Fault-Tolerant Offloading Framework for Real-Time Applications in Mobile Edge Computing

Chuanchao Gao[1,2] , Yiyang Gao[1], Michael Yuhas[1,2] Arvind Easwaran[1,2]
[1]*College of Computing and Data Science*
[2]*Energy Research Institute @ NTU, Interdisciplinary Graduate Programme*
Nanyang Technological University, Singapore
gaoc0008@e.ntu.edu.sg, yiyang009@e.ntu.edu.sg, michaelj004@e.ntu.edu.sg, arvinde@ntu.edu.sg

***Abstract*—This paper studies a Deadline-constrained task Offloading and resource Allocation Problem (DOAP) in a backhaul-network-enabled Mobile Edge Computing (MEC) system with both bandwidth and computational resource constraints. The goal is to maximize the total inference accuracy improvement across all tasks. We formulate this problem, denoted as P, as an Integer Linear Programming (ILP) problem and show it is NP-hard via a reduction from the maximum-weight three-dimensional matching problem. To address P, we develop `FastSA`, a linear-time $\frac{1}{10}$-approximation algorithm based on the local-ratio technique. To the best of our knowledge, `FastSA` provides the first constant-factor performance guarantee for backhaul-network-enabled DOAPs. In addition, we introduce a fault detection and recovery framework that enables robust MEC operation under backhaul link and edge server failures. Experiments using profiled image classification workloads and real-world taxi traces show that `FastSA` achieves an average accuracy improvement of 25.6% over benchmark algorithms under fault-free conditions while maintaining the lowest runtime. Under fault-injected scenarios, our framework reduces performance degradation by an average of 12.8%, demonstrating its effectiveness in fault handling.**

***Index Terms*—mobile edge computing, real-time offloading, fault-tolerance, approximation algorithm**

I. Introduction

Internet of Things (IoT) has emerged as a forefront technology (e.g., Cisco predicts 500 billion IoT devices by 2030 [1]), driven by continuous advancements in hardware, software, and communication technologies (e.g., ultra-reliable low-latency communication features of 5G) [2]. Meanwhile, the development of Artificial Intelligence (AI) has spurred growing interest in deploying AI services on IoT devices, enabling applications such as path planning and object detection in autonomous vehicles [3]. These applications are typically computation-intensive and subject to strict latency requirements, posing significant challenges for resource-constrained IoT devices. To achieve reasonable execution time and energy efficiency, model compression techniques such as quantization [4] are often applied when deploying AI services on IoT devices, albeit at the cost of reduced inference accuracy.

To obtain high-quality inference results while satisfying the latency requirements of End Devices (EDs), Mobile Edge Computing (MEC) has emerged as a promising paradigm. In MEC, Edge Servers (ESs) are deployed at the network edge to provide wireless bandwidth for task offloading and computational resources for task processing. A task generated by an ED can be offloaded to a nearby ES (termed offload-ES), which is then processed on the offload-ES or further forwarded to another ES for processing (termed process-ES) via the wired backhaul network. By leveraging ESs, MEC enables the use of full-scale AI models for high-quality inference while keeping computation close to EDs. This proximity substantially reduces communication latency compared to cloud computing, making MEC well-suited for latency-sensitive applications.

Despite the advantages of MEC, the bandwidth and computational capacity of ESs can become bottlenecks as the number of tasks increases, necessitating efficient strategies for *task offloading* (i.e., determining the offload-ES and process-ES for each task) and *resource allocation* (i.e., deciding the amount of bandwidth and computational resource to allocate for each task). Jointly optimizing task offloading and resource allocation enables adaptive resource management based on both ES resource availability and the wireless channel quality.

Numerous studies have explored Deadline-constrained task Offloading and resource Allocation Problems (DOAP), which can be broadly categorized into two groups. The first group (e.g., [5]–[22]) assumes that each offloaded task is processed on its selected offload-ES. The second group (e.g., [23]–[29]) considers a more flexible setting where tasks being offloaded can be further forwarded to another ES for processing via the backhaul network; we term this group as forwarding-enabled DOAP. While the first group simplifies system design, the second offers enhanced load-balancing capability by leveraging inter-ES cooperation. The limited coverage of 5G signals further emphasizes the value of backhaul-assisted task forwarding. When requests crowd around a single ES, resource congestion may occur, whereas other ESs with available capacity remain underutilized due to wireless network coverage constraints. Despite its advantage, the forwarding-enabled DOAP remains underexplored, and most existing studies rely on heuristic approaches without theoretical performance guarantees [23]–[28]. This gap motivates the design of efficient algorithms that can achieve both computational efficiency and solution quality.

Recently, a $\frac{1-\alpha}{2}$-approximation algorithm was proposed for

This work was supported by the MoE Tier-2 grant MOE-T2EP20224-0007. The work was also supported by DesCartes, National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme.

the forwarding-enabled DOAP [29], ensuring that the achieved utility value is at least $\frac{1-\alpha}{2}$ of the optimal. Here, α represents the maximum fraction of an ES's capacity that can be allocated to a single task. Although this algorithm provides a theoretical bound, its performance guarantee depends on α, leading to large variation under practical scenarios with different α requirements. Furthermore, most existing studies assume a fault-free MEC environment, overlooking the potential impact of backhaul network or ES failures. In real deployments, failures in backhaul links or ESs are inevitable and may cause severe service disruptions in the absence of effective fault-handling mechanisms. To the best of our knowledge, no prior work on DOAP has explicitly investigated fault detection and recovery in MEC systems.

In this paper, we investigate a forwarding-enabled DOAP in MEC, considering both bandwidth and computational resource constraints, i.e., the total resource that each ES can allocate to tasks is bounded by its resource capacity for any resource type. The objective is to maximize the total accuracy improvement across all tasks. We denote this problem as $\mathbf{P}$ and formulate it as an Integer Linear Programming (ILP) problem. The maximum weight three-dimensional matching problem [30], which is NP-hard, can be reduced to a special case of $\mathbf{P}$ where at most one task can be offloaded to or processed on each ES. Hence, $\mathbf{P}$ is also NP-hard.

To address $\mathbf{P}$, we propose a linear-time $\frac{1}{10}$-approximation algorithm, `FastSA`, based on a local-ratio technique [31]. To the best of our knowledge, it is the first constant-factor approximation algorithm developed for forwarding-enabled DOAPs. In addition, we design a distributed network synchronization framework similar to the Open Shortest Path First (OSPF) Protocol [32], where each ES periodically monitors its connections with neighboring ESs and broadcasts any detected changes. This framework enables the system to promptly detect and recover from failures in backhaul links and ESs, thereby ensuring service continuity in MEC systems. The main contributions of this paper are summarized as follows:

- We address a forwarding-enabled DOAP, $\mathbf{P}$, for real-time task offloading in MEC with both bandwidth and computational resource constraints. To maximize the total accuracy improvement across all tasks, we propose a linear-time $\frac{1}{10}$-approximation algorithm, `FastSA`, based on the local-ratio technique, providing the first constant-factor theoretical guarantee for forwarding-enabled DOAPs.
- We design a distributed backhaul network synchronization framework to detect and recover from failures in backhaul links and ESs. To validate its effectiveness, we extend the open-source simulator mecRT [33] with this framework, creating the first MEC simulator that supports fault-detection and recovery for real-time task offloading.
- We evaluate `FastSA` using profiled image classification workloads and a real-world taxi trace dataset [34] under fault-free and fault-injected MEC simulations. Results show that the average accuracy improvement of `FastSA` surpasses benchmark algorithms by 25.6% in fault-free scenarios while maintaining the lowest scheduling overhead. Under fault conditions, the algorithm performance degradation reduces 12.8% on average when the proposed framework is enabled, showing its effectiveness in maintaining service robustness when a fault occurs.

TABLE I
CATEGORY OF STUDIES ON DEADLINE-CONSTRAINED TASK OFFLOADING AND RESOURCE ALLOCATION

	Decide where to offload	Decide where to offload and process
Allocation for $\mathbf{B}$ or $\mathbf{C}$ Unspecified Contention on $\mathbf{B}$ or $\mathbf{C}$	[5]–[13]$^{\diamondsuit\heartsuit}$, [14]$^{\blacklozenge\spadesuit}$	[23], [24]$^{\diamondsuit\heartsuit}$
Allocation for $\mathbf{B}$ or $\mathbf{C}$ Unspecified Contention on $\mathbf{B}$ and $\mathbf{C}$	[15], [16]$^{\diamondsuit}$	[25]$^{\blacklozenge}$
Allocation for $\mathbf{B}$ and $\mathbf{C}$ Unspecified Contention on $\mathbf{B}$ and $\mathbf{C}$	[17]–[22]	[26]–[29]

$\mathbf{B} \leftarrow$ bandwidth; $\blacklozenge \leftarrow$ unspecified allocation for $\mathbf{B}$; $\spadesuit \leftarrow$ contention on $\mathbf{B}$; $\mathbf{C} \leftarrow$ comp. res.; $\diamondsuit \leftarrow$ unspecified allocation for $\mathbf{C}$; $\heartsuit \leftarrow$ contention on $\mathbf{C}$;

II. Related Works

Due to the promising potential of MEC, problems on task offloading and resource allocation in MEC have been extensively investigated [35]–[37]. These problems are typically categorized into two types: deadline-constrained and deadline-free (e.g., response-time minimization). Depending on whether resource allocation is pre-specified, they can be further divided into (i) problems determining task offloading under fixed resource allocation and (ii) problems jointly optimizing task offloading and resource allocation. This section focuses on studies addressing deadline-constrained task offloading and resource allocation problems (DOAP).

The rows of Table I classify existing DOAP studies based on resource contention and unspecified resource types. Studies considering single-resource contention mainly focused on computational resources with unspecified computational resource allocation [5]–[13], [23], [24], whereas others examined bandwidth contention with unspecified bandwidth allocation [14]. Among studies addressing joint bandwidth and computational resource contention, some assumed one unspecified resource type [15], [16], [25], while others considered both unspecified [17]–[22], [26]–[29].

The columns of Table I further classify DOAP studies by offloading type. In the first column [5]–[22], each task can be offloaded to at most one nearby ES (multiple tasks can be offloaded to the same ES under bandwidth constraints) and is processed on the offload-ES. Thus, the decision only involves selecting the offload-ES. The second column [23]–[29] represents forwarding-enabled MEC, where ESs are interconnected via a wired backhaul network and an offloaded task can be further forwarded to another ES for processing. Hence, both the offload-ES and process-ES must be jointly determined.

This paper addresses the DOAP under the most general setting (i.e., the blue category in Table I). A few studies have investigated this setting [26]–[29]. Among them, some [26], [28] assumed each task has a fixed offload-ES, whereas others [27], [29] allowed multiple offloading candidates per task.

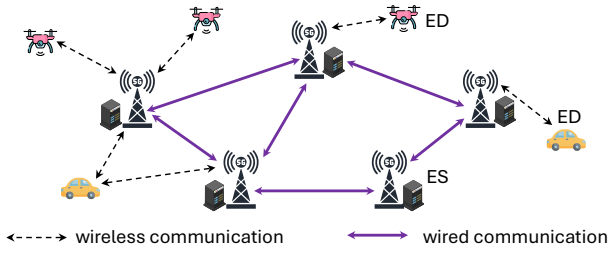

Fig. 1. An Example of the mobile edge computing system.

