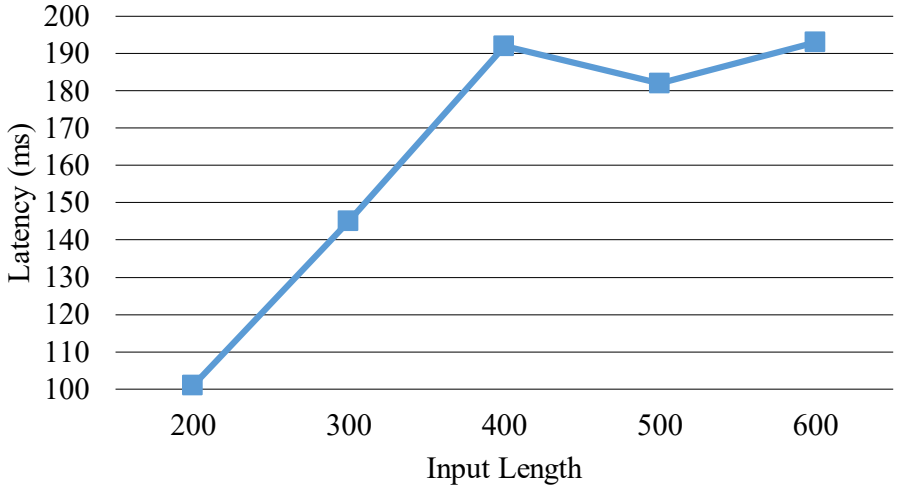

Figure 7. Task latency using the proposed method on edge devices.

Compared with the start-of-the-art method, our method was able to significantly reduce energy consumption, because our method slowed down the task execution while making sure that the QoS latency requirement was satisfied. This way, the processor runs tasks at low frequencies, and thus a large amount of energy is saved.

Apart from fixed input lengths, we use mixed different lengths for the input (from 200 to 600 with an interval of 100) and applied our method for energy optimization, as shown in Figure 8, where the x-axis represents the mixed task number that includes random input lengths and the y-axis denotes the energy consumption in mJ. We found that for tasks with

randomly mixed input lengths, our approach consumed 37% less energy on average (from 34% to 39%) compared with the method of Pagliari.

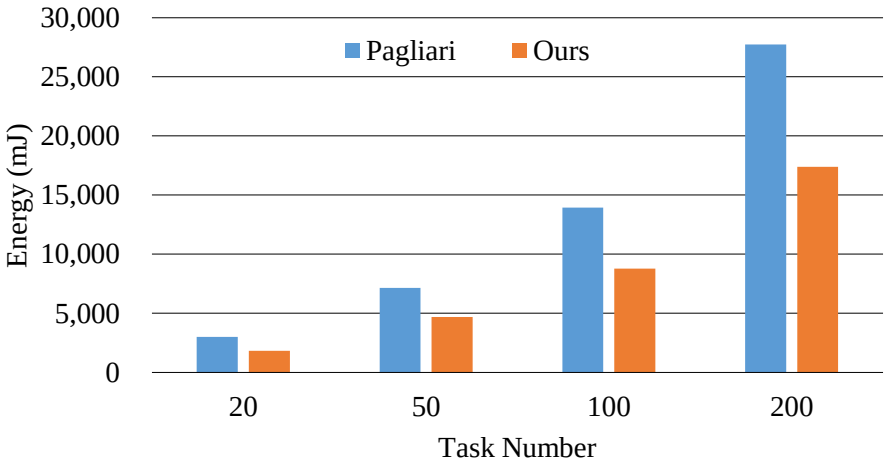

Figure 8. Energy consumption comparison on edge devices using mixed input lengths for the RNN model.

In addition to the normal RNN model, we also compared our approach with that of Pagliari using two specialized versions of RNN (i.e., long short-term memory (LSTM) and gated recurrent unit (GRU)). The results are displayed in Figures 9 and 10, where the x-axis represents the mixed task number that includes random input lengths (from 100 to 250 with an interval of 50) and the y-axis denotes the energy consumption in mJ. We found that for the LSTM model, our approach consumed 26% less energy on average—from 25% to 27%—compared with the method of Pagliari, and for the GRU model, our approach consumed 28% less energy on average—from 27% to 31%—compared with the method of Pagliari.

