


Article

# Service Function Chain Scheduling in Heterogeneous Multi-UAV Edge Computing

Yangang Wang<sup>1,2</sup> , Hai Wang<sup>1,\*</sup> , Xianglin Wei<sup>2,\*</sup> , Kuang Zhao<sup>2</sup>, Jianhua Fan<sup>2</sup>, Juan Chen<sup>1</sup>, Yongyang Hu<sup>2</sup> and Runa Jia<sup>1,2</sup><sup>1</sup> College of Communication Engineering, Army Engineering University of PLA, Nanjing 210007, China<sup>2</sup> The Sixty-Third Research Institute, National University of Defense Technology, Nanjing 210007, China

\* Correspondence: hai\_wang@aeu.edu.cn (H.W.); weixianglin@nudt.edu.cn (X.W.)

**Abstract:** Supporting Artificial Intelligence (AI)-enhanced intelligent applications on the resource-limited Unmanned Aerial Vehicle (UAV) platform is difficult due to the resource gap between the two. It is promising to partition an AI application into a service function (SF) chain and then dispatch the SFs onto multiple UAVs. However, it is still a challenging task to efficiently schedule the computation and communication resources of multiple UAVs to support a large number of SF chains (SFCs). Under the multi-UAV edge computing paradigm, this paper formulates the SFC scheduling problem as a 0–1 nonlinear integer programming problem. Then, a two-stage heuristic algorithm is put forward to solve this problem. At the first stage, if the resources are surplus, the SFCs are deployed to UAV edge servers in parallel based on our proposed pairing principle between SFCs and UAVs for minimizing the completion time sum of tasks. In contrast, a revenue maximization heuristic method is adopted to deploy the arrived SFCs in a serial service mode when the resource is insufficient. A series of experiments are conducted to evaluate the performance of our proposal. Results show that our algorithm outperforms other benchmark algorithms in the completion time sum of tasks, the overall revenue, and the task execution success ratio.

**Keywords:** edge computing; unmanned aerial vehicle; artificial intelligence; service function chain



**Citation:** Wang, Y.; Wang, H.; Wei, X.; Zhao, K.; Fan, J.; Chen, J.; Hu, Y.; Jia, R. Service Function Chain Scheduling in Heterogeneous Multi-UAV Edge Computing. *Drones* **2023**, *7*, 132. <https://doi.org/10.3390/drones7020132>

Academic Editor: Vishal Sharma

Received: 12 January 2023

Revised: 29 January 2023

Accepted: 9 February 2023

Published: 13 February 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Artificial Intelligence (AI)—in particular Deep Learning (DL)—techniques have been widely adopted as a powerful tool in wireless communications and mobile computation areas due to their unique advantages such as automated feature extraction and high generalizability [1]. DL has been utilized from different perspectives for intelligent applications, e.g., wireless spectrum sensing, object tracking, and channel estimation [2–4]. Although the advantages are brought by DL techniques, fulfilling their intensive computation needs is a new challenge for resource-limited Internet of Things (IoT) devices. Mobile edge computing (MEC) is seen as a promising technology for solving this challenge [5–8]. It can make computation resources closer to IoT devices, so that the computation-intensive and delay-sensitive tasks can be offloaded to edge computing servers for executing.

For the IoT devices distributed in a rural area or even a hostile environment, less communications and computation infrastructures are available for processing the sensed big data. Due to the low operational cost, deployment flexibility, and high mobility, unmanned aerial vehicle (UAV) is considered to be the optimal temporary platform for emergency scenarios without infrastructure [9,10]. With this backdrop, introducing multi-UAV-empowered edge computing paradigm for processing edge-side big data is necessary [11,12]. Multiple UAVs can cooperate with each other in the sky to accept the data processing tasks from the IoT devices on the ground to conduct DL-involved tasks. Typically, the application of processing a DL-involved task can be treated as a service function chain (SFC) [13], in which several service functions (SFs) are connected in a sequential

order. For instance, each SF could be a modular in the DL-involved application, such as data pre-processing functions, deep neural network components, and target tracking, etc. An SF is a static software template that can derive instances on demand based on the virtual machine (VM) or docker technology [13–15]. A corresponding SF instance (SI) has to be created whenever the UAV decides to process a task. Once all the SIs of SFs contained in an SFC are successfully created, a task can pass through each instance sequentially to obtain its required services. In addition, in order to be able to run SFCs with AI algorithms, it is necessary to equip UAVs with custom-made AI processors. Compared with graphics processing unit (GPU), field programmable gate array (FPGA) has obvious characteristics of low power consumption and small size [16,17], which has been treated as a promising solution on the UAV platforms without violating size, weight, and power constraints inherent to UAV design. Considering that the UAV has limited computation resources and storage resources, the SFs contained in one SFC and their corresponding SIs can be distributed on multiple UAVs. When a large number of tasks are offloaded onto the UAV network at the same time stage, massive SIs corresponding to the required SFs will have to be created. In order to make full use of the limited computation and communication resources of UAVs, an efficient SFC scheduling strategy is indispensable. However, it also faces many challenges: (1) There is a complex matching relationship between tasks, SIs, and SFCs due to the heterogeneity of UAVs and the resource requirements of the tasks. (2) There is a complex trade-off between the communication and computation resource scheduling. (3) It is hard to achieve a long-term multi-objective optimization for the scenario with continuous task arrival and unknown SFC requirements. Tremendous efforts have been made in designing task scheduling algorithms in multi-UAV edge computing paradigms [18–34]. However, they often assume that each task is served by only one UAV and less attention has been paid to SFC scheduling in a multi-UAV edge computing scenario. A detailed analysis of existing efforts is presented in Section 2. With this backdrop, this paper firstly formulates the SFC scheduling problem as a 0–1 nonlinear integer programming problem. Then, a two-stage heuristic algorithm is put forward to derive a sub-optimal solution of the problem. The main contributions of this paper are threefold:

- (1) The SFC scheduling problem in heterogeneous “CPU + FPGA” computation architecture is formulated as a 0–1 nonlinear integer programming problem. The overall revenue of the system and the completion time sum of tasks are optimized with various resource constraints. To the best of our knowledge, this is the first paper that has studied the SFC scheduling problem considering FPGA resources in the multi-UAV edge computing network;
- (2) To solve the NP-hard problem with coupling variables, a two-stage heuristic algorithm called ToRu is put forward. At the first stage, i.e., when the resources are abundant, the SFCs of all tasks are deployed to UAV edge servers in parallel based on our proposed pairing principle between SFCs and UAVs for minimizing the sum of all tasks’ completion time; at the second stage, i.e., when the resources are insufficient, a revenue maximization heuristic method is adopted to deploy the arrived SFCs in a serial service mode. In order to obtain the long-term optimization, a time-slot partitioning protocol is designed, based on which ToRu can operate repeatedly in each time-slot;
- (3) A series of experiments are conducted to evaluate the performance of our proposal. Experimental results show that our proposed ToRu algorithm outperforms other benchmark algorithms in the the sum of all tasks’ completion time, the overall revenue, and the task execution success ratio.

The remaining of this paper is organized as follows. Section 2 summarizes related work. The model and problem formulation of the proposed system are introduced in Section 3. Section 4 describes the details of our proposed ToRu algorithm. The experiments and analysis of the results are presented in Section 5. Finally, Section 6 concludes this paper.

## 2. Related Work

In the multi-UAV edge computing system, according to the granularity of services provided to users, the existing work can be roughly divided into two categories: task scheduling and SFC scheduling. The difference between the two is mainly reflected in the matching process between offloaded tasks and UAVs. The former only considers whether the UAV's hardware computation resources (such as CPU, RAM, etc.) meet the task requirements; the latter further considers whether the SFs deployed on UAVs match the offload tasks requirements and is closer to the real situation.

### 2.1. Task Scheduling

Considering the high complexity of the multi-UAV edge computing system, most task scheduling optimization problems need to be solved by jointly optimizing user association, computation resource allocation, trajectories of UAVs, the number of offloaded task bits, etc. Most corresponding optimization models are non-convex, and it is difficult to obtain the optimal solution in polynomial time. An alternating iteration method is currently widely used to solve such complex non-convex optimization problems. It decouples a complex non-convex problem into multiple simplified sub-problems that can be solved by typical convex optimization or heuristic algorithm. Based on the alternating iteration algorithm, Yu et al. [18] jointly optimized the number of local computing tasks and offloaded tasks, trajectories of UAVs, and offloading matching strategy between UAVs and user terminals for minimizing the energy consumption of user terminals; Zhang et al. [19] jointly optimized user association, allocation of CPU frequency, power and spectrum resources, as well as trajectory of UAVs based on the proposed double-loop structure with the aim of maximizing the computation efficiency maximization. Luo et al. [20] jointly optimized the task scheduling, bit allocation, and UAV trajectory for minimizing the energy consumption of ground users. Wang et al. [21] presented a two-layer optimization method for minimizing system energy consumption, through jointly optimizing the deployment of UAVs and offloading decision. Moreover, some scholars formulated the task scheduling problem as a Markov decision process (MDP). A series of methods based on deep reinforcement learning (DRL) are proposed: Chang et al. [22] proposed a reinforcement learning framework with applying synthetic considerations of the terminals' demand, risk, and geometric distance, so as to provide better Quality-of-Service and path planning; Ren et al. [23] proposed a real-time scalable scheduling approach in the dynamic edge computing environments based on a DRL method; Xue et al. [24] established the User, UAV cost, and UAV revenue model and then jointly optimized power control, resource allocation, and UE association for minimizing the system energy consumption by a multi-agent reinforcement deep learning algorithm (MADRL). Seid et al. [25] deployed a clustered multi-UAV to provide computing services to users' devices and proposed a MADRL-based approach for minimizing the overall network computation cost while ensuring the quality of service. Wu et al. [26] designed a pre-dispatch UAV-assisted vehicular edge computing networks to cope with the demand of vehicles in multiple traffic jams, and then proposed a DRL-based energy efficiency autonomous UAV deployment strategy. Furthermore, strategies based on game theory [27,28] are also adopted to solve optimization problems. Each UAV was considered as an individual player with private interests, the optimization problem was formulated as an offloading game with at least one Nash equilibrium. Asheralieva et al. [29] presented a novel game-theoretic and reinforcement learning framework for task offloading. A Stackelberg game approach is adopted in [30,31] for maximizing utilities.

### 2.2. SFC Scheduling

Qu et al. [32] studied the service provisioning in the UAV-enabled MEC networks, where the SFs placement, UAV trajectory, task scheduling, and computation resource allocation were jointly optimized, so as to minimize the overall energy consumption of all users. A sub-optimal solution was achieved based on a proposed two-stage alternating

optimization algorithm. However, the application considered in [32] only contains one SF, so that SFC scheduling is not involved.

