energies

MDPI

Article

An Intelligent Task Scheduling Mechanism for Autonomous Vehicles via Deep Learning

Gomatheeshwari Balasekaran [1,*], Selvakumar Jayakumar [1] and Rocío Pérez de Prado [2,*]

[1] Department of Electronics and Communication Engineering, SRM Institute of Science and Technology, Tamil Nadu 600026, India; selvakuj@srmist.edu.in
[2] Telecommunication Engineering Department, University of Jaén, 23700 Jaén, Spain
* Correspondence: gomatheeshwari_balasekaran@srmuniv.edu.in (G.B.); rperez@ujaen.es (R.P.d.P.)

Abstract: With the rapid development of the Internet of Things (IoT) and artificial intelligence, autonomous vehicles have received much attention in recent years. Safe driving is one of the essential concerns of self-driving cars. The main problem in providing better safe driving requires an efficient inference system for real-time task management and autonomous control. Due to limited battery life and computing power, reducing execution time and resource consumption can be a daunting process. This paper addressed these challenges and developed an intelligent task management system for IoT-based autonomous vehicles. For each task processing, a supervised resource predictor is invoked for optimal hardware cluster selection. Tasks are executed based on the earliest hyper period first (EHF) scheduler to achieve optimal task error rate and schedule length performance. The single-layer feedforward neural network (SLFN) and lightweight learning approaches are designed to distribute each task to the appropriate processor based on their emergency and CPU utilization. We developed this intelligent task management module in python and experimentally tested it on multicore SoCs (Odroid Xu4 and NVIDIA Jetson embedded platforms).Connected Autonomous Vehicles (CAV) and Internet of Medical Things (IoMT) benchmarks are used for training and testing purposes. The proposed modules are validated by observing the task miss rate, resource utilization, and energy consumption metrics compared with state-of-art heuristics. SLFN-EHF task scheduler achieved better results in an average of 98% accuracy, and in an average of 20–27% reduced in execution time and 32–45% in task miss rate metric than conventional methods.

Keywords: autonomous vehicles; deep learning; heterogeneous multicore; IoT; task mapping; scheduling; energy consumption

Citation: Balasekaran, G.; Jayakumar, S.; Pérez de Prado, R. An Intelligent Task Scheduling Mechanism for Autonomous Vehicles via Deep Learning. *Energies* **2021**, *14*, 1788. https://doi.org/10.3390/en14061788

Academic Editor: Hugo Morais

Received: 9 February 2021
Accepted: 15 March 2021
Published: 23 March 2021

1. Introduction

IoT is becoming ubiquitous that connects a million devices such as sensors, actuators, gateways, and hubs over the Internet [1]. Many embedded applications are connected to the physical world via the Internet, e.g., autonomous vehicles, avionics, home automation systems, health monitoring systems, smart cities, etc. This will lead to new developments in combining the embedded systems with IoT called real-time embedded IoT systems [2]. Of these applications, self-driving cars have received much attention in the recent past due to connectivity and artificial intelligence. Self-driving cars are the primary example for the real-time embedded IoT system, which includes multiple hardware sensors, actuators are connected through the IoT for processing each function. The self-driving car market is projected to grow by more than £ 7 trillion per year by 2050 [3].

However, there are many challenges in the transportation industry to make self-driving cars as public transportation to create a safe environment. Many autonomous vehicle manufacturers such as Uber, Waymo, and Google manufacturing are struggling to implement smart cars in the market due to the high cost of the computational systems and the lag of the software task management system [4]. According to a recent survey, around

56% of accident rates are increased by autonomous vehicles due to uncontrollable traffic, incorrect location, and mapping techniques [5]. Many recent articles are focused only on traffic prediction optimizations via machine learning algorithms. There are many other significant problems that need to be addressed in autonomous vehicle design in hardware and software optimization. The core components of autonomous vehicles are sensors, actuators, and processors to handle multiple functions based on the external environment conditions. This requires an efficient hardware platform and software kernel to achieve high performance and secure driving.

The real-time embedded IoT systems require an intelligent task management system to control and monitor the real-time processes to withstand uncertain conditions. Generally, real-time embedded systems are categorized as hard real-time systems, firm real-time systems, and soft real-time systems based on their deadline constraints [6]. A system with hard deadlines is considered high-criticality systems such as avionics, mission devices, autonomous vehicles, etc. The system with soft deadlines is low-criticality systems called multimedia, face recognition, etc. Deadline constraint is essential in real-time embedded systems. For example, unmanned aerial vehicles, health monitoring systems mandatorily need to follow the strict timing constraints and safety requirements to avoid catastrophic situations.

On the other hand, in an autonomous vehicle, many tasks have strict timing constraints in the form of hard deadlines, and if the deadline of any single task is missed, it will cause severe accidents. To minimize these accident rates, autonomous vehicles require resourceful hardware and software to handle multiple tasks concurrently and efficiently. The revolution on the hardware technology raised from uni-processor to multiprocessor system-on-chip (MPSoC). MPSoC is categorized as a homogeneous and heterogeneous multiprocessor system-on-chip. Homogeneous multiprocessor chip includes a similar category of CPUs on the same chip, and heterogeneous MPSoC (HMPSoC) comprises of distinct category such as CPUs, GPUs, AI processors on the same system-on-chip [7]. These asymmetric multiprocessors are energy-efficient and perform multi-tasking with low power consumption. In recent days, many real-time embedded IoT applications adopt this energy-efficient MPSoC as a hardware configuration to improve the overall performance with low-power consumption. Samsung Exynos 5422, Qualcomm snapdragon hardware kits are examples of MPSoC [8]. Despite its benefits, designing intelligent software to handle task distribution, resource utilization, and task scheduling are still NP-hard problems in heterogeneous MPSoCs. Michael G. Bechte et al. developed a low-cost autonomous vehicle using homogeneous quad-core MPSoC (Raspberry pi-3) with a deep neural network prediction algorithm for car steering detection via camera inputs [9]. Lu Cheng et al. exploited the heterogeneous MPSoC for autonomous driving and developed an event-based task scheduler to handle the driving tasks [10].

Similarly, Xiebang Wang et al. adopted the HMPSoC computing system with openCL software kernels for Advance Driver-Assistance System (ADAS) operations [11]. Tosiron et al. studied the role of multicore processor optimization in IoT systems. A review article comprises multicore processor enhancements, and its significant role in processing the IoT tasks with logical and temporal correctness is discussed in detail. Microprocessor configuration, which varies from single-core to multicore hardware units, is deliberated for IoT systems. In-depth learning modules are adapted widely in IoT applications because of their efficient data processing and intelligence in the decision-making process. The output accuracy is enhanced that compared to conventional algorithms for the specified inputs [12].

1.1. Conventional Scheduling Algorithms on Real-Time Embedded IoT Systems

Liu and Layland et al. introduced the Rate monotonic and earliest deadline first algorithms for real-time embedded systems [13]. These traditional algorithms are recently adopted for realtime IoT systems for accessing each job. Hongjun Dai et al. proposed two

scheduling approaches for autonomous vehicle driving services [14]. The tasks are modeled as periodic task sets with computational constraints in terms of execution cost, communication cost, waiting time, deadline constraints in a matrix format. These tasks are scheduled based on the two heuristics called direct execution strategy and task replacement model, allocating the tasks on mobile edge servers for execution. Hussein et al. developed a dynamic voltage frequency scaling-based task scheduling algorithm called Energy Saving-Dynamic Voltage Frequency Scaling(ES-DVFS) for autonomous systems. The proposed algorithm modeled the driving tasks as aperiodic tasks with soft deadlines [15]. Xiaoqiang et al. reshaped the Petri nets with Ant Colony Optimization (ACO) swarm-based optimizer for enhancing the performance of IoT networks. The job scheduling problem is analyzed with a schedule timed transition Petri net (TTPN) and search space optimization algorithm to enhance the makespan (overall schedule length) [16]. The IoT tasks are modeled as directed acyclic graphs (DAG) with precedence dependency constraints. The genetic algorithm is utilized to optimize the search space during the allocation and execution of real-time IoT tasks on virtual machines. Makespan is minimized, and the infrastructure's performance as a service (IaaS) model is improved using Genetic Algorithm (GA) scheduling modules by Xiaojin Ma et al. [17]. Bini et al. proposed the branch-bound search based task scheduler with fixed priorities in order to maximize the system performance without violation of deadlines [18]. Likewise, Hyun-Jun Cha et al. developed a deadline monotonic task scheduler for optimality of deadline and period constraints in automotive systems [19].

Research Gap on Existing Models

These methods' limitations are less efficient for multiprocessor system-on-chip and not suitable for high-complexity tasks such as object detection, path planning, localization, lane detection, etc. In recent days, hardware technology has become more efficient in combining multiple CPUs, GPUs, AI, and IoT processors in the same chip to handle multiple tasks simultaneously. The existing software kernel is less efficient in handling these heterogeneous resources that require the intelligent task management system to improve resource utilization with high-performance and low-energy consumption. Moreover, recently traffic prediction problems are focused much on autonomous vehicle systems. The proposed intellectual task management system resolved the hardware/software co-design challenges and provided the optimal solutions in resource utilization, task miss rate, and execution time.

1.2. Challenges on Conventional Task Scheduling Policy for Self-Driving Cars

The conventional algorithms are most suitable for uni-processor hardware computing systems that are not readily adaptable for multiprocessor systems. Moreover, recently autonomous vehicles are designed with HMPSoCs to provide efficient performance at runtime.

- Task miss ratio is the significant constraint to be minimized in the autonomous vehicles to minimize the accident rates, which is not considered in any previous scheduling methods.
- Hardware resources in autonomous vehicles to be efficiently utilized to maintain the lifetime of the devices, such as sensors, actuators, and processing units.
- Time complexity (overall execution time) is another essential metric for optimizing autonomous vehicles due to the battery-based device.
- A trade-off between task miss ratio and overall execution time in task scheduling is an NP-hard problem.