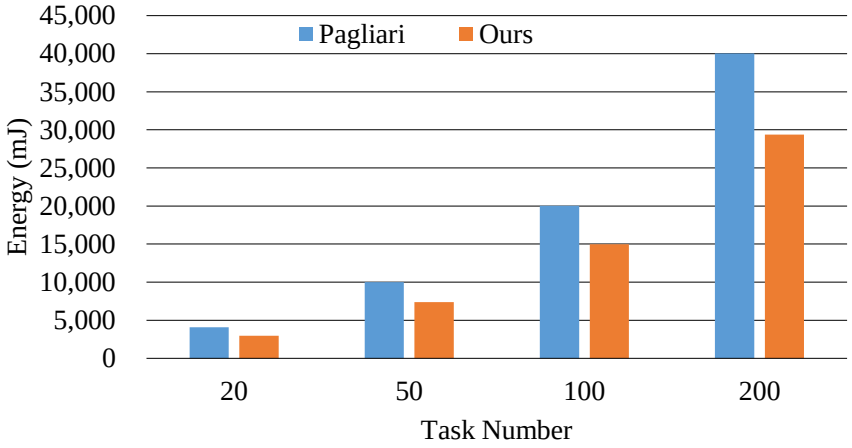

Figure 9. Energy consumption comparison for edge devices using mixed input lengths for the LSTM model.

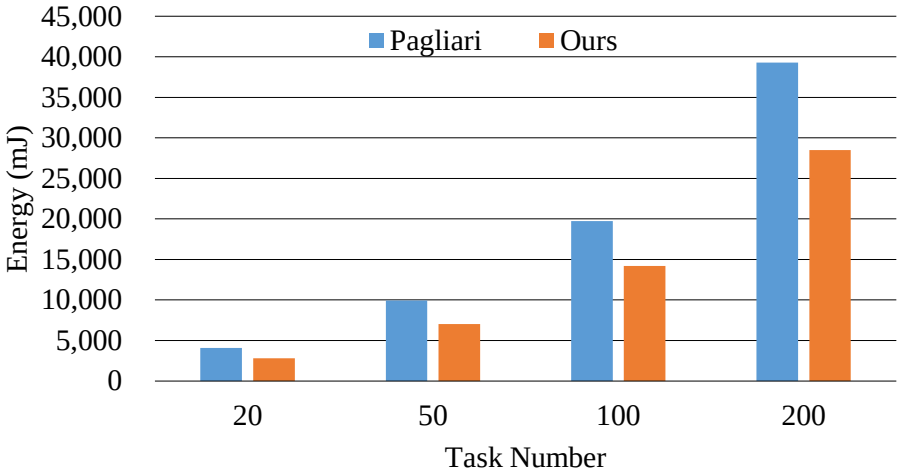

Figure 10. Energy consumption comparison for edge devices using mixed input lengths for the GRU model.

7. Conclusions

In this paper, we presented an energy-aware runtime manager that assigns RNN inference tasks to edge cloud systems. Our approach performs energy optimizations based on QoS requirements when processing tasks in edge systems. It decreases the operating frequency for tasks with short input lengths, and the experimental results reveal that it can reduce the energy consumption by up to 45% in edge devices compared with the state-of-the-art approach. Based on the successful experience of this paper, we conclude that we can improve the energy consumption of various projects by specifying the QoS requirement and slowing down the operating frequency using DVFS techniques. In the future, we plan to extend our manager to more neural network inference tasks.

Author Contributions: Conceptualization, C.C. and W.G.; methodology, C.C.; software, C.C., W.G., Z.W. (Zhuoyu Wu) and Y.Y.; validation, Z.W. (Zheng Wang); data curation, G.L.; writing—original draft preparation, C.C.; writing—review and editing, W.G. and Y.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Key-Area Research and Development Program of Guangdong Province (grant number 2019B010155003), National Natural Science and Foundation of China (NSFC 61902355), the Guangdong Basic and Applied Basic Research Foundation (grant number 2020B1515120044) and the joint fund of Science & Technology Department of Liaoning Province and State Key Laboratory of Robotics (2021-KF-22-12).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
2. Sahlol, A.T.; Abd Elaziz, M.; Tariq Jamal, A.; Damaševičius, R.; Farouk Hassan, O. A Novel Method for Detection of Tuberculosis in Chest Radiographs Using Artificial Ecosystem-Based Optimisation of Deep Neural Network Features. *Symmetry* **2020**, *12*, 1146. [CrossRef]
3. Maxwell, A.E.; Warner, T.A.; Guillén, L.A. Accuracy assessment in convolutional neural network-based deep learning remote sensing studies—part 1: Literature review. *Remote Sens.* **2021**, *13*, 2450. [CrossRef]
4. Dhaka, V.S.; Meena, S.V.; Rani, G.; Sinwar, D.; Ijaz, M.F.; Woźniak, M. A survey of deep convolutional neural networks applied for prediction of plant leaf diseases. *Sensors* **2021**, *21*, 4749. [CrossRef]