To solve these problems, heuristic approaches without theoretical guarantees were commonly adopted [26]–[28], while only a few works [29] provided theoretical performance bounds. Specifically, Gao *et al.* [29] developed a graph-matching-based algorithm achieving a theoretical bound of $\frac{1-\alpha}{2}$, where α denotes the maximum fraction of an ES's capacity that can be allocated to a single task. However, this bound depends on α, resulting in variation in the performance guarantee.

In contrast, this paper considers a setting where each task has multiple offloading candidates and proposes a linear-time approximation algorithm with a constant performance guarantee of $1/10$. To the best of our knowledge, this is the first constant-factor approximation algorithm for forwarding-enabled DOAPs. Moreover, none of the existing studies in Table I address fault-tolerant task offloading. To fill this gap, we design a detection and recovery framework that handles failures in both ESs and the backhaul network.

III. System Model and Problem Formulation

The MEC consists of a set of logical Edge Servers (ESs) and End Devices (EDs), as illustrated in Fig. 1. Each ES is logically associated with a 5G base station (gNB) that manages wireless bandwidth allocation during data offloading and a server that provides computational resources to host services for ED tasks. EDs and ESs communicate through the 5G network, while the ESs are interconnected via a wired backhaul network. A task can be offloaded by its ED to a nearby ES, where it is either processed locally at the associated server or forwarded to another ES through the backhaul network for processing. For ease of reading, a notation summary is provided in Table II.

Let $\mathcal{M}$ be the set of M ESs in the MEC, where $m_k \in \mathcal{M}$ denotes the k-th ES. The computational resources of each ES are measured in computing units (CUs), which abstract the execution capacity available on the underlying server hardware. For instance, with NVIDIA's Green Contexts feature [38], the execution capacity of a GPU server can be partitioned into multiple resource units, each comprising a predefined allocation of streaming multiprocessors and memory. Let C_k be the total CUs available at m_k, and C=$\max\{C_k|m_k \in \mathcal{M}\}$ be the maximum number of CUs that any ES can provide.

Orthogonal Frequency Division Multiple Access, a core technology in 5G networks, divides a wireless channel into multiple orthogonal subcarriers, with every 12 subcarriers forming a resource block, the smallest unit of bandwidth that can be allocated to users. Since PHY/MAC-layer resource block scheduling is performed by the gNB, we define a bandwidth unit (BU) as an abstraction of the PHY/MAC-layer resource block exposed to high-level control. Let B_k be the total BUs available at ES m_k, and B=$\max\{B_k|m_k \in \mathcal{M}\}$ be the maximum number of BUs that any ES can offer.

In this work, we focus on high-level offloading, forwarding, and processing decisions. Accordingly, we abstract the wireless transmission capacity available at gNBs as BUs, while direct control of PHY/MAC-layer resource block scheduling remains internal to the gNB. Similar abstraction is applied to computational resources at servers, which are represented at a higher level without exposing low-level execution scheduling.

Let $\mathcal{N}$ denote the set of N tasks pending to offload, where $n_i \in \mathcal{N}$ represents the i-th task. Let $\mathcal{M}_i$ denote the set of ESs accessible by task n_i's ED, i.e., the ESs where task n_i can be offloaded. Since these tasks typically operate periodically, task n_i has a period T_i, measured in seconds, which also serves as its implicit deadline. We focus on discriminative inference tasks (e.g., classification and detection), for which the output size is negligible compared to the input size.

For many computation-intensive tasks, local processing performance is often constrained by the hardware limitations of EDs. For example, deploying AI applications on resource-constrained EDs often requires model compression techniques to reduce computational and memory demands, usually at the cost of reduced model accuracy. In such cases, EDs can offload tasks to ESs to achieve more accurate inference results. However, the potential unavailability of ES resources due to faults or uncertainty in the wireless network necessitates the ability to process tasks locally to ensure operation safety. Let d_i^{loc} denote the local execution time of task n_i; besides, we assume $d_i^{\text{loc}} \leq T_i$ to ensure feasible local processing.

We consider two task offloading modes. In the first mode ψ_1, a task supports multiple execution stop points, where earlier termination yields lower-quality results (e.g., the anytime YOLO for object detection [39]). Task n_i first executes locally for d_i^{loc} after its release, and the intermediate data are then offloaded to the ES for continued execution to improve accuracy. The ES result is adopted if it is returned before T_i; otherwise, the locally obtained result is used. In the second mode ψ_2, the task is offloaded immediately after its release while also being executed locally in parallel. The ES result is adopted if it arrives before T_i; otherwise, the locally computed result is used. Let θ_i^1 denote the size of the intermediate data offloaded in mode ψ_1, and let θ_i^2 denote the size of the input data offloaded in mode ψ_2. Let $\Psi_i \subseteq \{\psi_1, \psi_2\}$ be the set of offloading modes that task n_i can choose.

For a given task (e.g., image classification), multiple AI models may be available that perform the same function but differ in inference accuracy and execution time. Each ED is assumed to adopt an offline strategy to select a fixed local model for each task. Let $\mathcal{A}_i$ denote the set of A_i candidate models available on ESs that can improve the inference accuracy of task n_i, and let A=$\max\{A_i \mid n_i \in \mathcal{N}\}$. When task n_i is granted to offload, an appropriate model is selected according to its latency requirement and the available resources. For a

selected model $a \in \mathcal{A}_i$ deployed on an ES to serve n_i, let u_{ia} denote the *accuracy improvement* (gain), defined as the difference in accuracy between the model a and the local model used by n_i. Specifically, when a job instance of n_i is offloaded to an ES and processed using model a, *and its result is returned before the deadline*, the obtained accuracy improvement for that job instance equals u_{ia}.

In 5G networks, each ED periodically transmits Sounding Reference Signals (SRSs) to nearby gNBs. Each gNB utilizes the received SRS to estimate the wireless channel quality between the ED and the gNB [40]. The channel quality can be affected by various factors, including physical distance, transmission power, ED mobility, obstacles, and interference caused by other ED transmissions [41]. Based on the estimated channel quality, each gNB selects an appropriate Modulation and Coding Scheme (MCS) to ensure reliable data transmission. The chosen MCS determines the number of bytes that a resource block can carry within a fixed time interval, thereby defining the achievable data offloading rate between an ED and the gNB, which is exposed to the high-level control plane. Let μ_{ik} denote the data offloading rate (in bytes per BU per second) when task n_i is offloaded to the gNB associated with ES m_k, as determined by the latest SRS feedback.

Mid-band 5G gNB typically covers 500–1000 meters in urban areas. An ED moving at 60 km/h (the average vehicle speed in urban areas) travels about 1.67 meters in 100 ms, which is negligible compared to the gNB coverage range. Thus, consistent with prior studies [42], [43], we assume that the wireless channel quality (and hence μ_{ik}), as estimated from uplink SRS, remains constant for the duration in which a task's data is offloaded (typically, of the order of 10ms).

In this MEC, we consider a wired *backhaul network* interconnecting the ESs. To support predictable and parallel data transmission across the backhaul network, we assume that deterministic Ethernet technologies, such as Time-Sensitive Networking (TSN), are available to provide bounded latency on wired links [44]. In addition, Software-Defined Networking (SDN) [45] can be employed to configure routing paths and to provision virtual links with predefined bandwidth allocations for forwarding a task's data. Wired backhaul links typically offer significantly higher bandwidth than wireless channels. For example, a 5G channel operating in the sub-6GHz band provides a bandwidth capacity of approximately 100MHz [46], whereas a commercial optical transmission system can offer bandwidths up to 4.5THz [47]. Given the substantial bandwidth available in wired networks, we ignore communication contention within the backhaul network, and each virtual link can be abstracted as providing a constant data transmission rate, denoted as β. Let h_{jk} denote the minimum number of link hops between ESs m_j and m_k.

Timing Model. Suppose task n_i is offloaded to ES $m_j \in \mathcal{M}_i$ with BU allocation $b_{ij} \in \{1, ..., B_j\}$, and is forwarded to ES m_k for processing with CU allocation $c_{ik} \in \{1, ..., C_k\}$ and candidate model $a \in \mathcal{A}_i$.

In offloading mode ψ_1, task n_i first executes locally for d_i^{loc} after its job instance is released. The intermediate data θ_i^1 is then offloaded to ES m_j. Let d_{ij}^{off} denote the offloading time, where $d_{ij}^{\text{off}} = \theta_i^1/(\mu_{ij} \cdot b_{ij})$. The forwarding time from ES m_j to ES m_k is given as $d_{ijk}^{\text{fwd}} = h_{jk} \cdot \theta_i^1/\beta$. Let d_{ik}^{pro} denote the processing time of task n_i on ES m_k, which is a function of the allocated CUs c_{ik}, the hardware type of the server associated with ES m_k, and the selected candidate model a. Since the result data size is much smaller than the input for discriminative inference tasks, the result forwarding time from m_k to m_j is negligible. Besides, the result can be downloaded from m_j using one BU within one time unit. Since the downlink bandwidth in 5G is generally larger than the uplink capacity and each offloaded task is guaranteed at least one BU in the uplink, we omit bandwidth contention for result downloading and define the result return time as a fixed constant d^{rr}. The deadline constraint in mode ψ_1 is given as

$$d_i^{\text{loc}} + d_{ij}^{\text{off}} + d_{ijk}^{\text{fwd}} + d_{ik}^{\text{pro}} + d^{\text{rr}} \leq T_i. \tag{1}$$

In offloading mode ψ_2, the job instance of task n_i is offloaded immediately after release. Thus, $d_{ij}^{\text{off}} = \theta_i^2/(\mu_{ij} \cdot b_{ij})$ and $d_{ijk}^{\text{fwd}} = h_{jk} \cdot \theta_i^2/\beta$. The processing time d_{ik}^{pro} and result return time d^{rr} remain the same as in mode ψ_1. The deadline constraint in ψ_2 is expressed as

$$d_{ij}^{\text{off}} + d_{ijk}^{\text{fwd}} + d_{ik}^{\text{pro}} + d^{\text{rr}} \leq T_i. \tag{2}$$

This paper aims to determine the offloading strategy for each task n_i (including offloading mode ψ, offload-ES m_j, BU allocation b_{ij}, process-ES m_k, CU allocation c_{ik}, and candidate model a) to maximize the total accuracy improvement per second. The optimization must satisfy the *offloading constraints* (i.e., $\psi \in \Psi_i$, $m_j \in \mathcal{M}_i$, and $b_{ij} \leq B_j$), the *processing constraints* (i.e., $c_{ik} \leq C_k$, $a \in \mathcal{A}_i$), the *task*

TABLE II
NOTATION (PARAMETERS AND VARIABLES)

symb.	definition
$\mathcal{M}$	set of ESs, where $\lvert\mathcal{M}\rvert = M$
C_k	total CUs of ES m_k; $C = \max\{C_k \lvert m_k \in \mathcal{M}\}$
B_k	total BUs of ES m_k; $B = \max\{B_k \lvert m_k \in \mathcal{M}\}$
$\mathcal{N}$	set of tasks, where $\lvert\mathcal{N}\rvert = N$
T_i	period (deadline) of task $n_i \in \mathcal{N}$
$\mathcal{M}_i$	set of ESs accessible by vehicle v_i
d_i^{loc}	local execution time of task $n_i \in \mathcal{N}$
Ψ_i	set of offloading modes of task n_i, $\Psi_i \subseteq \{\psi_1, \psi_2\}$
θ_i^1	intermediate data size of n_i (for offloading mode ψ_1)
θ_i^2	input data size of n_i (for offloading mode ψ_2)
$\mathcal{A}_i$	set of candidate models for task n_i if offloaded
u_{ia}	improved accuracy for n_i if choosing model $a \in \mathcal{A}_i$
μ_{ik}	offloading rate per BU per second from n_i to m_k
β	data rate in the virtual link of the backhaul network
h_{jk}	hop count in shortest path between ESs m_j and m_k
ℓ	$\ell \triangleq \langle n_i, m_j, m_k, b_\ell, c_\ell, \psi, a\rangle$, a service instance
$u(\ell)$	improved accuracy by serving a task following ℓ
$\mathcal{L}$	set of all feasible service instances
$\mathcal{L}_i^{\text{task}}$	$\mathcal{L}_i^{\text{task}} \subseteq \mathcal{L}$, set of service instances for task n_i
$\mathcal{L}_j^{\text{off}}$	$\mathcal{L}_j^{\text{off}} \subseteq \mathcal{L}$, set of service instances offloaded to ES m_j
$\mathcal{L}_k^{\text{pro}}$	$\mathcal{L}_k^{\text{pro}} \subseteq \mathcal{L}$, set of service instances processed on ES m_k
z_ℓ	binary selection variable for service instance ℓ

deadline constraints (Eq. (1) or Eq. (2)), and the *ES resource constraints* (i.e., the total allocated resources on each ES cannot exceed its available capacity for any resource type). We denote this optimization problem as $\mathbf{P}$.

