electronics

MDPI

Article

Traffic-Aware Intelligent Association and Task Offloading for Multi-Access Edge Computing

Avilia Kusumaputeri Nugroho and Taewoon Kim *

School of Computer Science and Engineering, Pusan National University, Busan 46241, Republic of Korea; avilia22@pusan.ac.kr
* Correspondence: taewoon@pusan.ac.kr

Abstract: Edge computing is a promising technology, especially for offloading users' computationally heavy tasks. The close proximity of edge computing units to users minimizes network latency, thereby enabling delay-sensitive applications. Although optimal resource provisioning and task offloading in edge computing are widely studied in the literature, there are still some critical research gaps. In this study, we propose a traffic-aware optimal association and task-offloading approach. The proposed method does not rely solely on the average rate of offloading requests, which can differ from actual values in real time. Instead, it uses an intelligent, high-precision prediction model to forecast future offloading requests, allowing resource provisioning to be based on future sequences of requests rather than average values. Additionally, we propose an optimization-based approach that can meet task deadlines, which is crucial for mission-critical applications. Finally, the proposed approach distributes the computing load over multiple time steps, ensuring future resource scheduling and task-offloading decisions can be made with a certain level of flexibility. The proposed approach is extensively evaluated under various scenarios and configurations to validate its effectiveness. As a result, the proposed deep learning model has resulted in a request prediction error of 0.0338 (RMSE). In addition, compared to the greedy approach, the proposed approach has reduced the use of local and cloud computing from 0.02 and 18.26 to 0.00 and 0.62, respectively, while increasing edge computing usage from 1.31 to 16.98, which can effectively prolong the lifetime of user devices and reduce network latency.

Keywords: multi-access edge computing; task offloading; traffic-awareness; optimization; time-series forecasting

Citation: Nugroho, A.K.; Kim, T. Traffic-Aware Intelligent Association and Task Offloading for Multi-Access Edge Computing. *Electronics* **2024**, *13*, 3130. https://doi.org/10.3390/electronics13163130

Academic Editors: Zhiwei Xu, Jianer Zhou, Xueshuo Xie and Zhao Huang

Received: 11 July 2024
Revised: 29 July 2024
Accepted: 5 August 2024
Published: 7 August 2024

1. Introduction

The proliferation of mobile devices is unmistakably pervasive, by recent statistics. According to the Pew Research Center, as of 2021, 97% of Americans own a cellphone, with 85% owning smartphones specifically, showcasing their ubiquitous presence in daily life [1]. Globally, there are over 6.4 billion smartphone users as of 2023, with projections indicating this number will surpass 7 billion by 2026 [2]. Furthermore, smartphone shipments totaled 1.35 billion units in 2021 alone [3], underscoring the continuous growth and adoption of mobile technology worldwide. These figures illustrate the widespread integration of mobile devices into personal and professional spheres, driving digital connectivity and innovation across various sectors.

In the meantime, mobile applications are evolving rapidly, becoming increasingly complex to meet the growing demands of users and advancements in technology. Today's mobile apps integrate sophisticated features such as real-time data processing [4], augmented reality (AR) [5], virtual reality (VR) [6], artificial intelligence (AI) [7], and complex backend integrations [8]. Despite advancements in computation and battery capacity, the complexity of mobile applications increases response time while decreasing battery life. Mobile/multi-access edge computing (MEC) holds significant promise by bringing

computational capabilities closer to end-users, reducing latency, and improving mobile network efficiency [9–12]. By leveraging MEC, applications can offload processing tasks from mobile devices to nearby edge servers (ESs), leading to faster response times for real-time applications such as AR, VR, and autonomous vehicles.

Task offloading in MEC addresses mobile computing challenges, including battery constraints, limited processing capacity, and delays in conventional cloud architectures [13]. This approach optimizes task execution, thereby reducing data travel distances and minimizing latency. MEC utilizes edge servers with a resource allocation system that dynamically allocates CPU, memory, and storage resources for fluctuating task offloading demands. Sophisticated algorithms optimize resource consumption, avoid conflicts over shared resources, maximize user experience (QoS), and enhance overall system efficiency. Additionally, load balancing distributes workloads among edge servers, ensuring none are overloaded and optimizing network utilization.

Moreover, user-base station (BS) association in task offloading involves strategic device connection to nearby BSs for optimal edge server utilization. Devices dynamically associate with BSs based on signal strength, available resources, and proximity to edge servers, ensuring efficient processing of offloaded tasks with minimal latency.

Extensive research has been conducted on the most effective strategies for provisioning and offloading tasks in MEC. A method was suggested to achieve optimal resource provisioning and task offloading in federated edge computing, involving the partitioning of edge servers into groups [14]. Another approach introduced a low-complexity task offloading solution through task routing [15]. Despite the technical progress the previous studies have successfully made, there are still some research gaps, including the following three. First, previous studies optimized resource provisioning and/or task offloading by assuming the average rate of task generation. However, the actual amount of tasks received at a particular moment may be different from the assumed average, which may cause resource under-/over-provisioning. Second, most studies assume that the received tasks can be completely processed right away or in the current time step (e.g., tasks are small enough). However, it may not be the case if the task is highly complex, requires sequential execution, or takes a long time. Lastly, some studies do not consider the fact that some applications are deadline-sensitive, which is especially important for mission-critical applications.

In this paper, we propose optimal resource provisioning and task offloading for MEC. By using a deep learning approach, the proposed method can predict future offloading requests with high precision by analyzing the network traffic history. The proposed approach optimizes multiple tasks for multiple future time steps to yield optimal scheduling decisions while guaranteeing deadline constraints. The summary of the contributions we make in this paper is given below.

- To avoid the mismatch between the average and actual offloading requests, we propose a deep learning-based multi-step offloading request prediction approach by learning from the past traffic, which includes the offloading requests.
- We propose a computing load balancing scheme over the time horizon so that each ES can make some resources available for accommodating unpredictable requests. This is particularly useful when the prediction of future requests is not perfect, as is the case in general.
- To maximize the offloading performance and fulfill the deadline constraints, we propose a multi-step optimization technique. That is, the proposed method optimizes both resource provisioning and task offloading for multiple future time steps.
- To maximize resource utilization and prevent excessive resource allocations, we propose an optimal network association solution that leads to the efficient use of edge servers.
- To validate the effectiveness of the proposed solution, we implement a time-slotted simulation environment and carry out extensive evaluations along with performance comparisons.

The remainder of this paper is structured as follows. The following Section 2 provides a comprehensive summary of related studies. In Section 3, the proposed approach is described in detail. In Section 4, the evaluation and comparison results are provided. Finally, Section 5 concludes this paper. The abbreviations and acronyms that frequently appear in this paper are summarized in Table 1.

Table 1. Abbreviations and acronyms used in this paper.

Abbreviation	Definition
BS	Base Station
CC	Cloud Computing
EC	Edge Computing
ES	Edge Server
MEC	Mobile/Multi-access Edge Computing
QoS	Quality of Service

2. Related Work

This section comprehensively reviews the optimization of resource provisioning and/or task offloading in MEC. For resource provisioning in MEC, the study in [16] optimizes MEC resource provisioning in IoT networks to meet performance requirements considering limited device resources and high data generation. Their method frames the provisioning of MEC resources and assignment of workloads as a mixed-integer program, utilizing a decomposition strategy to effectively manage workload variations. While this paper addresses workload assignment for immediate processing to achieve required response times, it assumes average user requests and does not consider delays for particularly heavy tasks. Xiang et al. [17] explore optimizing computing power allocation and traffic scheduling to minimize service response time in MEC, particularly for urgent or popular services. Their study introduces PCA-CATS, an online algorithm that effectively reduces service response time and expenses by optimizing resource allocation and traffic scheduling. However, the approach assumes immediate processing of received requests and does not address strict deadlines or time-sensitive tasks.

From the perspective of task offloading in MEC, various approaches have been proposed to enhance the efficiency of determining which tasks should be offloaded to edge servers or cloud data centers by considering aspects such as compute demands, network conditions, and device capabilities. Jiang et al. [18] proposed the Joint Offloading and Resource Allocation (JORA) framework, which employs Lyapunov optimization to maximize Quality of Experience (QoE) while considering long-term MEC energy constraints. JORA provides both centralized and distributed online approaches that distribute near-optimal performance. However, the immediate processing of received requests, which ignores task delay constraints such as processing deadlines, can result in prolonged processing times for heavy workloads. Apostolopoulos et al. [19] introduce an approach for optimizing data offloading in a multi-MEC server environment, focusing on users' risk-seeking tendencies rooted in Prospect Theory. Their research investigates the complexities of user decision-making in the face of uncertainty, placing it within a non-cooperative game framework aimed at achieving optimal offloading strategies. However, the study overlooks the task delay constraints, assumes average user request patterns, and employs immediate processing capabilities for all user requests. Han et al. [20] study impatience-driven user behaviors in MEC queue management, proposing an adaptive decision-making mechanism to balance latency outage risk and energy consumption. Their approach optimizes the allocation of tasks between local computing and MEC based on dynamic user demands and server conditions, with the goal of enhancing the QoE for applications that are sensitive to latency. However, they assume average user requests, which may differ from the actual values in real time.

An ES is, in general, co-located with a BS, and thus, making an optimal user-BS or user-edge association is important to further enhance the performance of MEC. Song et al. [21]

employ a genetic algorithm to optimize server deployment and user offloading in wireless edge networks with the objective of minimizing service latency. Feng et al. [22] concentrate on the optimization of task partitioning and user association in MEC systems to reduce the average latency. However, their approach depends on average requests and does not consider delay constraints for delay-sensitive tasks. In contrast, Charatsaris et al. [23] introduce Hierarchical Federated Learning (HFL) for wireless networks, focusing on user-to-edge association and power allocation to balance accuracy, energy, and time. However, they do not consider delay constraints for tasks. Wang et al. [24] introduced an optimization approach for associations, focusing on minimizing system delay and enhancing Quality of Service (QoS) through efficient user-BS pairing. However, they assume average user requests and that user requests are always processed immediately.

Traffic-aware task offloading strategies, introduced by Qi et al. [25] for vehicular edge computing, optimize task and wireless bandwidth ratios to minimize response time. Their algorithm efficiently handles this optimization, particularly with limited wireless capacity. Wang et al. [26] propose a method for IoV task offloading and content caching using traffic stream forecasting. However, their approach lacks explicit consideration of task processing deadlines and user demand variations. Oza et al. [27] introduce deadline-aware task offloading, which integrates traffic light data to optimize task scheduling. Their method ensures timely task processing by considering static and dynamic deadlines influenced by traffic conditions, yet immediate processing without queuing considerations may face challenges for longer tasks.

Intelligent approaches, such as those utilizing machine learning and deep learning, have also been proposed to enhance MEC systems. Guo et al. [28] introduce an intelligent task offloading scheme at the edge using supervised decision trees, improving prediction accuracy and processing delay in IoT scenarios. However, their method does not optimize resource provisioning and task offloading for multiple future time steps. Kim et al. [29] propose prediction-based sub-task offloading in MEC, employing linear regression to predict processing times across multiple nodes simultaneously. Their approach aims to enhance resource utilization efficiency and reduce application execution time on MEC servers but does not consider multiple future time forecasting for optimization. Zeng et al. [30] present a prediction-based task-offloading strategy for vehicular edge computing using deep-learning models. Their approach predicts task-offloading performance, including success/failure and service delay, yet does not optimize resource provisioning for multiple future time steps.