These are hardware/software co-design challenges that need to be solved in autonomous vehicles to provide safe driving and high-performance.

1.3. Significant Contributions

The proposed intellectual task management system developed on addressing the challenges mentioned above and evaluated on two different HMPSoC hardware configurations. A supervised resource predictor with a scheduling algorithm is designed and validated with standard autonomous vehicle benchmarks.

Resource Predictor using Supervised Learning Model: Smart vehicles comprise mixed-critical workloads in terms of sensors, image processing, high computational workloads to complete a single task. The challenge that arises in real-time is (i) Identification of task criticality and selection of best resources for execution is still lagging, which degrades the performance at runtime. In the proposed framework, we addressed this issue by developing an intellectual learning neural network to predict the best hardware cluster for each autonomous vehicle workload on the HMPSoC platform. Single Layer Feed-forward Network (SLFN) [20] and lightweight deep neural network (LW-DNN) are low-complex predictors designed and compared with traditional learning algorithms such as SVM, RF, decision tree.

Task Scheduling Mechanism: Another challenge is arisen on ordering the tasks for execution. Many existing scheduling approaches are more time-complex. We developed a low-complex "Earliest hyper period first (EHF)" algorithm and executed the tasks on the base cluster to resolve this. Consequently, tasks are executed based on their emergency and earliest hyper period to achieve better task miss rate (TMR) and performance.

Implementation Module: A heterogeneous multicore SoC module "Odroid Xu4, and NVIDIA jetson" platforms are used to implement the scheduling algorithm and provide real-time solutions for autonomous vehicle services. CAV and IoMT benchmarks are used for the evaluation of the proposed task management system. The hardware module is connected with a cloud server to maintain the regular monitoring and storage of obtained results for future use. SLEN-EHF task scheduler achieved better results in an average of 98% in prediction accuracy. An average of 2027% was reduced in execution time and 32–45% in task miss rate metric than conventional methods.

1.4. Outline

The remaining sections are illustrated as follows: Section 2 detailed relevant works performed on Scheduling algorithms for IoT-based AV applications. In Section 3, the Preliminary system and application modules are discussed with the proposed architecture. SLFN mathematical model and resource predictor with internal working structure is illustrated in Section 4. EHF algorithm with workflow diagram is explained in Section 5. In Section 6, an Experimental setup with validated results is described with performance charts—the conclusion with the proposed limitations as detailed in Section 7.

2. Related Works

Autonomous vehicle research emerges from the past decade. It has recently become ubiquitous due to IoT, artificial intelligence, and high computational platforms' availability. Meanwhile, they boost the research topic on hardware architecture and intelligent task scheduling mechanisms to satisfy the timing constraints and safety requirements with low power consumption. Chien-Ying et al. (2018) researched real-time IoT systems regarding architecture, security, and processing issues. The autonomous vehicle driving workloads are framed as real-time periodic and aperiodic jobs with various timing constraint are modeled, and different case studies of Real-Time(RT)–IoT is deliberated in detail. RT-scheduling for IoT devices with its open problems and challenge is explained [21]. Nasri et al. (2015) have developed a model to capture the harmonic coupling between the periods. After calculating the harmonic subinterval in the range of possible periods for a given set of tasks that can easily calculate and assign a harmonic period. To further reduce the hyper period's value for the harmonic period's resulting assignment [22]. Liu et al. (2017)

developed an autonomous driving system architecture that can run tasks on a heterogeneous Advanced Risc Machine (ARM) mobile system-on-chip [23]. They partition autonomous vehicle workloads into three categories in terms of sensing, perception, and decision-making.

Smruti et al. (2019) focused on IoT networks that assess various layers in terms of sensors, actuators in the application layer connected with hardware through efficient scheduler modules. Global and local preferable algorithms were designed for IoT devices. The proposed technique utilizes the multicore processors with a DVFS power optimizer to adjust the computational elements' voltage and the frequency at runtime [24]. Resource aware scheduler is framed for RT-IoT devices by Shabir et al. (2019) [25]. RT-Tasks are modeled in a periodic format in terms of execution time, period, and deadline constraints. The proposed module initially identified the hyper period for each periodic task set and executed by conventional schedulers such as Earlierst Deadline First(EDF), Rate Monotonic (RM). Resources such as processor energy consumption are considered for optimization and achieved by minimizing each periodic task set's hyper period during runtime. Sehrish Malik et al. (2019) developed an emergency first algorithm for real-time embedded workloads with hard and soft deadline constraints. Real-time IoT tasks are modeled with four diverse tags: emergency, regular, and most emergency periodic and aperiodic. Based on the tag attached to each task that multicore processors execute in [26]. Sensor tasks are modeled as periodic with a soft deadline, and actuator tasks are modeled as most emergency tasks prioritized first. The proposed model also includes the Artificial Neural Network (ANN)-based prediction module with urgency and failure measure metrics that helps in allocation and execution. Farzad Samie et al. (2019) published a survey on machine learning models for embedded IoT systems. The significant role of machine learning and deep learning algorithms in IoT applications in various categories is discussed generously [27].

Jie Tang et al. (2020) proposed the low-cost real-time autonomous vehicle (Dragonfly Pod) with three modules, such as LoPECS (Low-Power Edge Computing System) and CNN for real-time object detection and speech recognition module with the heterogeneous multicore platform at an affordable price of $10,000 [28]. Recently many autonomous vehicles are connected with mobile edge computing servers, and mobile devices are used to monitor and control the services in real-time. Xu et al. (2020) [29] developed a scheme based on a small adjustment of these problems' resulting periods. The adjusted periods of the task may not be entirely harmonious. However, they are closely linked, facilitating the entire schedule's calculation up to the execution time for the periodic task's final planning.

3. Groundworks

3.1. Autonomous Vehicle Service Module

Primary services of autonomous vehicles are five modules: (i) Sensor and Actuator module, (ii) Perception, (iii) Localization and Mapping, (iv) Path Planning, and (V) Control Module. Each module consists of monitoring (periodic), controller (event-driven), and complex tasks (large computational tasks). Figure 1 illustrates the basic structure of the proposed task management system.

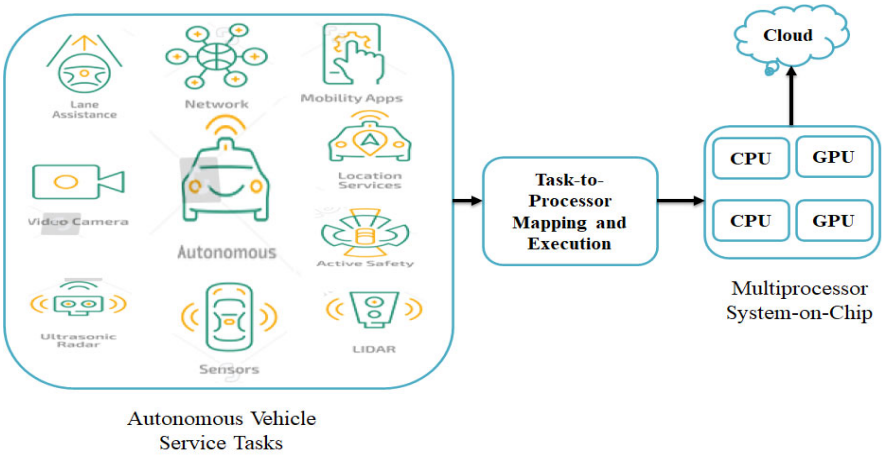


Figure 1. The building block of the proposed Intelligent Mapping System.

3.1.1. Application Layer Structure

In this work, the proposed framework is designed for all three category task sets. We assumed these tasks are represented as tree structures with nodes and edges [30]. A periodic (Sensor) task is represented as a single tree with its precedence tasks as child threads. Likewise, aperiodic and complex tasks are modeled as trees. Figure 2 (left) represents the sample periodic task graph. Figure 2 (right) denotes the aperiodic task graph, and the complex graph includes both periodic and aperiodic tasks, which are modeled as the same.

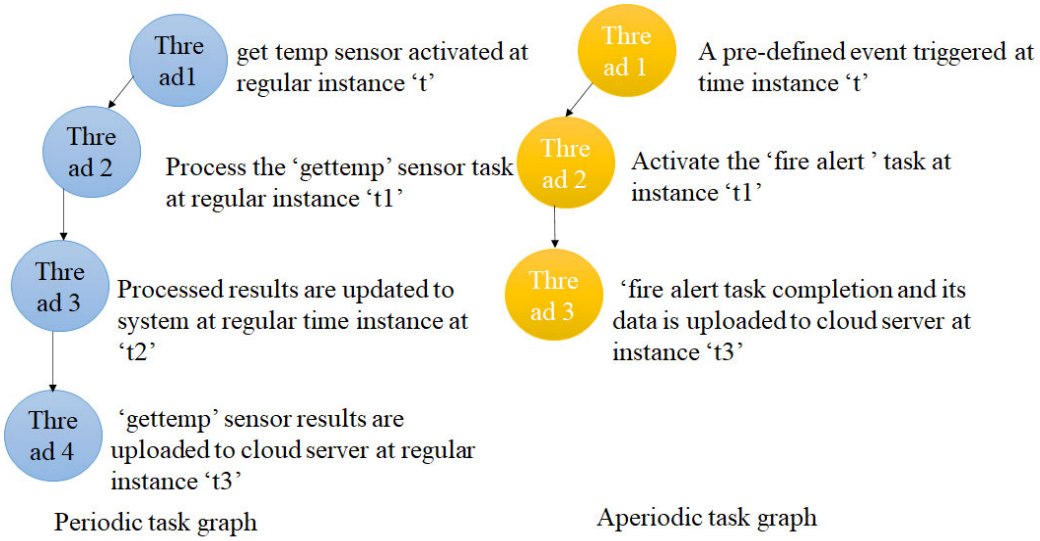


Figure 2. (**left**) Periodic workload "getemp" sensor processing task is represented with subtasks, (**right**). Aperiodic workload "fire alarm" task with its subtasks.