The direct formulation of $\mathbf{P}$ is nonlinear due to deadline constraints (1) and (2), making it difficult to solve directly. Here, we introduce a more concise ILP formulation for $\mathbf{P}$ based on service instances (defined in Definition 1) of all tasks.

Definition 1 (Service Instance)**.** *A service instance of task n_i is defined as $\ell \triangleq \langle n_i, m_j, m_k, b_\ell, c_\ell, \psi, a\rangle$, representing that task n_i selects offloading mode ψ, and it is offloaded to ES m_j with BU allocation b_ℓ; besides, it is forwarded to ES m_k for processing with CU allocation c_ℓ and candidate model a. Moreover, ℓ needs to satisfy (i) the offloading constraints (i.e., $\psi \in \Psi_i$, $m_j \in \mathcal{M}_i$, and $b_\ell \le B_j$), (ii) the processing constraints (i.e., $c_\ell \le C_k$, $a \in \mathcal{A}_i$), and (iii) the **deadline** constraints (i.e., Eq.* (1) *for $\psi = \psi_1$ or Eq.* (2) *for $\psi = \psi_2$).*

Let $\ell \triangleq \langle n_i, m_j, m_k, b_\ell, c_\ell, \psi, a\rangle$ denote a service instance of task n_i. Given task n_i and the selected model a, the resulting accuracy improvement per second is u_{ia}/T_i (if all job instances of n_i can be successfully offloaded and meet deadlines); we denote this utility value as $u(\ell)$, where $u(\ell) = u_{ia}/T_i$. Since both bandwidth and computational resource allocations are discrete, we can enumerate all feasible service instances for each task. Let $\mathcal{L}$ denote the set of all feasible service instances for all tasks in $\mathcal{N}$. We define $\mathcal{L}_i^{\text{task}} \subseteq \mathcal{L}$ as the subset of service instances corresponding to task n_i, $\mathcal{L}_j^{\text{off}} \subseteq \mathcal{L}$ as the subset of service instances in which the task is offloaded to ES m_j, and $\mathcal{L}_k^{\text{pro}} \subseteq \mathcal{L}$ as the subset of service instances in which the task is processed on ES m_k.

The MEC comprises N tasks and M ESs, where each ES provides up to B BUs and C CUs. Each task can adopt up to two offloading modes and select from A candidate models. For each bandwidth allocation, only the minimum required computational allocation needs to be determined to satisfy the task deadline (and vice versa). Consequently, the total number of service instances in $\mathcal{L}$ is at most $(2NM^2A \cdot \min\{B, C\})$. For each service instance $\ell \in \mathcal{L}$, let $z_\ell \in \{0, 1\}$ be its selection variable, where $z_\ell = 1$ indicates that ℓ is chosen as the service instance assigned to the corresponding task in the final solution. Then, we formulate $\mathbf{P}$ as follows.

$$(\mathbf{P}) \quad \max \textstyle\sum_{\ell \in \mathcal{L}} u(\ell) \cdot z_\ell \tag{3a}$$

subject to:

$$\textstyle\sum_{\ell \in \mathcal{L}_j^{\text{off}}} b_\ell \cdot z_\ell \le B_j,\ \forall m_j \in \mathcal{M} \tag{3b}$$

$$\textstyle\sum_{\ell \in \mathcal{L}_k^{\text{pro}}} c_\ell \cdot z_\ell \le C_k,\ \forall m_k \in \mathcal{M} \tag{3c}$$

$$\textstyle\sum_{\ell \in \mathcal{L}_i^{\text{task}}} z_\ell \le 1,\ \forall n_i \in \mathcal{N} \tag{3d}$$

$$z_\ell \in \{0, 1\},\ \forall \ell \in \mathcal{L} \tag{3e}$$

Eqs. (3b) and (3c) are the ES resource constraints, while Eq. (3d) ensures that at most one service instance is selected for each task. The offloading constraints, processing constraints, and *task deadline constraints* are implicitly enforced through the definition of service instances (Definition 1); thus, only ES resource constraints need to be explicitly enforced in the ILP formulation. As a result, an offloaded task contributes to the total accuracy improvement only if its deadline is met.

Problem Complexity. Given a solution to $\mathbf{P}$, we can efficiently verify whether it satisfies all the constraints in polynomial time, which indicates that the decision version of $\mathbf{P}$ belongs to the NP class. Furthermore, a well-known NP-Hard problem, the maximum weight three-dimensional matching problem [30], can be reduced to a special case of $\mathbf{P}$, where each task has only one offloading mode and one candidate model, and each ES allocates its entire bandwidth to the tasks offloaded to it and all its computational resources to the tasks processed on it. Since this simplified instance of $\mathbf{P}$ is NP-Hard, the original problem $\mathbf{P}$ is also NP-Hard.

IV. Fast Separate Assignment Algorithm

In this section, we present the Fast Separate Assignment Algorithm (`FastSA`) for problem $\mathbf{P}$ (Eqs. (3a)$\sim$(3e)). Due to task forwarding, a service instance may compete with different sets of service instances for bandwidth and computational resources, which complicates the enforcement of ES resource constraints. `FastSA` addresses this challenge by exploiting the structure of service instances and separating them into two disjoint groups, which are scheduled independently to simplify resource contention. The algorithm then returns the schedule with the higher total accuracy improvement.

We then show that `FastSA` achieves a $\frac{1}{10}$-approximation guarantee for $\mathbf{P}$ in Subsection IV-A, and present a linear-time implementation with respect to the size of $\mathcal{L}$ (i.e., $\mathcal{O}(|\mathcal{L}|)$) in Subsection IV-B. To the best of our knowledge, `FastSA` is the first constant-factor approximation algorithm for $\mathbf{P}$.

A. Fast Separate Assignment Algorithm (FastSA)

Based on the allocated resource, we divide all service instances in $\mathcal{L}$ into four categories, LL, LH, HL, and HH, and denote the set of service instances in $\mathcal{L}$ that belong to each category as $\mathcal{L}_{\text{LL}}$, $\mathcal{L}_{\text{LH}}$, $\mathcal{L}_{\text{HL}}$, and $\mathcal{L}_{\text{HH}}$, respectively. For a service instance $\ell \triangleq \langle n_i, m_j, m_k, b_\ell, c_\ell, \psi, a\rangle$,

- ℓ is in category LL if $b_\ell \le 1/2 \cdot B_j$ and $c_\ell \le 1/2 \cdot C_k$;
- ℓ is in category LH if $b_\ell \le 1/2 \cdot B_j$ and $c_\ell > 1/2 \cdot C_k$;
- ℓ is in category HL if $b_\ell > 1/2 \cdot B_j$ and $c_\ell \le 1/2 \cdot C_k$;
- ℓ is in category HH if $b_\ell > 1/2 \cdot B_j$ and $c_\ell > 1/2 \cdot C_k$.

`FastSA` (Algorithm 1) divides $\mathcal{L}$ into two sets: $\mathcal{L}_{\text{LL}} \cup \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$ and $\mathcal{L}_{\text{LH}}$. Let $\mathbb{L}_1$ be the vector of $\mathcal{L}_{\text{LL}} \cup \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$ that is sorted in the order of LL, HL, and HH categories. Let $\mathbb{L}_2$ be the vector of $\mathcal{L}_{\text{LH}}$. `FastSA` first calls function `Assign`$(\mathbf{u}, \mathbb{L}_1)$ to obtain a scheduling solution $\mathcal{S}_1$ for the combined set $\mathcal{L}_{\text{LL}} \cup \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$ (line 2), where $\mathbf{u}$ denotes the vector of $u(\ell)$ for all $\ell \in \mathcal{L}$. It then invokes `Assign`$(\mathbf{u}, \mathbb{L}_2)$ to derive a scheduling solution $\mathcal{S}_2$ for the remaining set $\mathcal{L}_{\text{LH}}$ (line 3). Finally, `FastSA` outputs the solution with the higher total accuracy improvement. `Assign` is a recursive function (invoked recursively in line 10). At each recursion level, `FastSA` updates the accuracy values of the remaining service instances (line 9). For clarity, we denote the *updated accuracy value* of service instance ℓ in the r-th level as $w^r(\ell)$, while $u(\ell)$ represents the *original accuracy value*. Let $\mathbf{w}^r$ denote

Algorithm 1: Fast Separate Assignment (`FastSA`)

1 Let $\mathbb{L}_1$ be the vector of $\mathcal{L}_{\text{LL}} \cup \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$ and is sorted in the order of LL, HL, and HH categories. Let $\mathbb{L}_2$ be the vector of $\mathcal{L}_{\text{LH}}$;
2 $\mathcal{S}_1 \leftarrow$ `Assign` $(\mathbf{u}, \mathbb{L}_1)$;
3 $\mathcal{S}_2 \leftarrow$ `Assign` $(\mathbf{u}, \mathbb{L}_2)$;
4 **if** $u(\mathcal{S}_1) \geq u(\mathcal{S}_2)$ **then return** $\mathcal{S}_1$ **else return** $\mathcal{S}_2$;
5 `Assign` $(\mathbf{w}^r, \mathbb{F})$**:**
6 Remove all instances ℓ from $\mathbb{F}$ with $w^r(\ell) \leq 0$;
7 **if** $\mathbb{F} = \emptyset$ **then return** $\mathcal{S}^r = \emptyset$;
8 Suppose $\ell^r \triangleq \langle n_i, m_j, m_k, b_{\ell^r}, c_{\ell^r}, \psi, a \rangle$ is the leftmost service instance in $\mathbb{F}$;
9 Decompose $\mathbf{w}^r$ into $\mathbf{w}_1^r$ and $\mathbf{w}_2^r$ (i.e., $\mathbf{w}^r$=$\mathbf{w}_1^r$+$\mathbf{w}_2^r$) such that $\forall \ell \in \mathbb{F}, w_1^r(\ell) =$

$$w^r(\ell^r) \cdot \begin{cases} 1 & \text{if } \ell \in \mathcal{L}_i^{\text{task}} \\ 2(\frac{b_\ell}{B_j} + \frac{c_\ell}{C_k}) & \text{if } \ell \in \mathcal{L}_j^{\text{off}} \cap \mathcal{L}_k^{\text{pro}} \setminus \mathcal{L}_i^{\text{task}} \\ 2\frac{b_\ell}{B_j} & \text{if } \ell \in \mathcal{L}_j^{\text{off}} \setminus (\mathcal{L}_k^{\text{pro}} \cup \mathcal{L}_i^{\text{task}}) \\ 2\frac{c_\ell}{C_k} & \text{if } \ell \in \mathcal{L}_k^{\text{pro}} \setminus (\mathcal{L}_j^{\text{off}} \cup \mathcal{L}_i^{\text{task}}) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

10 $\mathcal{S}^{r+1} \leftarrow$ `Assign` $(\mathbf{w}_2^r, \mathbb{F})$;
11 **if** $\mathcal{S}^{r+1} \cup \{\ell^r\}$ *is feasible* **then return** $\mathcal{S}^r \leftarrow \mathcal{S}^{r+1} \cup \{\ell^r\}$ **else return** $\mathcal{S}^r \leftarrow \mathcal{S}^{r+1}$;

the vector of $w^r(\ell)$. For any service instance set $\mathcal{S}$, we define $w^r(\mathcal{S}) = \sum_{\ell \in \mathcal{S}} w^r(\ell)$ and $u(\mathcal{S}) = \sum_{\ell \in \mathcal{S}} u(\ell)$.