To summarize, Table 2 provides a comparative analysis of previous studies, highlighting their approaches to real value optimization, deadline-awareness, offloading planning, and resource provisioning over multiple time steps, and while many studies focus on individual aspects, our proposed work addresses all four points comprehensively. This makes our approach distinct in its ability to handle task prediction-based multi-step optimization of resource provisioning and task offloading, effectively overcoming the limitations in previous work.

The aforementioned studies commonly optimize for average user request rates under constant conditions. However, real-time fluctuations in request rates can lead to inefficient resource allocation (e.g., under- or over-utilization) at specific times, thereby decreasing system efficiency. Moreover, they often prioritize the immediate processing of requests within the current time step without considering deadline awareness, which is similar to a greedy algorithm and may not be feasible in some cases. This overlooks the overall dynamics of the queue and may result in longer processing times, particularly for complex tasks such as deep learning. Furthermore, effective resource provisioning or task offloading should differentiate treatment based on application deadline sensitivity, which varies between those that require timely completion and those with flexible or no strict time constraints. In this study, we propose a task prediction-based multi-step optimization of resource provisioning and task offloading that effectively overcomes the aforementioned limitations.

Table 2. Comparison of selected previous studies under four critical aspects in edge computing research.

Reference	Real Value-Based Optimization	Deadline Awareness	Multi-Step Offloading Planning	Multi-Steps Resource Provisioning
Kheraf et al. [16]				
Xiang et al. [17]	✓			
Jiang et al. [18]	✓			
Apostolopoulus et al. [19]				
Han et al. [20]		✓		
Song et al. [21]	✓	✓		
Feng et al. [22]				
Charatsaris et al. [23]	✓			
Wang et al. [24]		✓		
Qi et al. [25]	✓	✓		
Wang et al. [26]			✓	✓
Oza et al. [27]	✓	✓		
Guo et al. [28]	✓	✓		
Kim et al. [29]	✓			
Zeng et al. [30]	✓	✓	✓	
our work	✓	✓	✓	✓

3. Proposed Idea

In this section, we provide a description of the assumed system and the proposed approach in detail. The notations that frequently appear in this section are summarized in Table 3.

Table 3. Summary of the mathematical notations used in this study.

Notation	Definition
t	Index of time step whose length is Δ_t
$\mathbf{A}^{(t)}$	BS-user accessibility matrix at t
AC_b	Association capacity of BS b
$\mathbf{c}_{local}^{(t)}$	Normalized, available computing capacity of users at t
$\mathbf{c}_{bs}^{(t)}$	Normalized, available computing capacity of ESs at t
$\mathbf{c}_{cloud}$	Normalized, available computing capacity of cloud at t
$\mathbf{d}^{(t)}$	Deadlines of users' requests at t
$\mathbf{f}_{local}^{(t)}$	Portion of tasks to be processed locally at t
$\mathbf{F}_{es}^{(t)}$	Portion of tasks to be offloaded to ESs at t
$\mathbf{f}_{cloud}^{(t)}$	Portion of tasks to be offloaded to cloud at t
h_u	User u's task-offloading request history
$\mathbf{I}^{(t)}$	BS-user association indication matrix at t
l_{bs}	Scalar determining round-trip latency to the BS (ES), being $l_{bs} \times \Delta_t$
l_{cloud}	Scalar determining round-trip latency to cloud, being $l_{cloud} \times \Delta_t$
N_{bs}	Number of BSs
N_{es}	Number of ESs
N_{fh}	Forecasting horizon length
N_{lw}	Lookback window length
N_{sh}	Number of time steps over which an optimal resource scheduling is carried out
N_{user}	Number of users
$\mathbf{o}_{local}^{(t)}$	Binary task offloading decision vector for local processing at t
$\mathbf{O}_{es}^{(t)}$	Binary task offloading decision matrix between an ES and user at t
$\mathbf{o}_{cloud}^{(t)}$	Binary task offloading decision vector for cloud processing at t
$q_u^{(t)}$	Task-offloading requests of users at t

We assume a hierarchical networked system, as shown in Figure 1. A mobile user is connected to the access network via a BS, and each BS is connected to an ES with a

dedicated, high-speed link. The BS is connected to the Internet, as is the cloud data center. From the computing perspective, the assumed system is composed of three layers. At the lowest level is the local computing layer, enabling mobile users to process tasks directly on their devices. In the middle layer lies edge computing, facilitating task offloading to nearby edge servers with minimal network latency. Finally, the top layer is the cloud computing infrastructure, where abundant computing resources are available with a relatively large network latency. In this work, we assume that both mobile devices and edge servers are resource-limited, while cloud computing is not.

If a mobile user is near the edge of the base station (BS) with which it is currently associated, and begins to move away, a handover event will occur. This handover allows the user to connect to a different BS, referred to as $BS_{b'}$. Consequently, this may involve either migrating virtual resources from the edge server (ES) at BS_b or allocating new resources on the ES at $BS_{b'}$. However, this study does not account for the effects of user mobility. Given that the average walking speed of individuals ranges from 94.3 cm/s to 143.4 cm/s [31], and considering the random deployment of users and the coverage area of the BS, it is reasonable to assume that the accessibility matrix will remain nearly static over the period considered for resource and offloading optimization.

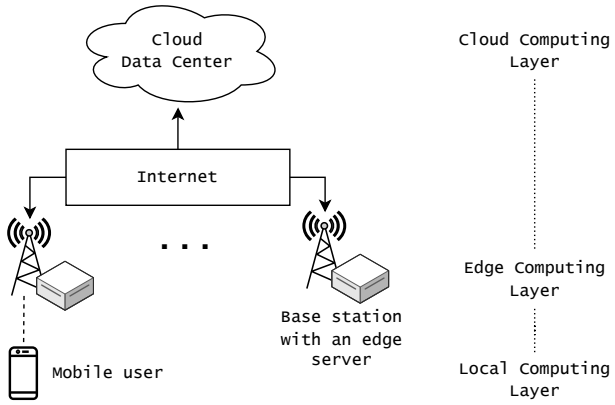


Figure 1. Illustration of the considered 3-layer computing system in this study where an edge server is co-located with a base station.

The overview of the proposed traffic-aware intelligent association and task offloading scheme is illustrated in Figure 2, which has both offline and online components. The offline operation includes training a prediction model given each user u's offloading request history (h_u). Then, by using the predicted future requests, the online logic iteratively performs the optimal scheduling subsystem that optimizes both resource provisioning and task offloading. In the figure, N_{lw} and N_{fh} refer to the lookback window and forecasting horizon size, determining the input and output length of the prediction model, respectively.

3.1. Multi-Step Task-Offloading Prediction

The development of the deep learning-based sequence-to-sequence prediction is a well-known problem [32,33]. The available approaches range from conventional statistical approaches, e.g., ARIMA (Auto Regressive Integrated Moving Average) and ETS (Error Trend and Seasonality, or exponential smoothing) [34], to state-of-the-art deep learning models, e.g., one-dimensional convolutional network (1DCNN), long short-term memory (LSTM) [35], and Transformer [36]. In this study, 1DCNN, LSTM, and Transformer are taken for predicting future task-offloading requests. The performance comparison among the three models are presented in Section 4.1.

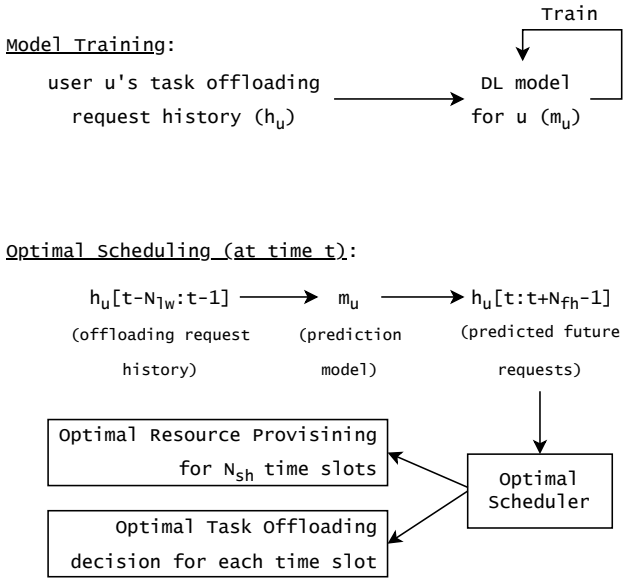


Figure 2. Overall workflow of the proposed idea where the model training and optimal scheduling is carried out offline and online, respectively.

The 1DCNN is particularly effective for processing and predicting time series data. The 1DCNN takes a sequence of time series data as input, and the convolutional layers therein apply convolution operations using filters (kernels) that slide over the input data to extract local patterns or features. In a 1DCNN, the filters move along one dimension (time). The major advantages of 1DCNN include the following three: (i) Feature extraction: The convolutional layers learn to automatically extract important features from the input time series data. Early layers capture simple patterns, while deeper layers capture more complex structures; (ii) Temporal dependencies: By applying convolutions across the time dimension, the model can capture temporal dependencies and correlations between different time points; (iii) Down-sampling: Pooling layers reduce the size of the data representation, helping to summarize the learned features and make the model more robust.

LSTM is a type of recurrent neural network (RNN) model designed to effectively learn temporal sequences and long-range dependencies by using three distinctive features, i.e., memory cells, forget/input/output gates, and recurrent architecture. The key features and advantages of LSTM include the following three: (i) Long-term dependency handling: LSTMs are designed to remember information over long periods of time, which is useful for tasks where context is important; (ii) Mitigation of the vanishing gradient problem: The design of LSTM gates helps in propagating gradients throughout the network, thus addressing the vanishing gradient problem that traditional RNNs face; (iii) Stateful processing: LSTMs can maintain a state across different sequences, which is valuable in applications like continuous speech recognition and video analysis.

The Transformer model was first introduced to handle tasks involving natural language processing, and is now widely used for analyzing and predicting time series datasets. Transformers bring various advantages, as listed below, from its features such as self-attention mechanisms, positional encoding, and encoder-decoder architecture: (i) Parallelization: Unlike RNN-type networks, Transformers allow for parallel processing of input sequences, leading to significantly faster training times; (ii) Handling Long-Range Dependencies: The self-attention mechanism allows the model to capture dependencies between tokens regardless of their distance in the sequence; (iii) Reduced Vanishing Gradient Problem: Since Transformers rely on attention mechanisms rather than sequential processing, they are less susceptible to the vanishing gradient problem that plagues RNNs and LSTMs.

3.2. Optimal Association and Task Offloading

In this work, we assume a time-slotted system where each equal-length time step is indexed by t. Given the user u's task-offloading request history, h_u, a deep learning model is trained to make precise predictions on future offloading requests. In contrast to most previous studies utilizing only the average request rates, we propose to make predictions on the future offloading requests for N_{fh} time steps. This is because the actual offloading request each time step can be different from the average. Thus, the optimal resource provisioning and offloading decisions made upon the average offloading rate may experience resource under-/over-provisioning, reducing resource utilization or QoS. Once the prediction model is trained, it is used to make predictions on future offloading requests, with which the optimal decision on resource provisioning and task offloading will be made. It is worth noting that in this study the optimal offloading decision is made each time step, while the optimal resource provisioning is made for the future N_{sh} time steps. The reason for making such multi-step provisioning decisions is that some procedures that are tightly related to resource provisioning, e.g., horizontal scaling, migration, and starting/stopping virtual machines, are time-consuming [37], and thus, it is impractical to carry out resource scheduling on every time step.