Periodic Task Graph: Each periodic task set includes these four subtasks such as sensing, processing, monitoring, and uploading at regular time instance '*t*'.

Definition 1. *Each node in the periodic tree is denoted as* $Per_i = \{AT_i, ET_i, D_i, t_i, Token_i\}$ *that need to be executed at regular time instance* $'t_i'$ *(i.e., period). Where* $'Per_i'$ *denotes the periodic task graph, ,'AT' is the arrival time of a periodic task, ET is the execution time of a periodic task, D is the deadline of a periodic task, 't' is the period of a periodic task, Token is the priority bit associated to a periodic task were Token = 0 it is a regular task if Token = 1 emergency task.*

Aperiodic Task Graph: Aperiodic tasks are event-based tasks that are highly prioritized in IoT-based Autonomous vehicle applications. Generally, aperiodic tasks are not periodic (i.e., not steady job). Here, we represent the time instance as execution instance time of event triggered.

Definition 2. *Each node in the Aperiodic tree is denoted as* $Aper_i = \{e_i, AT_i, ET_i, p_i, Di, t_i, E'_i\}$ *that needs to be executed* $'t_i'$ *when triggered. Where* $'APer_i'$ *denotes the aperiodic task graph* $'e_i'$ *is the event triggered for a particular task,* $'AT_i'$ *is the arrival time of a periodic task,* $'ET_i'$ *is the execution time of a periodic task,* $'D_i'$ *is the deadline of a periodic task,* $'t_i'$ *is the event trigger instance in nanoseconds,* E' *is the priority bit associated to an aperiodic task were* $E' = 1$ *always because aperiodic is a control task that to be executed first in real-time systems to avoid the catastrophic situation.*

Complex Task Graph: Complex tasks are Localization, Path planning, obstacle detection, etc.

Definition 3. *WTi denotes a large computational complex task* $\boldsymbol{Compl_i = \{AT_i, ET_i, D_i, t_i, E'_i\}}$ *It should be executed repeatedly by CPUs or GPUs to control the entire system and safe driving.*

For example, Object tracking, localization tasks are emergency tasks that need to be monitored and controlled continuously.

3.1.2. Constraints involved in the Application Layer

Arrival time—Tasks arrived at the processor for execution are denoted by '*AT*', which varies dynamically for each task.

Deadline—Worst case completion time of each task is denoted by '*D*', which has been pre-defined before execution.

Response time—The time duration between released time and finish time is the total response time of task '*i*' at processor '*j*'.

Execution time—Worst case processing time of task '*i*' is denoted by '*ET*' during processor execution.

RT: Task-to-processor Mapping: Each node (i.e., task) is allocated to a particular processor for execution is a mapping problem; designing an optimal mapping for an RT–IOT system is critical due to limited resources in terms of memory, processors, and network availability. In this paper, a dynamic core mapping algorithm based on the SLFN predictor is developed for task allocation on multicore processors in real-time.

RT-Task scheduling: A feasible or optimal sequence order of task execution pre-defined in the RT–IoT system. In this paper, the earliest hyper period first (EHF) scheduling sequence is designed and experimentally validated with real-time periodic and aperiodic task sets.

3.2. Hardware Layer Multicore Processor System-on-Chip

In this paper, we assumed that CPU's are represented as Little cores = {L_1, L_2,…,L_k}, GPU's as big cores = {B_1, B_2,…,B_g}, with respective operating frequencies and voltage levels. Two different heterogeneous multicores SoC are targeted as hardware clusters named as (Hardware configuration:1 ("Odroid Xu4 SoC: 4L1b, 3L1b, 0L1b") and (Hardware configuration: 2 (Nvidia Jetson SoC- "8L0b, 0L384b, and 8L384b") which is used for the implementation of the proposed intelligent task management framework. The targeted heterogeneous multicore architecture comprises 16 CPU cores and 384 GPU cores. The number of big and little cores in each cluster differs based on the core combinations. These are the clusters we used for dataset preparation Clusters (CL): where "L" denotes the CPU

core and "b" represents the GPU core. The operating voltage for the entire system is denoted as, $Vl = \{Vl_1, Vl_2, \ldots Vl_V\}, v \in k, g$ and the frequencies are denoted as $fl = \{fl_1, fl_2, \ldots, fl_f\}, f \in k, g$ respectively.

4. Intelligent Task Management for IoT Based AV

In this proposed framework, tasks are allocated on an optimal hardware cluster that is predicted by learning networks, and it has been scheduled based on the EHF algorithm. The SLFN and LW-DNN supervised learning predictors select the best hardware cluster based on the task and execution constraints listed in Table 1. The overview structure of the proposed model is illustrated in Figure 3.

Table 1. Summarizing the Existing models with key findings.

Existing Models	Inference	System Structure	Application Model	Resource Utilization	Execution Time Optimization	Task Miss Ratio Optimization
[21]	Formation of RT–IoT with the embedded system is deliberated with its current challenges in hardware/software, task scheduling, and security issues.	Multicore platforms, IoT networks are studied.	IoT tasks with mixed-criticalities	-	-	-
[22]	Fixed-priority and Rate monotonic scheduling algorithms designed with optimized period bounds	Hardware is not focused on this work.	IoT tasks with mixed-criticalities (control applications are modeled as periodic task sets)	Resource utilization is improved with harmonic period selections.	-	-
[23]	Autonomous driving workloads are experimentally tested on various HMPSoC platforms.	HMPSoCs are utilized for evaluation.	Autonomous vehicle workloads are considered.	Resources are utilized up to 100%	Execution time is improved based on the appropriate selection of processors.	-
[24]	Energy consumption is targeted in this work. DVFS based heuristics are developed for the RT-IoT system	HMPSoC	RT–IoT workloads are considered for evaluation.	-	25–30% of execution time is minimized based on the DVFS adjustment	Task miss rates are reduced with a penalty
[25]	A resource-based task scheduler is designed to optimize the hyper-period metric.	Raspberry pi-3 HMPSoC used for evaluation	Sensor tasks are modeled as Periodic task sets with a mixed-period range.	-	20% improved	-
[26]	Fair emergency first scheduling algorithms is designed with ANN resource predictor.	Intel PC has been used for evaluation.	Sensor and actuator tasks are modeled with mixed-criticalities.	-	30% improved	35% task miss ratio is improved
[27]	Various machine learning algorithms are discussed	MPSoC platform used for analysis	Embedded workloads are used	-	-	-
[28]	Dragonfly's low-cost autonomous vehicle is designed and executed with an inter-core scheduling algorithm.	HMPSoC	Autonomous vehicle workloads are utilized.	-	10–27% minimized the overall execution time	-
[29]	Task periods are minimized using the DVFS technique.	MPSoC	Embedded workloads are considered.	-	10% minimized	-
Proposed Framework	Two heuristics are modeled, such as EHF, SLFN-EHF with intelligent resource predictors to solve hardware/software challenges of autonomous vehicles. The proposed modules are evaluated on two HMPSoC testbeds.	HMPSoC	The autonomous vehicle, IoT workloads as a Tree structure	Resources are clustered and utilized fully	35% compared with traditional methods CFS and fairness algorithms	65% improvement is achieved

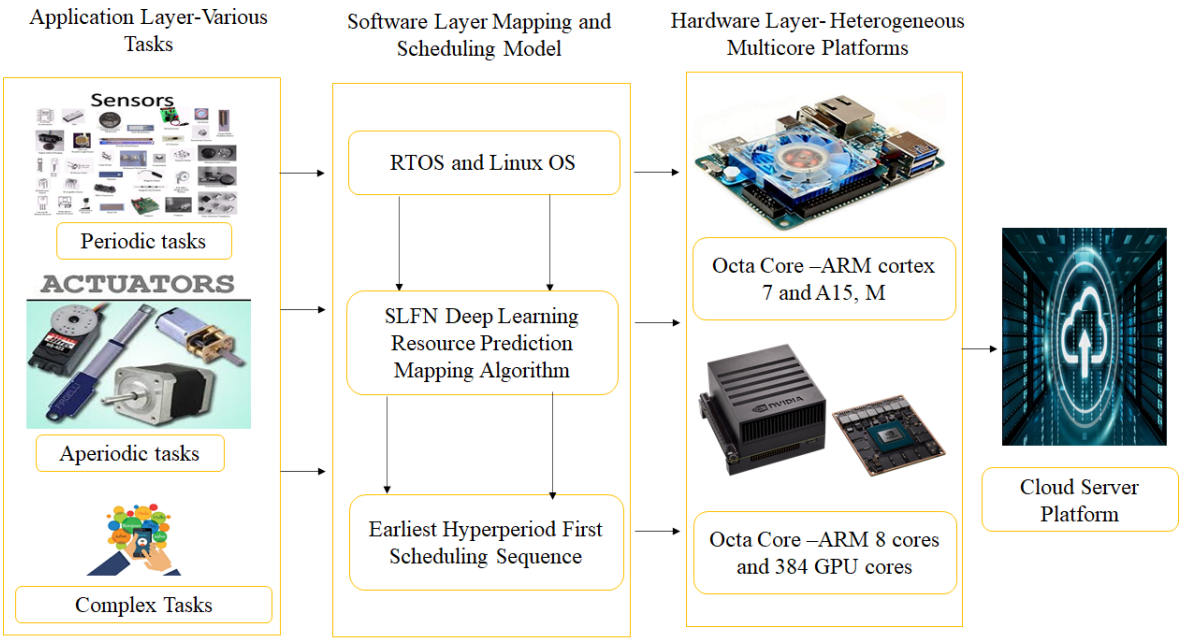


Figure 3. Overview structure of Proposed Intelligent System.