In line 9, the accuracy value $w^r(\ell)$ for each remaining ℓ is decomposed into $w_1^r(\ell)$ and $w_2^r(\ell)$ based on Eq. (4) and the chosen ℓ^r in line 8. The first four conditions in Eq. (4) correspond to service instances that (i) belong to the same task as ℓ^r, (ii) are offloaded to and processed on the same ESs as ℓ^r, (iii) are only offloaded to the same ES as ℓ^r, and (iv) are only processed on the same ES as ℓ^r. The intuition is *selecting ℓ^r reduces the opportunity for other service instances to be selected*. Specifically, $w_1^r(\ell)$ estimates the impact on ℓ if ℓ^r is selected, whereas $w_2^r(\ell)$ measures the marginal gain with respect to considered service instances in previous recursive levels. $\mathbf{w}_2^r$ is then used as the input accuracy vector $\mathbf{w}^{r+1}$ for the next recursive level (line 10). Based on the solution $\mathcal{S}^{r+1}$ returned from the $(r+1)$-th level, ℓ^r is included in the solution $\mathcal{S}^r$ if it satisfies the feasibility constraints (3b)–(3d) (line 11).

In the following, we first show that `FastSA` achieves a $\frac{1}{5}$-approximation for $\mathbf{P}$ when only the set $\mathcal{L}_{\text{LL}} \cup \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$ is considered. We then prove that `FastSA` also achieves a $\frac{1}{5}$-approximation for $\mathbf{P}$ when considering only $\mathcal{L}_{\text{LH}}$. Finally, we show that `FastSA` is a $\frac{1}{10}$-approximation algorithm for $\mathbf{P}$.

Theorem 1. *Let $\mathcal{Q}_1$ be an optimal solution of $\mathbf{P}$ when only considering the set $\mathcal{L}_{LL} \cup \mathcal{L}_{HL} \cup \mathcal{L}_{HH}$. Let $\mathcal{S}_1$ be the solution returned in line 2 of* `FastSA`*. Then, we have $u(\mathcal{S}_1) \geq \frac{1}{5} u(\mathcal{Q}_1)$.*

Proof. Let the deepest recursive level of `Assign` be level δ (i.e., when $\mathbb{F} = \emptyset$ in line 7), and the initial call be level 1. We use simple induction to prove that $w^r(\mathcal{S}^r) \geq \frac{1}{5} w^r(\mathcal{Q}_1)$ for $r = \delta, \delta - 1, ..., 1$. Since $\mathbf{w}^1 = \mathbf{u}$ and $\mathcal{S}^1$ is the solution returned in line 2 of `FastSA`, this theorem can thus be proved.

Base Case ($r = \delta$): When $r = \delta$, $\mathbb{F} = \emptyset$ after removing all service instances with non-positive accuracy value (line 6), implying that $w^r(\ell) \leq 0, \forall \ell \in \mathbb{F}$, before line 6 is executed. The returned $\mathcal{S}^r = \emptyset$, so $w^r(\mathcal{S}^r) \geq w^r(\mathcal{Q}_1) \geq \frac{1}{5} w^r(\mathcal{Q}_1)$.

Inductive Step ($r < \delta$): When $r < \delta$, we show that if $w^{r+1}(\mathcal{S}^{r+1}) \geq \frac{1}{5} w^{r+1}(\mathcal{Q}_1)$ holds, then $w^r(\mathcal{S}^r) \geq \frac{1}{5} w^r(\mathcal{Q}_1)$. Here, $\mathcal{S}^{r+1}$ is the solution returned in line 10 from the lower level. Suppose $w^{r+1}(\mathcal{S}^{r+1}) \geq \frac{1}{5} w^{r+1}(\mathcal{Q}_1)$. Since we assign $\mathbf{w}_2^r$ as the accuracy vector for the lower level in line 10, $\mathbf{w}_2^r = \mathbf{w}^{r+1}$. Then, we have $w_2^r(\mathcal{S}^{r+1}) \geq \frac{1}{5} w_2^r(\mathcal{Q}_1)$. Let $\ell^r \triangleq \langle n_i, m_j, m_k, b_{\ell^r}, c_{\ell^r}, \psi, a \rangle$ (line 8). Based on the first condition in Eq. (4), $w_1^r(\ell^r) = w^r(\ell^r)$, so $w_2^r(\ell^r) = 0$; thus,

$$w_2^r(\mathcal{S}^r) = w_2^r(\mathcal{S}^{r+1}) \geq \tfrac{1}{5} w_2^r(\mathcal{Q}_1). \tag{5}$$

If ℓ^r can be added to $\mathcal{S}^r$ in line 11, $w_1^r(S^r) \geq w_1^r(\ell^r) = w^r(\ell^r)$. If ℓ^r cannot be added to $\mathcal{S}^r$ in line 11 due to constraint (3d) (i.e., at most one instance of the same task can be selected), one $\ell \in \mathcal{L}_i^{\text{task}}$ must already exist in $\mathcal{S}^{r+1}$. Based on the first condition in Eq. (4), $w_1^r(S^r) = w_1^r(S^{r+1}) \geq w^r(\ell^r)$.
If ℓ^r cannot be added to $\mathcal{S}^r$ in line 11 due to the ES resource constraints (3b) or (3c), we proceed with the analysis based on the category of ℓ^r. Given the sorted vector $\mathbb{L}_1$ and the "leftmost" selection strategy in line 8, the service instances in line 8 are chosen in the order of LL, HL, and HH categories. Based on lines 10–11, **adding ℓ^r into the solution $\mathcal{S}^r$ occurs in the reverse category order**, i.e., HH, HL, and LL. Thus, we consider the following three cases:

- Case 1: $\ell^r \in \mathcal{L}_{\text{HH}}$. In this case, $\mathcal{S}^{r+1}$ is a subset of $\mathcal{L}_{\text{HH}}$. Since ℓ^r cannot be added into $\mathcal{S}^{r+1}$ due to ES resource constraint, there must exist one $\ell \in \mathcal{S}^{r+1}$ in which the task is offloaded to ES m_j or processed on ES m_k (maybe both). By definition, $\frac{b_\ell}{B_j} > \frac{1}{2}$ and $\frac{c_\ell}{C_k} > \frac{1}{2}$ for all $\ell \in \mathcal{L}_{\text{HH}}$. Based on the second, third, or fourth condition in Eq. (4), we have $w_1^r(S^r) = w_1^r(S^{r+1}) \geq w^r(\ell^r)$.
- Case 2: $\ell^r \in \mathcal{L}_{\text{HL}}$. In this case, $\mathcal{S}^{r+1}$ is a subset of $\mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$. By definition, $\frac{b_\ell}{B_j} > \frac{1}{2}$ and $\frac{c_\ell}{C_k} \leq \frac{1}{2}$ for all $\ell \in \mathcal{L}_{\text{HL}}$. If ℓ^r cannot be added into $\mathcal{S}^r$ due to the ES bandwidth constraint, there must exist one service instance $\ell \in \mathcal{S}^{r+1}$ in which the task is offloaded to ES m_j. As $\frac{b_\ell}{B_j} > \frac{1}{2}$ for all $\ell \in \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$, based on the second or third condition in Eq. (4), $w_1^r(S^r) = w_1^r(S^{r+1}) \geq w^r(\ell^r)$. If ℓ^r cannot be added into $\mathcal{S}^{r+1}$ due to the ES computational resource constraint, at least $\frac{1}{2}C_k$ amount of ES m_k's computational resource have been allocated to service instances in $\mathcal{S}^{r+1}$. Based on the second or fourth condition in Eq. (4), $w_1^r(S^r) = w_1^r(S^{r+1}) \geq w^r(\ell^r)$.
- Case 3: $\ell^r \in \mathcal{L}_{\text{LL}}$. Since $\frac{b_\ell}{B_j} \leq \frac{1}{2}$ and $\frac{c_\ell}{C_k} \leq \frac{1}{2}$ for all $\ell \in \mathcal{L}_{\text{LL}}$, if ℓ^r cannot be added into $\mathcal{S}^r$ due to ES resource constraints, at least $\frac{1}{2}B_j$ amount of ES m_j's bandwidth or $\frac{1}{2}C_k$ amount of ES m_k's computational resource have been allocated to service instances in $\mathcal{S}^{r+1}$ (maybe both); based on the second to forth conditions in Eq. (4), we have $w_1^r(S^r) = w_1^r(S^{r+1}) \geq w^r(\ell^r)$.

In summary, regardless of whether ℓ^r can be added to $\mathcal{S}^{r+1}$ in line 11, $w_1^r(S^r) \geq w^r(\ell^r)$. Meanwhile, based on Eq. (4), the value $w_1^r(\mathcal{Q}_1)$ can be *maximized* if the optimal solution $\mathcal{Q}_1$ contains one service instance in $\mathcal{L}_i^{\text{task}} \setminus (\mathcal{L}_j^{\text{off}} \cup \mathcal{L}_k^{\text{pro}})$, and all bandwidth of ES m_j (i.e., B_j) and all computational resource of ES m_k (i.e., C_k) are allocated to service instances in $(\mathcal{L}_j^{\text{off}} \cup \mathcal{L}_k^{\text{pro}}) \backslash \mathcal{L}_i^{\text{task}}$ (only these service instances have non-zero $w_1^r(\ell)$); in this case, $w_1^r(\mathcal{Q}_1) \leq w^r(\ell^r) + 4w^r(\ell^r) = 5w^r(\ell^r)$. Hence,

$$w_1^r(S^r) \geq \tfrac{1}{5} \cdot w_1^r(\mathcal{Q}_1). \tag{6}$$

Given Eq. (5), Eq. (6), and $\mathbf{w}^r = \mathbf{w}_1^r + \mathbf{w}_2^r$, we have $w^r(S^r) \geq \frac{1}{5} w^r(\mathcal{Q}_1)$. Combining the results in the base case and the inductive step, we have $w^r(S^r) \geq \frac{1}{5} w^r(\mathcal{Q}_1)$ for $r = \delta, ..., 1$ based on simple induction. □

Theorem 2. *Let $\mathcal{Q}_2$ be an optimal solution of $\mathbf{P}$ when only considering the set $\mathcal{L}_{LH}$, and $\mathcal{S}_2$ be the solution returned in line 3 of* `FastSA`*. Then, $u(\mathcal{S}_2) \geq \frac{1}{5} u(\mathcal{Q}_2)$.*

Proof. The proof of this theorem follows a similar structure to that of Theorem 1. The key difference is that when ℓ^r cannot be added to $\mathcal{S}^r$ in line 11 due to constraints (3b) or (3c) (i.e., the ES resource limitations), only one case (i.e., analogous to the Case 2 analysis in Theorem 1) needs to be considered. The complete proof is omitted here. □

Combining the results from both Theorem 1 and Theorem 2, we have the following conclusion.