Upon generating the future offloading requests, the proposed resource provisioning and task offloading algorithms will operate periodically and on each time step, respectively, to process the users' offloading requests without violating the deadline constraints while evenly distributing the computing load of edge servers over time. The reasons for distributing the load are to make a certain amount of computing resources available on each edge server each time step so that future resource scheduling and task offloading can be optimized with a certain level of flexibility. To achieve an optimal load balancing, it is also important to optimize the association between BSs and users as well, since BS-user association leads to the association between the user and the ES which is co-located with the BS.

In this work, each user can offload their task to an ES and/or the cloud, in addition to processing tasks locally on their device. Such diverse access options for task offloading enhance flexibility in resource optimization but also increase the complexity of the optimization which will be effectively managed in what follows. This multi-access capability allows for more adaptable resource management, addressing varying load conditions and user demands.

To address these needs, the optimal resource provisioning shown in Figure 2 includes the following two components, i.e., user-BS association and computing resource provisioning. Once the decisions are made, the optimal task offloading will be carried out at each time step. Next, we present the formulated problem of optimal resource provisioning and task offloading as proposed. We first formulate a greedy optimization problem that schedules both resource provisioning and task offloading only for the current time step. Then, we extend the problem so that optimal decisions spanning multiple time steps can be made while guaranteeing the deadlines.

Let N_{bs}, N_{es}, and N_{user} be the number of BSs, ESs, and users in the system, respectively. In this work, we assume a cellular communication technology for its wide coverage [38], though others, such as WiFi, can be applied with little modification. In general, BSs periodically broadcast beacon signals to notify their presence [39]. Upon receiving the signal, a user can enumerate the accessible BSs. In this work, accessibility is guaranteed if the measured signal-to-interference plus noise ratio (SINR) exceeds the predefined threshold. By collecting the accessible BS lists from users, the central scheduler can configure an accessibility matrix $\mathbf{A}^{(t)} \in \{0,1\}^{N_{bs} \times N_{user}}$ at time t where the (b,u)-th element therein is denoted by $a_{bu}^{(t)}$. Here, b is the integer-valued index of BSs and ESs ranging from 1 to N_{bs} or N_{es}, and u is the integer-valued user index ranging from 1 to N_{user}.

A single scheduler is assumed responsible for configuring the accessibility matrix and solving the proposed optimization problem while having a single central entity can introduce a risk of failure affecting overall system reliability, we assumed that the sched-

uler operates within a cloud infrastructure equipped with high-availability clusters and redundancy features [40,41]. In this configuration, if the primary scheduler fails, a standby scheduler is available to assume its responsibilities without delay.

Let the decision variable $\mathbf{I}^{(t)} \in \{0,1\}^{N_{bs} \times N_{user}}$ be the association indication matrix. Each (b,u)-th element $i_{bu}^{(t)}$ in $\mathbf{I}^{(t)}$ takes a value of either 0 or 1, indicating that the association between BS b and user u is to be made or not, respectively. Then, the following constraint Equation (1) forces that an association can only be made if a user is accessible to a BS.

$$\forall b, u : i_{bu}^{(t)} \leq a_{bu}^{(t)} \tag{1}$$

In this work, a mobile device is assumed to have a single antenna, and thus, for a time step, a user can access only a single BS.

$$\forall u : \sum_{\forall b} i_{bu}^{(t)} = 1 \tag{2}$$

Ingeneral, at each time step, a BS b can associate with a predefined number, AC_b, of users due to its limited resource [38,39], where AC_b stands for the association capacity of a BS b.

$$\forall b : \sum_{\forall i} i_{bu}^{(t)} \leq AC_b \tag{3}$$

Once the BS-user association is made, the user can utilize the ES connected to the BS for task offloading. Let $\mathbf{o}_{local}^{(t)} \in \{0,1\}^{N_{user}}$ be the task offloading decision vector for local processing at time t, where $o_{local,u}^{(t)}$ being 1 or 0 indicates that the user u has decided to process (some) tasks locally or not, respectively. In the same manner, $\mathbf{O}_{es}^{(t)} \in \{0,1\}^{N_{es} \times N_{user}}$ is the task offloading decision matrix between the ES b and user u at time t. Since a BS b has a dedicated ES by assumption, the same index notation b is used to indicate the ES connected to the BS b. Furthermore, $\mathbf{o}_{cloud}^{(t)}$ is the task offloading decision vector to cloud at time t. Regardless of which BS a user is connected to at the moment, the user can choose to offload locally or to the cloud. However, offloading to a particular BS b is possible only when the user is associated with the BS b, which is formulated as a constraint as below.

$$\forall b, u : o_{es,bu}^{(t)} \leq i_{bu}^{(t)} \tag{4}$$

The decision variables $\mathbf{o}_{local}^{(t)}$, $\mathbf{O}_{es}^{(t)}$, and $\mathbf{o}_{cloud}^{(t)}$ indicate whether or not to offload tasks, and the specific amount of task to offload is determined by the following decision variables. Let $\mathbf{f}_{local}^{(t)} \in \mathcal{R}_{[0,1]}^{N_{users}}$ be the normalized decision variable determining the number of tasks to be offloaded locally. The subscript indicates that each $f_{local,u}^{(t)} \in \mathbf{f}_{local}^{(t)}$ takes a value from the range $[0,1]$. Similarly, $\mathbf{F}_{es}^{(t)} \in \mathcal{R}_{[0,1]}^{N_{es} \times N_{user}}$ is a matrix determining the offloading portion between ESs and users, whereas $\mathbf{f}_{cloud}^{(t)} \in \mathcal{R}_{+}^{N_{users}}$ is for offloading to cloud. Since decisions regarding the amount of tasks to be offloaded can only be made when the corresponding offloading is allowed, the following constraints are formulated. Here the scaling constant κ_{cloud} is multiplied to the right-hand side of Equation (7) so that the decision variable determining the actual amount of cloud offloading can take values without being limited to the $[0,1]$ range.

$$\forall u : f_{local,u}^{(t)} \leq o_{local,u}^{(t)} \tag{5}$$

$$\forall b, u : f_{es,bu}^{(t)} \leq o_{es,bu}^{(t)} \tag{6}$$

$$\forall u : f_{cloud,u}^{(t)} \leq \kappa_{cloud} \times o_{cloud,u}^{(t)} \tag{7}$$

The amount of requests to be offloaded is also limited by the available computing capacity. Let $\mathbf{c}_{local}^{(t)} \in \mathcal{R}_{[0,1]}^{N_{user}}$ be the normalized, available local computing capacity of users at time t, where the subscript indicates that each $c_{local,u}^{(t)} \in \mathbf{c}_{local}^{(t)}$ takes a value from the range $[0, 1]$. The superscript t does not indicate that the physical capacity changes over time. However, if a mobile device has some tasks to process at the moment, its computing capacity should be reduced, which necessitates the time step notation. In the same manner, the computing capacity of ESs and CC is defined as $\mathbf{c}_{es}^{(t)} \in \mathcal{R}_{[0,1]}^{N_{es}}$ and $\mathbf{c}_{cloud} \in \mathcal{R}_{+}^{N_{user}}$, respectively. The CC resource is large enough by assumption, and thus, the notation t is omitted from the CC capacity. As a result, the following constraints are formulated to limit the amount of offloading to the available capacity.

$$\forall u : f_{local,u}^{(t)} \leq c_{local,u}^{(t)} \tag{8}$$

$$\forall b : \sum_{\forall u} f_{es,bu}^{(t)} \leq c_{es,b}^{(t)} \tag{9}$$

The resource provisioning and task offloading scheduling problems should maximize the QoS by fulfilling the user's offloading demand. Let a vector $\mathbf{q}^{(t)} \in \mathcal{R}_{+}^{N_{user}}$ be the offloading request of users, where the subscript $+$ indicates that each element $q_u^{(t)} \in \mathbf{q}^{(t)}$ is non-negative. To guarantee the completion of task offloading for each user, the total amount of tasks offloaded to the local device, ES, and cloud should be equal to the request. As a result, the following constraint is formulated as follows:

$$\forall b : f_{local,u}^{(t)} + \sum_{\forall b} f_{es,bu}^{(t)} + f_{cloud,u}^{(t)} = q_u^{(t)} \tag{10}$$

Finally, the objective function is formulated to minimize the response time for the user's request. In this problem formulation, each task is assumed to be completely processed within the current time step. Thus, the factor impacting the response time is which computing resource the user has utilized. It is assumed in this work that the network latency for local processing is zero and the latency for accessing a BS is much smaller than that for accessing the cloud. Let l_{bs} and l_{cloud} be the integer scalar determining the round-trip network latency to the BS/ES (i.e., $l_{bs} \times \Delta_t$) and cloud (i.e., $l_{cloud} \times \Delta_t$), respectively. Then, the response time $L_u^{(t)}$ that user u experiences for task offloading can be written as below:

$$\forall u : L_u^{(t)} = \Delta_t \times l_{bs} \sum_{\forall b} o_{es,bu}^{(t)} + l_{cloud} \times \Delta_t \times o_{cloud,u}^{(t)} + \Delta_t, \tag{11}$$

where the first term is the latency for accessing the BS (ES), the second term is the latency for accessing the cloud, and the last term is the time spent on the current time step. Finally, the objective function is formulated as below using Equation (11). The main objective here is to minimize the total response time, and in addition to that, the second term in the objective function minimizes the use of local processing, as shown below.

$$\min . \ (1 - \alpha) \sum_{\forall u} L_u^{(t)} + \alpha \times \sum_{\forall u} f_{local,u}^{(t)} \tag{12}$$

The $\alpha \in [0, 1]$ represents the service provider's preference to prolonging the lifetime of mobile users by minimizing the amount of local processing. As expected when $\alpha = 0$, the local processing on the user devices will be carried out as much as possible to minimize the response time. On the other hand, as α increases, the use of local processing will be minimized and the use of ESs will increase. Still, the offloading to the cloud will not be heavily exercised for causing high response time.

Although the above optimization problem, called OPT(1), can produce an effective scheduling solution for the current time step, it has the following drawbacks. In general,

the length of a time step Δ_t is short, and resource provisioning may consume computing resources and take time, as discussed earlier. Thus, carrying out resource provisioning every time step may not be practical, especially for real-time scheduling. Thus, in this work, we propose to perform resource provisioning for $N_{sh} > 1$ time steps, within which both the network association and computing resource allocation decisions remain the same. OPT(1) optimizes only a single time step at a time and thus, what it can do as to the deadline constraint is process as many tasks as possible at the current time step, operating as a greedy solution. In the following multi-time step approach, we propose an optimization formulation that guarantees the deadline requirements of tasks by scheduling over multiple time steps. In addition, we propose to distribute the offloading requests over multiple time steps so that future resource provisioning and task offloading can be made with a certain level of flexibility. Furthermore, it helps to effectively adapt to the case of misprediction on future requests.

It is assumed in this work that on each time step, a user can generate up to one offloading request. However, this can be extended to various scenarios where a user generates multiple requests. Since the amount of the offloading request can be any arbitrary number, the scenario where a user generates multiple requests can be treated as a single request with the request amount being the sum of all simultaneous requests. Let $\mathbf{d}^{(t)}$ be the deadlines of the tasks generated by users at t. The deadline is application-specific and that of each request is assumed to be known when the request arrives. Each value is represented by a strongly positive integer if the user has generated a task, indicating the number of time steps from t the deadline is not violated. After collecting the deadlines, the scheduler computes the largest deadline which becomes the number of time steps to optimize the computing resource and the offloading decision. Let $\sigma^{(t)}$ be the largest deadline among the offloading requests to be scheduled together at t, which is defined as follows.

$$\sigma^{(t)} = \max\{d_u^{(t)} : \forall u\} \tag{13}$$

It is worth noting that the value of $\sigma^{(t)}$ is an integer value and thus the actual timespan of it can be derived, if needed, by multiplying the length of each time step, Δ_t.