4.1. Resource Prediction Model

Researchers recently developed deep learning-based optimization algorithms to enhance application mapping performance on multicore architectures [31]. The limitation of traditional scheduling techniques was targeted only a single metric like energy or power or time constraints, and task miss ration is not observed in the existing techniques which is most significant in RT-IoT systems. In this paper, we developed two different supervised learning algorithms named LW_DNN and SLFN. The DNN network is adopted from the previous work, which is utilized for comparison.

Feature Extraction

We executed IoT based Autonomous vehicle tasks on two different hardware configuration clusters named as (Hardware config:1 (“Odroid Xu4 SoC: 4L1b, 3L1b, 0L1b”) and (Hardware config: 2 (Nvidia Jetson SoC- “8L0b, 0L384b, and 8L384b”) with a mixed category of CPUs and GPUs. Table 2. describes the significant features utilized for selecting the appropriate cores for execution.

IoT and autonomous vehicle benchmark programs are repeatedly executed in each cluster and examined the execution characteristics. Workload characteristics are extracted in terms of execution features and optimized in this phase.

As mentioned above, each workload is executed on every hardware cluster and observed PC values, which in total 10 K data. The primary purpose of the feature preprocessing algorithm is to minimize the time complexity and training complexity. We developed a feature optimization algorithm similar to the previous work [31]. Table 1 shows the optimized features used for core selection. These are the significant features utilized as input for resource predictors on selecting optimal core at runtime. The application and execution features are automatically recognized and feedforwarded into the modeling process as input vector values.

Table 2. Optimized Feature of Workloads [31].

Data Type	Workload Features
performance	Instruction per count (IPC)
Time	Deadline, period
Emergency value	Token ID
Runtime utilization	No of active CPUs and GPUs
Load	No of assigned processes count on each core
Threshold limit	CPUs and GPUs capacity (pre-defined)
memory	L1-D cache, L1-I cache, L2-cache access and cache miss ratio
Instructions (%)	Arithmetic integer/float/add/mul
branch	Branch misprediction data's
TLB	dTLB misses, iTLB misses
power	average power consumption in watts for a time slice
Time	The overall execution time of the workload
Total tasks	Total no.of tasks on each queue
Queue id	Each task queue id

4.2. Modeling of SLFN Predictor

Single Hidden Layer Feedforward Neural Network Prediction Model is developed to predict the best hardware cluster for each task execution to optimize resource utilization and power consumption. Machine and Deep learning classifiers, predictors have become popular in IoT, computer vision, medical devices, and autonomous vehicle applications. In this work, we adopted a single-layer feedforward network (G.B. Hung 2006) to predict the multicore in terms of "4L1b,3L1b, 0L1b, 8L0b, 0L384b, and 8Lb". Each task's best hardware cluster is selected based on the task emergency value and core utilization value. Based on this prediction, each ready task arrived from queues is allocated and executed on respective cores. In this network, 'J' neurons in the concealed layer are expected to operate with a widely differentiable activation function (e.g., the Relu, sigmoid function). However, the output layer is straight. The concealed layers in ELM do not have to be tuned obligatorily. Loads of the concealed layer are given blindly (inclusive of bias loads). Figure 4 depicts the basic structure of the SLFN resource predictor with its input features are represented as 'X', concealed layer nodes are denoted as 'H', and otuput score is denoted as 't1 to tm'

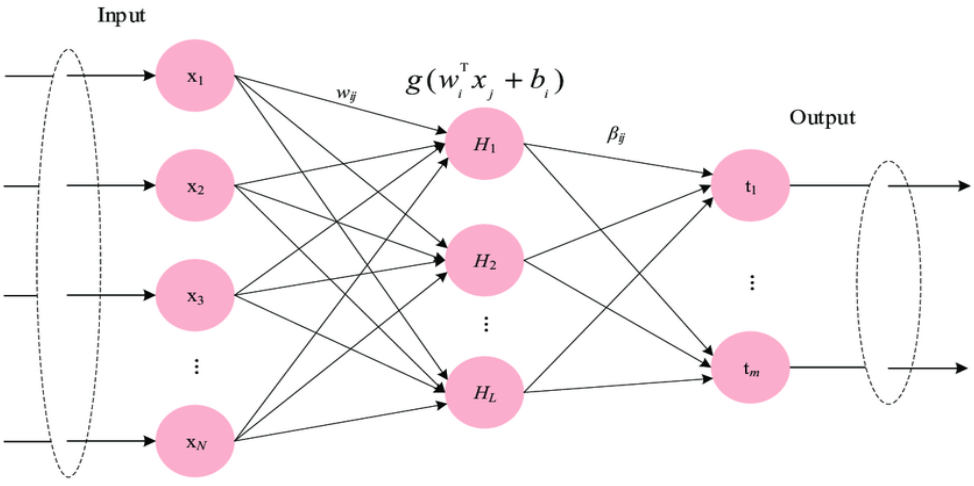


Figure 4. Basic structure of single-layer feedforward neural network (SLFN) predictor.

The condition is not that the hidden neurons are inapplicable; instead, there is no need for tuning. Even in advance, the hidden node's parameters can be chaotically generated. The concealed neurons' parameters can be randomly produced even in advance. Before taking care of the training set data, for an SLFN resource predictor, the system yield is given by

$$SLFN_L(k) = \sum_{i=1}^{J} \mu_i \, \partial_i(k) = \partial(k)\mu \tag{1}$$

where $k \in R^d$ denotes the input layer vector of 'd' dimension, $\mu = [\mu_1 \dots \dots \dots \dots \dots \mu_L]^T$ denotes the vector between hidden to output weights, J is the No. of neurons in the new layer, and $\partial(k) = [\partial_i(k) \dots \dots \partial_J(k)]$ is the hidden layer output vector.

The following Equation provides the 'J' unknown layer output vector. $\partial_i(k)$

$$\partial_i(k) = G(a_i, b_i, k) = \frac{1}{1 + exp(-(a_i k + b_i))} \tag{2}$$

Based on Equation (9), 'G' is considered the general sigmoid function with (a,b) node values. The hidden output layer vector μ is utilized to diminish the training dataset error that connects the output as

$$H\mu = T \tag{3}$$

$$\mu = H^{+}T \tag{4}$$

Based on the Equations (10) and (11), '∂' remain hidden or unknown layer output matrix, 'T' denotes the target training set data matrix and H^{+} denotes the Moore Penrose inverse of H. The above equation. can also be given as

$$\mu = H^T \left(\frac{1}{c} HH^T\right)^{-1} OP \tag{5}$$

Hence the output function can be found using the above equation.

$$\partial(x)\mu = \partial(x)H^T \left(\frac{1}{c} HH^T\right)^{-1} OP \tag{6}$$

$$SLFN_L(x): OP = min' \, C \, L' \tag{7}$$

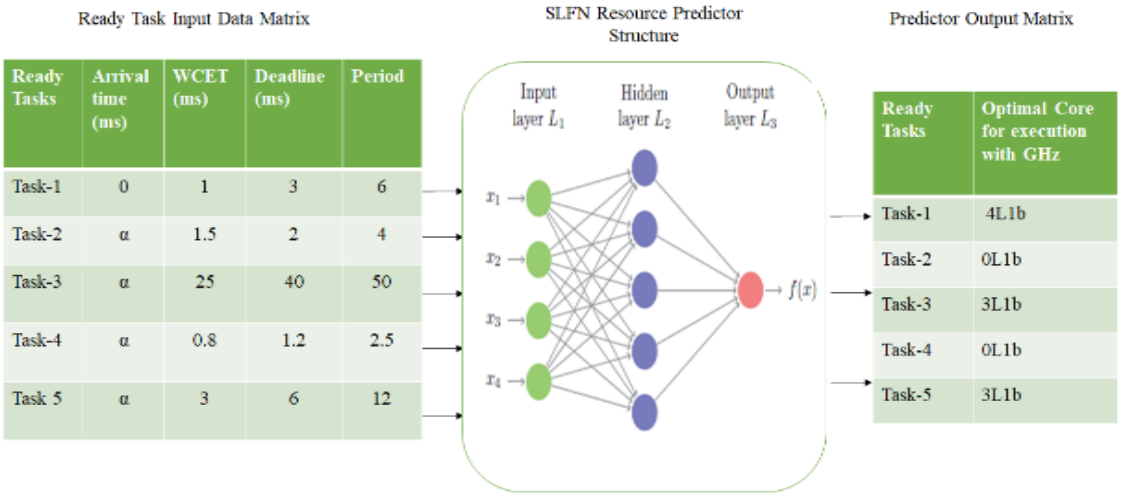
where 'c' is the approximate constant, and 'OP' is the output matrix.

ELM uses the kernel function to yield good accuracy for better performance. The significant advantages of the ELM are minimal training error and better approximation. Since ELM uses the auto-tuning of the weight biases and non-zero activation functions, a detailed description of ELM equations can be found in the literature [32]. The pseudocode for the ELM is shown in Algorithm 1.

Algorithm1: SLFN Resource Predictor
Input: Trained dataset with a random number of weights and bias values
Output: Predicted Score in terms of optimal core and frequency parameter
1: The network is initialized with input neurons 'X.'
2: A set of concealed neurons are generated as a single layer.
3: Sigmoid and relu functions are defined as activation functions.
4: Synapse parameters are generated through the gaussian random method.

5: After each iteration concealed matrix are updated
6: The output matrix values are calculated.
7: Final prediction scores are observed for the entire system.

Ready tasks parameters such as arrival time, Worst Case Execution Time (WCET) values with a respective deadline, and period with execution parameters are entered as input features to the developed SLFN network, which deviated from {1 to N}. The proposed SLFN resource predictor is a supervised learning network that is trained with several sensors and actuator workloads execution characteristics along with hardware configuration labels were trained statically. Figure 5 illustrates the working procedure of SLFN resource predictor executed for sample task set. Initially, the task features are feed-forwarded into SLFN modeling to obtain the best processor. The sample task graph is executed on the hardware configuration 1.