Theorem 3. `FastSA` *is a $\frac{1}{10}$-approximation guarantee for $\mathbf{P}$.*

Proof. Let $\mathcal{S}$ be the solution returned by `FastSA`, $\mathcal{S}_1$ be the solution returned in line 2 of `FastSA`, and $\mathcal{S}_2$ be the solution returned in line 3 of `FastSA`. Let $\mathcal{Q}$ be the optimal solution of $\mathbf{P}$, $\mathcal{Q}_1$ be the optimal solution of $\mathbf{P}$ when only considering service instances in $\mathcal{L}_{\text{LL}} \cup \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$, and $\mathcal{Q}_2$ be the optimal solution of $\mathbf{P}$ when only considering service instances in $\mathcal{L}_{\text{LH}}$.

Any solution of $\mathbf{P}$ can be divided into a subset of $\mathcal{L}_{\text{LL}} \cup \mathcal{L}_{\text{HL}} \cup \mathcal{L}_{\text{HH}}$ and a subset of $\mathcal{L}_{\text{LH}}$. Therefore, $u(\mathcal{Q}) \leq u(\mathcal{Q}_1) + u(\mathcal{Q}_2)$. Besides, $u(\mathcal{S}) = \max\{u(\mathcal{S}_1), u(\mathcal{S}_2)\}$. Based on Theorem 1 and Theorem 2, we have $u(\mathcal{Q}_1) \leq 5u(\mathcal{S}_1)$ and $u(\mathcal{Q}_2) \leq 5u(\mathcal{S}_2)$. Then, we have $u(\mathcal{Q}) \leq u(\mathcal{Q}_1) + u(\mathcal{Q}_2) \leq 5u(\mathcal{S}_1) + 5u(\mathcal{S}_2) \leq 10 \cdot \max\{u(\mathcal{S}_1), u(\mathcal{S}_2)\} = 10u(\mathcal{S})$. □

In `FastSA`, the function `Assign` has at most $|\mathcal{L}|$ recursive levels (i.e., the size of $\mathcal{L}$), and the accuracy values of at most $|\mathcal{L}|$ service instances are updated in each recursive level. Thus, the time complexity of `FastSA` is $\mathcal{O}(|\mathcal{L}|^2)$. In Subsection IV-B, we show how to implement `FastSA` with a linear runtime complexity with respect to the size of $\mathcal{L}$, i.e., $\mathcal{O}(|\mathcal{L}|)$.

B. Linear-Time Implementation for FastSA

Let $\ell^r \triangleq \langle n_i, m_j, m_k, b_{\ell^r}, c_{\ell^r}, \psi, a \rangle$ denote the service instance considered in the r-th recursive level of the function `Assign` (line 8). According to Eq. (4), the accuracy value of ℓ^r is reduced only when service instances in $\mathcal{L}_i^{\text{task}} \cup \mathcal{L}_j^{\text{off}} \cup \mathcal{L}_k^{\text{pro}}$ are processed in earlier recursive levels $s < r$. Let $\tilde{u}(\ell^r)$ denote the total accuracy value *reduction* for ℓ^r when it is considered, i.e., $w^r(\ell^r) = u(\ell^r) - \tilde{u}(\ell^r)$. Based on the decomposition rule in Eq. (4), $\tilde{u}(\ell^r)$ can be expressed as

$$\tilde{u}(\ell^r) = U_i^{\text{task}} + 2 \cdot U_j^{\text{off}} \cdot \tfrac{b_{\ell^r}}{B_j} + 2 \cdot U_k^{\text{pro}} \cdot \tfrac{c_{\ell^r}}{C_k}, \tag{7}$$

where U_i^{task}, U_j^{off}, and U_k^{pro} represent the reduced accuracy caused by service instances processed in earlier recursive levels that belong to the same task n_i (first condition in Eq. (4)), are offloaded to the same ES m_j (second or third condition in Eq. (4)), and are processed on the same ES m_k (second or fourth condition in Eq. (4)), respectively. Specifically, $U_i^{\text{task}} = \sum_{s=1, \ell^s \in \mathcal{L}_i^{\text{task}}}^{r-1} w^s(\ell^s)$, $U_j^{\text{off}} = \sum_{s=1, \ell^s \in \mathcal{L}_j^{\text{off}} \setminus \mathcal{L}_i^{\text{task}}}^{r-1} w^s(\ell^s)$, and $U_k^{\text{pro}} = \sum_{s=1, \ell^s \in \mathcal{L}_k^{\text{pro}} \setminus \mathcal{L}_i^{\text{task}}}^{r-1} w^s(\ell^s)$.

Motivated by these observations, we simplify `FastSA` as shown in Algorithm 2. We introduce five auxiliary vectors, $\mathbf{U}_n$, $\mathbf{U}_m^{\text{off}}$, $\mathbf{U}_{nm}^{\text{off}}$, $\mathbf{U}_m^{\text{pro}}$, and $\mathbf{U}_{nm}^{\text{pro}}$, to store accumulated accuracy reductions. Specifically, $\mathbf{U}_n[i]$, $\mathbf{U}_m^{\text{off}}[j]$, $\mathbf{U}_{nm}^{\text{off}}[i,j]$, $\mathbf{U}_m^{\text{pro}}[k]$, and $\mathbf{U}_{nm}^{\text{pro}}[i,k]$ store the accuracy decrements for service instances in $\mathcal{L}_i^{\text{task}}$, $\mathcal{L}_j^{\text{off}}$, $\mathcal{L}_i^{\text{task}} \cap \mathcal{L}_j^{\text{off}}$, $\mathcal{L}_k^{\text{pro}}$, and $\mathcal{L}_i^{\text{task}} \cap \mathcal{L}_k^{\text{pro}}$, respectively. In addition, the vector $\mathbb{L}_{candi}$ stores candidate service instances, and the set $\mathcal{S}$ stores the final solution.

Algorithm 2 first calls the function `Assign`([LL, HL, HH]) to handle service instances in categories LL, HL, and HH sequentially (line 1). For each category, `Assign` traverses the entire $\mathcal{L}$ and only considers the service instances belonging to the current category (lines 6–8). For each considered service instance ℓ^r, the updated accuracy value $w^r(\ell^r)$ is computed according to Eq. (7) (lines 10–12). If $w^r(\ell^r)$ is positive, ℓ^r is appended to $\mathbb{L}_{candi}$, and the auxiliary vectors are updated accordingly (lines 13–15). Then, candidates in $\mathbb{L}_{candi}$ are added to the solution $\mathcal{S}$ in reverse order. Algorithm 2 subsequently calls `Assign`([LH]) to schedule service instances in category

Algorithm 2: Linear-Time Implementation for `FastSA`

```
/* consider categories LL, HL, and HH in order */
1  S_1 ← Assign([LL, HL, HH]);
2  S_2 ← Assign([LH]);                    // category LH only
3  if u(S_1) ≥ u(S_2) then return S_1 else return S_2;
4  Assign(CategoryList):
5      Initialize vectors U_n, U_m^off, U_nm^off, U_m^pro, U_nm^pro, L_candi, S;
6      for category ∈ CategoryList do
7          for r ← 1 to |L| do            // ℓ^r: r-th element in L
8              if ℓ^r is not in category then continue;
9              Suppose ℓ^r ≜ ⟨n_i, m_j, m_k, b_ℓr, c_ℓr, ψ, a_r⟩;
10             Let U_i^task = U_n[i], U_j^off = U_m^off[j] − U_nm^off[i,j],
               and U_k^pro = U_m^pro[k] − U_nm^pro[i,k];
11             ũ(ℓ^r) = U_i^task + 2U_j^off · b_ℓr/B_k + 2U_k^pro · c_ℓr/C_k (Eq. (7));
12             w^r(ℓ^r) = u(ℓ^r) − ũ(ℓ^r);
13             if w^r(ℓ^r) <= 0 then continue;
14             Append ℓ^r into the end of vector L_candi;
15             U_n[i]=U_n[i]+w^r(ℓ^r), U_m^off[j]=U_m^off[j]+w^r(ℓ^r),
               U_nm^off[i,j]=U_nm^off[i,j]+w^r(ℓ^r), U_m^pro[k]=U_m^pro[k]+w^r(ℓ^r),
               U_nm^pro[i,k]=U_nm^pro[i,k]+w^r(ℓ^r);
16     for r ← |L_candi| to 1 do          // ℓ^r: r-th element in L_candi
17         if S ∪ {ℓ^r} is feasible then S ← S ∪ {ℓ^r};
18     return S;
```

LH (line 2). Finally, the solution achieving the higher total accuracy improvement is selected as the output of `FastSA`.

The sequence of candidate service instances in $\mathbb{L}_{candi}$ generated by the `Assign` function in Algorithm 2 exactly mirrors the sequence produced by the recursive `Assign` function in `FastSA`. Moreover, the accuracy reduction applied to each instance remains identical. Therefore, **the theoretical guarantee established in Theorem 3 still holds.** Algorithm 2 performs four traversals over $\mathcal{L}$, during which the accuracy value of each service instance is only updated once. Thus, the time complexity of Algorithm 2 is $\mathcal{O}(|\mathcal{L}|) = \mathcal{O}(2NM^2A \cdot \min\{B, C\})$.

V. Fault-Tolerant Offloading Framework

This section presents a fault-tolerant task offloading framework for real-time tasks. We first introduce the fault detection and recovery mechanism to handle both backhaul link and ES failures, and then present the task offloading workflow.

A. Link-State Synchronization and Global Scheduler Election

To maintain global network awareness, we adopt a flooding-based synchronization protocol (similar to OSPF [32]) to disseminate backhaul connectivity information among all ESs.

Each ES is associated with a gNB, a server, a scheduler, and a router. The gNB handles data offloading, and the server hosts services for EDs. The scheduler remains idle unless it is elected as the global scheduler, which then determines service placement and resource allocation for ED requests. The router is responsible for backhaul network synchronization. It maintains a Link State Advertisement (LSA) database that stores both the LSA generated by itself (referred to as the self-LSA) and LSAs received from other ESs (Fig. 2). Each router records the *link state* of its ES in the self-LSA, which includes the origin ES, a timestamp indicating the freshness of the link state, and a list of neighboring ESs it has detected. When a link state change is detected (e.g., a new neighbor is discovered or an existing one is disconnected), the router updates the timestamp and the list of neighboring ESs in the self-LSA, then broadcasts the updated self-LSA message to all directly connected ESs.

Upon receiving an LSA from another ES, the router compares the timestamp of the received LSA with that of the corresponding LSA (with the same origin ES) in its database. If the received LSA is newer, the router updates its database to maintain freshness and forwards the new LSA to all neighboring ESs *except the one from which it was received*, as that ES already possesses the update; otherwise, the received LSA is discarded. This ensures that each updated LSA traverses every link in each direction at most once, thereby preventing redundant traffic and minimizing backhaul overhead. An example of this process is illustrated in Fig. 2.

Through this flooding process, once a link-state change is detected, all ESs within the same backhaul network (e.g., in which every pair of ESs is connected by a path) can be informed within a bounded time, as shown in the following.