The following optimization problem addresses resource provisioning and task offloading over $\sigma^{(t)} = N_{sh}$ time steps for offloading requests generated at time t. The entire time horizon is indexed by $t = \tau_1, \tau_2, \cdots, \tau_{\sigma^{(t)}}$. The length of each time step is determined by system-specific parameters. In this study, the duration of each time step is set to 1 ms. However, the proposed method is designed to be flexible and can be adapted to accommodate time step lengths of varying sizes.

The optimization model, OPT(N_{sh}), aims to minimize a weighted sum of three components: the overall system cost, local device processing cost, and cloud processing cost. The weights α and β control the relative importance of local and cloud processing costs. This allows for flexible adjustment based on system requirements. The deadline-constrained OPT(N_{sh}), multiple-time step resource provisioning and task offloading problem is formulated as follows:

$$\min. \ (1-\alpha-\beta)\phi + \alpha \times \sum_{\forall t}\sum_{\forall u} f_{local,u}^{(t)} + \beta \times \sum_{\forall t}\sum_{\forall u} f_{cloud,u}^{(t)} \tag{14a}$$

$$\text{s.t.} \quad \forall b,u : i_{bu}^{(\tau_1)} \le a_{bu}^{(\tau_1)} \tag{14b}$$

$$\forall u : \sum_{\forall b} i_{bu}^{(\tau_1)} = 1 \tag{14c}$$

$$\forall b : \sum_{\forall i} i_{bu}^{(\tau_1)} \le AC_b \tag{14d}$$

$$\forall b,u,t : o_{es,bu}^{(t)} \le i_{bu}^{(\tau_1)} \tag{14e}$$

$$\forall u,t : f_{local,u}^{(t)} \le o_{local,u}^{(t)} \tag{14f}$$

$$\forall b, u, t : f_{es,bu}^{(t)} \leq o_{es,bu}^{(t)} \tag{14g}$$

$$\forall u, t : f_{cloud,u}^{(t)} \leq \kappa_{cloud} \times o_{cloud,u}^{(t)} \tag{14h}$$

$$\forall u, t : f_{local,u}^{(t)} \leq c_{local,u}^{(t)} \tag{14i}$$

$$\forall b, t : \sum_{\forall u} f_{es,bu}^{(t)} \leq c_{es,b}^{(t)} \tag{14j}$$

$$\forall b, t : \sum_{t} (f_{local,u}^{(t)} + \sum_{\forall b} f_{es,bu}^{(t)} + f_{cloud,u}^{(t)}) = q_u^{(\tau_1)} \tag{14k}$$

$$\forall u : \sum_{t=d_u^{(\tau_1)}+1}^{\tau_{\sigma^{(t)}}} o_{local,u}^{(t)} \leq 0 \tag{14l}$$

$$\forall u : \sum_{t=d_u^{(\tau_1)}+1-l_{bs}}^{\tau_{\sigma^{(t)}}} \sum_{\forall b} o_{bs,bu}^{(t)} \leq 0 \tag{14m}$$

$$\forall u : \sum_{t=d_u^{(\tau_1)}+1-l_{cloud}}^{\tau_{\sigma^{(t)}}} o_{cloud,u}^{(t)} \leq 0 \tag{14n}$$

$$\forall t, b : \sum_{\forall u} f_{es,bu}^{(t)} \leq \phi \tag{14o}$$

$$\forall b, u, t : o_{local,u}^{(t-1)} \geq o_{local,u}^{(t)}, o_{es,bu}^{(t-1)} \geq o_{es,bu}^{(t)}, o_{cloud,u}^{(t-1)} \geq o_{cloud,u}^{(t)} \tag{14p}$$

The OPT(N_{sh}) is similar to OPT(1) except for the following major differences: (i) OPT(N_{sh}) performs optimization over multiple time steps, while OPT(1) does only for a single time step; (ii) OPT(N_{sh}) guarantees the deadline of each task, while OPT(1) minimizes the response time only; (iii) OPT(N_{sh}) distributes the computing load over multiple time steps, while OPT(1) cannot.

The constraints in Equations (14b)–(14k) for OPT(N_{sh}) are similar to those in OPT(1), but they differ in that they apply across multiple time steps rather than a single time step. Constraints in Equations (14b)–(14d) ensure that each user is associated with exactly one BS and that no BS exceeds its capacity. This requirement is extended to cover all time steps in OPT(N_{sh}). On the other hand, constraints in Equations (14e)–(14k) handle offloading decisions by ensuring that offloading is only permitted if a user is associated with a BS, that local device and edge server processing limits are not exceeded, and that user task requirements are satisfied. These constraints are applied over multiple time steps, accommodating the dynamic nature of the system.

Key differences in OPT(N_{sh}) are introduced with constraints Equations (14l)–(14n), which enforce deadlines. Constraint Equation (14l) prevents offloading if a task's deadline has passed, thus avoiding offloading beyond the deadline. Constraint Equation (14m) applies this rule to edge server offloading, while constraint Equation (14n) applies it to cloud offloading. Finally, constraint Equation (14o) and the term ϕ in the objective function Equation (14a) are used to balance computing load over multiple time steps. Constraint Equation (14o) ensures that the total load on edge servers does not exceed ϕ at any time step, while the term ϕ in the objective function penalizes uneven load distribution, encouraging balanced resource allocation.

The objective function also raises some penalties for the use of local resources and the cloud for prolonging the lifetime of mobile devices and reducing network latency, respectively. The objective function has two preference parameters, $\alpha \in [0, 1]$ and $\beta \in [0, 1]$, highlighting the importance of reducing local processing and cloud offloading, respectively.

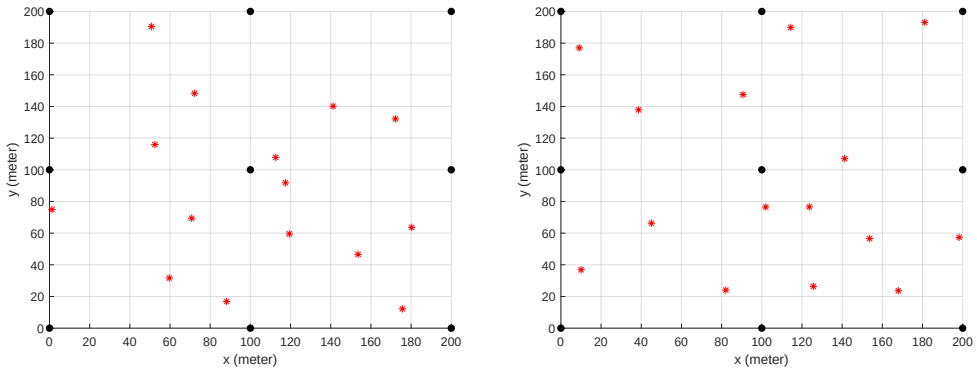
It is assumed in this work that users are stationary for a small number of time steps, thereby causing no changes in the accessibility matrix, $\mathbf{A}^{(t)}$, over $\sigma^{(t)}$ time steps. In addition, as mentioned earlier, frequent rescheduling of resources (e.g., association, horizontal scaling, and migration) causes computational overhead and delays. Thus, we propose to make

optimal associations and resource provisioning for multiple time steps, which is possible by making accurate predictions on future requests. As in Equations (14b)–(14e), the optimal decision on BS-user association remains the same over $\sigma^{(t)}$ time steps. Furthermore, if offloading is to happen, the related resources should be allocated in advance. For example, a containerized virtual environment should be created and started in advance. The constraint Equation (14p) states that if offloading is to be enabled at some t, it should be enabled in advance. On the other hand, the actual allocated computing portion determined by $\mathbf{f}_{local}^{(t)}$, $\mathbf{F}_{es}^{(t)}$, and $\mathbf{f}_{cloud}^{(t)}$ changes over time steps. In the widely used Docker platform [42], for example, the change in CPU/MEM resources (i.e., vertical scaling) can be performed in real time by using Docker's built-in runtime options [43], enabling frequent scheduling.

It is worth noting that in both OPT(1) and OPT(N_{sh}), we intentionally omitted the constraints for limiting the feasible value ranges of the decision variables since they are already stated in the manuscript. Additionally, the greedy version of the proposed approach, OPT(1), processes all tasks within the current time step without considering deadline constraints. In contrast, OPT(N_{sh}) optimizes resource allocation and task offloading decisions over multiple time steps while ensuring that deadlines are consistently met by incorporating these constraints.

4. Evaluation

For evaluation, we have configured networks as shown in Figure 3. On a 300 m-by-300 m area, there are nine BSs and ESs located in a planned manner with 100 m spacing vertically and horizontally, while mobile users are randomly placed. The transmit/receive range is set to 150 m. Each mobile device has a computing budget that is randomly drawn from Uniform [0.05, 0.20], while the available computing resources at each ES are randomly drawn from Uniform [0.40, 0.80]. Each user generates an offloading request, which requires computing resources drawn from Distribution [0.00, 1.00] $\times\eta$, where $\eta \in \mathcal{R}_{++}$ is a scaling factor to make low or high request rate scenarios. During the evaluation, the *Distribution* is configured to one of the following: sine-, triangle-, and sawtooth-shaped patterns or uniform random. Regarding the network latency, l_{bs} and l_{cloud} are set to 1 and 2, resulting in $1 \times \Delta_t$ and $2 \times \Delta_t$, respectively. The simulation and evaluation are carried out on a high-performance workstation with an Intel Core i9 10940X CPU, 128 GB of memory, and NVIDIA GeForce RTX 3090 graphics card, and the reported values in this section are the average of 10 evaluations.



(**a**) (**b**)

Figure 3. Some randomly sampled layouts of the assumed 300 m-by-300 m area where the nine red stars and fifteen black dots are the locations of users and BSs, respectively. (**a**) Example layout 1. (**b**) Example layout 2.

4.1. Traffic Offloading Dataset and Prediction Model

Task offloading patterns (or traffic patterns) we assume in this study include sine-, triangle-, and sawtooth-shaped patterns and a random pattern as shown in Figure 4. The

values are generated at 1000 Hz for $T = 50$ s, yielding 50,000 data samples. We used an 80:10:10 split for training, validation, and testing. As a pre-processing step, the dataset is normalized. The input/output length of the model is configured to $l_{lb} = 128$ and $l_{fh} = 5$. The deep learning models are implemented with Tensorflow and Keras framework. The Adam optimizer is used with a learning rate of 0.001, and the number of epochs and the batch size are configured to 10 and 128, respectively.