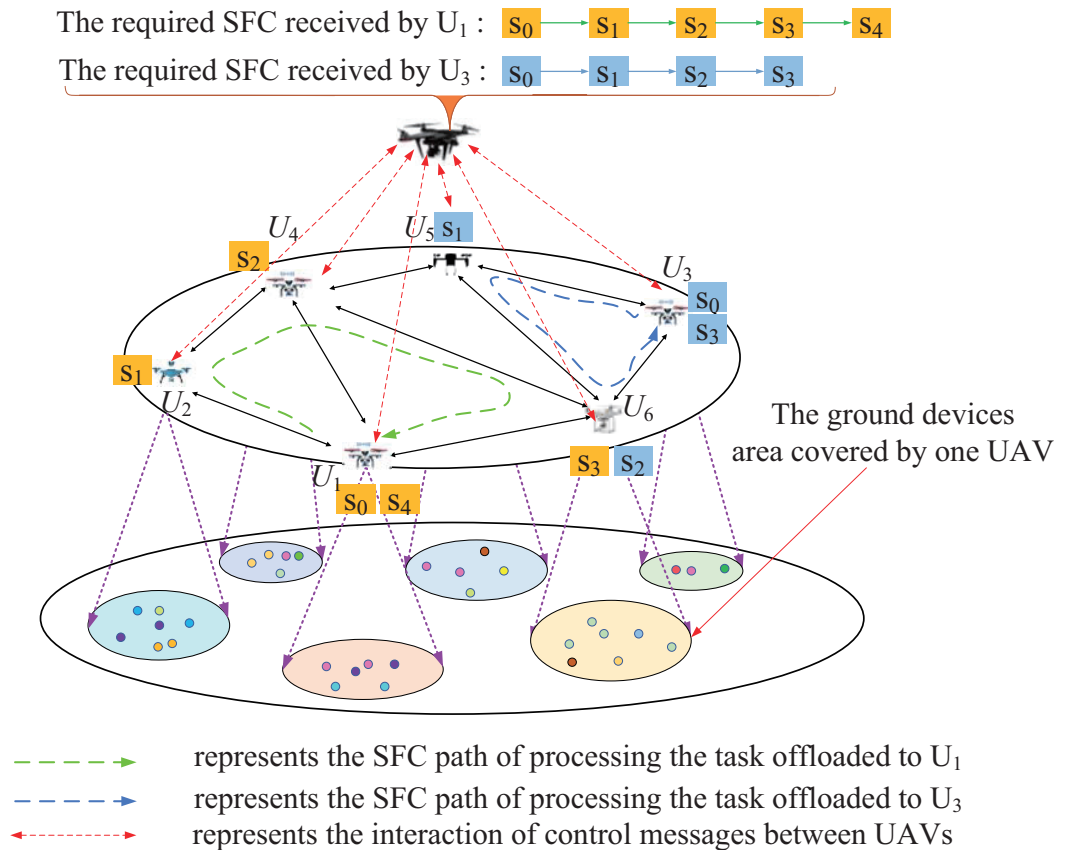
Wang et al. [33] proposed a reconfigurable service provisioning framework based on SFCs for Space–Air–Ground–Integrated Networks (SAGINs), where the computation and communications resource consumptions are balanced. Li et al. [34] investigated the online mapping and scheduling of dynamic virtual network functions (VNFs) in SAGINs, in which an Internet of Vehicles (IoV) service can be represented by a SFC formed with a set of chained VNFs. However, in the proposed SFC construction process, only the capacity constraints of CPU and buffer on NVF nodes are considered, and the channel resources constraints between NVF nodes are ignored. In fact, the channel resources between NVF nodes are very scarce in the wireless environment, and it often becomes the bottleneck factor affecting the SFC construction. In addition, the SFC scheduling mentioned in [33,34] aims to establish an end-to-end route for servicing data, where the data source and destination nodes are separately located in different geographical regions. However, in the edge computing scenario considered in this paper, the SFC scheduling aims to provide services for requesting users, where the input of raw data and the output of computing results are all located in the same access node. Moreover, dedicated hardware resources (such as GPU, FPGA, etc.) supporting AI algorithm operation are also not considered in [33,34].

In summary, it is difficult to solve the SFC scheduling problem involved in this paper with existing algorithms. To the best of our knowledge, compared with existing works, this is the first paper that considers the SFC scheduling problem in a multi-UAV edge computing network with FPGAs resources, which can provide services for complicated intelligent application-oriented tasks in the weak infrastructure areas.

### 3. System Model and Problem Formulation

#### 3.1. Network Model

In order to achieve a long-term SFC scheduling optimization, the management and allocation of UAV network resources is particularly important. Inspired by [35], a two-layer UAV network architecture is designed, in which one UAV with relatively high computing and communication resource capability is used as the master, other UAVs as the slaves. As shown in Figure 1, the edge computing network consists of one master UAV and  $M$  slave UAVs with heterogeneous computation resources, which evenly hovers over the mission area in a fully interconnected manner for performing time-sensitive AI tasks. The master UAV controls all the computation and communication resources on the slave UAVs, and it is responsible for instantiating the required SFCs. The slave UAVs are mainly used to cover devices on the ground, receive the SFC requests and offloaded time-sensitive AI tasks. When a slave UAV receives the SFC requirement of a ground device, it immediately reports to the master UAV. Then, the master UAV creates an SFC instance for the requiring ground device based on the resource of one or multiple slave UAVs. Eventually, the task offloaded from the requiring ground device is processed on the created SFC instance. For simplicity of description, the “heterogeneous Multi-UAV edge computing network” will be referred to as “UAV network” for short in the rest of this paper. The slave UAV network can be denoted by an directed complete graph  $\mathcal{G} = (\mathcal{U}, \mathcal{L})$ , where  $\mathcal{U}$  is the slave UAV set, denoted as  $\mathcal{U} = \{U_1, U_2, \dots, U_m, \dots, U_M\}$ ,  $1 \leq m \leq M$ . Several SFs are deployed on each slave UAV, which can be run by virtual machines (VMs) or docker technology [13–15].  $\mathcal{L}$  is the set of wireless links between slave UAVs, denoted as  $\mathcal{L} = \{L_1, L_2, \dots, L_m, \dots, L_M\}$ ,  $1 \leq m \leq M$ .  $L_m = \{L_{m,1}, L_{m,2}, \dots, L_{m,m'}, \dots, L_{m,M}\}$  represents the wireless link sets of  $U_m$  transmitting data to other UAVs, in which  $L_{m,m'}$  ( $1 \leq m' \leq M$ ) indicates the wireless link of  $U_m$  to  $U_{m'}$ . We assume that UAVs use orthogonal frequency channels for communication without collisions (e.g., OFDMA [36]). Thus,  $L_{m,m'}$  is represented as a two-tuple:  $\{r_{m,m'}, N_m^c\}$ .  $r_{m,m'}$  is the data transmission rate on one sub-channel,  $N_m^c$  represents the current idle sub-channel number. We define  $N_{max}^c$  as the maximum number of sub-channels on each slave UAV.  $N_m^c \leq N_{max}^c$  always holds.



**Figure 1.** A heterogeneous Multi-UAV edge computing scenario. Each UAV deploys several SFs and can receive the SFC required from the devices on the ground. The master UAV controls the resources of the edge computing network, which can instantiate the required SFCs.

Considering that UAVs have the characteristics of line-of-sight (LoS) communication, a free space attenuation model is adopted [36]. Then,  $r_{m,m'}$  can be expressed as:

$$r_{m,m'} = B \log_2 \left( 1 + \frac{p_m \beta_0}{N_0 B d_{m,m'}^2} \right), 1 \leq m, m' \leq M, m \neq m'. \quad (1)$$

where  $\beta_0$  is the transmit power gain at a reference distance of one meter.  $p_m$  is the transmission power of  $U_m$ .  $N_0$  is the noise power spectrum density.  $B$  is the bandwidth of a sub-channel.  $d_{m,m'}$  is the distance between  $U_m$  and  $U_{m'}$ . Note that when  $m = m'$ ,  $r_{m,m'} = \infty$ . In other words, this paper ignores the data exchanging time between two SF instances at the same UAV. For ease of description, the main notations used in this paper are listed in Table 1.

**Table 1.** Notations.

Notation	Definition
$U_n$	The $n$ -th UAV in $\mathcal{U}$
SF	Service Function
SI	SF instances
SFC	Service Function Chain
$SF_n$	The $n$ -th SF
$\tilde{\mathcal{S}}_1$	The set of FPGA-independent SFs
$\tilde{\mathcal{S}}_2$	The set of FPGA-dependent SFs
$Q_1$	The number of FPGA-independent SFs in $\tilde{\mathcal{S}}_1$
$Q_2$	The number of FPGA-dependent SFs in $\tilde{\mathcal{S}}_2$

Table 1. Cont.

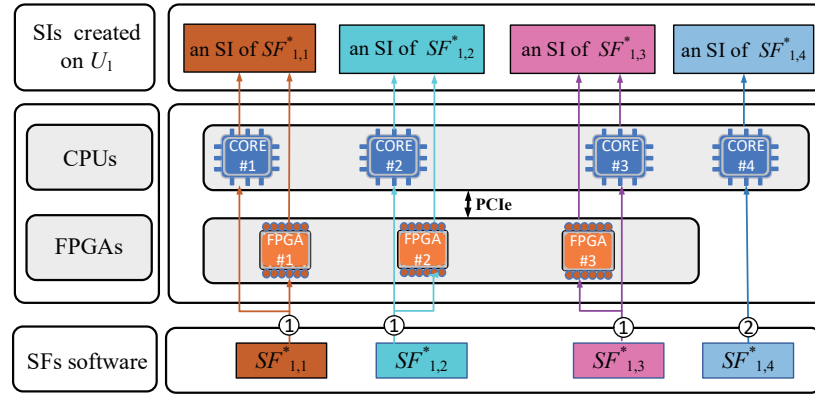
Notation	Definition
$\mathcal{S}_m$	The set of SF deployed on $U_m$
$SF_{m,q}^*$	The $q$ -th SF deployed on $U_m$
$\mathcal{T}$	The set of tasks currently requesting services
$T_n$	The $n$ -th requesting task
$N_m^{cpu}$	The number of CPU cores on $U_m$
$N_m^{fpga}$	The number of FPGAs on $U_m$
$N_{m,a}^{cpu}$	The number of the current idle CPU cores on $U_m$
$N_{m,a}^{fpga}$	The number of the current idle FPGAs on $U_m$
$n_{m,q}^i$	The number of SIs currently created corresponding to $SF_{m,q}^*$
$f_{m,q}$	The frequency of CPU core on $U_m$
$a_{m,q}$	The processing speed of CP when running SI of $SF_{m,q}^*$
$L_{m,m'}$	The wireless link of $U_m$ to $U_{m'}$
$r_{m,m'}$	The data transmission rate on one sub-channel of the link $L_{m,m'}$
$N_m^c$	The number of current idle sub-channels on the link $L_{m,m'}$
$o_n$	The source UAV that receives $T_n$
$s_n$	The required SFC of $T_n$
$s_n^k$	The $k$ -th SF contained in the SFC required by $T_n$
$l_n$	The properties of $T_n$ before its entering into an SFC
$l_n^k$	The properties of $T_n$ when arriving at the instance of $s_n^k$ in its SFC
$l_n^{k,0}$	The length of $T_n$ when arriving at $s_n^k$
$l_n^{k,1}$	The number of CPU cycles required for processing one bit task on $s_n^k$
$l_n^{k,2}$	The number of AI accelerator operations required for processing one bit task on $s_n^k$
$r_n$	The minimum requirement set for the transmission rate between the instances of SFCs required by $T_n$
$r_n^{k,k+1}$	The minimum transmission rate requirement from the instance of $s_n^k$ to $s_n^{k+1}$
$c_n$	The minimum computation resources requirement set of $T_n$ for the instances of SFC.
$c_n^{k,0}$	The minimum CPU processing speed requirement of $T_n$ on the instances of $s_n^k$ .
$c_n^{k,1}$	The minimum AI accelerator processing speed requirement of $T_n$ on the instances of $s_n^k$ .
$v_n$	The revenue obtained through completing $T_n$
$T_n^c$	The completion time of $T_n$
$X_{n,k}^m$	The decision variable of creating the instance of $s_n^k$

### 3.2. Service Model

In this paper, SFs are divided into two categories based on their resource requirements: FPGA-independent SFs and FPGA-dependent SFs [15]. An SI corresponding to the FPGA-independent SF is created only based on CPU resource. In contrast, only the combination of CPU and FPGA resources can support the SI corresponding to the FPGA-dependent SF. The set of FPGA-independent SFs are denoted by  $\widetilde{\mathcal{S}}_1 = \{SF_1, SF_2, \dots, SF_n, \dots, SF_{Q_1}\}$ .  $SF_n$  represents the  $n$ -th type of FPGA-independent SF,  $Q_1$  represents the number of FPGA-independent SF types. Moreover, the set of FPGA-dependent SFs are denoted by  $\widetilde{\mathcal{S}}_2 = \{SF_{Q_1+1}, SF_{Q_1+2}, \dots, SF_{n'}, \dots, SF_{Q_1+Q_2}\}$ .  $SF_{n'}$  represents the  $n'$ -th type of FPGA-dependent SF,  $Q_2$  is the number of FPGA-dependent SF types. Considering the heterogeneity of UAVs, the SF set deployed on each UAV may be different. Therefore, the set of SFs deployed on  $U_m$  are denoted by  $\mathcal{S}_m = \{SF_{m,1}^*, SF_{m,2}^*, \dots, SF_{m,q}^*, \dots, SF_{m,Q_m}^*\}$ ,  $Q_m \leq (Q_1 + Q_2)$  and  $\mathcal{S}_m \subseteq (\widetilde{\mathcal{S}}_1 \cup \widetilde{\mathcal{S}}_2)$ .  $SF_{m,q}^*$  is the  $q$ -th SF deployed on  $U_m$ .