5. Ackerson, J.M.; Dave, R.; Seliya, N. Applications of recurrent neural network for biometric authentication & anomaly detection. *Information* **2021**, *12*, 272.
6. Lin, J.C.W.; Shao, Y.; Djenouri, Y.; Yun, U. ASRNN: A recurrent neural network with an attention model for sequence labeling. *Knowl. Based Syst.* **2021**, *212*, 106548. [CrossRef]
7. Anagnostis, A.; Benos, L.; Tsaopoulos, D.; Tagarakis, A.; Tsolakis, N.; Bochtis, D. Human activity recognition through recurrent neural networks for human–robot interaction in agriculture. *Appl. Sci.* **2021**, *11*, 2188. [CrossRef]
8. Rahman, M.M.; Watanobe, Y.; Nakamura, K. A Bidirectional LSTM Language Model for Code Evaluation and Repair. *Symmetry* **2021**, *13*, 247. [CrossRef]
9. Du, L.; Du, Y.; Li, Y.; Su, J.; Kuan, Y.C.; Liu, C.C.; Chang, M.C.F. A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2017**, *65*, 198–208. [CrossRef]
10. Chen, Y.; Xie, Y.; Song, L.; Chen, F.; Tang, T. A survey of accelerator architectures for deep neural networks. *Engineering* **2020**, *6*, 264–274. [CrossRef]
11. Yin, H.; Wang, Z.; Jha, N.K. A hierarchical inference model for Internet-of-Things. *IEEE Trans. Multi-Scale Comput. Syst.* **2018**, *4*, 260–271. [CrossRef]
12. Thomas, A.; Guo, Y.; Kim, Y.; Aksanli, B.; Kumar, A.; Rosing, T.S. Hierarchical and distributed machine learning inference beyond the edge. In Proceedings of the 2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC), Banff, AB, Canada, 9–11 May 2019; pp. 18–23.
13. Kang, Y.; Hauswald, J.; Gao, C.; Rovinski, A.; Mudge, T.; Mars, J.; Tang, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Comput. Archit. News* **2017**, *45*, 615–629. [CrossRef]
14. Eshratifar, A.E.; Esmaili, A.; Pedram, M. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In Proceedings of the 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Lausanne, Switzerland, 29–31 July 2019; pp. 1–6.
15. Pagliari, D.J.; Chiaro, R.; Chen, Y.; Vinco, S.; Macii, E.; Poncino, M. Input-dependent edge-cloud mapping of recurrent neural networks inference. In Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 20–24 July 2020; pp. 1–6.
16. Pagliari, D.J.; Chiaro, R.; Macii, E.; Poncino, M. CRIME: Input-Dependent Collaborative Inference for Recurrent Neural Networks. *IEEE Trans. Comput.* **2020**, *70*, 1626–1639.
17. Tann, H.; Hashemi, S.; Bahar, R.I.; Reda, S. Runtime configurable deep neural networks for energy-accuracy trade-off. In Proceedings of the 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Pittsburgh, PA, USA, 2–7 October 2016; pp. 1–10.
18. Jahier Pagliari, D.; Panini, F.; Macii, E.; Poncino, M. Dynamic Beam Width Tuning for Energy-Efficient Recurrent Neural Networks. In Proceedings of the GLSVLSI '19, 2019 on Great Lakes Symposium on VLSI, Tysons Corner, VA, USA, 9–11 May 2019; Association for Computing Machinery: New York, NY, USA, 2019; p. 69–74. [CrossRef]
19. 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]
20. Chen, J.; Ran, X. Deep Learning With Edge Computing: A Review. *Proc. IEEE* **2019**, *107*, 1655–1674. [CrossRef]
21. Liu, F.; Tang, G.; Li, Y.; Cai, Z.; Zhang, X.; Zhou, T. A Survey on Edge Computing Systems and Tools. *Proc. IEEE* **2019**, *107*, 1537–1562. [CrossRef]
22. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [CrossRef]
23. Nabavinejad, S.M.; Baharloo, M.; Chen, K.C.; Palesi, M.; Kogel, T.; Ebrahimi, M. An Overview of Efficient Interconnection Networks for Deep Neural Network Accelerators. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2020**, *10*, 268–282. [CrossRef]
24. Belabed, T.; Coutinho, M.G.F.; Fernandes, M.A.C.; Carlos, V.; Souani, C. Low Cost and Low Power Stacked Sparse Autoencoder Hardware Acceleration for Deep Learning Edge Computing Applications. In Proceedings of the 2020 5th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), Sousse, Tunisia, 2–5 September 2020; pp. 1–6. [CrossRef]
25. Xia, M.; Huang, Z.; Tian, L.; Wang, H.; Chang, V.; Zhu, Y.; Feng, S. SparkNoC: An energy-efficiency FPGA-based accelerator using optimized lightweight CNN for edge computing. *J. Syst. Archit.* **2021**, *115*, 101991. [CrossRef]
26. Liu, X.; Yang, J.; Zou, C.; Chen, Q.; Yan, X.; Chen, Y.; Cai, C. Collaborative Edge Computing With FPGA-Based CNN Accelerators for Energy-Efficient and Time-Aware Face Tracking System. *IEEE Trans. Comput. Soc. Syst.* **2022**, *9*, 252–266. [CrossRef]
27. Xu, C.; Jiang, S.; Luo, G.; Sun, G.; An, N.; Huang, G.; Liu, X. The Case for FPGA-Based Edge Computing. *IEEE Trans. Mob. Comput.* **2022**, *21*, 2610–2619. [CrossRef]
28. Zhou, J.; Dai, H.N.; Wang, H. Lightweight Convolution Neural Networks for Mobile Edge Computing in Transportation Cyber Physical Systems. *ACM Trans. Intell. Syst. Technol.* **2019**, *10*, 67. [CrossRef]
29. Kim, R.; Kim, G.; Kim, H.; Yoon, G.; Yoo, H. A Method for Optimizing Deep Learning Object Detection in Edge Computing. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 21–23 October 2020; pp. 1164–1167. [CrossRef]
30. Li, G.; Ma, X.; Wang, X.; Liu, L.; Xue, J.; Feng, X. Fusion-Catalyzed Pruning for Optimizing Deep Learning on Intelligent Edge Devices. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *39*, 3614–3626. [CrossRef]