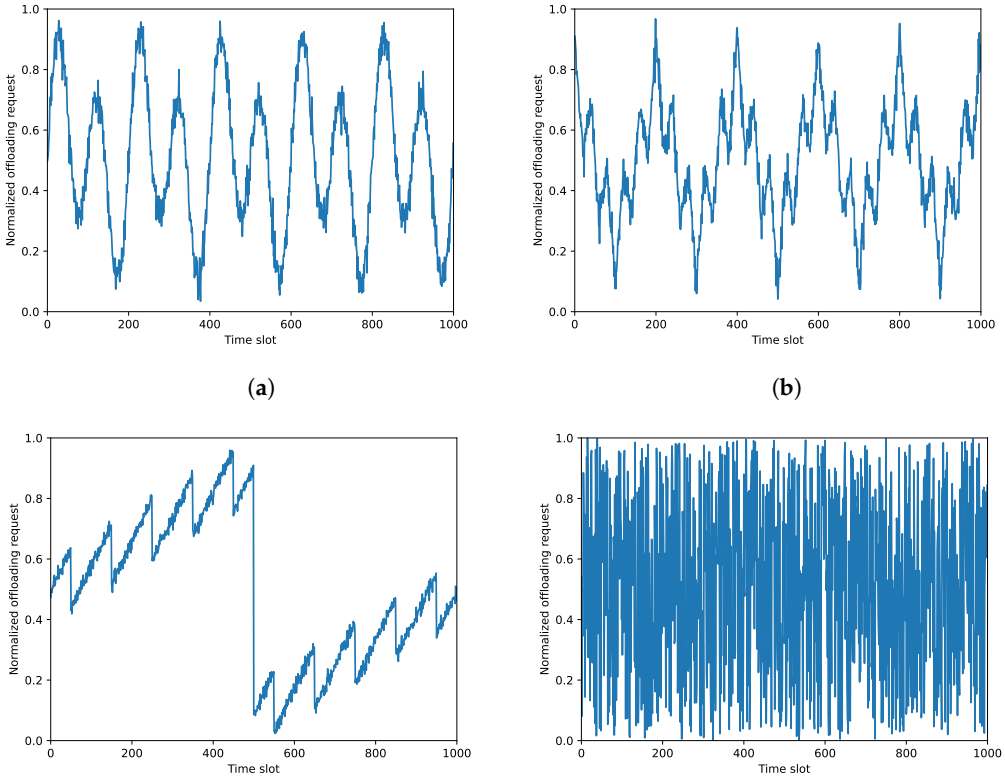


Figure 4. The task-offloading request patterns assumed in this study where the x-axis shows only up to the first 1000 samples out of 50,000. (**a**) Sine pattern. (**b**) Triangle pattern. (**c**) Sawtooth pattern. (**d**) Random.

Real patterns can often be decomposed into multiple primitive patterns. This implies that the sum of one, two, or three basic patterns can result in a realistic pattern. Therefore, by studying these fundamental shapes, we can gain insights into more complex and realistic traffic patterns. According to a study analyzing mobile traffic using a real dataset collected from over 150,000 users [44], such decomposition is a valid approach for capturing the variability in real-world traffic patterns. This view is further supported by techniques such as Fourier analysis and other effective methods, which can decompose compound signals into multiple basic, primitive signals [45–48]. In this study, we demonstrate that the proposed prediction model can accurately forecast these compound signals using the generated composite signals. Furthermore, if the given offloading pattern exhibits high complexity, it can be decomposed into simpler, more manageable components.

The measured accuracy of the considered neural network models is reported in Table 4, where the reported values therein are the root mean square errors (RMSE). Overall, the 1DCNN model outperformed the rest with respect to mean accuracy, and thus, it is used for evaluation as the offloading request prediction model in what follows.

Table 4. Prediction accuracy performance (RMSE) of the three neural networks with respect to different request patterns.

Request Pattern	1DCNN	LSTM	Transformer
Sine	0.0350	0.0337	0.0350
Triangle	0.0336	0.0337	0.0347
Sawtooth	0.0328	0.0575	0.0349
Random	0.2874	0.2869	0.2868
mean (total)	0.0972	0.1030	0.0979
mean (except Random)	0.0338	0.0416	0.0349

The predicted results of 1DCNN are shown in Figure 5, where Figure 5a is for sine-shaped requests, Figure 5b is for triangle-shaped requests, Figure 5c is for sawtooth-shaped requests, and Figure 5d is for randomly generated requests. As can be seen from the figures, requests exhibiting a certain pattern are predicted with high precision. However, in the case of the randomly generated requests, it is impossible to learn the underlying patterns that do not exist, and thus, the model outputs the mean values all the time.

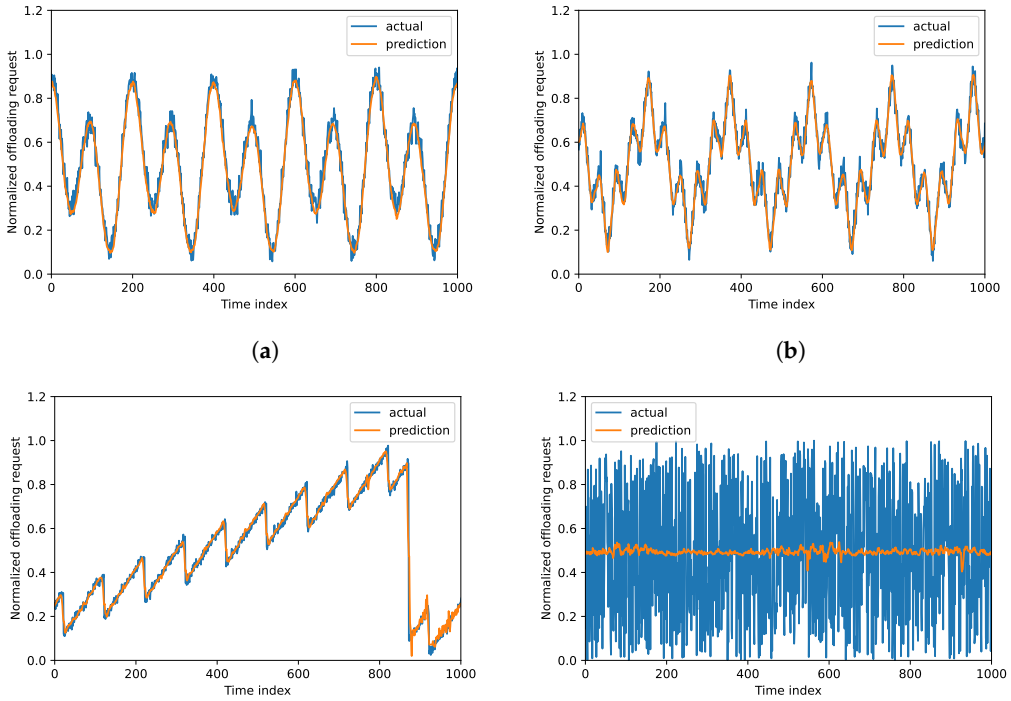


Figure 5. Task-offloading prediction results with 1DCNN for the four different shapes or patterns. (**a**) Sine pattern. (**b**) Triangle pattern. (**c**) Sawtooth pattern. (**d**) Random.

The predicted requests are then used in the following subsection for optimizing resource provisioning and task offloading. By carrying out extensive evaluations, we found that the results from using sine-, triangle-, and sawtooth-shaped requests are very similar to each other for making highly accurate predictions. On the other hand, the evaluations with random tasks showed different outcomes. Thus, in the following sections, we report the cases with the sine-, triangle-, and sawtooth-shaped requests as a single result after taking the average of all three. In addition, such requests will be called *pattern requests*.

4.2. Multi-Step Optimal Resource Provisioning and Task Offloading

We implemented the proposed algorithm and a discrete-time simulation environment on MATLAB R2020b [49]. To implement and solve the proposed optimization problem,

we used CVX 2.1 [50] and Gurobi 11.0 [51]. To evaluate the performance of the proposed approach and to make a comparison analysis, we implemented the following algorithms:

- OPT(N_{sh}): the optimal method OPT(N_{sh}) proposed in this paper, which optimizes resources and offloads over multiple time steps with deadline constraints.
- OPT(N_{sh})/SSF: the proposed OPT(N_{sh}) without optimal BS-user association; the conventional strongest signal first (SSF) association rule [52] is used instead.
- OPT(1): The optimal method OPT(1) proposed in this paper, which optimizes resources and offloading for the current time step only in a greedy manner.

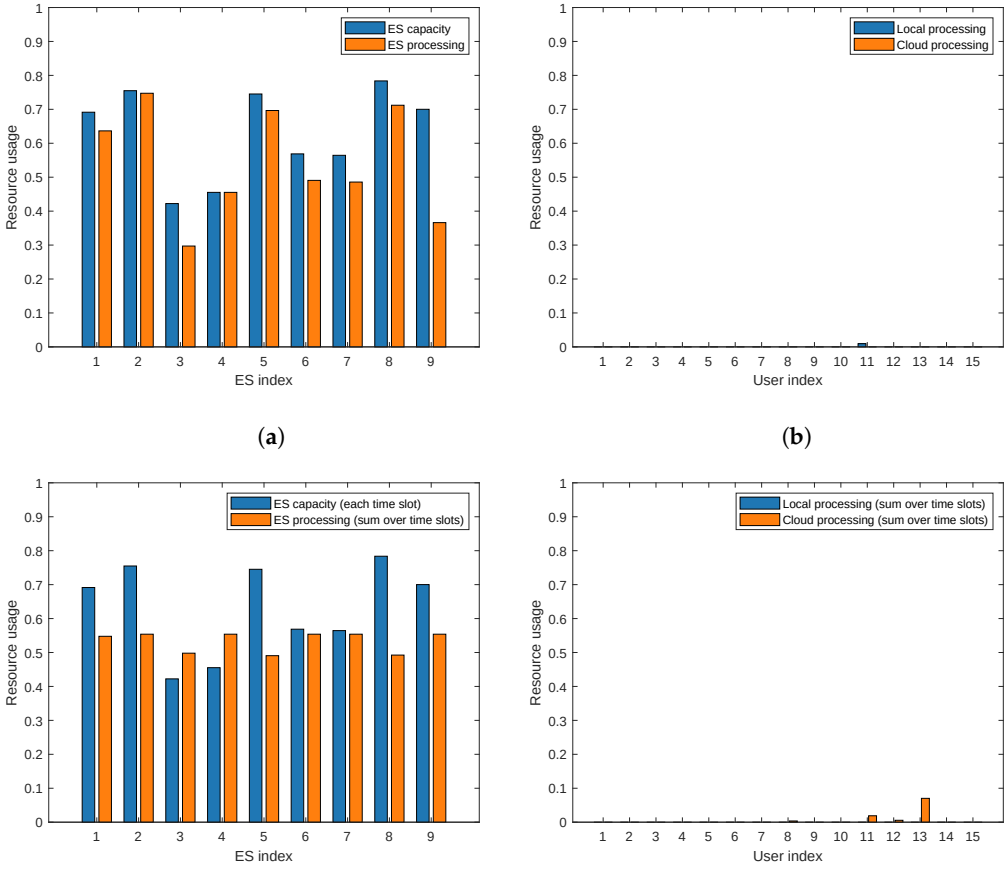
It is important to note that the greedy version of the proposed approach, OPT(1), reflects the methodologies discussed in previous studies. However, unlike earlier methods, OPT(1) includes advanced features such as optimization of BS-user associations and minimization of delays. Additionally, by comparing OPT(1) with OPT(N_{sh})/SSF, we can highlight the benefits of BS-user association optimization, which is also an important aspect of the proposed approach.

The design parameters or preference values are set as follows, and the most suitable values are found empirically. To be specific, for OPT(1), the value of α is set to 0.1. On the other hand, for both OPT(N_{sh}) and OPT(N_{sh})/SSF, the following configurations for the two parameters are used: $\alpha = 0.2$ and $\beta = 0.15$.