Proposition 1. *Let Δ denote the longest shortest path between any two ESs in a backhaul network (i.e., the diameter). The updated self-LSA of any ES can be disseminated to all other ESs within at most Δ hops of LSA message propagation.*

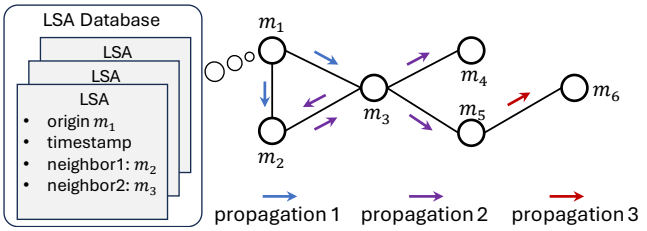

Fig. 2. An example of ES m_1 disseminates its self-LSA to other ESs. When the self-LSA of m_1 is updated, m_1 broadcasts the self-LSA to its neighboring ESs m_2 and m_3 (first propagation). In the second propagation, m_3 forwards the LSA to m_2, m_4, and m_5, while m_2 also forwards the LSA to m_3. During the third propagation, m_5 further forwards the LSA to m_6.

Global Scheduler Election. Whenever an ES receives an LSA message, it initiates a global scheduler election timer using the timestamp contained in the message. If multiple LSAs are received from different ESs, the timer is set according to the largest timestamp among them, which represents the time when the most recent link-state change is detected. Since any LSA can be disseminated to all ESs within a bounded time (Proposition 1), an appropriate delay can be set for the timer based on the largest timestamp such that, upon expiration, all link-state updates are fully disseminated across the network.

Once the backhaul network is synchronized, each ES within the same backhaul holds an identical LSA database, which is used by Dijkstra's algorithm to determine the shortest paths among ESs in the backhaul network. A deterministic and self-consistent election procedure is then independently executed by all ESs in the same backhaul to select a common global scheduler without additional coordination. The election rule ensures a unique outcome within each backhaul: the ES connected to the largest number of neighboring ESs is selected as the global scheduler, and in case of a tie, the ES with the smallest identifier is chosen. This rule ensures consistency among all ESs in the same backhaul. When a backhaul network is partitioned into multiple disjoint backhaul networks due to backhaul link or ES failures, each backhaul independently elects its own global scheduler.

After a new global scheduler is elected, a periodic scheduling timer is initiated to repeatedly invoke the scheduling algorithm, which determines service deployment and resource allocation for pending ED requests by solving $\mathbf{P}$ based on the resource in its backhaul. When the backhaul network is partitioned, multiple global schedulers operate concurrently, each making decisions exclusively for its own backhaul.

B. Failure Detection and Recovery

Each ES connects neighboring ESs via Ethernet interfaces using the Point-to-Point Protocol (PPP). Each ES can have multiple interfaces (termed pppIf), each of which connects to at most one neighboring ES, while unused interfaces remain unconnected. At the system-design level, these point-to-point links represent dedicated backhaul connections between ESs. To support predictable end-to-end behavior, the wired

backhaul is assumed to provide deterministic transmission with fixed data rates, as can be realized using TSN-capable Ethernet or similar technologies. Accordingly, backhaul delay is modeled as a deterministic function of data size and hop count, as defined in Section III.

To monitor link availability, every ES *periodically* broadcasts HELLO messages through all its pppIfs. Upon receiving a HELLO message, the neighboring ES replies with an acknowledgment (ACK). If the ACK is received within a predefined time interval, the corresponding link is marked as *active*. Conversely, if no ACK is received for two consecutive HELLO message broadcasts (to avoid excessive sensitivity), the link is considered *inactive*.

This framework considers both ES and backhaul link failures. When a link between two ESs fails, both ESs detect the connection loss. Upon confirmation after two consecutive missed ACKs, each ES updates its self-LSA. Similarly, if an ES (whether hosting the global scheduler or not) fails, all its neighboring ESs will detect the loss of connectivity and update their respective self-LSAs after the same confirmation interval. Then, each ES with updated-LSA broadcasts its self-LSA following the synchronization procedure in Subsection V-A.

Upon receiving an LSA, an ES (e.g., m_k) forwards the message following the procedure in Subsection V-A and updates its LSA database accordingly. In addition, ES m_k terminates all services hosted on its server and sends termination grants to the tasks that have offloaded to it. The ES then releases its occupied bandwidth and computational resources, and reports its updated status to the newly elected global scheduler once it is ready. If ES m_k previously hosts a global scheduler, the scheduler state is set to *idle*.

Furthermore, the discovery of a new neighboring ES automatically triggers a link-state update. Consequently, when a previously failed link or ES recovers, it is reintegrated into the existing topology. This failure detection and recovery mechanism provides strong robustness against both ES and backhaul link failures, ensuring continuous MEC service.

C. Task Offloading Workflow

A detailed task offloading workflow is illustrated in Fig. 3. When an ED decides to offload a task (e.g., n_i), it first transmits a service request to its nearest ES (e.g., m_j) via the 5G wireless network, including its deadline T_i and offload data sizes θ_i^1, θ_i^2, etc. Upon receiving the request, ES m_j temporarily stores it in a local buffer. Once the global scheduler is elected, all buffered requests are forwarded to it for scheduling.

The global scheduler is invoked periodically to continuously serve surrounding EDs. Each invocation defines a *scheduling cycle*, and its duration is referred to as the *scheduling interval*.

At the beginning of each scheduling cycle, every ED broadcasts an SRS to nearby gNBs. Upon receiving the SRS, each gNB selects an appropriate MCS to ensure reliable wireless transmission, thereby determining the achievable uplink data rate μ_{ij} between the ED and the ES (e.g., m_j) associated with that gNB. This uplink rate, together with the current abstracted

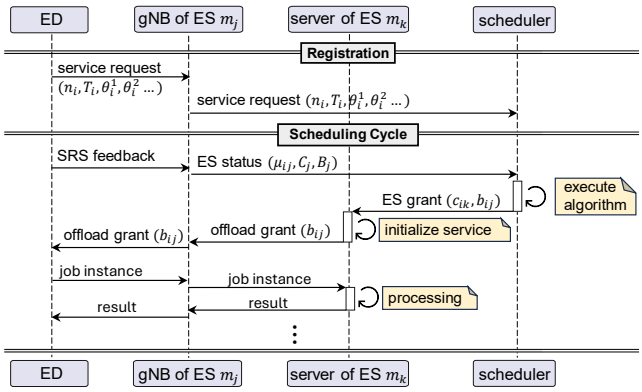

Fig. 3. Task Offloading Workflow (task n_i, ESs m_j, m_k are used as example).

resource availability at ES m_j (i.e., B_j and C_j), is reported to the global scheduler.

Using the latest resource reports from all ESs, the global scheduler invokes the scheduling algorithm (e.g., `FastSA`) to schedule pending requests by solving problem $\mathbf{P}$ (defined in Section III). Once a scheduling decision is obtained, an ES grant is issued to the server associated with each designated process-ES (e.g., m_k) to initialize the corresponding service. After successful service initialization, an offload grant is forwarded to the gNB associated with the designated offload-ES (e.g., m_j). The gNB interprets the offload grant to perform PHY/MAC-layer resource block scheduling and to notify the corresponding ED that task n_i may begin offloading.

When a job instance (i.e., task input data) of n_i is generated, it is first offloaded to the gNB associated with offload-ES m_j, and then forwarded to the server associated with process-ES m_k via the backhaul network for processing. After processing completes, the result is returned to ES m_j and then delivered to the ED. During offloading, the gNB continuously monitors the wireless channel quality. If channel conditions degrade such that timely offloading cannot be guaranteed, the gNB temporarily suspends the offloading process until channel quality recovers. During this suspension period, the task uses locally computed results to ensure safe operation; however, no utility (e.g., accuracy improvement) is accrued in this case.

The scheduling cycle repeats until a link-state change is detected in the backhaul network. After the network is resynchronized and a new global scheduler is elected, a new scheduling cycle begins.

Implementation. We implement the proposed framework on top of the open-source MEC simulator for real-time applications, *mecRT* [48], which provides a basic task offloading framework. However, the original *mecRT* only supports scenarios where all ESs are directly connected to a fixed centralized global scheduler. It lacks data forwarding, backhaul network synchronization, and fault detection and recovery mechanisms. In this work, we extend *mecRT* by integrating our fault-tolerant framework[1], thereby enabling general and resilient MEC simulation scenarios.

[1]The source code is available at https://github.com/gaochuanchao/mecRT.

VI. Experimental Evaluation

This section evaluates FastSA by comparing it with three state-of-the-art benchmark algorithms from the literature. We first examine FastSA in a fault-free MEC environment and then analyze the impact of backhaul link and ES failures on FastSA. All experiments are conducted on a desktop PC with an Intel(R) Xeon(R) W-2235 CPU and 32 GB of RAM.

A. Simulation Setup

We evaluate FastSA using the simulator *mecRT* [48], which provides detailed 5G network modeling and task offloading control logic. The simulation emulates an intelligent transportation scenario, where MEC services are provided to vehicles. Real taxi GPS trajectory data [34] is used to model vehicle mobility. Specifically, we extract all taxi trajectories passing through a 1 km $\times$ 1 km urban area in Shanghai over a 15-minute period, during which 80 taxis traverse the area.

We deploy 15 ESs evenly along the roads to provide MEC services. Each ES manages a 5G network with 270 BUs (i.e., $B_k = 270$), delivering a total data rate of 37 MB/s (obtained from profiling) under optimal channel quality. Additionally, each ES is equipped with a GPU server containing 20 CUs (i.e., $C_k = 20$), where the GPU type is randomly selected from three models (RTX3090, RTX4090, RTX4500). Notably, for the same allocated CUs, the execution time of a given service varies across different GPU types. We consider four image classification services (ResNet-101/152 [49], Convnext-L [50], RegNet-Y-L [51]), and profile their execution on each GPU type. Furthermore, the service initialization time on various ESs is randomly sampled from the range of $[10, 50]$ ms.

Each vehicle runs multiple applications (tasks). For each task n_i, its input data is randomly sampled from the ImageNet ILSVRC dataset [52], with data sizes ranging from 0.14MB to 0.6MB. The local application model is randomly selected from three lightweight image classification networks: ResNet-18 [49], GoogleNet [53], and RegNet-X-S [51]. We profile the Nvidia Jetson Nano, an embedded GPU-based device, to obtain local execution time d_i^{loc} and accuracy for each local model. The period is set to 60, 70, or 80ms for tasks using model ResNet-18, GoogleNet, and RegNet-X-S, respectively.

Benchmark Algorithms. We compare FastSA with three related algorithms adapted from prior studies: Greedy [27], Game [18], and Graph [29]. Among them, only Graph provides a theoretical guarantee of $\frac{1-\alpha}{2}$, where α denotes the maximum fraction of an ES's capacity that can be allocated to a single task. In contrast, Greedy and Game offer no such theoretical guarantee for problem $\mathbf{P}$. Since FastSA, Greedy, and Game do not require α, we set $\alpha = \frac{1}{2}$ for Graph in our experiment.

- Greedy: For each $\ell \in \mathcal{L}$, Greedy defines resource efficiency as $e(\ell) = u(\ell)/(\frac{b_\ell}{B_j} \times \frac{c_\ell}{C_k})$, where m_j and m_k are the offload-ES and process-ES of ℓ. It then processes all $\ell \in \mathcal{L}$ in non-increasing order of $e(\ell)$, selecting each service whenever feasibility constraints are met.
- Game: Game models $\mathbf{P}$ as a non-cooperative game, in which each vehicle acts as a player, and selects the $\ell \in \mathcal{L}$ that maximizes total utility in each iteration.
- Graph: Graph converts $\mathbf{P}$ into a tripartite graph matching problem and solves it with an LP rounding method.