As shown in Figure 2, a general computation model for UAVs is considered in this paper, which includes two parts: CPU and FPGA resources. The former includes several CPU cores of the same type; the latter refers to some FPGAs, which can assist CPUs in performing AI workload, such as FPGA-based convolutional network accelerator (CNN) [17]. The connection between CPU cores and FPGAs can be established dynamically through internal bus (e.g., PCIe bus) [37]. The number of CPU cores and FPGAs on  $U_m$  are denoted by  $N_m^{cpu}$  and  $N_m^{fpga}$ , respectively. Accordingly,  $N_{m,a}^{cpu}$  and  $N_{m,a}^{fpga}$  separately represent the number of current idle CPU cores and FPGAs. As shown in Figure 2, when creating an SI of  $SF_{1,1}^*$ ,  $U_1$  firstly assigns idle CPU core #1 and FPGA #1 to it; then, the corresponding SF software is separately loaded onto them. In contrast, only idle CPU core #4 is assigned to  $SF_{1,4}^*$  before loading its SF software. The processing capacity of an SI corresponding to  $SF_{m,q}^*$

is expressed as a two-tuple:  $\{f_{m,q}, a_{m,q}\}$ .  $f_{m,q}$  is the frequency of CPU core on  $U_m$ , measured in GHz;  $a_{m,q}$  is the processing speed of FPGA when running the SI of  $SF_{m,q}^*$ , measured in GOP/s [38]. Note that  $a_{m,q} = 0$  when  $SF_{m,q}^*$  is an FPGA-independent SF.



- ①  $U_1$  loads the software components of  $SF$  to CPU cores and FPGAs
- ②  $U_1$  loads the software components of  $SF$  to CPU cores

**Figure 2.** The creation process of SIs on  $U_1$ . The SI of FPGA-dependent SF (i.e.,  $SF_{1,1}^*$ ,  $SF_{1,2}^*$ , and  $SF_{1,3}^*$ ) occupies a CPU core and an FPGA. Moreover, the SI of FPGA-independent SF (i.e.,  $SF_{1,4}^*$ ) only occupies a CPU core.

From the perspective of information security, we stipulate that an SI independently occupies a CPU core or a combination of ‘CPU + FPGA’. Therefore, the maximum number of SIs that can be created simultaneously on one UAV is bounded. Define  $n_{m,q}^{si}$  to represent the number of currently created SIs corresponding to  $SF_{m,q}^*$  on  $U_m$ , and the following constraints must be satisfied [15].

$$n_{m,q}^{si} \leq N_m^{cpu}, SF_{m,q}^* \in \mathcal{S}_1, 1 \leq m \leq M. \tag{2}$$

$$n_{m,q}^{si} \leq N_m^{fpga}, SF_{m,q}^* \in \mathcal{S}_2, 1 \leq m \leq M. \tag{3}$$

$$\sum_{SF_{m,q}^* \in (\mathcal{S}_m \cap \tilde{\mathcal{S}}_2)} n_{m,q}^{si} \leq N_m^{fpga}, 1 \leq m \leq M. \tag{4}$$

$$\sum_{SF_{m,q}^* \in \mathcal{S}_m} n_{m,q}^{si} \leq N_m^{cpu}, 1 \leq m \leq M. \tag{5}$$

Equation (2) ensures that the number of currently created SIs corresponding to any FPGA-independent SF on  $U_m$  does not exceed the number of CPU cores on it. Equation (3) means that the number of currently created SIs corresponding to any FPGA-dependent SF on  $U_m$  does not exceed the number of FPGAs on it. Equation (4) guarantees that the total number of currently created SIs corresponding to all FPGA-dependent SFs on  $U_m$  does not exceed the number of FPGAs on it. Equation (5) restricts the total number of currently created SIs on  $U_m$  to not exceed the number of CPU cores on it.

### 3.3. Task Model

The tasks currently requesting service are denoted as a set  $\mathcal{T} = \{T_1, T_2, \dots, T_n, \dots, T_{N_t}\}$ .  $N_t = |\mathcal{T}|$  denotes the total number of tasks. We define a six-tuple for  $T_n$  as follows:  $T_n = \{o_n, s_n, l_n, r_n, c_n, v_n\}$ ,  $1 \leq n \leq N_t$ .  $o_n$  shows the source UAV that receives  $T_n$ .  $s_n$  indicates its required SFC, denoted as  $s_n = \{s_n^0, s_n^1, \dots, s_n^k, \dots, s_n^{N_n^s}, s_n^{N_n^s+1}\}$ ,  $0 \leq k \leq N_n^s + 1$ .  $s_n^0$  is the SF that receives the tasks from the ground and transmits it to the  $s_n^1$ , which has to be instantiated at the source UAV, as shown in Figure 1.  $s_n^{N_n^s+1}$  is the SF that receives the

computing result from  $s_n^{N_n^s}$  and transmits it to the ground, which also has to be instantiated at the source UAV.  $N_n^s$  represents the total number of SFs contained in the required SFC. When  $1 \leq k \leq N_n^s$ ,  $s_n^k \in \mathcal{S}$  represents the  $k$ -th SF contained in the required SFC, which can be instantiated on any slave UAV that satisfies its resource requirements.  $l_n$  means the current properties of  $T_n$  when it arrives at each SF contained in  $s_n$ , denoted as  $l_n = \{l_n^0, l_n^1, l_n^2, \dots, l_n^k, \dots, l_n^{N_n^s}, l_n^{N_n^s+1}\}$ ,  $0 \leq k \leq N_n^s + 1$ .  $l_n^k$  is the property of  $T_n$  when it arrives at  $s_n^k$ , denoted as  $l_n^k = \{l_n^{k,0}, l_n^{k,1}, l_n^{k,2}\}$ .  $l_n^{k,0}$  is the current length of  $T_n$ ;  $l_n^{k,1}$  is the number of CPU cycles required for processing one bit task;  $l_n^{k,2}$  is the number of AI accelerator operations required for processing one bit task. The computation resources consumed by  $s_n^0$  and  $s_n^{N_n^s+1}$  are considered to be negligible in this paper, since their instances require far less computation resources than other AI instances. Therefore,  $l_n^{0,0}$  and  $l_n^{N_n^s+1,0}$  represents the initial length of  $T_n$  and its computing result, respectively;  $l_n^{0,1}$ ,  $l_n^{0,2}$ ,  $l_n^{N_n^s+1,1}$  and  $l_n^{N_n^s+1,2}$  are equal to 0.  $r_n$  represents the minimum transmission rate requirement between the SFs contained in the required SFC, denoted as  $r_n = \{r_n^{0,1}, r_n^{1,2}, \dots, r_n^{N_n^s-1, N_n^s}, r_n^{N_n^s, N_n^s+1}\}$ .  $r_n^{k,k+1}$  ( $0 \leq k \leq N_n^s$ ) indicates the minimum transmission rate requirement from  $s_n^k$  to  $s_n^{k+1}$ .  $c_n$  represents the minimum computation resources requirement when creating the instances of SFs contained in the required SFC, denoted as  $c_n = \{c_n^0, c_n^1, c_n^2, \dots, c_n^k, \dots, c_n^{N_n^s}, c_n^{N_n^s+1}\}$ ,  $0 \leq k \leq N_n^s + 1$ .  $c_n^k$  is the minimum computation resource requirement of initiating  $s_n^k$ , denoted as  $c_n^k = \{c_n^{k,0}, c_n^{k,1}\}$ .  $c_n^{k,0}$  is the minimum processing speed requirement for CPU, measured in GHz;  $c_n^{k,1}$  is the minimum processing speed requirement for AI accelerator, measured in GOP/s. For  $c_n^0$  and  $c_n^{N_n^s+1}$ , their values are 0.  $v_n$  represents the revenue obtained through completing  $T_n$ .

As shown in Figure 1, an SFC is unidirectional, and the task goes through each SF sequentially. Furthermore, UAVs adopt the communication mode of orthogonal frequency division multiple access. Therefore, the communication resources consumed between adjacent SFs belong to the UAV instantiating the upstream SF. To sum up, for task  $T_n$ , it is more reasonable to create the SF instances in the reverse order of the required SFC, i.e., the instance of  $s_n^{N_n^s+1}$  is created first, and the instance of  $s_n^0$  is created last. Assume that the instance of  $s_n^{k+1}$  ( $0 \leq k \leq N_n^s$ ) has been created on  $U_{m'}$ . If we want to continue creating the instance of  $s_n^k$  on  $U_m$ , the following conditions have to be satisfied at the same time:

$$s_n^k = SF_{m,q}^*, 1 \leq m \leq M, 1 \leq q \leq Q_m \tag{6}$$

$$c_n^{k,0} \leq f_{m,q}, 1 \leq m \leq M, 1 \leq q \leq Q_m \tag{7}$$

$$c_n^{k,1} \leq a_{m,q}, 1 \leq m \leq M, 1 \leq q \leq Q_m \tag{8}$$

$$\lceil \frac{r_n^{k,k+1}}{r_{m,m'}} \rceil \leq N_m^c, m \neq m'. \tag{9}$$

Equation (6) means that  $U_m$  has to deploy the SF matching  $s_n^k$ . Equations (7) and (8) ensure that the SI of  $s_n^k$  on  $U_m$  can satisfy the computing requirement of  $T_n$ . Equation (9) guarantees that  $U_m$  has enough idle sub-channels for transmitting  $T_n$  to  $U_{m'}$  at a rate no smaller than the required. Moreover, define  $t_n^k$  as the stay time of  $T_n$  on the SI of  $s_{n,k}$ , which includes the executing time  $t_n^{k,1}$  and transmitting time  $t_n^{k,2}$ .  $t_n^k$  can be expressed as:

$$t_n^k = \begin{cases} \frac{l_n^{k,0} l_n^{k,1}}{f_{m,q}} + \frac{l_n^{k,0} l_n^{k,2}}{a_{m,q}}, & 1 \leq k \leq N_n^s. \\ 0, & k = 0, k = N_n^s + 1. \end{cases} \tag{10}$$



$$t_n^{k,2} = \begin{cases} \frac{t_n^{k+1}}{\lceil \frac{t_n^{k,k+1}}{r_{m,m'}} \rceil r_{m,m'}}, & m \neq m', 0 \leq k \leq N_n^s \\ 0, & m = m', 0 \leq k \leq N_n^s. \end{cases} \tag{11}$$

We define the binary variable  $X_{n,k}^m$  to represent the decision of creating the SI of  $s_n^k$ , which can be expressed as:

$$X_{n,k}^m = \begin{cases} 1, & \text{if the SI of } s_n^k \text{ is created on } U_m \text{ with satisfying the constraints} \\ & \text{of Equations (6) to (9)} \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

Considering that the SI of each SF contained in the required SFC is created at most one time, the following constraints must be satisfied:

$$\sum_{m=1}^{m=M} X_{n,k}^m \leq 1 \tag{13}$$

Note that  $X_{n,0}^{o_n} = 1$  and  $X_{n,N_n^s+1}^{o_n} = 1$  always hold, since the SIs of  $s_n^0$  and  $s_n^{N_n^s+1}$  have to be created on the source UAV.  $T_n$  can be successfully executed only if the SIs of all SFs contained in its required SFC have been successfully created. At this time, the following formula holds.

$$\prod_{k=0}^{k=N_n^s+1} \sum_{m=1}^{m=M} X_{n,k}^m = 1. \tag{14}$$

Lastly, we define  $T_n^c$  as the completion of  $T_n$ , which is expressed as follows:

$$T_n^c = \sum_{m=1}^{m=M} \sum_{k=0}^{k=N_n^s+1} X_{n,k}^m (t_n^{k,1} + t_n^{k,2}), 1 \leq n \leq N \tag{15}$$

*s.t. : Equations (1) to (14).*

### 3.4. Problem Formulation

For the required SFC, there may be multiple strategies of creating its SIs. For a task, it wants to be executed on the UAVs that can minimize its completion time. On the other hand, the UAV network wants to process all received tasks by making full use of their limited computation resources and communication resources, so as to maximize the revenue. In this paper, we aim at minimizing the completion time sum of all task while maximizing the overall revenue of the UAV network by optimizing  $X = \{X_{n,k}^m | 1 \leq n \leq N, 1 \leq m \leq M, 0 \leq k \leq N_n^s + 1\}$ .