4.2.1. Effects of Multi-Time Step Optimization: Comparison between OPT(N_{sh}) and OPT(1)

We first carry out an evaluation to show the advantages of OPT(N_{sh}) over OPT(1). That is, the benefits of optimizing the resources over multiple time steps will be evaluated. For this evaluation, we assume that both OPT(1) and OPT(N_{sh}) can perfectly predict the future requests. The reported values are the average over the three different traffic patterns and the random traffic. During evaluations, OPT(N_{sh}) has successfully satisfied the deadline constraints, which is the case in the following evaluations as well. The following Figures 6 and 7 depict the amount of computing resource used for task offloading when the users' offloading requests are small and large, respectively. In the small request rate scenario (see Figure 6), the overall computing resource usage is quite similar to each other in that most computations are carried out on ESs. The notable difference is that OPT(1) offloaded tasks as much as possible to an ES to minimize the response time and to minimize the local processing according to its objective function Equation (12). On the other hand, OPT(N_{sh}) optimizes over multiple time steps, and thus, excessive use of ESs does not occur. Rather, OPT(N_{sh}) evenly distributes the computing resource allocations on ESs, and a small portion of the tasks are offloaded to the cloud. This is because, as long as the deadline is not violated, the use of the cloud is not penalized much. The fair distribution of the load is desired in this study. It is worth noting that for OPT(N_{sh}), the use of ESs can exceed the ES's capacity as shown in Figure 6c. This is because the ES capacity in the figure is the capacity for each time step, while the resource use is the accumulation over multiple time steps.

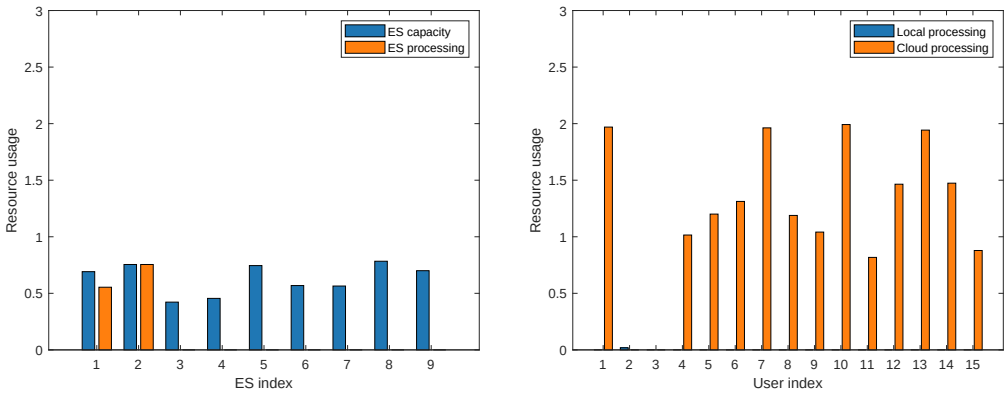
In the large request rate scenario (see Figure 7), the two methods show quite different results. The OPT(1) takes an extreme approach in that each user offloads their tasks completely to either the cloud or ES, and most tasks are offloaded to cloud for not having enough capacity in an ES to process all within a single time step. As a result, users with relatively small requests offload their tasks to ESs completely. On the other hand, users with large amounts of requests completely offload their traffic to the cloud. However, OPT(N_{sh}) can utilize ESs over multiple time steps, and thus, it has decided to offload much of the user's tasks to ESs while maintaining the balance of the ES resources. To be specific, the use of ESs in OPT(1) and OPT(N_{sh}) has been 1.31 and 16.98, respectively. On the other hand, the use of local and cloud computing in OPT(1) are 0.02 and 18.26, respectively, while those of OPT(N_{sh}) are 0.00 and 0.62, respectively. This demonstrates that optimizing across multiple time steps can effectively distribute the workload over time, thereby minimizing reliance on both local and cloud resources.



(**a**) (**b**)

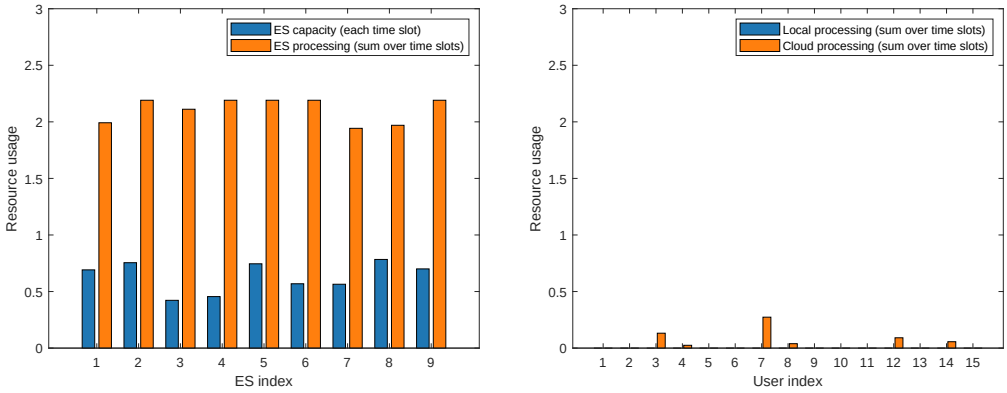
(**c**) (**d**)

Figure 6. Comparison of computing resource usage between OPT(1) and OPT(N_{sh}) under a low request rate ($\eta = 0.5$). Subfigures (**a**,**c**) show edge server (ES) usage, while subfigures (**b**,**d**) display the combined usage of local devices and cloud resources. (**a**) ES usage of OPT(1). (**b**) Local device and cloud usage of OPT(1). (**c**) ES usage of OPT(N_{sh}). (**d**) Local device and cloud usage of OPT(N_{sh}).



(**a**) (**b**)

Figure 7. *Cont.*



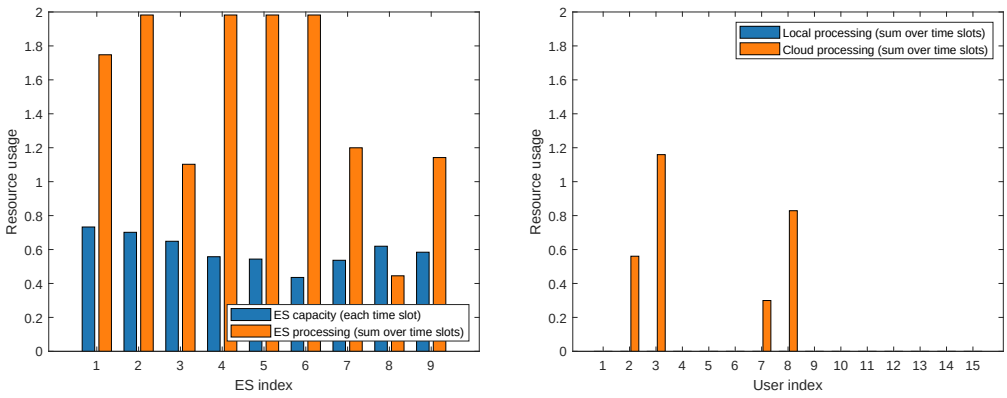
(**c**) (**d**)

Figure 7. Comparison of computing resource usage for OPT(1) and OPT(N_{sh}) at a high request rate ($\eta = 2.0$). Subfigures (**a**,**c**) show edge server usage, while (**b**,**d**) display local device and cloud resource usage. (**a**) ES usage of OPT(1). (**b**) Local device and cloud usage of OPT(1). (**c**) ES usage of OPT(N_{sh}). (**d**) Local device and cloud usage of OPT(N_{sh}).

4.2.2. Effects of Optimal Association: Comparison between OPT(N_{sh}) and OPT(N_{sh})/SSF

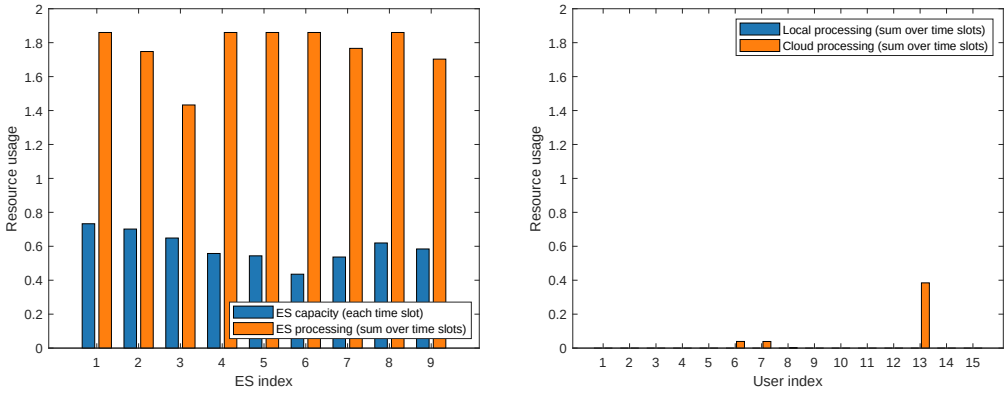
To validate the importance and effectiveness of the proposed optimal association approach, we make a comparison between OPT(N_{sh}) and one variant of it, called OPT(1)/SSF. As similar to the previous subsection, we assumed the case where the future request rates can be perfectly predicted in this evaluation. It is the same as OPT(N_{sh}) except that the BS-user association is made based on the strongest signal first rule. For SSH, we assume a free space path loss model [53] where the distance between the transceiver and receiver dominates the path loss: $P_L dB = -10 \times log_{10} G_l \lambda^2 / (4\pi d)^2$ with d being the distance.

When the mean request rates are small (i.e., $\eta = 0.5$), there was not much difference between OPT(N_{sh}) and OPT(1)/SSF. However, when the mean rate increased (i.e., $\eta = 0.5$), OPT(1)/SSF is much outperformed by OPT(N_{sh}) as shown in Figure 8. Despite the optimal computing resource provisioning and task offloading, OPT(1)/SSF has failed to make balanced use of ES computing resources, as shown in Figure 8a. On the other hand, OPT(1) has achieved better load balancing among ESs, as shown in Figure 8c. In addition, compared to OPT(1)/SSF which has utilized the cloud resource much (see Figure 8b), OPT(1) has used much less (see Figure 8d). Such performance difference arises from the optimal association strategy. The BS-user association determines the ES-user association as well. Thus, poor association may result in the case where many users associate with a small number of BSs, which prevents fair load distribution.



(**a**) (**b**)

Figure 8. *Cont.*

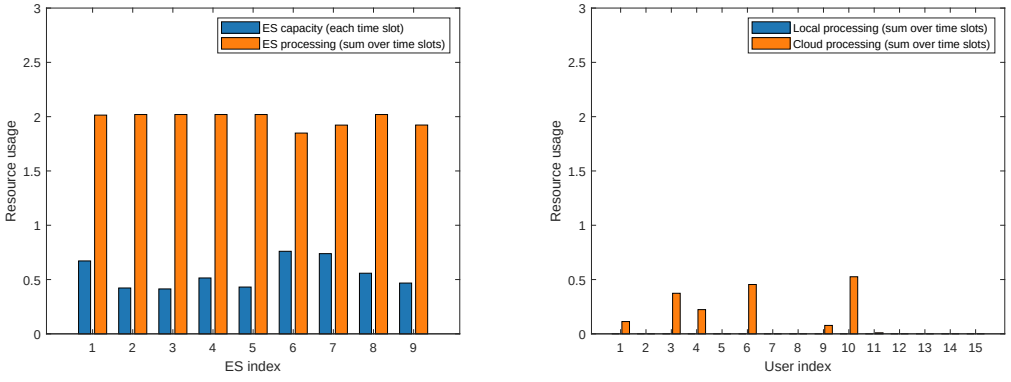


(**c**) (**d**)

Figure 8. Computing resource usage comparison between OPT(N_{sh}) and OPT(N_{sh})/SSF with a large request rate ($\eta = 2.0$). (**a**) ES usage of OPT(N_{sh})/SSH. (**b**) Local device and cloud usage of OPT(N_{sh})/SSH. (**c**) ES usage of OPT(N_{sh}). (**d**) Local device and cloud usage of OPT(N_{sh}).