Ready Tasks	Arrival time (ms)	WCET (ms)	Deadline (ms)	Period
Task-1	0	1	3	6
Task-2	α	1.5	2	4
Task-3	α	25	40	50
Task-4	α	0.8	1.2	2.5
Task 5	α	3	6	12

Ready Tasks	Optimal Core for execution with GHz
Task-1	4L1b
Task-2	0L1b
Task-3	3L1b
Task-4	0L1b
Task-5	3L1b

Figure 5. SLFN Resource Predictor Results for periodic task graph mapped on hardware configuration-1 (Odroid XU4).

4.3. Task Scheduler Method

Earliest Hyperperiod First Scheduling Sequence

In this work, mixed category workloads are allocated on optimal cores for execution. Liu and Layland et al. developed optimal uni-processor algorithms such as RM and EDF, which computes the hyper period of ready tasks to the total processor utilization (i.e., total execution cycles). Likewise, Sherik et al. acknowledged individual task periods play a vital role in resource consumption complexity. To address this issue, the author developed an adaptive hyper period generator to optimize each task period. We are motivated by this; our algorithm initially identifies the optimized hyper period for each task based on the earliest hyper period, each task is executed on the allocated processor. The optimal core executes aperiodic tasks. If the optimal core is running a high-priority periodic or complicated task, it will enter into the suspended state, and aperiodic tasks are executed whenever it arrives. This pre-defined condition is assumed to avoid the RT-IoT system devastation. Each task graph includes the '*N*' number of task nodes represented by periodic, aperiodic, or complex sets. A task node includes essential parameters such as WCET of task ω_1 denoted by task inter-arrival instance is denoted by period and hyper period (i.e., the completion time of each task set), and core utilization is calculated based on Equation (8).

$$Cu_i = ET_i/t_i \quad (8)$$

The Cluster utilization is denoted by Equation (9),

$$CU_i = \sum_{i=1}^{cl} Cu_i \quad (9)$$

These two factors find whether a given task set ω is schedulable on a given CPU of 'M' cores.

$$H\omega_i = \{LCM(t_i = t_1, t_2 \ldots t_n)\} \quad (10)$$

The hyper-period '$H\omega_i$' of a set of n tasks 'ω' can be found by Equation (9), where LCM is the least common multiple of tasks' period, and 'ti' is an individual task. Task nodes with the "earliest hyper period are executed first" from the allocated queue. As we mentioned, Liu and Layland fixed the schedulability criteria as $'TU' \leq 1$. The problem arises when the algorithm passes the schedulability test. However, the embedded IoT objects do not have sufficient resources to execute tasks due to the requirement of higher clock cycles than devices can support. In this work, we assumed $H\omega_i \leq Threshold_M$; then, the entire task set is schedulable, where the threshold is the total load capacity of core 'M' at runtime.

5. Proposed SLFN_EHF Scheduling Algorithm

Three modules form the entire RT-IoT system, which is elaborated below (Algorithm 2).

Algorithm 2 Task Scheduling Mechanism

Input: Periodic, Aperiodic, and Complex tasks
$\boldsymbol{Per_i} = \{\boldsymbol{AT_i}, \boldsymbol{ET_i}, \boldsymbol{D_i}, \boldsymbol{t_i}, \boldsymbol{Token_i}\} \boldsymbol{APer_i} = \{\boldsymbol{e_i}, \boldsymbol{AT_i}, \boldsymbol{ET_i}, \boldsymbol{D_i}, \boldsymbol{t_i E'_i}\} \boldsymbol{Compl_i} = \boldsymbol{AT_i}, \boldsymbol{ET_i}, \boldsymbol{D_i}, \boldsymbol{t_i}, \boldsymbol{E'_i}$With respective arrival time, Average case Execution time, deadline, period, priority list.

Outcome: Prediction Accuracy, Task Miss rate, Execution time metrics

1: Ready tasks =[ ]
2: Arrival time =[ ]
3: The average Case Execution time of each task is calculated by ET = WCEC/period.
4: Task parameters are modeled as neurons (i.e., features), which enter the core prediction model.
5: Single Hidden Layer Feedforward Neural Network core prediction network
6: X1…Xn = []
7: Randomly generated bias and weights for SLFN at runtime.
8: Activation function = relu, optimizer = adam,
9: Trained with workload characteristics which executed using Nvidia SoC
10: Ready tasks are feedforwarded to the SLFN resource predictor in order to predict the optimal hardware processor.
11: Predicted optimal core in the final layer using the softmax function
12: Tasks are allocated in the optimal core (i.e., queue), given the SLFN network.
13: The pre-assumed condition is Aperiodic tasks alone executed immediately on current processors to avoid devastation.
14: Periodic and complex tasks on each core are scheduled and executed based on the hyper period method.
15: Hyperperiod (Hp) = {LCM(t_i) 1< 0 < n }
16: Tasks are sorted in non-decreasing order of Hp.
17: **If** $Hw_i < Threshold_M$ condition has been checked
18: The task set is schedulable and executed on a predicted processor
19: **else**

20: The task set is moved to the waiting queue, and the same is executed on the base cluster
21: **end**
22: Task miss deadline = miss/Total tasks on each queue
23: Execution time is calculated based on CPU processing time for the entire process.
24: Every iteration results are updated to the cloud server automatically at a regular 2: time interval.
25: **End**

The tasks are allocated on the predicted processor, which is optimal for each task at runtime based on the two conditions verified by the task manager. The hyper-period metric needs to be less than the active cluster threshold, and its emergency is very high. If this condition is satisfied, then the task graph is allocated on the optimal cluster selected by the proposed SLFN resource predictor. Otherwise, the task graph will be allocated to the waiting list and executed on the base cluster. The optimal clusters are activated only if the tasks are allocated to them. Otherwise, it will be deactivated (idle state). Both hardware configurations are activated initially, and high-processing tasks such as obstacle detection, lane detection, path planning are executed on the hardware configuration.2. Figure 6 illustrate the working procedure of SLFN_EHF task management technique.

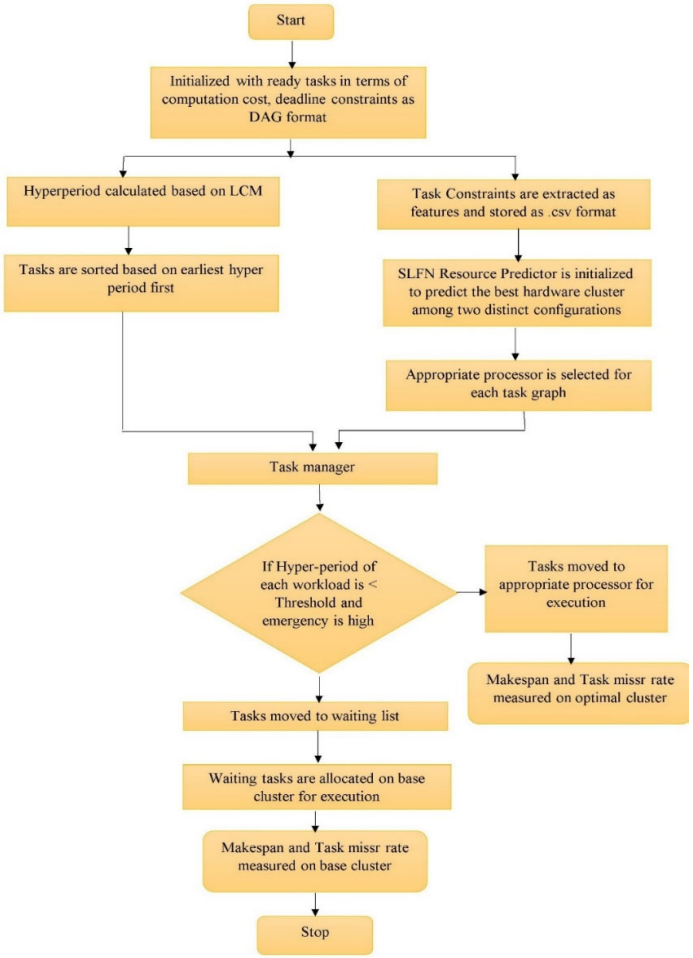


Figure 6. Flowchart of proposed task management system for autonomous vehicles5. Performance Analysis of Task Management Framework.

This section presents three sub-modules called simulation environments, implementation setups, and real-time test descriptions to develop and test the proposed mechanism. Results and discussion follow, and the corresponding performance graphs are shown below.

5.1. Simulation Environment

The famous "Python IDE-version 3.8" with machine learning libraries is used to develop the learning predictors and installed on two embedded kits and PCs. Task Graph For Free (TGFF) [33] is used to generate the synthetic real-time task graphs and their constraints. The "psutils and Perf" tools are used to evaluate the performance metrics after each execution.

5.2. Implementation Setup's

In this work, we have used two different embedded multicore boards, such as the "Odroid Xu4 and Nvidia Jetson xavior" [34] board for evaluation and validated the proposed models. Jetson modules include an on-board power monitor, the INA3221, to monitor the voltage and current of curtain rails. Table 3 shows the hardware configuration board details.

Table 3. SLFN resource Predictor testing parameters.