Therefore, the first optimization goal is formulated as:

$$\mathcal{F}_1 = \sum_{n=1}^{n=N} T_n^c \prod_{k=0}^{k=N_n^s+1} \sum_{m=1}^{m=M} X_{n,k}^m \tag{16}$$

The second optimization goal is formulated as:

$$\mathcal{F}_2 = \sum_{n=1}^{n=N} v_n \prod_{k=0}^{k=N_n^s+1} \sum_{m=1}^{m=M} X_{n,k}^m \tag{17}$$

According to Equations (16) and (17), a multi-objective optimization problem can be formulated as:

$$\begin{aligned}
 \mathbf{P}_1 : & \min_{\{X\}} (\mathcal{F}_1, -\mathcal{F}_2) \\
 \text{s.t.} & C_1 : 1 \leq n \leq N, 1 \leq m \leq M, 1 \leq q \leq Q_m, 0 \leq k \leq N_n^s + 1 \\
 & C_2 : s_n^k = SF_{m,q}^*, \forall X_{n,k}^m = 1 \\
 & C_3 : X_{n,0}^{o_n} = 1, X_{n,N_n^s+1}^{o_n} = 1 \\
 & C_4 : \text{Equations (1) } \sim \text{(15)}
 \end{aligned} \tag{18}$$

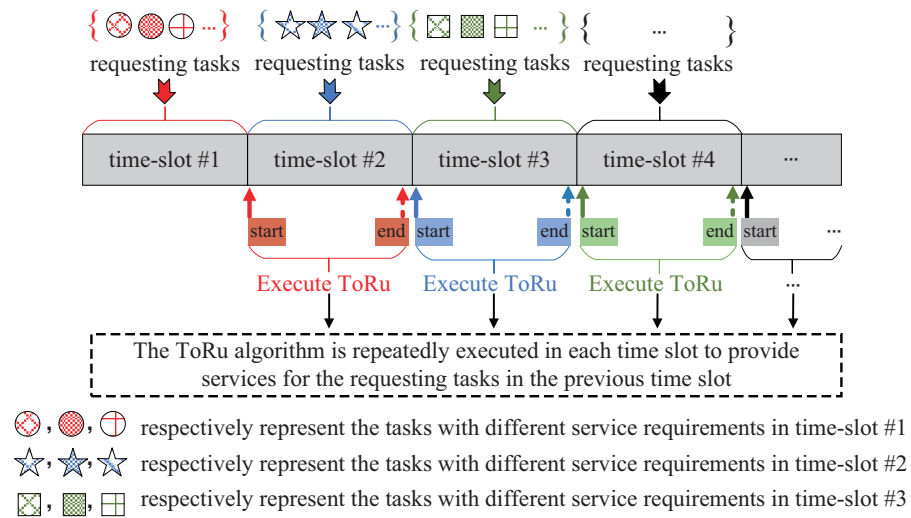
Constraint (C<sub>1</sub>) specifies the valid ranges of the involved variables in Constraint (C<sub>2</sub>)~(C<sub>4</sub>). Constraint (C<sub>2</sub>) guarantees that the UAV of instantiating the required SF should deploy this SF in advance. Constraint (C<sub>3</sub>) restricts that the SIs of the first SF and last SF of an SFC have to be created on the source UAV. Constraint (C<sub>4</sub>) includes several constraints related to the resource requirements and the resource capacities, which are the described in detail after Equations (1)~(15).

#### 4. Proposed Approach

In  $P_1$ , minimizing  $\mathcal{F}_1$  is a 0–1 nonlinear integer programming problem. Minimizing  $-\mathcal{F}_2$  is equivalent to maximizing  $\mathcal{F}_2$ , which is also a 0–1 nonlinear integer programming problem. At the same time, there is a close coupling relationship between  $\mathcal{F}_1$  and  $\mathcal{F}_2$ . Furthermore, the task properties considered in this paper are not known in advance. Therefore, it is difficult to solve  $P_1$  effectively with traditional optimization algorithms in an online manner. To tackle this problem in an online manner, we propose an efficient online two-stage heuristic algorithm named ToRu with much lower complexity. In ToRu, the minimization of the completion time sum of tasks is pursued in the case of abundant resources (i.e., the first stage); on the contrary, when the resources are insufficient (i.e., the second stage), the maximization of the UAV network revenue is pursued.

##### 4.1. The ToRu Framework

The proposed ToRu is deployed on the master UAV. In order to providing a long-term service for unknown tasks in an online manner, a time-slot partition protocol is designed, as shown in Figure 3. The mission period is divided into time-slots with equal length. One time-slot is the basic unit for SFC scheduling. At the beginning of each time-slot, the master UAV starts ToRu with the requesting tasks arriving in the previous time-slot and the current idle resources on the UAV network as input parameters. ToRu completes SFC scheduling before the end of a time-slot and exits. According to the above process, ToRu is executed repeatedly in each time-slot, thus enabling the UAV network to provide the long-term service. Note that ToRu must complete SFC scheduling before the end of a time-slot, otherwise the time-slot length needs to be increased, which indicates that the number of task requesting service is too large. If ToRu completes SFC scheduling very early before the end of the time-slot, it means that the number of tasks requesting service is small and the time-slot length should be shortened, thus reducing the service waiting time of tasks. Therefore, the time-slot partition protocol proposed in this paper has good dynamic scalability.



**Figure 3.** The time-slot partitioning protocol. The ToRu algorithm is started by the master UAV at the beginning of each time-slot and ends at the end of each time-slot, which is repeatedly executed in each time-slot to provide services for the requesting tasks in the previous time-slot. For example, ToRu is executed in time-slot #2 to only provide services for requesting tasks in time-slot #1.

The pseudocode of the ToRu algorithm is illustrated in Algorithm 1. Firstly, it calls Algorithm 2 with the task set  $\mathcal{T}$ , the idle communication resource set  $\mathcal{L}$  and idle computation resource set  $\mathcal{R}$  in the current time-slot as the input arguments. Note that  $\mathcal{R}$  is expressed as  $\mathcal{R} = \{R_1, R_2, \dots, R_m, \dots, R_M\}$ . The  $R_m = \{N_{m,a}^{cpu}, N_{m,a}^{fpga}, S_m\}$  means the current idle computation resource and deployed SFs on  $U_m$ . Then, the result returned by Algorithm 2 is denoted as a four-tuple:  $\{\tilde{X}, \tilde{S}_u, \tilde{F}_1, \tilde{F}_2\}$ .  $\tilde{X}$  indicates a sub-optimal of problem  $P_1$ .  $\tilde{S}_u$  means the task execution success ratio based on  $\tilde{X}$ , that is, the ratio of the number of successfully completed tasks to the total number.  $\tilde{F}_1$  and  $\tilde{F}_2$  shows the sum of the task completion time and the overall revenue based on  $\tilde{X}$ , respectively. If  $\tilde{S}_u = 1$ , the UAV network obtains the complete revenue from all tasks, and tasks also obtain the approximate minimum completion time sum; otherwise, it indicates that the UAV resources are insufficient, and Algorithm 3 is called for maximizing the overall revenue, regardless of the task completion time. This is reasonable, since when resources are insufficient, the number of the successfully completed tasks is more important than the task completion time. Algorithm 3 has the same type of the input parameters and output results with Algorithm 2. However, the  $\tilde{F}_1$  returned by Algorithm 3 only represents the completion time sum of tasks that have been executed successfully.

---

**Algorithm 1:** General Framework of ToRu.

---

**Input:** the task set  $\mathcal{T}$ , the idle communication resource set  $\mathcal{L}$ , and the idle computation resource set  $\mathcal{R}$ .

**Output:**  $\{X, S_u, F_1, F_2\}$ .

- 1: Initialize:  $X = \mathbf{0}, S_u = 0, F_1 = 0, F_2 = 0$ ;  $S_u$  represents the task execution success ratio based on  $X$ .
  - 2: Obtain  $\{\tilde{X}, \tilde{S}_u, \tilde{F}_1, \tilde{F}_2\}$  by calling Algorithm 2 with  $\mathcal{T}, \mathcal{L}$  and  $\mathcal{R}$  as the input parameters.
  - 3: **if**  $S_u < 1$  **then**
  - 4:   Obtain  $\{\tilde{X}, \tilde{S}_u, \tilde{F}_1, \tilde{F}_2\}$  by calling Algorithm 3 with  $\mathcal{T}, \mathcal{L}$  and  $\mathcal{R}$  as the input parameters.
  - 5: **end if**
  - 6:  $\{X, S_u, F_1, F_2\} = \{\tilde{X}, \tilde{S}_u, \tilde{F}_1, \tilde{F}_2\}$ .
  - 7: **return**  $\{X, S_u, F_1, F_2\}$ .
-

**Algorithm 2:** Minimizing the completion time sum.

---

**Input:** the task set  $\mathcal{T}$ , the idle communication resource set  $\mathcal{L}$ , the idle computation resource set  $\mathcal{R}$ .  
**Output:**  $\{X, S_u, \mathcal{F}_1, \mathcal{F}_2\}$ .

- 1: Initialize:  $X = \mathbf{0}$ ,  $S_u = 0$ ,  $\mathcal{F}_1 = 0$ ,  $\mathcal{F}_2 = 0$ ,  $L_{max} = 0$ ,  $N_t = 0$ ,  $sp = 0$ ,  $n = 0$ ,  $k = 0$ .
- 2:  $N_t = |\mathcal{T}|$ .
- 3:  $L_{max}$  = the maximum length of SFCs in  $\mathcal{T}$ .
- 4: Instantiate the last SF of the SFC on its source UAV for each task.
- 5:  $T_{temp} = Null$ . \\\ Define a temporary set.
- 6: **for**  $p = 0$  to  $p = L_{max} - 2$  **do**
- 7:   **for**  $n = 1$  to  $N_t$  **do**
- 8:      $k = N_n^s - p$ .
- 9:     **if**  $k < 0$  **then**
- 10:       Continue. \\\ The SFC of  $T_n$  has been successfully instantiated.
- 11:     **else**
- 12:       Add the  $T_n$  to the set  $T_{temp}$ .
- 13:     **end if**
- 14:   **end for**
- 15:   **while**  $T_{temp} \neq NULL$  **do**
- 16:     **for**  $i = 1$  to  $|T_{temp}|$  **do**
- 17:       Extract the  $i$ -th task  $T_{i^*}$  from the set  $T_{temp}$ ,  $T_{i^*} \in \mathcal{T}$ .
- 18:       Compute the candidate UAVs of the SF  $s_{i^*}^k$  for task  $T_{i^*}$  according to  $C_2 \sim C_4$ .
- 19:       **if no** the candidate UAVs **then**
- 20:          $S_u = 0$ .
- 21:         **return**  $\{X, S_u, \mathcal{F}_1, \mathcal{F}_2\}$ . \\\ There are tasks that cannot be completed as required.
- 22:       **end if**
- 23:       **end for**
- 24:       Select one task  $T_{n^*}$  from the set  $T_{temp}$  and instantiate its SF  $s_{n^*}^k$  on its optimal candidate UAV  $U_{m^*}$  based on our proposed principle in Section 4.2.
- 25:       The remaining resources of  $U_{m^*}$  are refreshed through subtracting the resources consumed by the instance of  $s_{n^*}^k$ .
- 26:        $X_{n^*,k}^m = 1$ , and delete  $T_{n^*}$  from  $T_{temp}$ .
- 27:     **end while**
- 28:   **end for**
- 29:  $S_u = 1$ .
- 30: Obtain the value of  $\mathcal{F}_1$  and  $\mathcal{F}_2$  according to Equation (16) and Equation (17).
- 31: **return**  $\{X, S_u, \mathcal{F}_1, \mathcal{F}_2\}$ .