31. Matsubara, Y.; Callegaro, D.; Baidya, S.; Levorato, M.; Singh, S. Head Network Distillation: Splitting Distilled Deep Neural Networks for Resource-Constrained Edge Computing Systems. *IEEE Access* **2020**, *8*, 212177–212193. [CrossRef]
32. Gordienko, Y.; Kochura, Y.; Taran, V.; Gordienko, N.; Rokovyi, O.; Alienin, O.; Stirenko, S. Chapter Nine—"Last mile" optimization of edge computing ecosystem with deep learning models and specialized tensor processing architectures. In *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*; Advances in Computers; Kim, S., Deka, G.C., Eds.; Elsevier: Amsterdam, The Netherlands, 2021; Volume 122, pp. 303–341. [CrossRef]
33. Li, G.; Ma, X.; Wang, X.; Yue, H.; Li, J.; Liu, L.; Feng, X.; Xue, J. Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning. *J. Syst. Archit.* **2022**, *124*, 102431. [CrossRef]
34. Zhou, X.; Canady, R.; Bao, S.; Gokhale, A. Cost-effective hardware accelerator recommendation for edge computing. In Proceedings of the 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20), Virtual Event, 25–26 June 2020.
35. 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]
36. Feng, J.; Richard Yu, F.; Pei, Q.; Chu, X.; Du, J.; Zhu, L. Cooperative Computation Offloading and Resource Allocation for Blockchain-Enabled Mobile-Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Internet Things J.* **2020**, *7*, 6214–6228. [CrossRef]
37. Liu, Y.; Xie, S.; Zhang, Y. Cooperative Offloading and Resource Management for UAV-Enabled Mobile Edge Computing in Power IoT System. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12229–12239. [CrossRef]
38. Kennedy, J.; Varghese, B.; Reaño, C. AVEC: Accelerator Virtualization in Cloud-Edge Computing for Deep Learning Libraries. In Proceedings of the 2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC), Melbourne, VIC, Australia, 10–13 May 2021; pp. 37–44. [CrossRef]
39. Kuang, Z.; Ma, Z.; Li, Z.; Deng, X. Cooperative computation offloading and resource allocation for delay minimization in mobile edge computing. *J. Syst. Archit.* **2021**, *118*, 102167. [CrossRef]