Network Type	Single-Layer Neural Network
Activation variable	Sigmoid and Relu functions
No of the hidden nodes	20
Input parameters	N (Ready Task parameters form Allocation Queue)
Output parameters	1 (Optimal Hardware cluster)

5.3. Realtime Benchmark Programs

IoMT [35] and autonomous vehicle application comprises of taskset-1: Object Tracking, and Path planning programs are developed in python. These are the standard autonomous vehicle application which is mentioned on the CAV benchmark [36]. The IoMT database includes nine programs named taskset-2: Activity, Imhist, Aes, Iradon, Squ_wab, Hrv, Lzw, and Adpet. These programs are frequently executed on both embedded kits to observe the different characteristics in terms of features are listed in Section 3. Various sensors and actuators based programs such as taskset-3: "getpressure, gethumidity, getmotion, gettemp, LED" are used for data collection. The synthetic database also generated using TGFF and pseudorandom values that are listed in Table 4. In total, more than 10k data have been used for training and testing purposes. The synthetic task graphs are generated. Tables 5 and 6 depicts the parameters used for synthetic and benchmark workloads with its task description

Table 4. Hardware Configuration Details.

Hardware Configurations	Multicore Processor Config.1	Multicore Processor Config.2
No. of. cores	8 ('ARM Cortex-A15, A7')	384-core Volta GPU with 48 Tensor Cores, 6-Core
(CPUs and GPUs)	Mali-T628MP6	ARM v8.2 64-bit CPU, 8-MB L2 + 4 MB L3
Operating freq. max	2.0 GHz and	1100 MHz and 1.4 GHz
Operating voltage. max	5 V	5 V
Power	10–20 W	10–15 W
Memory	8 GB	16 GB
Cache used	L1-64 KB & L2-2 MB	16 GB 256-bit LPDDR4x| 137 GB/s
Execution order	Out-of-order	In-order

Table 5. Synthetic Task Graphs used for validation.

Synthetic Task Graphs	Task Graphs	Task Nodes	Task Edges	Deadline (s)	Period (s)
Synthetic Tasks-20	Graphs {0–20}	{1–20}	{1–20}	{0.0001–30}	{0.0001–30}
Synthetic Tasks-40	Graphs {0–40}	{1–40}	{1–39}	{0.0001–60}	{0.0001–60}
Synthetic Tasks-60	Graphs {0–60}	{1–60}	{1–59}	{0.0001–90}	{0.0001–90}
Synthetic Tasks-80	Graphs {0–80}	{1–80}	{1–79}	{0.0001–50}	{0.0001–50}

Table 6. Real-world Benchmark Task Graphs with Constraints.

Real-Time Benchmark Programs	Task Graphs	Task Nodes	Task Edges	Deadline (s)	Period (s)
Object_tracking-graph	Graph-0	165	164	280	400
Video_Analysis-graph	Graph-0	120	119	165	250
Path_planning-graph	Graph-0	145	144	233	300
Navigation-graph	Graph-0	100	99	120	500
AES-graph	Graph-0	4	4	0.01	0.01
	Graph-1	6	6	0.01	0.02
	Graph-2	6	6	0.01	0.02
	Graph-3	3	2	0.001	0.001
	Graph-4	3	2	0.001	0.001
	Graph-5	2	1	0.0004	0.0003
	Graph-6	2	1	0.0001	0.0005
	Graph-7	2	1	0.0004	0.0005
	Graph-8	2	1	0.0004	0.0005
Imhist-graph	Graph-0	8	8	0.002	0.002
	Graph-1	5	4	0.004	0.004
Activity-graph	Graph-0	6	6	0.05	0.05
	Graph-1	4	3	0.004	0.004
	Graph-2	7	5	0.06	0.08
Iradon-graph	Graph-0	4	3	0.001	0.002
	Graph-1	4	4	0.004	0.003
	Graph-2	3	3	0.003	0.003
Squ_web-graph	Graph-0	8	8	0.002	0.002
	Graph-1	5	4	0.004	0.004
HRV-graph	Graph-0	6	6	0.05	0.05
	Graph-1	4	3	0.004	0.004
LZW-graph	Graph-0	8	10	0.08	0.08
Apet-graph	Graph-0	15	14	0.2	0.5
	Graph-1	10	9	0.4	0.4
	Graph-2	8	7	0.8	0.8
gettemp-graph	Graph-0	6	6	0.05	0.05
		4	3	0.004	0.004
		7	5	0.06	0.08
gethumidity-graph	Graph-0	3	2	0.01	0.03
getmotion-graph	Graph-0	3	3	0.01	0.03
Getpressure-graph	Graph-0	3	3	0.01	0.03
LED-graph	Graph-0	8	7	0.5	0.8

6. Results and Discussion

The proposed algorithms are validated using an estimated few scheduling metrics such as task miss rate, execution time (makespan), cluster utilization, and prediction accuracy were estimated as per the below equations.

$$\phi = \frac{\mu}{\rho} \times 100 \tag{11}$$

where $'\phi'$ accuracy rate of core prediction $'\mu'$ denotes the predicted number of cores divided by $'\rho'$ an actual number of active cores.

$$TMR = \frac{miss_i}{\omega} \tag{12}$$

where TMR is called the missed task rate, which is calculated by dividing the missed task instance by the total number of completed task instances on that particular task graph (ω).

$$Makespan = \max\left(completiontime(\omega)\right) \tag{13}$$

where makespan is the total time taken to complete the entire set of tasks ('*M*') on the entire set of allocated cores ('*N*'). These metrics are estimated for each task graph and normalized the values.

Tables 7 and 8 clarifies that the proposed SLFN network achieved better results compared to other machine learning algorithms, where all the strategies have been optimally tuned for the problem in hand. LW-DNN and SLFN achieved nearly 96–98% accuracy for both synthetic and real-time benchmarks. SLFN achieved better performance for sensors, actuator workloads in real-time. These results are obtained based on the intellectual selection of processors and the best order of execution. Few workloads such as object detection, edge video analysis only utilized the hardware configuration.2 and the resources are optimized through this appropriate selection and utilization.

Table 7. Normalized Execution time obtained standard benchmark workloads Hardware configuration -1.

Task Models	Normalized Execution Time (s)		Task Miss Rate (%)	
Hardware Clusters	**Base Cluster**	**Optimal Cluster**	**Base Cluster**	**Optimal Cluster**
IoT based AV Workloads {1–150 nodes}	1800	1400	0.21	0.026
IoMT workloads {1–50 nodes}	1000	855	0.18	0.22
Synthetic task workloads {1–80 nodes}	200	120	-	0.01

Table 8. Normalized Execution time obtained for synthetic and AV workloads on Hardware configuration -2.

Task Models	Normalized Execution Time (s)		Task Miss Rate (%)	
Hardware Clusters	**Base Cluster**	**Optimal Cluster**	**Base Cluster**	**Optimal Cluster**
IoT based AV Workloads {1–150 nodes}	1000	855	0.33	0.31
IoMT workloads {1–50 nodes}	500	399	0.22	0.25
Synthetictask workloads {1–80 nodes}	110	89	-	0.01

Synthetic workloads are executed on appropriate processors selected by the proposed resource predictor. The accuracy is achieved by nearly 96.3 to 98% for synthetic workloads, which is highly utilized in the hardware configuration.1. Figures 7 and 8 illustrates the resource prediction accuracy obtained for sythetic task sets and benchmark workloads at runtime. The proposed SLFN_EHF task scheduler achieved nearly 97.5% in an average for benchmark workloads.

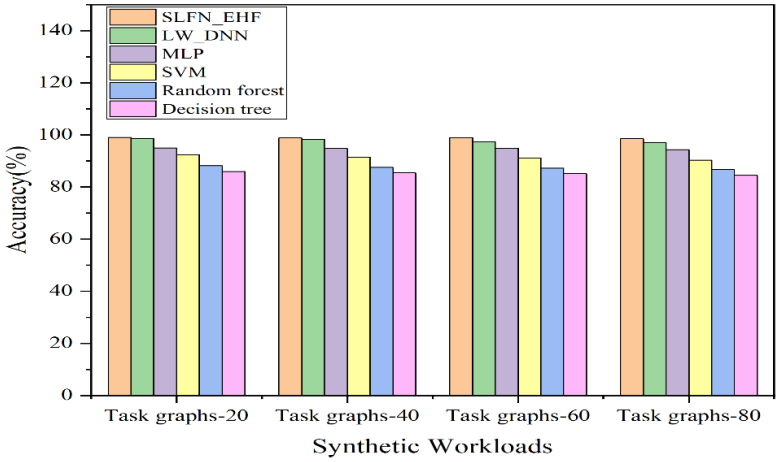


Figure 7. Prediction Accuracy for Synthetic workloads on Hardware Configuration.1.

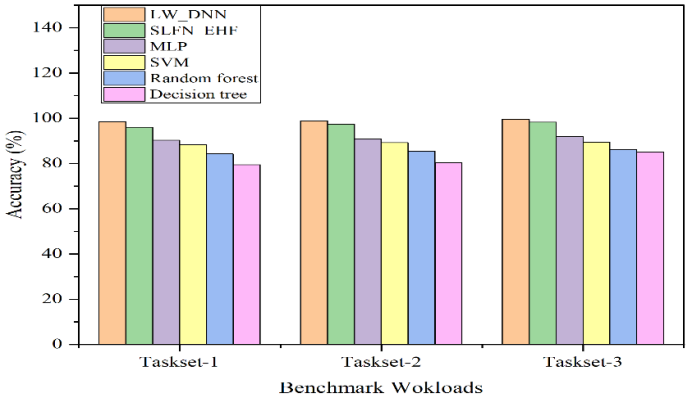


Figure 8. Prediction Accuracy for Benchmark workloads on Hardware Configuration.1.

Figure 9 illustrate the task miss instance rate observed for synthetic workloads execution on hardware configuration 1. The TMR is minimized in an average of 15–22% compared to traditional schedulers due to selection of best processors for ready tasks at runtime.