---

**Algorithm 3:** Maximizing the overall revenue.

---

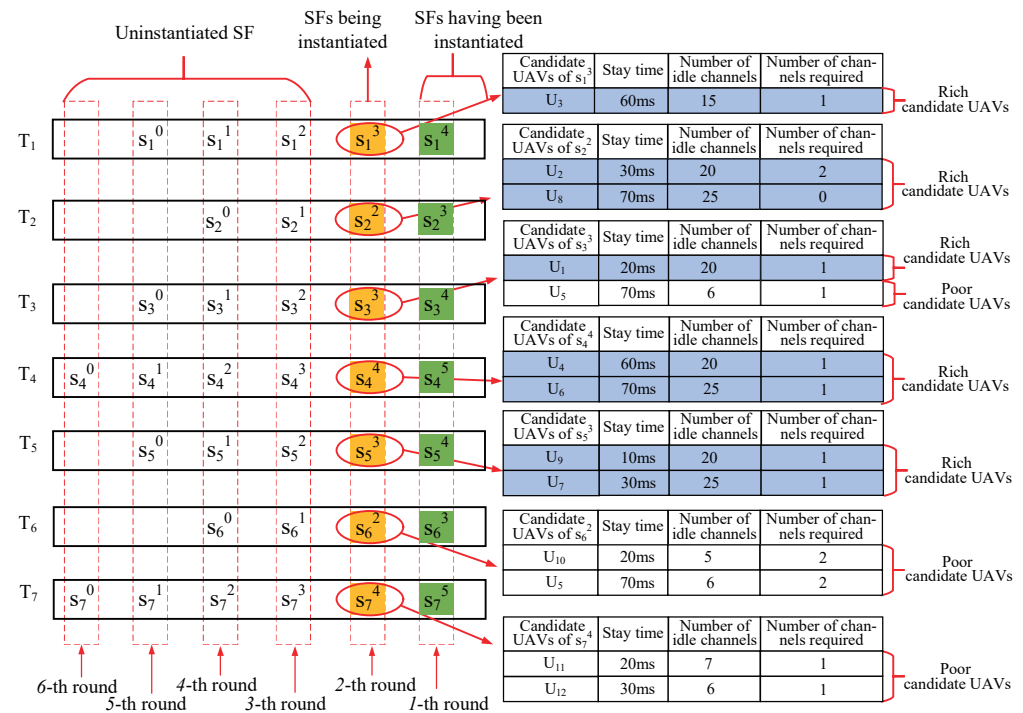
**Input:** the task set  $\mathcal{T}$ , the idle communication resource set  $\mathcal{L}$ , and the idle computation resource set  $\mathcal{R}$ .  
**Output:**  $\{X, S_u, \mathcal{F}_1, \mathcal{F}_2\}$ .

- 1: Initialize:  $X = \mathbf{0}$ ,  $S_u = 0$ ,  $\mathcal{F}_1 = 0$ ,  $\mathcal{F}_2 = 0$ ,  $N_t = 0$ ,  $N_{fail} = 0$ ,  $n = 0$ ,  $k = 0$ ;
- 2:  $N_t = |\mathcal{T}|$ ;
- 3: Sort the tasks in  $\mathcal{T}$  according to the value of  $v_n / N_n^s$ . The larger the value, the higher the ranking.
- 4: **for**  $n = 1$  to  $N_t$  **do**
- 5:   Instantiate the SF  $s_n^{N_n^s+1}$  of the task  $T_n$
- 6:   **for**  $k = N_n^s$  to 0 **do**
- 7:     Compute the candidate UAVs of the SF  $s_n^k$  for the task  $T_n$  according to  $C_2 \sim C_4$ ;
- 8:     **if no** the candidate UAVs **then**
- 9:        $N_{fail} ++$ ;
- 10:       Release the resources previously allocated to the task  $T_n$ , and clear the corresponding decision variable in  $X$ ;
- 11:       Continue;
- 12:     **end if**
- 13:     Select an optimal candidate UAV  $U_m^*$  based on the principle proposed in Section 4.3 for instantiating the SF  $s_n^k$ ;
- 14:     The remaining resources of  $U_m^*$  are refreshed through subtracting the resources consumed by the instance of  $s_n^k$ .
- 15:      $X_{n,k}^m = 1$ ;
- 16:   **end for**
- 17: **end for**
- 18:  $S_u = (N_t - N_{fail}) / N_t$ ;
- 19: Obtain the value of  $\mathcal{F}_1$  and  $\mathcal{F}_2$  according to Equation (16) and Equation (17).
- 20: **return**  $\{X, S_u, \mathcal{F}_1, \mathcal{F}_2\}$ .

---

### 4.2. Suboptimal Solution to Minimize the Completion Time Sum

When instantiating an SFC for a requesting task, the UAV network not only needs to provide sufficient computation resources for each SF instance, but also to ensure that there are sufficient wireless link resources between adjacent SF instances. In order to improve the efficiency of SFC instantiation, this paper implements SF instantiation one-by-one according to the reverse order of SFC. As shown in Figure 4, the downstream SF is first instantiated on a UAV; then, the candidate UAVs that can instantiate the upstream SF are found according to  $C_2 \sim C_4$ ; finally, in order to efficiently match SFCs and computing resources, so as to minimize the completion time sum of all tasks, this paper instantiates the SFCs of all tasks in parallel mode with sufficient UAV network resources. This parallel mode refers to the fact that only one SF in each SFC can be instantiated in a round of SF instantiation. After multiple rounds of SF instantiation, the instantiations of all SFCs can be completed. If all SFCs have the same length, the instantiations of all SFCs are completed at the same time.



**Figure 4.** Workflow chart of Algorithm 2. The last SF of the SFCs of  $T_1 \sim T_7$  is simultaneously instantiated on the source UAV in the 1-st round. In the 2-nd round, according to our proposed principle, the candidate UAVs for the upstream SFs (i.e.,  $s_1^3, s_2^2, s_3^3, s_4^4, s_5^3, s_6^2$  and  $s_7^4$ ) is obtained, respectively; then, these upstream SFs is instantiated one by one; next, go into the next round. When the 6-th round is successfully completed, the SFCs of all tasks are successfully instantiated and Algorithm 2 exits.

A typical process of the SFCs instantiation in parallel mode is shown in Figure 4. Assume that the SFCs of task  $T_1 \sim T_7$  need to be instantiated. The last SF of each SFC is first simultaneously instantiated on the source UAV (i.e., 1-st round). Then, according to  $C_2 \sim C_4$ , we separately identify candidate UAVs that can instantiate the upstream SF (i.e., 2-nd round). Furthermore, we calculate the corresponding task stay time and the number of sub-channels occupied when the upstream SF is instantiated on different candidate UAVs. Finally, we select one optimal UAV for each upstream SF from the candidates based on our proposed principle, which is described below:

1. The upstream SF with only one candidate UAV firstly are instantiated, which is beneficial to maximizing  $\mathcal{F}_2$ . As shown in Figure 4,  $s_1^3$  contained in  $T_1$  has one

- candidate UAV, so it is firstly instantiated. When facing multiple such upstream SFs (like  $s_1^3$ ), randomly select one of them for instantiation;
2. When all upstream SFs have multiple candidate UAVs, select the upstream SF with the candidate UAV that does not occupy the sub-channel, and instantiate it on this candidate UAV. As shown in Figure 4,  $s_2^2$  contained in  $T_2$  has multiple candidate UAVs, and if it is instantiated on the candidate  $U_8$ , no sub-channel is occupied. Therefore,  $s_2^2$  should be first initialized in the current situation. When facing multiple such upstream SFs (like  $s_2^2$ ), randomly select one of them for instantiation;
  3. If there is no upstream SF that satisfies the above principle 1 or principle 2, the candidate UAVs of each upstream SF are divided into two categories based on the number of idle channels: the candidate UAVs with the number of idle channels greater than  $N_e$  are called “rich candidate UAVs”, the remaining UAVs are called “poor candidate UAVs”. Considering the shortage of UAV wireless link resources, the upstream SFs should be instantiated preferentially on candidate UAVs with abundant link resources, i.e., “rich candidate UAVs”, which is beneficial to maximizing  $\mathcal{F}_2$ . Therefore, we first select an optimal UAV for the upstream SF with “rich candidate UAVs”, the specific principles are as follows:
    - (a) The upstream SFs with only one “rich candidate UAV” are first instantiated. As shown in Figure 4,  $s_3^3$  contained in  $T_3$  has only one “rich candidate UAV”, so it is instantiated first. When facing multiple such upstream SFs (e.g.,  $s_3^3$ ), randomly select one of them for instantiation;
    - (b) When the remaining upstream SFs have multiple “rich candidate UAVs”, we rank their candidate UAVs according to the stay time of a task executed on them, and the candidate UAV with short stay time is ranked higher. The upstream SF with the largest gap in the stay time between its first-ranked candidate UAV and its second-ranked candidate one will first be instantiated on the first candidate one, which is beneficial to minimizing  $\mathcal{F}_1$ . As shown in Figure 4, both  $s_4^4$  contained in  $T_4$  and  $s_5^3$  contained in  $T_5$  have two “rich candidate UAV”. The gap in the stay time of  $s_4^4$  ( $s_5^3$ ) on its different candidate UAVs is 10 ms (20 ms), so  $s_5^3$  is first instantiated on the candidate  $U_9$ . When the gap is the same, select one of them at random for instantiation

In addition, when all upstream SFs with “rich candidate UAVs” have been instantiated and there are still uninstantiated upstream SFs, i.e., the upstream SFs with only “poor candidate UAVs”, we regard the “poor candidate UAVs” as “rich candidate UAVs” and select the optimal UAVs for the uninstantiated SFs according to the above principle (a) and (b). As shown in Figure 4, both  $s_6^2$  contained in  $T_6$  and  $s_7^4$  contained in  $T_7$  have only “poor candidate UAV”, so they are lastly instantiated according to the above principle (a) and (b). Note that once an SF is successfully instantiated, all candidate UAVs belonging to uninstantiated SFs must be updated immediately before starting to select the optimal UAV for the next uninstantiated SF (i.e., step 16~25 in Algorithm 2). Repeat the above operations until the SFCs of all tasks are instantiated, whose pseudocode is as shown in Algorithm 2.

#### 4.3. Suboptimal Solution to Maximize the Overall Revenue

When the UAV network resources are insufficient, in order to obtain more revenue, UAVs naturally give priority to serving tasks that can pay more on average for each SF instance, i.e., the task with the greatest value of  $v_n/N_n^s$  is given priority. Different from Algorithm 2, we adopt a serial service mode, that is, only after completing the SFC instantiation of one task, we start to instantiate the SFC of the next task. Thus, the principle of selecting one optimal UAV for each SF is also different from that in Section 4.2, which is described as follows:

1. The UAV network first instantiates the SFC for a task with the greatest value of  $v_n/N_n^s$ . When facing multiple tasks with the same payment, the UAV randomly selects one to serve;

2. An SF is preferentially instantiated on the candidate UAV that does not occupy the sub-channel;
3. If there is no task that satisfies the above principle 2, the candidate UAVs of each task are divided into “rich candidate UAVs” and “poor candidate UAVs” according to the principle in Section 4.2. Then, we do the following:
  - (a) When the number of “rich candidate UAVs” is greater than 0, the candidate UAV with the lowest performance is selected, and the high-performance UAVs are left for subsequent tasks with higher computation requirement, which is beneficial to maximizing  $\mathcal{F}_2$ ;
  - (b) When the number of “rich candidate UAVs” is equal to 0, the above operations are performed among “poor candidate UAVs”.

Repeat the above operations for the required SFCs of all tasks are instantiated, whose pseudocode is as shown in Algorithm 3.