4.2.3. Effects of Request Prediction Accuracy on OPT(N_{sh})

Lastly, we conduct evaluations to examine how errors in predicting future offloading requests impact resource utilization. As aforementioned, the proposed 1DCNN-based prediction model can make highly accurate predictions for the three patterns, and thus, the resource usage outcomes are similar to each other. In this regard, we configured two scenarios in the following manner. In one scenario, during the simulation, each user generates requests by using one of the sine-, triangle-, and sawtooth-shaped patterns. In the other scenario, all users will generate requests randomly. The difference between the two is the level of precision in predicting future requests. As a result, we configured the three scenarios as follows: (i) users' requests can be perfectly predicted; (ii) users' requests can be predicted with high accuracy (i.e., each user is generating requests by using sine-, triangle-, and sawtooth-shaped pattern); (iii) users' requests cannot be predicted accurately (i.e., each user is generating requests at uniform random). When the users' request rates are small with $\eta = 0.5$, the results among the above three cases are not much different, while the more accurate scenario has yielded better performance in terms of the fair distribution among ESs and less use of local and cloud computing resource. Thus, we present only the results when the users' request rates are large with $\eta = 0.5$, as shown in Figure 9.



(**a**) (**b**)

Figure 9. *Cont.*

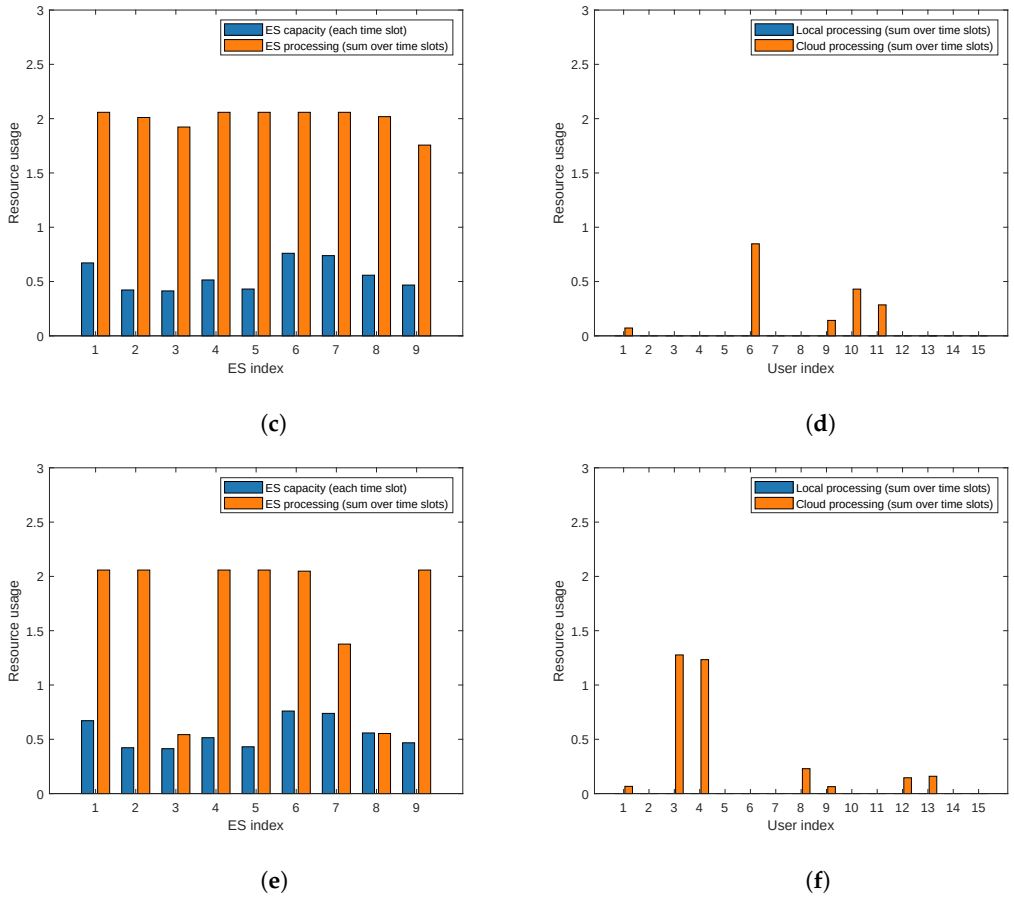


Figure 9. Computing resource usage comparison of OPT(N_{sh}) between the cases with different levels of prediction accuracy with a large request rate ($\eta = 2.0$). (**a**) ES usage of OPT(N_{sh}) when the request prediction is perfect. (**b**) Local device and cloud usage of OPT(N_{sh}) when the request prediction is perfect. (**c**) ES usage of OPT(N_{sh}) when the request prediction is highly accurate. (**d**) Local device and cloud usage of OPT(N_{sh}) when the request prediction is highly accurate. (**e**) ES usage of OPT(N_{sh}) when the request prediction is inaccurate. (**f**) Local device and cloud usage of OPT(N_{sh}) when the request prediction is inaccurate.

When the future task requests can be predicted with high accuracy, the resource use in ESs and user/cloud (see Figure 9c and Figure 9d, respectively) is quite similar to that of optimum (see Figure 9a and Figure 9b, respectively). Such little differences is mainly because the future requests have been predicted with high accuracy. In addition, OPT(N_{sh}) makes some resources available by fairly distributing the load over time. Thus, in the case of under-provisioning, it can allocate additional computing resources on ESs without accessing cloud or local device. As a result, the use of ESs can still be balanced, and the use of cloud can be minimized. Yet, in scenarios where all users generate offloading requests uniformly at random, accurately predicting future requests becomes impossible. Thus, OPT(N_{sh}) suffers much from under-/over-provisioning problems. Still, OPT(N_{sh}) distributes the load on ESs over time, but the level of load balancing is worse than the rest. In addition, the use of the cloud has increased much, which is not preferred under the current objective function (see Equation (14a)). This result shows the importance of highly accurate predictions on the future request. However, random traffic cannot be captured, and the proposed approach has shown that under the random scenario, it has a certain level of load balancing and the use of cloud has been successfully controlled.

We have also carried out additional experiments to evaluate the effect of misprediction on the performance of OPT(1) and OPT(N_{sh})/SSF, whereas the performance of the two with perfect prediction is presented in Section 4.2.1 and Section 4.2.2, respectively. In the case of

OPT(1) with a low request rate ($\eta = 0.5$), when the prediction is highly accurate, the use of local, ES, and cloud computing has been similar to the case when the prediction is perfect. When the user request is generated uniformly at random, on the other hand, the use of computing resources has been less efficient. The most notable difference is, compared to the perfect prediction case, the use of local computing resources increased from 0.0095 to 0.1010, on average. This is because most users have closed their access to the cloud during the optimization, which considers the average rate only. This is clearly against the goal we programmed in the objective function since it will shorten the battery lifetime of the user device.

In the case of OPT(1) with a high request rate ($\eta = 2.0$), we have observed the following. Similar to the case with low request rates, when the prediction is highly accurate, the use of local, ESs, and cloud computing has been similar to that of the perfect prediction case. When the user request is generated uniformly at random, the offloading scheduling has become infeasible frequently. As can be seen in Figure 7b, both users 2 and 3 do not allocate resources on the cloud. It might be the desired outcome under the average request rates. However, in real time, the offloading request can be much larger than the average, but the closed access to the cloud for both users causes failure in fulfilling the user's request.

Lastly, in the case of OPT(N_{sh})/SSF with a high request rate ($\eta = 2.0$), we have learned the following. Similar to the above cases, when the prediction is highly accurate, the use of local, ESs, and cloud computing has been similar to that of the perfect prediction case. When the user request is generated uniformly at random, the use of cloud resources is increased from 2.8482 to 3.7279, on average. Due to the non-optimal approach taken for user-BS association, OPT(N_{sh})/SSF can result in a few BSs with a large number of associations. When the real-time requests exceed the average for the users associated with such busy BSs, part of the excessive amount of requests are forwarded to the cloud, resulting in a longer response time.

4.2.4. Discussions

Enhanced resource utilization and reduced task offloading latency significantly improve service quality for applications such as real-time streaming and gaming. Efficient resource usage reduces costs for providers, particularly in large-scale deployments, while improved load distribution enhances network scalability, allowing for the accommodation of more users and devices. However, average-based optimization has limitations, as real-time offloading request amounts can deviate from the average. Consequently, this work employs real-value-based optimization to better address these variations, making it more suitable for delay-constrained or sensitive applications.

However, some limitations exist. The framework's effectiveness relies on the accuracy of predicting future offloading requests; unpredictable traffic can reduce performance. Implementing multi-time step optimization adds computational overhead, which may be challenging in resource-constrained or large-scale networks. Stable evaluation conditions may not reflect real-world scenarios where dynamic network changes, such as user mobility and signal variations, can affect performance.

Finally, the summary of findings regarding the feasibility and practical challenges of implementing the proposed approach is as follows. The proposed deep learning model is implemented by using the widely used combination of Keras and Tensorflow frameworks [54] and thus, the resulting trained model can be easily ported or migrated to different operating systems and CPU architectures. The iterative inference step takes a negligible amount of time, making it applicable for real-time operations. On the other hand, each training epoch during the training process takes approximately two seconds on the aforementioned computing machine, but the training process is supposed to be carried out offline before the deployment of the proposed approach. As aforementioned, the proposed optimal approach as well as its variants are implemented by using the widely available computer solver. Although the proposed OPT(N_{sh}) utilizes binary variables, by leveraging advanced algorithms for handling binary/integer variables such as branch-and-bound,

cutting planes, and parallelism [55], the proposed optimization problem is solved in at most tens of milliseconds which is applicable for real-time scheduling since the proposed method can plan in advance by using the predicted future requests. In addition, by leveraging advanced techniques such as Lagrangian relaxation and problem decomposition [15] or gradual one-by-one removal [39] the computation time can be further reduced.

5. Conclusions

In this study, we proposed a multi-step request prediction and optimization scheme for MEC. The proposed scheme first forecasts the multi-step task-offloading requests. Then, using the predictions, the proposed approach called OPT(N_{sh}) optimizes network association, computing resource provisioning, and task offloading so that users' tasks are efficiently processed with reduced battery consumption, guaranteed deadlines, and reduced cloud use which causes larger delays. The proposed multi-time step optimization framework can schedule the computing resources and task offloading over multiple time steps, and thus, the deadline-constrained optimization problem can effectively be implemented while making a balanced use of ESs over time. Extensive evaluations have been carried out to show that the proposed approach can effectively predict future offloading requests with a pattern, and with these predictions, edge computing resources can be optimally utilized without violating deadlines. Even under the cases where the requests cannot be predicted for randomness, the proposed scheme has shown its effectiveness in not exploding the use of local and cloud computing resources and for making balanced use of ESs to some extent.

Author Contributions: Conceptualization, A.K.N. and T.K.; methodology, A.K.N. and T.K.; software, A.K.N.; validation, A.K.N. and T.K.; formal analysis, T.K.; investigation, A.K.N.; resources, T.K.; data curation, A.K.N.; writing—original draft preparation, A.K.N. and T.K.; writing—review and editing, T.K.; visualization, A.K.N.; supervision, T.K.; project administration, T.K.; funding acquisition, T.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a 2-Year Research Grant of Pusan National University.

Data Availability Statement: The data presented in this study along with the related code are available at the following GitHub repository: https://github.com/CISLAB-PNU/Traffic-Aware-Intelligent-Association-and-Task-Offloading-for-Multi-Access-Edge-Computing.git (accessed on 29 July 2024).