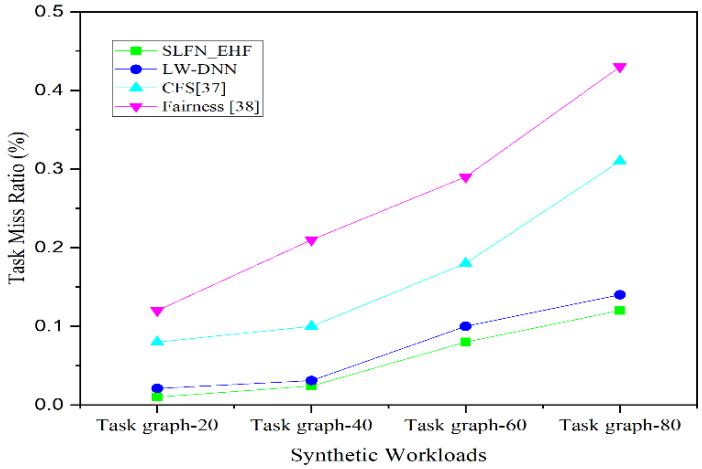


Figure 9. Normalized Task Miss Ratio observed for Synthetic workloads on Hardware Configuration.1.

Figure 10 depicts the task miss instance rate on hardware configuration 1 for CAV and IoMT benchmark workloads. SLFN_EHF and LW_DNN task schedulers are optimized the TMR compared to the traditional algorithms.

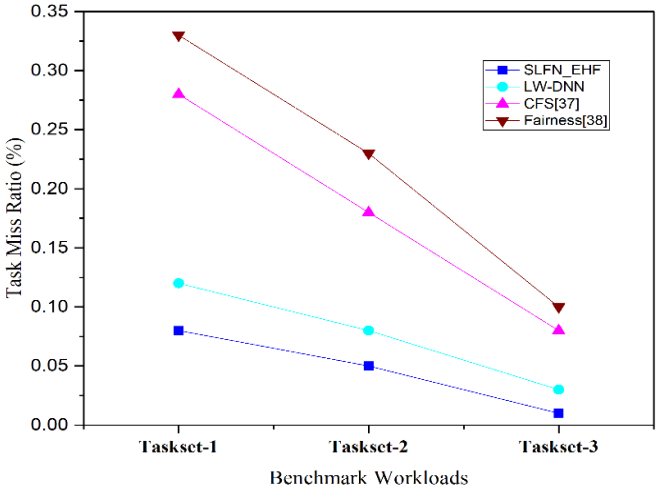


Figure 10. Normalized Task Miss Ratio observed for Benchmark workloads on Hardware Configuration.1.

Figures 11 and 12 represents the overall execution time (Makespan) observed for both synthetic tasksets and benchmark workloads. The proposed task schedulers reduced the makespan nearly 27–32% compared with traditional algorithms. The makespan of the taskset −3 is nearly equal for all methods due to its low-level execution cost compared to other tasksets.

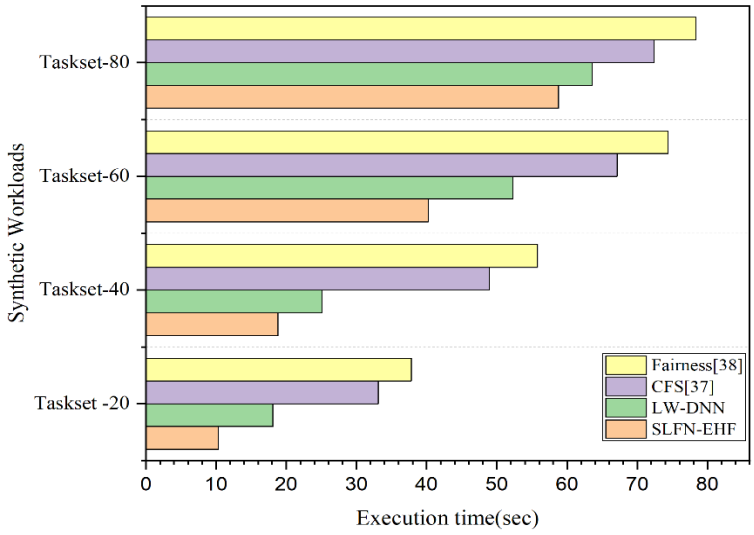


Figure 11. Normalized Execution time observed for Synthetic workloads on Hardware Configuration.1.

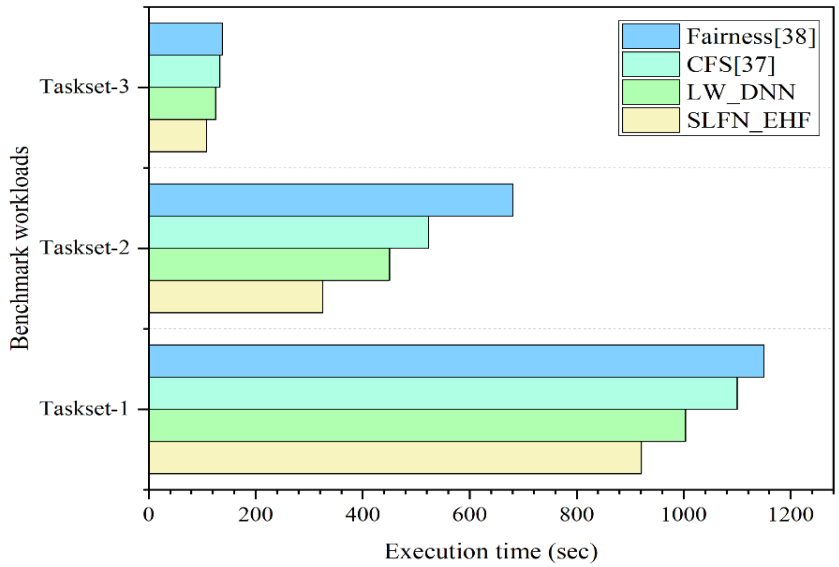


Figure 12. Normalized Execution time observed for Benchmark Workloads on Hardware Configuration.1.

The overall schedule length has been measured as per Equation (12). Figures 13 and 14 represents the normalized task miss rate and execution time that is optimized using the proposed predictor based task scheduler. Both synthetic and benchmarak workloads are executed on hardware configuration.2 Few synthetic task sets are nearly executed within micro seconds on hardware configuration.2 due to its high-speed and large number of GPUs. The SLFN core predictor with EHF task scheduler outperformed traditional CFS [37] and Fairness [38] algorithms. The proposed algorithm achieved better results on each cluster in an average of 20–27% in makespan metric and 32–45% reduction in task miss rate metric than CFS and fairness schedulers.

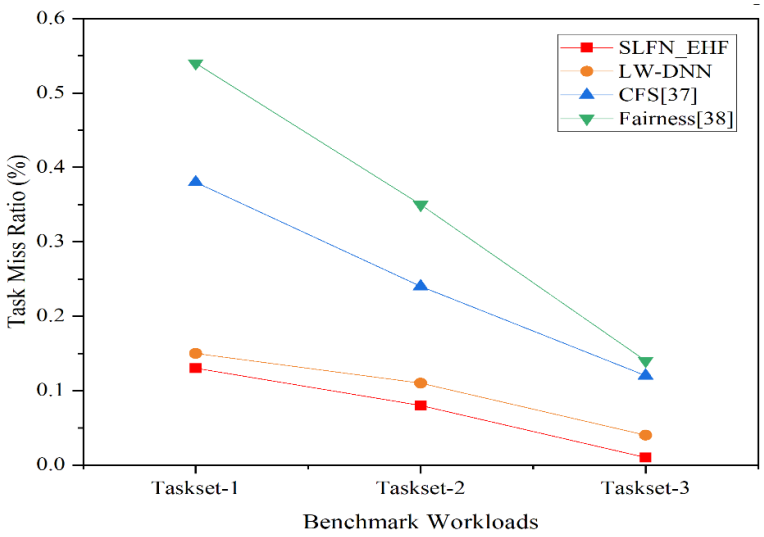


Figure 13. Task miss rate (TMR) analysis observed on Hardware Configuration-2 for the execution of Benchmark Workloads.

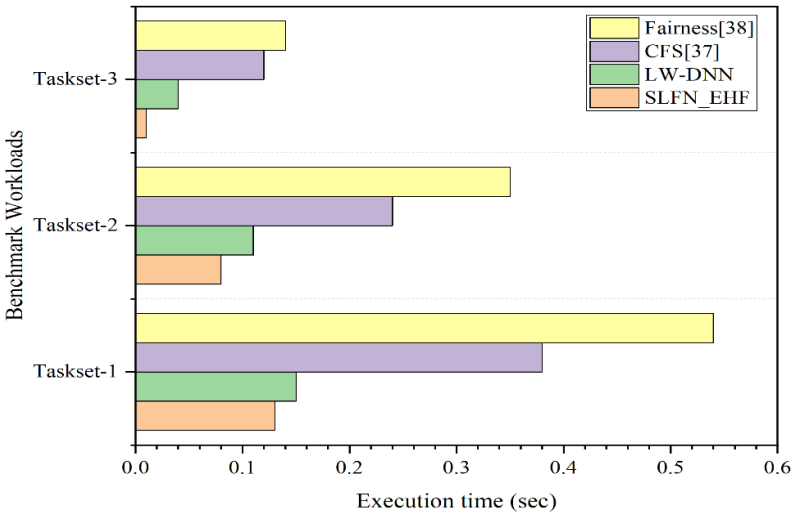


Figure 14. Overall Execution time observed on Hardware Configuration-2 for the execution of Benchmark Workloads.

7. Conclusions

Autonomous vehicle and IoT applications have become everywhere in recent days. Sensors, actuators are the core components of the applications mentioned above. The software challenges in terms of task distribution, appropriate allocation, execution on-time without delay are still NP-hard issues in real-time. This paper developed an intelligent kernel that comprises the SLFN core predictor and EHF task scheduler algorithm for IoT-based autonomous vehicles. The essential purpose is to minimize the task miss rate and overall schedule length by intelligent mapping and execution kernel. The proposed framework improves the scheduling performance by selecting and executing optimal hardware configuration in terms of core type, frequency–voltage level for each workload at runtime.