#### 4.4. Computational Complexity Analysis

In order to show the feasibility and efficiency of the proposed ToRu algorithm, we focus on its time computational complexity in this section. As shown in Algorithm 1, when the idle resources of the UAV network are rich and the number of tasks requesting service is small, ToRu only executes Algorithm 2; otherwise, it first executes Algorithm 2, and then executes Algorithm 3. Next, we analyze the time complexity of Algorithm 2 and Algorithm 3, respectively. As shown in Algorithm 2, the maximum value of variable  $|T_{temp}|$  is equal to  $N_t$ , and its value decreases with the increase of the control variable  $p$ . Therefore, the worst time complexity of Algorithm 2 is  $O(L_{max} \cdot N_t)$ , where  $L_{max}$  represents the maximum number of SFs contained in an SFC,  $N_t$  represents the number of tasks requesting service. Similarly, the maximum value of variable  $N_n^s$  in Algorithm 3 is equal to  $L_{max}$ , so that the worst time complexity of Algorithm 3 is also  $O(L_{max} \cdot N_t)$ . To sum up, the worst time complexity of the proposed ToRu algorithm is  $O(L_{max} \cdot N_t)$ , which can obtain the sub-optimal solution to the problem  $P_1$  in polynomial time.

### 5. Simulation and Results Analysis

This section first presents the experimental settings, then analyzes the results.

#### 5.1. Experimental Settings

Platform Settings. All experiments were conducted on a PC that runs Ubuntu 18.04 with 3.2 GHz CPU and 16 GB RAM. The proposed ToRu algorithm was designed and implemented using C++ language.

Parameter Settings. Consider a service scenario with 25 UAVs that are 500 m apart. The number of tasks input into the simulation environment varies from 10 to 190. The main parameter settings are included in Table 2.

Table 2. Parameter settings.

Parameter	Value	Parameter	Value
$M$	25	$Q$	30
$f_{m,q}$	[1, 10] GHz	$a_{m,q}$	[2, 20] GOPS
$P_m$	1 W	$N_0$	$10^{-20}$ W/Hz
$B$	1 MHz	$\beta_0$	$1.42 \times 10^{-4}$
$r_n^{k,k+1}$	10 Mbit	$N_m^c$	8
$I_n^{k,0}$	[0.1, 10] Mbit	$I_n^{k,1}$	[100, 1,000,000] Cycles
$N_n^s$	[2, 5]	$I_n^{k,2}$	[200, 2,000,000] OPs
$N_m^{cpu}$	[10, 20]	$N_m^{fpga}$	[10, 20]
$d_{n,m}$	500 m	$v_n$	[2, 20]

Experimental process. When the number of input tasks is given, the geographic location of each task is generated randomly, and then the attributes of each task (such as

task length, required SFC, task complexity, etc.) are generated randomly. Finally, each task randomly requests service from a UAV that covers it. In addition, for a given number of input tasks, we simulate 100 times and take the average of the simulation results as the final value.

**Comparison Benchmarks.** To validate the necessity of each component of the comparison benchmarks design, we adopt a step-by-step evaluation philosophy in the experimental design. For each benchmark algorithm, there are two steps: the order in which these SFCs are instantiated, and the principle of instantiating SI contained in an SFC. For the first step, similar to the greedy algorithm [20], two sorting strategies are chosen as comparison benchmarks: (1) Revenue: the task with highest payment is firstly served; (2) Length: the task with the shortest SFC length is firstly served. For the second step, three strategies are chosen as comparison benchmarks: (1) Random: an SF is instantiated on a random candidate UAV, similar to the random algorithm [32]; (2) Greedy: an SF is instantiated on a candidate UAV with the best performance, similar to the greedy algorithm [20]; (3) Local: an SF is only instantiated on a local UAV, similar to the local algorithm [32]. Similar to these step-wise algorithms [20,32], we have 6 combination algorithms for comparison, marked as “Revenue + Random”, “Revenue + Greedy”, “Revenue + Local”, “Length + Random”, “Length + Greedy”, and “Length + Local”.

1. “Revenue + Random”: it first selects the task with the highest payment and performs SFC scheduling for it; next, when instantiating one SF contained in an SFC, it always randomly selects one from the candidate UAVs to instantiate this SF.
2. “Revenue + Greedy”: this algorithm first selects the task with the highest payment and performs SFC scheduling for it; next, when instantiating one SF contained in an SFC, it always selects the best performance one from the candidate UAVs to instantiate this SF.
3. “Revenue + Local”: this algorithm first selects the task with the highest payment and performs SFC scheduling for it; next, when instantiating one SF contained in an SFC, it always selects the local one from the candidate UAVs to instantiate this SF.
4. “Length + Random”: this algorithm first selects the task with the shortest SFC length and performs SFC scheduling for it; next, when instantiating one SF contained in an SFC, it always randomly selects one from the candidate UAVs to instantiate this SF.
5. “Length + Greedy”: this algorithm first selects the task with the shortest SFC length and performs SFC scheduling for it; next, when instantiating one SF contained in an SFC, it always selects the best performance one from the candidate UAVs to instantiate this SF.
6. “Length + Local”: this algorithm first selects the task with the shortest SFC length and performs SFC scheduling for it; next, when instantiating one SF contained in an SFC, it always selects the local one from the candidate UAVs to instantiate this SF.

## 5.2. Results and Analysis

### 5.2.1. The Completion Time Sum of Tasks

Figure 5 shows the completion time sum under the different tasks offloaded to the UAV network. We can see that our proposed ToRu algorithm significantly outperforms other algorithms in the completion time sum of tasks at the first stage, i.e., the stage before the number of tasks reaches critical point. Figure 6 shows the simulation results of the first stage in more detail. This is mainly because on the one hand, we instantiate each SF of the SFCs of all tasks in a parallel mode; on the other hand, the pairing rules between SFs and their candidate UAVs are designed from the global perspective, and each SF considers the impact on other SFs when selecting candidate UAVs. Then, the performance of “Revenue + Greedy” and “Length + Greedy” comes second. It is also reasonable, since both algorithms greedily choose the UAV with the highest processing performance to instantiate the SFC. Lastly, the Algorithm “Revenue + Random”, “Revenue + Local”, “Length + Random”, and “Length + Local” have the worst performance because they do not consider the effect of UAVs on the completion time when making their choices. However, with the increase



of the number of tasks, the completion time sum of tasks also increases. This is because the length and type of newly added tasks are random, so UAV computation resources are used more fully and more tasks are executed successfully. Furthermore, as the number of tasks continues to increase, the completion time sum of tasks in all algorithms no longer increases, it even starts to decrease (e.g., “Length + Greedy”). The reason is that tasks with less execution time are prioritized. To sum up, when the number of tasks exceeds the critical point (i.e., tasks can not be executed 100%), it is meaningless to evaluate the completion sum of tasks.

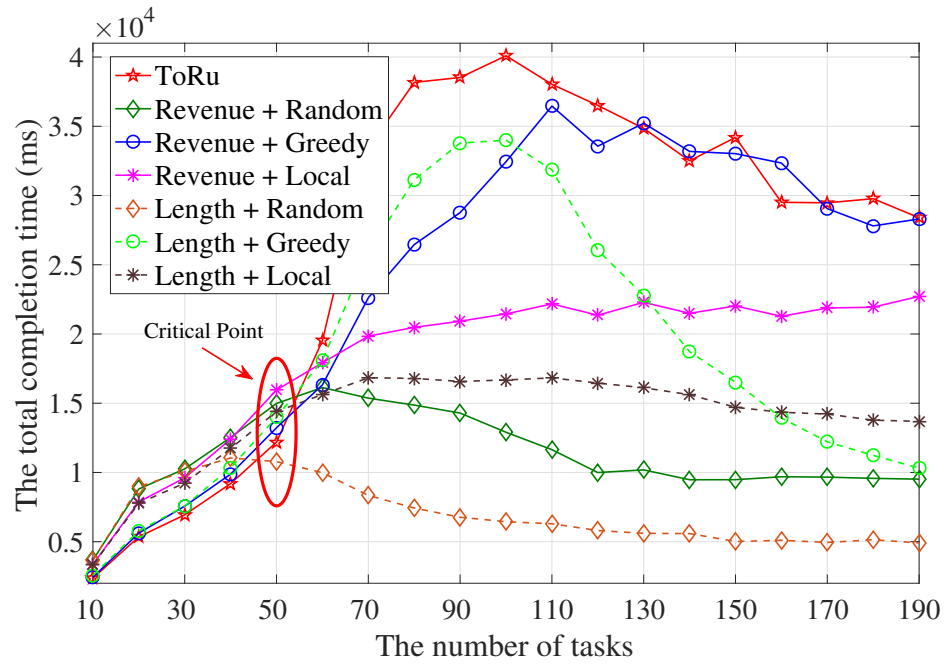


Figure 5. The completion time sum with different algorithms during the whole simulation phase.

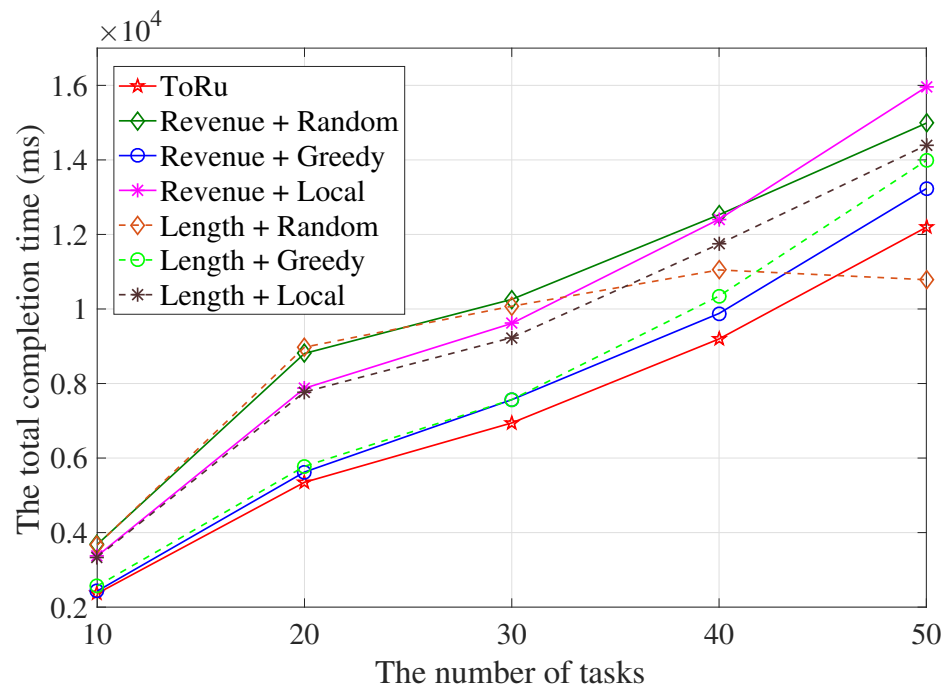


Figure 6. The completion time sum with different algorithms in the stage of abundant resources.

### 5.2.2. The Task Execution Success Ratio

Figure 7 shows the average execution success ratio of algorithms under different tasks. The result shows that algorithm ToRu has the highest success ratio, especially with the gradual increase of tasks, it can still maintain a high success ratio. This is mainly because ToRu only selects candidate UAVs with low performance to meet the requirements when instantiating SF, in other words, the improvement in the task execution success ratio comes at the expense of individual task execution performance. The algorithms “Length + Local” and “Revenue + Local” show the worst performance before the critical point, because they do not fully utilize the resources of the UAV network. With the increase of the number of tasks, “Length + Greedy” has the highest task execution success ratio. This is reasonable since it serves tasks with the shortest SFCs first, which consume less computation and communication resources. Finally, the algorithms “Length + Random” and “Revenue + Random” shows the poor performance when the network load is heavy.