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

References

1. Pew Research Center. Mobile Fact Sheet. 2021. Available online: https://www.pewresearch.org/internet/fact-sheet/mobile/ (accessed on 6 July 2024).
2. Statista. Number of Smartphone Users Worldwide from 2016 to 2023. 2023. Available online: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (accessed on 6 July 2024).
3. International Data Corporation (IDC). Smartphone Shipments Totaled 1.35 Billion Units in 2021, According to IDC. 2022. Available online: https://www.idc.com/getdoc.jsp?containerId=prUS52032524 (accessed on 6 July 2024).
4. Yew, H.T.; Ng, M.F.; Ping, S.Z.; Chung, S.K.; Chekima, A.; Dargham, J.A. Iot Based Real-Time Remote Patient Monitoring System. In Proceedings of the 2020 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA), Langkawi, Malaysia, 28–29 February 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 176–179.
5. Herskovitz, J.; Wu, J.; White, S.; Pavel, A.; Reyes, G.; Guo, A.; Bigham, J.P. Making mobile augmented reality applications accessible. In Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility, Virtual Event, Greece, 26–28 October 2020; pp. 1–14.
6. Chen, Y.L.; Hsu, C.C. Self-regulated mobile game-based English learning in a virtual reality environment. *Comput. Educ.* **2020**, *154*, 103910. [CrossRef]
7. Imran, A.; Posokhova, I.; Qureshi, H.N.; Masood, U.; Riaz, M.S.; Ali, K.; John, C.N.; Hussain, M.I.; Nabeel, M. AI4COVID-19: AI enabled preliminary diagnosis for COVID-19 from cough samples via an app. *Inform. Med. Unlocked* **2020**, *20*, 100378. [CrossRef] [PubMed]
8. Vaz, D.; Matos, D.R.; Pardal, M.L.; Correia, M. MIRES: Intrusion Recovery for Applications Based on Backend-As-a-Service. *IEEE Trans. Cloud Comput.* **2023**, *11*, 2011–2027. [CrossRef]

9. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
10. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358. [CrossRef]
11. Chen, C.L.; Brinton, C.G.; Aggarwal, V. Latency minimization for mobile edge computing networks. *IEEE Trans. Mob. Comput.* **2021**, *22*, 2233–2247. [CrossRef]
12. Hu, H.; Song, W.; Wang, Q.; Hu, R.Q.; Zhu, H. Energy Efficiency and Delay Tradeoff in an MEC-Enabled Mobile IoT Network. *IEEE Internet Things J.* **2022**, *9*, 15942–15956. [CrossRef]
13. Antonopoulos, N.; Gillam, L. *Cloud Computing*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 51.
14. Nugroho, A.K.; Shioda, S.; Kim, T. Optimal Resource Provisioning and Task Offloading for Network-Aware and Federated Edge Computing. *Sensors* **2023**, *23*, 9200. [CrossRef] [PubMed]
15. Kim, T.; Lin, J.W.; Hsieh, C.T. Delay and QoS Aware Low Complex Optimal Service Provisioning for Edge Computing. *IEEE Trans. Veh. Technol.* **2023**, *72*, 1169–1183. [CrossRef]
16. Kherraf, N.; Alameddine, H.A.; Sharafeddine, S.; Assi, C.M.; Ghrayeb, A. Optimized provisioning of edge computing resources with heterogeneous workload in IoT networks. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 459–474. [CrossRef]
17. Xiang, Z.; Deng, S.; Jiang, F.; Gao, H.; Tehari, J.; Yin, J. Computing power allocation and traffic scheduling for edge service provisioning. In Proceedings of the 2020 IEEE International Conference on Web Services (ICWS), Beijing, China, 18–24 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 394–403.
18. Jiang, H.; Dai, X.; Xiao, Z.; Iyengar, A. Joint task offloading and resource allocation for energy-constrained mobile edge computing. *IEEE Trans. Mob. Comput.* **2022**, *22*, 4000–4015. [CrossRef]
19. Apostolopoulos, P.A.; Tsiropoulou, E.E.; Papavassiliou, S. Risk-aware data offloading in multi-server multi-access edge computing environment. *IEEE/ACM Trans. Netw.* **2020**, *28*, 1405–1418. [CrossRef]
20. Han, B.; Sciancalepore, V.; Xu, Y.; Feng, D.; Schotten, H.D. Impatient queuing for intelligent task offloading in multiaccess edge computing. *IEEE Trans. Wirel. Commun.* **2022**, *22*, 59–72. [CrossRef]
21. Song, H.; Gu, B.; Son, K.; Choi, W. Joint optimization of edge computing server deployment and user offloading associations in wireless edge network via a genetic algorithm. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 2535–2548. [CrossRef]
22. Feng, M.; Krunz, M.; Zhang, W. Joint task partitioning and user association for latency minimization in mobile edge computing networks. *IEEE Trans. Veh. Technol.* **2021**, *70*, 8108–8121. [CrossRef]
23. Charatsaris, P.; Diamanti, M.; Papavassiliou, S. Joint User Association and Resource Allocation for Hierarchical Federated Learning Based on Games in Satisfaction Form. *IEEE Open J. Commun. Soc.* **2023**, *5*, 457–471. [CrossRef]
24. Wang, H.; Wang, Y.; Sun, R.; Su, R.; Liu, B. Joint user association and power allocation for minimizing multi-bitrate video transmission delay in mobile-edge computing networks. In *Innovative Mobile and Internet Services in Ubiquitous Computing: Proceedings of the 12th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2018), Sydney, NSW, Australia, 3–5 July 2019*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 467–478.
25. Qi, Y.; Zhou, Y.; Liu, Y.F.; Liu, L.; Pan, Z. Traffic-aware task offloading based on convergence of communication and sensing in vehicular edge computing. *IEEE Internet Things J.* **2021**, *8*, 17762–17777. [CrossRef]
26. Wang, P.; Wang, Y.; Qiao, J.; Hu, Z. Traffic-Aware Optimization of Task Offloading and Content Caching in the Internet of Vehicles. *Appl. Sci.* **2023**, *13*, 13069. [CrossRef]
27. Oza, P.; Hudson, N.; Chantem, T.; Khamfroush, H. Deadline-aware task offloading for vehicular edge computing networks using traffic light data. *ACM Trans. Embed. Comput. Syst.* **2024**, *23*, 1–25. [CrossRef]
28. Guo, H.; Liu, J.; Lv, J. Toward intelligent task offloading at the edge. *IEEE Netw.* **2019**, *34*, 128–134. [CrossRef]
29. Kim, K.; Lynskey, J.; Kang, S.; Hong, C.S. Prediction based sub-task offloading in mobile edge computing. In Proceedings of the 2019 International Conference on Information Networking (ICOIN), Kuala Lumpur, Malaysia, 9–11 January 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 448–452.
30. Zeng, F.; Tang, J.; Liu, C.; Deng, X.; Li, W. Task-offloading strategy based on performance prediction in vehicular edge computing. *Mathematics* **2022**, *10*, 1010. [CrossRef]
31. Bohannon, R.W.; Andrews, A.W. Normal walking speed: A descriptive meta-analysis. *Physiotherapy* **2011**, *97*, 182–189. [CrossRef] [PubMed]
32. Benidis, K.; Rangapuram, S.S.; Flunkert, V.; Wang, Y.; Maddix, D.; Turkmen, C.; Gasthaus, J.; Bohlke-Schneider, M.; Salinas, D.; Stella, L.; et al. Deep learning for time series forecasting: Tutorial and literature survey. *ACM Comput. Surv.* **2022**, *55*, 1–36. [CrossRef]
33. Kim, N.; Balaraman, A.; Lee, K.; Kim, T. Multi-Step Peak Power Forecasting with Constrained Conditional Transformer for a Large-Scale Manufacturing Plant. *IEEE Access* **2023**, *11*, 136692–136705. [CrossRef]
34. Brownlee, J. *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*; Machine Learning Mastery: Vermont, Australia, 2018.
35. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
36. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6000–6010.

37. Kim, T.; Al-Tarazi, M.; Lin, J.W.; Choi, W. Optimal container migration for mobile edge computing: Algorithm, system design and implementation. *IEEE Access* **2021**, *9*, 158074–158090. [CrossRef]
38. Kim, T.; Chang, J.M. Profitable and energy-efficient resource optimization for heterogeneous cloud-based radio access networks. *IEEE Access* **2019**, *7*, 34719–34737. [CrossRef]
39. Kim, T.; Chang, J.M. QoS-aware energy-efficient association and resource scheduling for HetNets. *IEEE Trans. Veh. Technol.* **2017**, *67*, 650–664. [CrossRef]
40. Amazon Web Services, High Availability and Scalability on AWS—Real-Time Communication on AWS, Amazon Web Services. 2024. Available online: https://docs.aws.amazon.com/whitepapers/latest/real-time-communication-on-aws/high-availability-and-scalability-on-aws.html (accessed on 26 July 2024).
41. Microsoft Azure, Azure Availability Zones—High Availability at Scale. Available online: https://azure.microsoft.com/en-us/explore/global-infrastructure/availability-zones#features (accessed on 25 July 2024).
42. Docker Inc. Docker. Available online: https://www.docker.com/ (accessed on 6 July 2024).
43. Docker Inc. Runtime Options with Memory, CPUs, and GPUs. Available online: https://docs.docker.com/config/containers/resource_constraints/ (accessed on 6 July 2024).
44. Wang, H.; Xu, F.; Li, Y.; Zhang, P.; Jin, D. Understanding mobile traffic patterns of large scale cellular towers in urban environment. In Proceedings of the 2015 Internet Measurement Conference, Tokyo, Japan, 28–30 October 2015; pp. 225–238.
45. James, J.F. *A Student's Guide to Fourier Transforms with Applications in Physics and Engineering*; Cambridge University Press: Cambridge, UK, 2011.
46. Roonizi, A.K.; Sassi, R. ECG signal decomposition using Fourier analysis. *EURASIP J. Adv. Signal Process.* **2024**, *2024*, 79. [CrossRef]
47. Huang, H.; Chen, J.; Sun, R.; Wang, S. Short-term traffic prediction based on time series decomposition. *Phys. A Stat. Mech. Its Appl.* **2022**, *585*, 126441. [CrossRef]
48. Shi, J.; Leau, Y.-B.; Li, K.; Park, Y.-J.; Yan, Z. Optimization and Decomposition Methods in Network Traffic Prediction Model: A Review and Discussion. *IEEE Access* **2020**, *8*, 202858–202871. [CrossRef]
49. The MathWorks Inc. *MATLAB*, Version: 9.13.0 (R2022b); The MathWorks Inc.: Portola Valley, CA, USA, 2022.
50. Grant, M.; Boyd, S. *CVX: Matlab Software for Disciplined Convex Programming*, Version 2.1; CVX Research, Inc.: Austin, TX, USA, 2014.
51. Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, Version 11.0; Gurobi Optimization, LLC: Beaverton, OR, USA, 2024.
52. Khan, M.A.; Hamila, R.; Gastli, A.; Kiranyaz, S.; Al-Emadi, N.A. ML-based handover prediction and AP selection in cognitive Wi-Fi networks. *J. Netw. Syst. Manag.* **2022**, *30*, 72. [CrossRef]
53. Goldsmith, A. *Wireless Communications*; Cambridge University Press: Cambridge, UK, 2005.
54. Tensorflow, Keras: The High-Level API for TensorFlow, Tensorflow. Available online: https://www.tensorflow.org/guide/keras (accessed on 15 June 2024).
55. Gurobi Optimization, Mixed-Integer Programming (MIP)—A Primer on the Basics, Gurobi Optimization. Available online: https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics (accessed on 15 June 2024).