Additionally, two different hardware setups have been utilized for experimental validation of the proposed system performance. The proposed SLFN core predictor is compared with the previous work LW-DNN core predictor and achieved better accuracy of nearly 96–98% approximately and in an average of 27% reduced in execution time and 45% in task miss rate metric. The limitation of the proposed method is targeted only the

IoT-based autonomous vehicle workloads and embedded workloads for training and testing purposes which can be extended with other multimedia benchmarks in future work.

Author Contributions: The paper investigation, resources, data curation, writing—original draft preparation, writing—review and editing, visualization, were done by G.B. The paper conceptualization, methodology, software, validation, and formal analysis were done by S.J. Supervision, project administration, and final approval of the version to be published were conducted by R.P.d.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Amalraj, J.J.; Banumathi, S.; John, J.J. IOT Sensors and Applications: A Survey. *Int. J. Sci. Technol. Res.* **2019**, *8*, 998–1003.
2. Shafique, K.; Khawaja, B.A.; Sabir, F.; Qazi, S.; Mustaqim, M. 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* **2020**, *8*, 23022–23040, doi:10.1109/access.2020.2970118.
3. Barabás, I.; Todoruţ, A.; Cordoş, N.; Molea, A. Current challenges in autonomous driving. *IOP Conf. Ser. Mater. Sci. Eng.* **2017**, *252*, 012096.
4. Martínez-Díaz, M.; Soriguera, F. Autonomous vehicles: Theoretical and practical challenges. *Transp. Res. Procedia* **2018**, *33*, 275–282, doi:10.1016/j.trpro.2018.10.103.
5. Yurtsever, E.; Lambert, J.; Carballo, A.; Takeda, K. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* **2020**, *8*, 58443–58469, doi:10.1109/access.2020.2983149.
6. Mohammadi, A.; Akl, S.G. *Scheduling Algorithms for Real-Time Systems*; Technical Report; School of Computing Queens University: Kingston, ON, Canada, 2005.
7. Wolf, W. The future of multiprocessor systems-on-chips. In Proceedings of the 41st Design Automation Conference, San Diego, CA, USA, 7–11 July 2004; pp. 681–685, doi:10.1145/996566.996753.
8. Singh, A.K.; Dey, S.; McDonald-Maier, K.; Basireddy, K.R.; Merrett, G.V.; Al-Hashimi, B.M. Dynamic Energy and Thermal Management of Multi-core Mobile Platforms: A Survey. *IEEE Des. Test* **2020**, *37*, 25–33, doi:10.1109/mdat.2020.2982629.
9. Bechtel, M.G.; McEllhiney, E.; Kim, M.; Yun, H. DeepPicar: A Low-Cost Deep Neural Network-Based Autonomous Car. In Proceedings of the 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hakodate, Japan, 28–31 August 2018; pp. 11–21, doi:10.1109/RTCSA.2018.00011.
10. Cheng, L. Mixed-Criticality Scheduling of an Autonomous Driving Car. Master's Thesis, Institute for Integrated Systems, Technische Universitat Munchen, Munchen, Germany, 2016.
11. Wang, X.; Huang, K.; Knoll, A. Performance Optimisation of Parallelized ADAS Applications in FPGA-GPU Heterogeneous Systems: A Case Study with Lane Detection. *IEEE Trans. Intell. Veh.* **2019**, *4*, 519–531, doi:10.1109/tiv.2019.2938092.
12. Adegbija, T.; Rogacs, A.; Patel, C.; Gordon-Ross, A. Microprocessor Optimizations for the Internet of Things: A Survey. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2017**, *37*, 7–20, doi:10.1109/tcad.2017.2717782.
13. Chetto, H.; Chetto, M. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Trans. Softw. Eng.* **1989**, *15*, 1261–1269, doi:10.1109/tse.1989.559777.
14. Dai, H.; Zeng, X.; Yu, Z.; Wang, T. A Scheduling Algorithm for Autonomous Driving Tasks on Mobile Edge Computing Servers. *J. Syst. Arch.* **2019**, *94*, 14–23, doi:10.1016/j.sysarc.2019.02.004.
15. El Ghor, H.; Aggoune, E.-H.M. Energy efficient scheduler of aperiodic jobs for real-time embedded systems. *Int. J. Autom. Comput.* **2016**, *17*, 733–743, doi:10.1007/s11633-016-0993-3.
16. Wu, X.; Tian, S.; Zhang, L. The Internet of Things Enabled Shop Floor Scheduling and Process Control Method Based on Petri Nets. *IEEE Access* **2019**, *7*, doi:10.1109/access.2019.2900117.
17. Ma, X.; Gao, H.; Xu, H.; Bian, M. An IoT-based task scheduling optimization scheme considering the deadline and costaware scientific workflow for cloud computing. *EURASIP J. Wirel. Commun. Netw.* **2019**, *2019*, 249.
18. Bini, E.; Buttazzo, G.C. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Comput.* **2004**, *53*, 1462–1473, doi:10.1109/tc.2004.103.
19. Cha, H.-J.; Jeong, W.-H.; Kim, J.-C. Control-Scheduling Codesign Exploiting Trade-Off between Task Periods and Deadlines. *Mob. Inf. Syst.* **2016**, *2016*, 1–11, doi:10.1155/2016/3414816.
20. Huang, G.-B.; Zhu, Q.-Y.; Siew, C.-K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501, doi:10.1016/j.neucom.2005.12.126.
21. Chen, C.-Y.; Hasan, M.; Mohan, S. Securing Real-Time Internet-of-Things. *Sensors* **2018**, *18*, 4356, doi:10.3390/s18124356.

22. Nasri, M.; Fohler, G. An Efficient Method for Assigning Harmonic Periods to Hard Real-Time Tasks with Period Ranges. In Proceedings of the 27th Euromicro Conference on Real-Time Systems, Lund, Sweden, 7–10 July 2015; pp. 149–159.
23. Liu, S.; Tang, J.; Zhang, Z.; Gaudiot, J.-L. Computer Architectures for Autonomous Driving. *Computer* **2017**, *50*, 18–25, doi:10.1109/mc.2017.3001256.
24. Sarangi, S.R.; Goel, S.; Singh, B. Energy efficient scheduling in IoT networks. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, Pau, France, 9–13 April 2018; pp. 733–740.
25. Ahmad, S.; Malik, S.; Ullah, I.; Fayaz, M.; Park, D.-H.; Kim, K.; Kim, D. An Adaptive Approach Based on Resource-Awareness Towards Power-Efficient Real-Time Periodic Task Modeling on Embedded IoT Devices. *Processes* **2018**, *6*, 90, doi:10.3390/pr6070090.
26. Malik, S.; Ahmad, S.; Ullah, I.; Park, D.H.; Kim, D. An Adaptive Emergency First Intelligent Scheduling Algorithm for Efficient Task Management and Scheduling in Hybrid of Hard Real-Time and Soft Real-Time Embedded IoT Systems. *Sustainability* **2019**, *11*, 2192, doi:10.3390/su11082192.
27. Samie, F.; Bauer, L.; Henkel, J. Iot Technologies for Embedded Computing: A Survey. In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, Pittsburgh, PA, USA, 2–7 October 2016; pp. 1–10.
28. Tang, J.; Yu, B.; Liu, S.; Zhang, Z.; Fang, W.; Zhang, Y. π-SoC: Heterogeneous SoC Architecture for Visual Inertial SLAM Applications. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 8302–8307.
29. Xu, J. A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems. In Proceedings of the 2010 IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications (MESA), Qingdao, China, 15–17 July 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 288–294.
30. Brüning, J.; Forbrig, P. TTMS: A Task Tree Based Workflow Management System. In Proceedings of the International Conference, BPMDS 2011, and 16th International Conference, EMMSAD 2011, London, UK, 20–21 June 2011; Volume 81, pp. 186–200.
31. Gomatheeshwari, B.; Selvakumar, J. Appropriate allocation of workloads on performance asymmetric multicore architectures via deep learning algorithms. *Microprocess. Microsyst.* **2020**, *73*, 102996, doi:10.1016/j.micpro.2020.102996.
32. Akusok, A. Extreme Learning Machines: Novel Extensions and Application to Big Data. Ph.D. Thesis, University of Iowa, Iowa City, IA, USA, 2016; doi:10.17077/etd.i9q1uhwn.
33. Topcuoglu, H.; Hariri, S.; Wu, M.-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274, doi:10.1109/71.993206.
34. Tang, J.; Liu, S.; Liu, K.; Yu, B.; Shi, W. LoPECS: A Low-Power Edge Computing System for Real-Time Autonomous Driving Services. *IEEE Access* **2020**, *8*, 30467–30479, doi:10.1109/access.2020.2970728.
35. Limaye, A.; Adegbija, T. HERMIT: A Benchmark Suite for the Internet of Medical Things. *IEEE Internet Things J.* **2018**, *5*, 4212–4222, doi:10.1109/jiot.2018.2849859.
36. Wang, Y.; Liu, S.; Wu, X.; Shi, W. CAVBench: A Benchmark Suite for Connected and Autonomous Vehicles. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), San Jose, CA, USA, 14–17 December 2018; pp. 30–42.
37. Saez, J.C.; Pousa, A.; Castro, F.; Chaver, D.; Prieto-Matias, M. Towards completely fair scheduling on asymmetric single-ISA multicore processors. *J. Parallel Distrib. Comput.* **2017**, *102*, 115–131, doi:10.1016/j.jpdc.2016.12.011.
38. van Craeynest, K.; Akram, S.; Heirman, W.; Jaleel, A.; Eeckhout, L. Fairnessaware Scheduling on single-is heterogeneous multicores. In Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Boston, MA, USA, 20–23 October 1996; pp. 177–187.