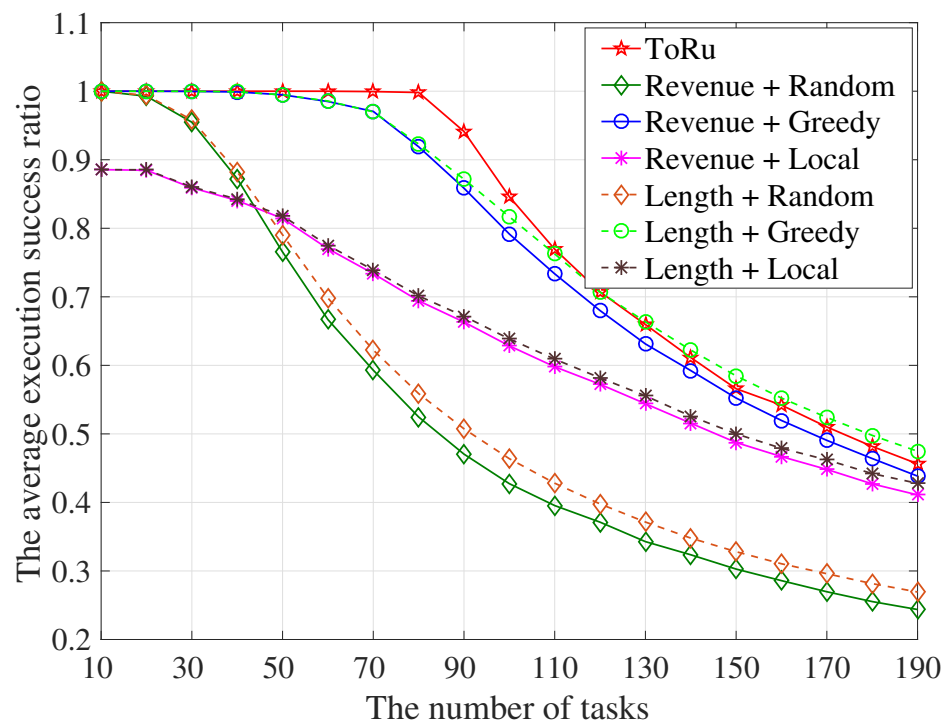
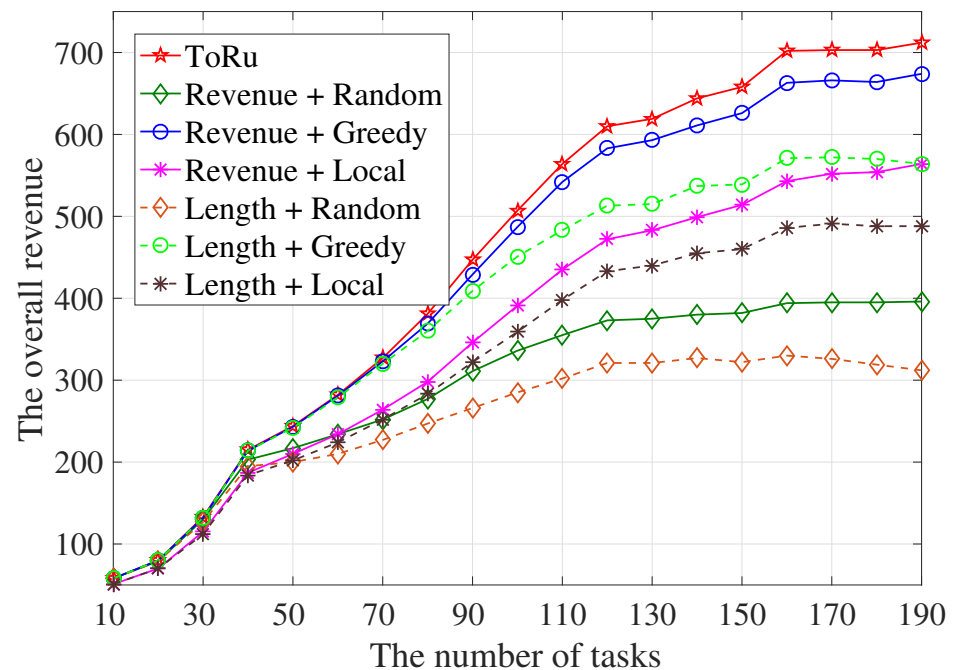


Figure 7. The task execution success ratio with different algorithms.

### 5.2.3. The Overall Revenue

Figure 8 presents the overall revenue under different tasks. As can be seen in Figure 7, when the number of tasks is small, the task execution success ratios of the algorithms except “Length + Local” and “Revenue + Local” are all 100%. Therefore, they have equal revenue. With the increase of tasks, the overall revenue of each algorithm increases rapidly before the critical point, then, the growth becomes slow. Moreover, the overall revenue of algorithm ToRu is always the largest among the seven algorithms because it not only ensures that tasks with higher payments are executed, but also ensures a higher task execution success rate. The algorithm “Revenue + Greedy” performs better, which is mainly because it first completes tasks that pay more. In addition, we can see that the performance of algorithms based on revenue sorting principle is better than that of algorithms based on length sorting, which is reasonable.



**Figure 8.** The overall revenue with different algorithms.

#### 5.2.4. The Resources Utilization

Figures 9 and 10 show the utilization of channel and computation resources of different algorithms, respectively. We can find that the channel resources consumed by the algorithm ToRu increase with the number of tasks before the critical point. This is because the resources are sufficient before the critical point, and the algorithm ToRu seeks to minimize the completion time sum of tasks without caring about resource consumption. Due to the limited communication resources between UAVs, it often becomes the bottleneck of the task execution success ratio. When the number of tasks requesting service is small, Algorithm 2 is executed. It instantiates each requested SFC in parallel mode, that is, it selects the most preferred UAV for each SF contained in different SFCs at the same time, which will lead to the premature consumption of the most preferred UAVs early. Therefore, the probability that all SFs contained in one SFC are deployed on the same UAV will be reduced. Different SFs contained in one SFC have to interact with each other, resulting in the high channel utilization. On the contrary, when the number of tasks requesting service is large, Algorithm 3 is executed. It instantiates each requested SFC in serial mode, that is, it starts to instantiate the next SFC after having instantiated all SFs contained in the previous SFC, so that the probability that all SFs contained in one SFC are instantiated on the same UAV will be greatly increased. The interaction between different SFs on the same UAV will no longer consume channel resources, thus, the channel utilization will be rapidly reduced. As shown in Figure 9, after the critical point, the channel utilization in the algorithm ToRu decreases greatly and remains stable at a low value. This inevitably leads to the increase of the completion time sum of tasks (as shown in Figure 5), but it can improve the task execution success ratio (as shown in Figure 7). The algorithms “Revenue + Local” and “Length + Local” do not consume channels because they only execute tasks on the local UAV. The channel utilization of other algorithms increases sharply with the increase of the number of tasks, so that their computation resources cannot be fully utilized, as shown in Figure 10. This will reduce the task execution success ratio and the overall revenue, as shown in Figures 7 and 8. It is obvious that the computation utilization of the algorithm ToRu has reached 100%, which is the reason why its task execution success ratio and overall revenue are the highest.

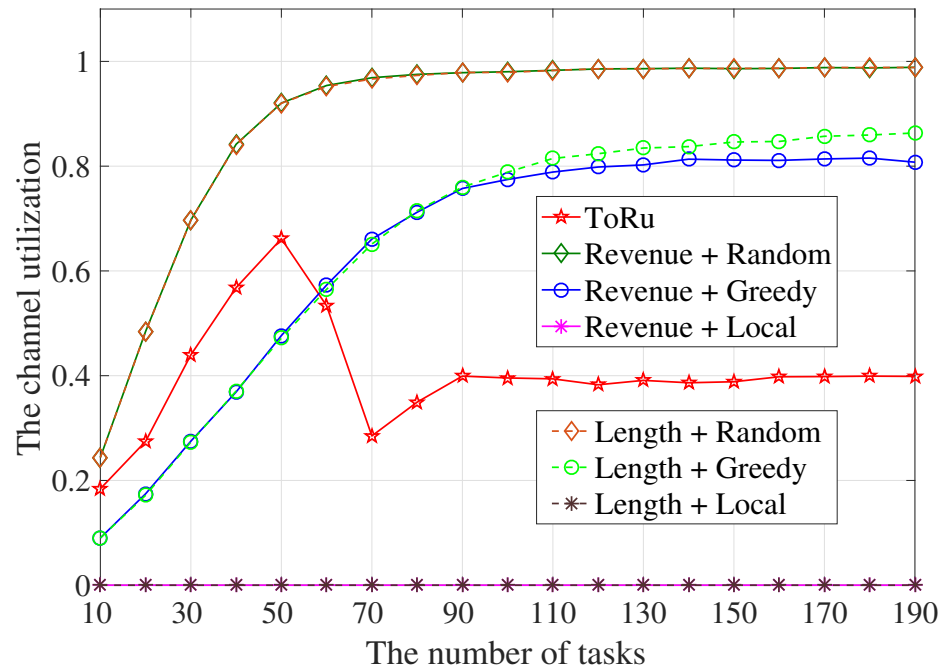


Figure 9. The channel utilization with different algorithms.

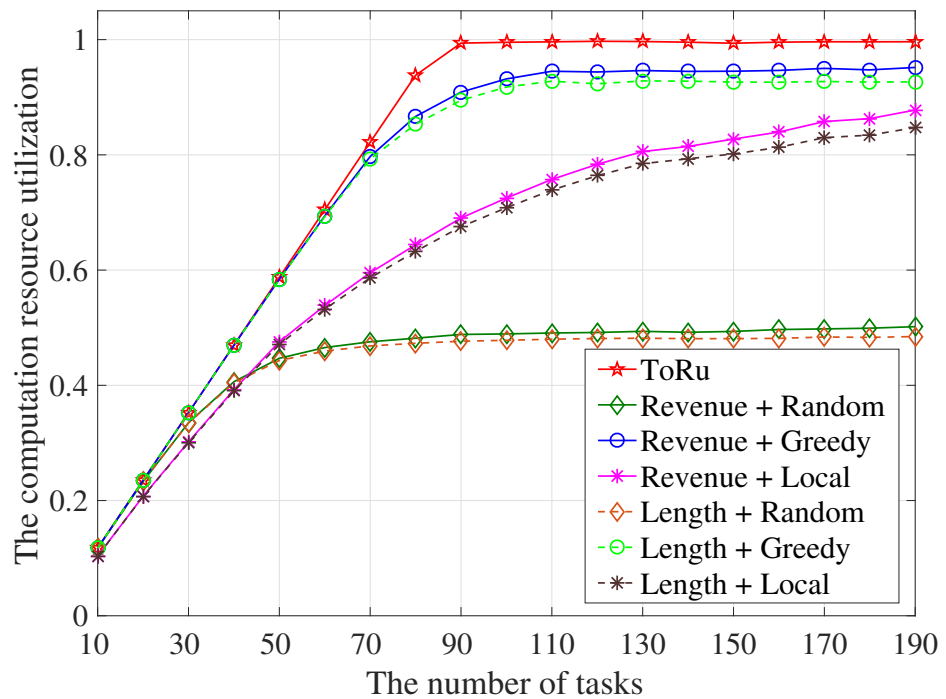


Figure 10. The computation resource utilization with different algorithms.

### 5.2.5. The Operation Time

Table 3 provides the operation time results of our proposed ToRu algorithm on average under different number of tasks requesting service. We can see that the operation time of ToRu is relatively small in the first stage (that is, before the number of tasks requesting service reaches “critical point”, as shown in Figure 5). This is because as there are sufficient resources at this stage, ToRu can exit after executing Algorithm 2. When the number of tasks requesting service exceeds the “critical point” (i.e., 50), ToRu can judge that the resources are insufficient after executing Algorithm 2, and it then continues to execute Algorithm 3 for rescheduling SFCs. This undoubtedly increases the operation time, as

shown in Table 3. However, as the number of tasks requesting service continues to increase, the shortage of resources will become more obvious, which can be easily judged by several loop operations of Algorithm 2. Therefore, the operation time of Algorithm 2 can be ignored, and the operation time of ToRu only includes the one of Algorithm 3. As shown in Table 3, the operation time of the ToRu algorithm decreased sharply after the number of tasks requesting service exceeds 110, and then maintained a stable small increase. This is consistent with our complexity analysis results in Section 4.4, indicating that our proposed algorithm has good execution efficiency and scalability.