Scheduling algorithm performance is assessed using two metrics: predicted and measured accuracy improvement. That is, the improved image classification accuracy as a result of successfully offloading the task and receiving the results within the deadline when compared to executing the task only locally on the ED. The *predicted accuracy improvement* represents the result estimated by the algorithm itself, assuming static network conditions for the whole scheduling interval and neglecting execution overhead. In contrast, the *measured accuracy improvement* is obtained from simulation results based on the actual offloaded job instances, capturing the impact of network fluctuations and algorithm execution overhead. Since the simulation lasts 900 seconds, the results are reported as the total accuracy improvement per second for consistency.

B. Result Discussion

Two sets of experiments are conducted. Since the scheduling algorithm is invoked once for each scheduling interval, the first experiment evaluates FastSA in a fault-free MEC under different scheduling intervals. The second experiment introduces backhaul link and ES failures during simulation to assess FastSA under our fault detection and recovery framework.

1) Fault-Free MEC: In the first experiment, we evaluate FastSA in a fault-free MEC under different scheduling intervals: 5s, 10s, and 15s. The average predicted and measured accuracy improvements are presented in Figs. 4(a) and 4(b), respectively. On average, the predicted accuracy improvement of FastSA is 27% lower than that of Greedy, but 34.2% and 38.7% higher than those of Game and Graph, respectively. Besides, the measured accuracy improvement of FastSA surpasses Greedy, Game, and Graph by 15.3%, 31.1%, and 33.3%, respectively. Although FastSA achieves the second-highest predicted accuracy improvement, its measured performance significantly exceeds that of all benchmark algorithms, demonstrating its superior resource utilization efficiency in realistic, dynamic MEC environments.

A comparison between Figs. 4(a) and 4(b) reveals a clear discrepancy between the predicted and measured accuracy improvements. On average, the measured accuracy improvements of FastSA, Greedy, Game, and Graph are 38.1%, 58.7%, 35.2%, and 32.6% lower than their predicted values, respectively. The underlying cause is shown in Fig. 4(c), which illustrates the average fallback ratio in each scheduling interval. The *fallback ratio* is defined as the proportion of jobs successfully served (i.e., offloaded and completed within deadlines) relative to the total number of generated jobs for tasks that are granted in each scheduling interval. It consists of two components: "-O" in Fig. 4(c) denotes jobs not offloaded due to scheduling algorithm overhead, while "-N" corresponds to those not offloaded due to network quality fluctuations (the channel quality decides the MCS selection, and hence the data offloading rate). As shown, the average fallback ratios of FastSA, Greedy, Game, and Graph are 38.7%, 59.5%, 34.3%, and 29.9%, respectively. Moreover, the fallback ratios

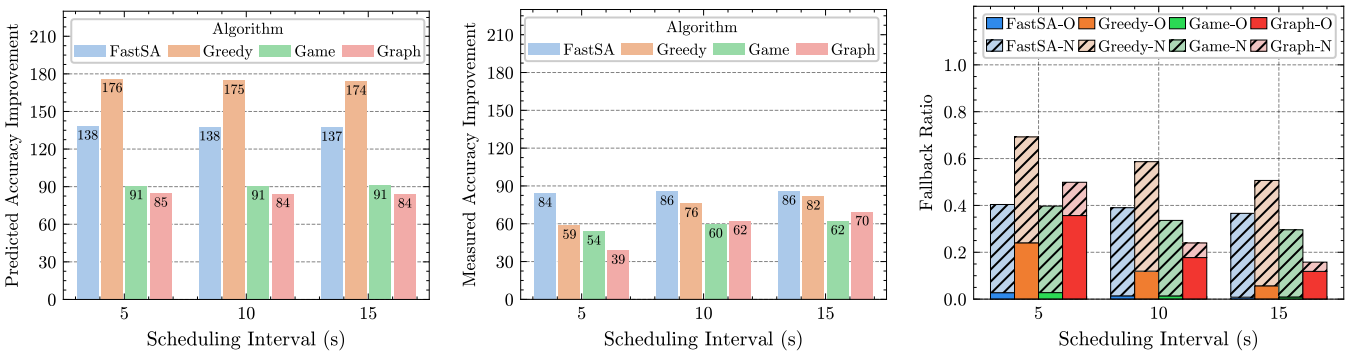

Fig. 4. In fault-free MEC, for different scheduling intervals, the average (a) *predicted* and (b) *measured* accuracy improvements, and (c) fallback ratio.

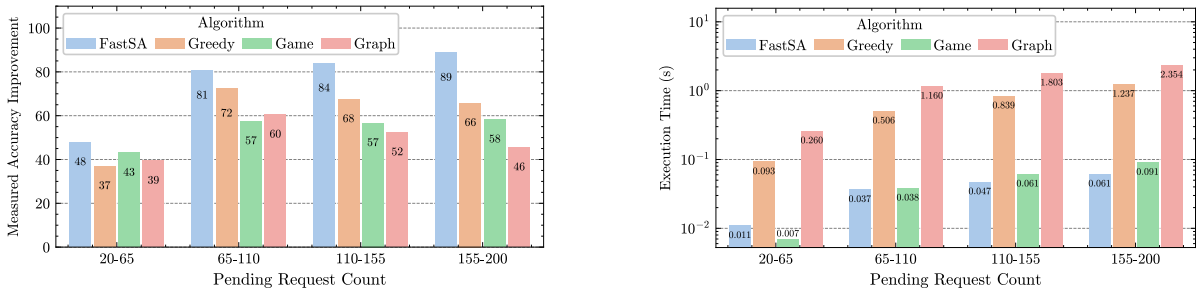

Fig. 5. In fault-free MEC, for different request counts, the results for (a) measured accuracy improvement and (b) algorithm execution time.

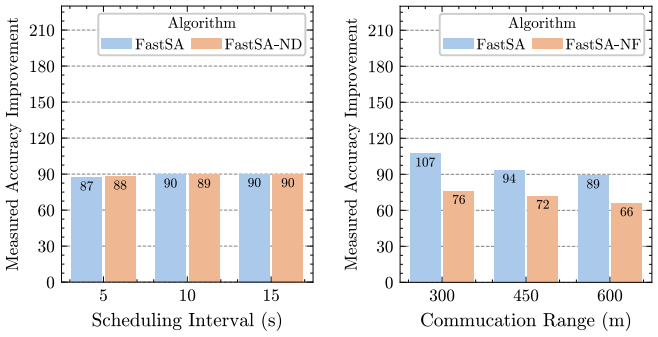

Fig. 6. Ablation results for `FastSA` in fault-free MEC. (a) The impact of service-instance division under varying scheduling intervals; (b) the impact of task forwarding under varying effective communication ranges. Measured accuracy improvements are shown.

of `FastSA`, `Greedy`, and `Game` are primarily influenced by network variations, whereas that of `Graph` is dominated by algorithmic overhead. Besides, the 38.7% fallback ratio of `FastSA` highlights a promising research direction: incorporating wireless network variation awareness into the problem formulation to enhance robustness against such dynamics.

We further examine how the measured accuracy improvement and execution time vary with the number of pending requests, as shown in Figs. 5(a) and 5(b). In Fig. 5(a), the measured accuracy improvement of all algorithms generally increases with the request count, except for `Graph`, whose performance degrades due to its scheduling overhead (and hence a higher fallback ratio). `FastSA` consistently outperforms `Greedy`, `Game`, and `Graph`, with the performance gap widening as the number of requests grows. Meanwhile, the average execution time (Fig. 5(b)) of `FastSA` is 0.05s, which is 18.4×, 1.4×, and 37.5× faster than `Greedy`, `Game`, and `Graph`, respectively. These results demonstrate the superior practical performance and runtime efficiency of `FastSA`.

2) Ablation Evaluation in Fault-Free MEC: In the second experiment, we evaluate the impact of service-instance division and task forwarding on `FastSA` in a fault-free MEC setting. The average measured accuracy improvement under different scheduling intervals is shown in Fig. 6(a). `FastSA-ND` denotes a variant of `FastSA` that processes all service instances in a unified category order (LL, LH, HL, HH) without division. As shown, `FastSA` and `FastSA-ND` achieve similar measured accuracy improvements across all scheduling intervals, indicating that instance division does not incur noticeable performance loss. Meanwhile, `FastSA` additionally provides a constant-factor approximation guarantee, improving theoretical robustness without sacrificing practical performance.

We next evaluate the impact of task forwarding. Since 5G coverage is spatially limited, enabling forwarding increases the flexibility of task placement and allows the system to better utilize ES resources. Here, we consider different effective gNB communication ranges, where a task may only be offloaded to a gNB if its ED lies within the gNB's communication range. The measured accuracy improvement

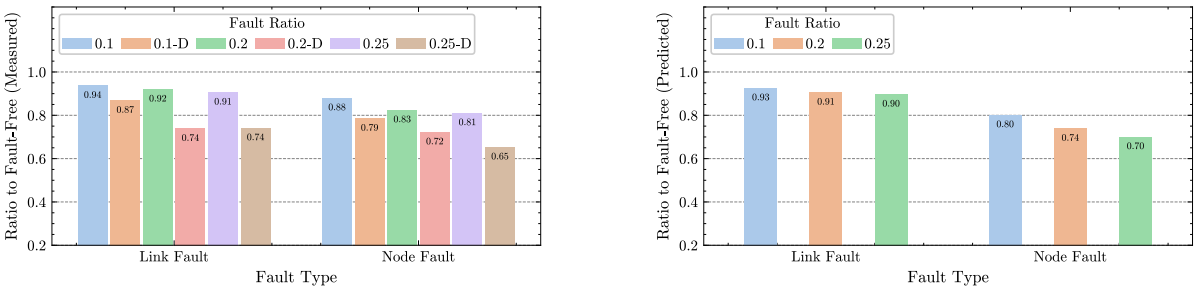

Fig. 7. In fault-injected MEC, the normalized average (a) measured and (b) predicted accuracy improvement for different fault types and fault ratios.

under different effective gNB communication ranges is shown in Fig. 6(b), where `FastSA-NF` disables task forwarding. On average, `FastSA` achieves 26.2% higher measured accuracy improvements than `FastSA-NF`, indicating that task forwarding effectively improves performance by enabling more flexible use of ES resources.

3) Fault-Injected MEC: In the third experiment, we evaluate the accuracy improvement of `FastSA` under the proposed fault detection and recovery framework by manually injecting backhaul link faults (termed *link fault*) and ES faults (termed *node fault*). The simulation runs for 900s with a scheduling interval of 10s. Three fault ratios are considered for both link and node faults: 0.1, 0.2, and 0.25. For each ratio, randomly sampled link or node faults are injected into the MEC in the middle of every 100s time interval and are removed at the end of the same interval, e.g., during the time interval $[0, 100]$, a fault is injected at 50s and cleared at 100s.

In the simulation, the period of the HELLO message for link state monitoring is set to 100ms. The recovery time is defined as the duration between fault injection and the election of a new global scheduler. During this interval, all tasks are processed locally. Simulation results show that the recovery time does not exceed 108ms, which is negligible compared to the 10s scheduling interval. This demonstrates the superior efficiency of our framework in fault detection and recovery.