**Table 3.** Operation time of ToRu algorithm.

Number of Tasks	Operation Time	Number of Tasks	Operation Time
$N_t = 20$	~ 2 ms	$N_t = 110$	~ 34 ms
$N_t = 30$	~6 ms	$N_t = 120$	~36 ms
$N_t = 40$	~14 ms	$N_t = 130$	~38 ms
$N_t = 50$	~32 ms	$N_t = 140$	~40 ms
$N_t = 60$	~50 ms	$N_t = 150$	~46 ms
$N_t = 70$	~90 ms	$N_t = 160$	~50 ms
$N_t = 80$	~95 ms	$N_t = 170$	~56 ms
$N_t = 90$	~110 ms	$N_t = 180$	~62 ms
$N_t = 100$	~126 ms	$N_t = 190$	~74 ms

## 6. Conclusions

This paper formulates the SFC scheduling problem as a 0–1 nonlinear integer programming problem in the multi-UAV edge computing network with CPU + FPGA computation architecture. A two-stage heuristic algorithm named ToRu is put forward to derive a sub-optimal solution of the problem. At the first stage, the SFCs of all tasks are scheduled to UAV edge servers in parallel based on the our proposed pairing principle between SFCs and UAVs for minimizing the completion time sum of tasks; at the second stage, a revenue maximization heuristic is adopted to schedule the arrived SFCs in a serial service method. A series of experiments were conducted to evaluate the performance of our proposal. The results show that our algorithm outperforms other benchmark algorithms in the completion time sum of tasks, the overall revenue, and the task execution success ratio.

The main limitation of ToRu algorithm lies in the fact that it is designed to realize the online long-term SFC scheduling based on the stable UAV network topology. In other words, it cannot be applied directly in the scenario where UAVs frequently join and exit. In our future work, we plan to design a supplemental algorithm with network topology prediction capability, which can help ToRu adapt to the dynamic scenario.

**Author Contributions:** Conceptualization, Y.W. and H.W.; methodology, Y.W. and X.W.; software, K.Z.; validation, J.F. and J.C.; formal analysis, Y.H.; investigation, R.J.; resources, K.Z.; data curation, K.Z.; writing—original draft preparation, Y.W.; writing—review and editing, X.W.; visualization, J.C.; supervision, H.W.; project administration, Y.H.. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the Natural Science Foundation of China under Grant No. 62171465.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Acknowledgments:** The authors would like to thank all coordinators and supervisors involved and the anonymous reviewers for their detailed comments that helped to improve the quality of this article.

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

## References

1. Chen, W.; He, R.; Wang, G.; Zhang, J.; Wang, F.; Xiong, K.; Ai, B.; Zhong, Z. Ai assisted phy in future wireless systems: Recent developments and challenges. *China Commun.* **2021**, *18*, 285–297. [\[CrossRef\]](#)
2. Sarikhani, R.; Keynia, F. Cooperative spectrum sensing meets machine learning: Deep reinforcement learning approach. *IEEE Commun. Lett.* **2020**, *24*, 1459–1462. [\[CrossRef\]](#)
3. Zheng, S.; Chen, S.; Qi, P.; Zhou, H.; Yang, X. Spectrum sensing based on deep learning classification for cognitive radios. *China Commun.* **2020**, *17*, 138–148. [\[CrossRef\]](#)
4. Xie, J.; Liu, C.; Liang, Y.-C.; Fang, J. Activity pattern aware spectrum sensing: A cnn-based deep learning approach. *IEEE Commun. Lett.* **2019**, *23*, 1025–1028. [\[CrossRef\]](#)
5. Deng, S.; Zhao, H.; Fang, W.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet Things J.* **2020**, *7*, 7457–7469. [\[CrossRef\]](#)
6. Wang, J.; Wei, X.; Fan, J.; Duan, Q.; Liu, J.; Wang, Y. Request pattern change-based cache pollution attack detection and defense in edge computing. *Digit. Commun. Netw.* **2022**. [\[CrossRef\]](#)
7. Liu, Z.; Cao, Y.; Gao, P.; Hua, X.; Zhang, D.; Jiang, T. Multi-uav network assisted intelligent edge computing: Challenges and opportunities. *China Commun.* **2022**, *19*, 258–278. [\[CrossRef\]](#)
8. Wu, W.; Zhou, F.; Wang, B.; Wu, Q.; Dong, C.; Hu, R.Q. Unmanned Aerial Vehicle Swarm-Enabled Edge Computing: Potentials, Promising Technologies, and Challenges. *IEEE Wirel. Commun.* **2022**, *29*, 78–85. [\[CrossRef\]](#)
9. Zhao, N.; Lu, W.; Sheng, M.; Chen, Y.; Tang, J.; Yu, F.R.; Wong, K.K. Uav-assisted emergency networks in disasters. *IEEE Wirel. Commun.* **2019**, *26*, 45–51. [\[CrossRef\]](#)
10. Cao, B.; Li, M.; Liu, X.; Zhao, J.; Cao, W.; Lv, Z. Many-Objective Deployment Optimization for a Drone-Assisted Camera Network. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 2756–2764. [\[CrossRef\]](#)
11. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [\[CrossRef\]](#)
12. Dong, C.; Shen, Y.; Qu, Y.; Wang, K.; Zheng, J.; Wu, Q.; Wu, F. Uavs as an intelligent service: Boosting edge intelligence for air-ground integrated networks. *IEEE Netw.* **2021**, *35*, 167–175. [\[CrossRef\]](#)
13. Behraves, R.; Harutyunyan, D.; Coronado, E.; Riggio, R. Time-sensitive mobile user association and sfc placement in mec-enabled 5g networks. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 3006–3020. [\[CrossRef\]](#)
14. Xu, Z.; Gong, W.; Xia, Q.; Liang, W.; Rana, O.F.; Wu, G. Nfv-enabled iot service provisioning in mobile edge clouds. *IEEE Trans. Mob. Comput.* **2021**, *20*, 1892–1906. [\[CrossRef\]](#)
15. Wang, Y.; Wei, X.; Wang, H.; Fan, J.; Chen, J.; Zhao, K.; Hu, Y. Joint UAV deployment, SF placement, and collaborative task scheduling in heterogeneous multi-UAV-empowered edge intelligence. *IET Commun. Early Access Artic.* **2023**. [\[CrossRef\]](#)
16. Xu, C.; Jiang, S.; Luo, G.; Sun, G.; An, N.; Huang, G.; Liu, X. The case for fpga-based edge computing. *IEEE Trans. Mob. Comput.* **2022**, *21*, 2610–2619. [\[CrossRef\]](#)
17. Li, J.; Un, K.-F.; Yu, W.-H.; Mak, P.-I.; Martins, R.P. An fpga-based energy-efficient reconfigurable convolutional neural network accelerator for object recognition applications. *IEEE Trans. Circuits Syst. Express Briefs* **2021**, *68*, 3143–3147. [\[CrossRef\]](#)
18. Yu, X.; Niu, W.; Zhu, Y.; Zhu, H. UAV-assisted cooperative offloading energy efficiency system for mobile edge computing. *Digit. Commun. Netw.* **2022**. [\[CrossRef\]](#)
19. Zhang, J.; Zhou, L.; Zhou, F.; Seet, B.-C.; Zhang, H.; Cai, Z.; Wei, J. Computation-efficient offloading and trajectory scheduling for multi-uav assisted mobile edge computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 2114–2125. [\[CrossRef\]](#)
20. Luo, Y.; Ding, W.; Zhang, B. Optimization of task scheduling and dynamic service strategy for multi-uav-enabled mobile-edge computing system. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 970–984. [\[CrossRef\]](#)
21. Wang, Y.; Ru, Z.-Y.; Wang, K.; Huang, P.-Q. Joint deployment and task scheduling optimization for large-scale mobile users in multi-uav-enabled mobile edge computing. *IEEE Trans. Cybern.* **2020**, *50*, 3984–3997. [\[CrossRef\]](#)
22. Chang, H.; Chen, Y.; Zhang, B.; Doermann, D. Multi-uav mobile edge computing and path planning platform based on reinforcement learning. *IEEE Trans. Emerg. Top. Comput. Intell.* **2022**, *6*, 489–498. [\[CrossRef\]](#)
23. Ren, T.; Niu, J.; Dai, B.; Liu, X.; Hu, Z.; Xu, M.; Guizani, M. Enabling efficient scheduling in large-scale uav-assisted mobile-edge computing via hierarchical reinforcement learning. *IEEE Internet Things J.* **2022**, *9*, 7095–7109. [\[CrossRef\]](#)
24. Xue, J.; Wu, Q.; Zhang, H. Cost optimization of UAV-MEC network calculation offloading: A multi-agent reinforcement learning method. *Ad Hoc Netw.* **2022**, *136*, 102981. [\[CrossRef\]](#)
25. Seid, A.M.; Boateng, G.O.; Mareri, B.; Sun, G.; Jiang, W. Multi-Agent DRL for Task Offloading and Resource Allocation in Multi-UAV Enabled IoT Edge Network. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4531–4547. [\[CrossRef\]](#)
26. Wu, Z.; Yang, Z.; Yang, C.; Lin, J.; Liu, Y.; Chen, X. Joint deployment and trajectory optimization in UAV-assisted vehicular edge computing networks. *J. Commun. Netw.* **2022**, *24*, 47–58. [\[CrossRef\]](#)
27. Moura, J.; Hutchison, D. Game Theory for Multi-Access Edge Computing: Survey, Use Cases, and Future Trends. *IEEE Commun. Surv. Tutor.* **2019**, *2*, 260–288. [\[CrossRef\]](#)
28. Liu, L.; Zhang, S.; Zhang, L.; Pan, G.; Yu, J. Multi-UUV Maneuvering Counter-Game for Dynamic Target Scenario Based on Fractional-Order Recurrent Neural Network. *IEEE Trans. Cybern. Access Artic.* **2022**. [\[CrossRef\]](#)

29. Asheralieva, A.; Niyato, D. Hierarchical game-theoretic and reinforcement learning framework for computational offloading in uav-enabled mobile edge computing networks with multiple service providers. *IEEE Internet Things J.* **2019**, *6*, 8753–8769. [[CrossRef](#)]
30. Wu, Q.; Chen, J.; Xu, Y.; Qi, N.; Fang, T.; Sun, Y.; Jia, L. Joint Computation Offloading, Role, and Location Selection in Hierarchical Multicoalition UAV MEC Networks: A Stackelberg Game Learning Approach. *IEEE Internet Things J.* **2022**, *9*, 18293–18304. [[CrossRef](#)]
31. Zhou, H.; Wang, Z.; Min, G.; Zhang, H. UAV-aided Computation Offloading in Mobile Edge Computing Networks: A Stackelberg Game Approach. *IEEE Internet Things J. Early Access Artic.* **2022**. [[CrossRef](#)]
32. Qu, Y.; Dai, H.; Wang, H.; Dong, C.; Wu, F.; Guo, S.; Wu, Q. Service provisioning for uav-enabled mobile edge computing. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 3287–3305. [[CrossRef](#)]
33. Wang, G.; Zhou, S.; Zhang, S.; Niu, Z.; Shen, X. SFC-Based Service Provisioning for Reconfigurable Space-Air-Ground Integrated Networks. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1478–1489. [[CrossRef](#)]
34. Li, J.; Shi, W.; Wu, H.; Zhang, S.; Shen, X. Cost-Aware Dynamic SFC Mapping and Scheduling in SDN/NFV-Enabled Space–Air–Ground-Integrated Networks for Internet of Vehicles. *IEEE Internet Things J.* **2022**, *9*, 5824–5838. [[CrossRef](#)]
35. Xia, J.; Wang, P.; Li, B.; Fei, Z. Intelligent task offloading and collaborative computation in multi-UAV-enabled mobile edge computing. *China Commun.* **2022**, *19*, 244–256. [[CrossRef](#)]
36. Liu, S.; Yang, T. Delay aware scheduling in uav-enabled ofdma mobile edge computing system. *IET Commun.* **2020**, *14*, 3203–3211. [[CrossRef](#)]
37. Tan, G.; Shui, C.; Wang, Y.; Yu, X.; Yan, Y. Optimizing the lincpack algorithm for large-scale pcie-based cpu-gpu heterogeneous systems. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 2367–2380. [[CrossRef](#)]
38. Kowsalya, T. Area and power efficient pipelined hybrid merged adders for customized deep learning framework for FPGA implementation. *Microprocess. Microsyst.* **2020**, *72*, 102906. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.