The average measured accuracy improvement is presented in Fig. 7(a), where all values are normalized to the fault-free baseline. For comparison, results with the fault detection and recovery framework disabled are also included (denoted as "-D" in Fig. 7(a)). When the framework is disabled, the ES hosting the global scheduler is excluded from fault injection; otherwise, offloading would not be possible. As shown in Fig. 7(a), under link faults, the average measured accuracy improvements with and without the framework drop to 92.2% and 78.3% of the fault-free baseline, respectively. With the framework enabled, performance remains close to the fault-free level and only slightly declines as the fault ratio increases. This is because link failures do not reduce the total ES resources in the MEC, and the backhaul network can re-route data through alternative paths after resynchronization. In contrast, disabling the framework leads to a significant drop in accuracy improvement as the fault ratio grows, as the scheduler is not aware of the infeasible backhaul paths, and deadlines will be missed for job instances routed through those paths.

For node faults, as shown in Fig. 7(a), the measured accuracy improvement decreases with higher fault ratios, with a notably greater degradation when the framework is disabled. On average, the measured accuracy improvements with and without the framework are 83.9% and 72.1% of the fault-free baseline, respectively. Node faults result in more severe performance degradation than link faults because they directly reduce the available ES resources in the MEC. These results verify the effectiveness of our framework in maintaining system performance under both link and node faults.

The average predicted accuracy improvement is presented in Fig. 7(b), where all results are normalized to the fault-free baseline. On average, the predicted accuracy improvement decreases to 91% and 74.8% of the fault-free baseline under link and node faults, respectively. A comparison between Figs. 7(a) and 7(b) indicates that the reduction in measured accuracy improvement is smaller than that in the predicted values, particularly under node faults. This discrepancy arises because, after a fault is injected, fewer tasks are granted for offloading. Consequently, network interference due to concurrent data offloading is reduced, leading to a smaller fallback ratio caused by network quality fluctuations.

VII. Conclusion

This paper investigated a forwarding-enabled DOAP in MEC, denoted as $\mathbf{P}$, which jointly optimizes task offloading and resource allocation under wireless bandwidth and computational resource constraints to maximize the total inference accuracy improvement. To solve $\mathbf{P}$, we proposed `FastSA`, a linear-time $\frac{1}{10}$-approximation algorithm that offers the first constant-factor guarantee for the forwarding-enabled DOAP. We also designed a fault-tolerant task offloading framework capable of detecting and recovering from backhaul link and ES failures. Experimental results show that `FastSA` outperforms all benchmark algorithms in fault-free settings while achieving the lowest execution time, and that the proposed fault-handling framework effectively preserves service quality under fault-injected scenarios. Lastly, we observed a gap between the predicted and measured accuracy improvements of `FastSA` due to wireless network quality fluctuations. As future work, we plan to incorporate wireless variation awareness into the problem formulation to enhance robustness against such dynamics.

REFERENCES

[1] W. Ji, T. Ebrahimi, Z. Li, J. Yuan, D. O. Wu, and Y. Xin, "Guest editorial: Emerging visual iot technologies for future communications and networks," *IEEE Wireless Communications*, vol. 28, no. 4, pp. 10–11, 2021.

[2] C. Gao, N. Kumar, and A. Easwaran, "Energy-efficient real-time job mapping and resource management in mobile-edge computing," in *2024 IEEE Real-Time Systems Symposium (RTSS)*, 2024, pp. 15–28.

[3] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[4] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," 2018. [Online]. Available: https://arxiv.org/abs/1802.05668

[5] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint offloading and resource allocation in vehicular edge computing and networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[6] C. Zhu, J. Tao, G. Pastor, Y. Xiao, Y. Ji, Q. Zhou, Y. Li, and A. Ylä-Jääski, "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4150–4161, 2019.

[7] C. Xu, G. Zheng, and X. Zhao, "Energy-minimization task offloading and resource allocation for mobile edge computing in noma heterogeneous networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 16 001–16 016, 2020.

[8] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5g beyond," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12 175–12 186, 2020.

[9] Z. Xie, X. Song, J. Cao, and W. Qiu, "Providing aerial mec service in areas without infrastructure: A tethered-uav-based energy-efficient task scheduling framework," *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 25 223–25 236, 2022.

[10] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism," *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 3017–3030, 2023.

[11] Q. Qi, X. Chen, and C. Yuen, "Joint offloading selection and resource allocation for integrated localization and computing in edge-intelligent networks," *IEEE Transactions on Vehicular Technology*, pp. 1–15, 2024.

[12] X. Song, Q. Chen, S. Wang, and T. Song, "Cross-domain resources optimization for hybrid edge computing networks: federated drl approach," *Digital Communications and Networks*, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864824000361

[13] Z. Liao, C. Yuan, B. Zheng, and X. Tang, "An adaptive deployment scheme of unmanned aerial vehicles in dynamic vehicle networking for complete offloading," *IEEE Internet of Things Journal*, vol. 11, no. 13, pp. 23 509–23 520, 2024.

[14] Q. He, J. Li, X. Zhu, A. Jolfaei, Z. Feng, A. Tolba, K. Yu, and Y. Fu, "Joint data offloading and energy-efficient secure mec resource allocation method for iot device data in ran communication," *IEEE Transactions on Green Communications and Networking*, vol. 8, no. 3, pp. 1008–1017, 2024.

[15] M. A. Hossain and N. Ansari, "Network slicing for noma-enabled edge computing," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 811–821, 2023.

[16] W. Jiang, B. Ai, M. Li, W. Wu, Y. Pei, and X. Shen, "Aerial-irss-assisted energy-efficient task offloading and computing," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 20 178–20 193, 2024.

[17] H. Wang, X. Chen, H. Xu, J. Liu, and L. Huang, "Joint job offloading and resource allocation for distributed deep learning in edge computing," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 734–741.

[18] Q. Li, J. Zhao, and Y. Gong, "Cooperative computation offloading and resource allocation for mobile edge computing," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2019, pp. 1–6.

[19] T. T. Vu, D. N. Nguyen, D. T. Hoang, E. Dutkiewicz, and T. V. Nguyen, "Optimal energy efficiency with delay constraints for multi-layer cooperative fog computing networks," *IEEE Transactions on Communications*, vol. 69, no. 6, pp. 3911–3929, 2021.

[20] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, and Y. Liu, "Joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 4, pp. 4277–4292, 2023.

[21] J. Lin, S. Huang, H. Zhang, X. Yang, and P. Zhao, "A deep-reinforcement-learning-based computation offloading with mobile vehicles in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 17, pp. 15 501–15 514, 2023.

[22] C. Gao and A. Easwaran, "Local ratio based real-time job offloading and resource allocation in mobile edge computing," in *Proceedings of the 4th International Workshop on Real-Time and IntelliGent Edge Computing*, ser. RAGE '25. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3722567.3727843

[23] S. Hu and G. Li, "Dynamic request scheduling optimization in mobile edge computing for iot applications," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1426–1437, 2020.

[24] L. Zhong, Y. Li, M.-F. Ge, M. Feng, and S. Mao, "Joint task offloading and resource allocation for leo satellite-based mobile edge computing systems with heterogeneous task demands," *IEEE Transactions on Vehicular Technology*, pp. 1–15, 2025.

[25] L. Li, Q. Qiu, Z. Xiao, Q. Lin, J. Gu, and Z. Ming, "A two-stage hybrid multi-objective optimization evolutionary algorithm for computing offloading in sustainable edge computing," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 735–746, 2024.

[26] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Transactions on Communications*, vol. 66, no. 4, pp. 1594–1608, 2018.

[27] C. Gao, A. Shaan, and A. Easwaran, "Deadline-constrained multi-resource task mapping and allocation for edge-cloud systems," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 5037–5043.

[28] Z. Wang, B. Lin, Q. Ye, Y. Fang, and X. Han, "Joint computation offloading and resource allocation for maritime mec with energy harvesting," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 19 898–19 913, 2024.

[29] C. Gao and A. Easwaran, "Energy-efficient joint offloading and resource allocation for deadline-constrained tasks in multi-access edge computing," in *2025 IEEE 31st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2025, pp. 55–67.

[30] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Philadelphia, PA: SIAM, 2009, chapter 7: Multi-dimensional Assignment Problems.

[31] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz, "Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon even 1935-2004," *ACM Comput. Surv.*, vol. 36, no. 4, p. 422–463, dec 2004. [Online]. Available: https://doi.org/10.1145/1041680.1041683

[32] J. Moy, "Rfc2328: Ospf version 2," USA, 1998.

[33] G. Chuanchao, "mecrt: an mec simulator for real-time applications," Oct. 2025. [Online]. Available: https://github.com/gaochuanchao/mecRT

[34] SODA, "Shanghai qiangsheng taxi gps data trace (2018-04-01)," 2018. [Online]. Available: https://github.com/hetianzhang/Edge-DataSet?tab=readme-ov-file#taxi-trajectory-data

[35] B. Jamil, H. Ijaz, M. Shojafar, K. Munir, and R. Buyya, "Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–38, 2022.

[36] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[37] S. Ramanathan, N. Shivaraman, S. Suryasekaran, A. Easwaran, E. Borde, and S. Steinhorst, "A survey on time-sensitive resource allocation in the cloud continuum," *it-Information Technology*, vol. 62, no. 5-6, pp. 241–255, 2020.

[38] NVIDIA Corporation, *CUDA Toolkit Documentation*, 2025, green Contexts. [Online]. Available: https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__GREEN__CONTEXTS.html

[39] L. Yang, Z. Zheng, J. Wang, S. Song, G. Huang, and F. Li, "Adadet: An adaptive object detection system based on early-exit neural networks," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 16, no. 1, pp. 332–345, 2024.

[40] 3GPP, "Physical channels and modulation (Release 16)," 3GPP, Tech. Rep. TS 38.211 V16.2.0, 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138200_138299/138211/16.02.00_60/ts_138211v160200p.pdf

[41] M. Kutila, K. Kauvo, P. Aalto, V. G. Martinez, M. Niemi, and Y. Zheng, "5g network performance experiments for automated car functions," in *2020 IEEE 3rd 5G World Forum (5GWF)*, 2020, pp. 366–371.

[42] W. Zhan, C. Luo, G. Min, C. Wang, Q. Zhu, and H. Duan, "Mobility-aware multi-user offloading optimization for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3341–3356, 2020.

[43] S. Raza, S. Wang, M. Ahmed, M. R. Anwar, M. A. Mirza, and W. U. Khan, "Task offloading and resource allocation for iov using 5g nr-v2x communication," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 10 397–10 410, 2022.

[44] M. Szalay, P. Mátray, and L. Toka, "Real-time faas: Towards a latency bounded serverless cloud," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1636–1650, 2023.

[45] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.

[46] ShareTechnote, "5g/nr - fr/operating bandwidth," 2025. [Online]. Available: https://www.sharetechnote.com/html/5G/5G_FR_Bandwidth.html

[47] T. J. Xia and G. A. Wellbrock, "Commercial 100-gbit/s coherent transmission systems," *Optical Fiber Telecommunications*, pp. 45–82, 2013.

[48] C. Gao and A. Easwaran, "Real-time service subscription and adaptive offloading control in vehicular edge computing," in *2025 IEEE Real-Time Systems Symposium (RTSS)*, 2025, pp. 175–188.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[50] A. Trockman and J. Z. Kolter, "Patches are all you need?" *arXiv preprint arXiv:2201.09792*, 2022.

[51] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 428–10 436.

[52] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[53] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.