EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

GTBTL-IoT: An Approach of Curtailing Task Offloading Time for Improved Responsiveness in IoT-MEC Model

E. F. Siddiqui[1] and T. Ahmed[1,*]

[1]Advanced Computing Research Laboratory, Department of Computer Application, Integral University, Lucknow, India

Abstract

INTRODUCTION: The Internet of Things (IoT) has transformed daily life by interconnecting digital devices via integrated sensors, software, and connectivity. Although IoT devices excel at real-time data collection and decision-making, their performance on complex tasks is hindered by limited power, resources, and time. To address this, IoT is often combined with cloud computing (CC) to meet time-sensitive demands. However, the distance between IoT devices and cloud servers can result in latency issues.
OBJECTIVES: To mitigate latency challenges, Mobile Edge Computing (MEC) is integrated with IoT. MEC offers cloud-like services through servers located near network edges and IoT devices, enhancing device responsiveness by reducing transmission and processing latency. This study aims to develop a solution to optimize task offloading in IoT-MEC environments, addressing challenges like latency, uneven workloads, and network congestion.
METHODS: This research introduces the Game Theory-Based Task Latency (GTBTL-IoT) algorithm, a two-way task offloading approach employing Game Matching Theory and Data Partitioning Theory. Initially, the algorithm matches IoT devices with the nearest MEC server using game-matching theory. Subsequently, it splits the entire task into two halves and allocates them to both local and MEC servers for parallel computation, optimizing resource usage and workload balance.
RESULTS: GTBTL-IoT outperforms existing algorithms, such as the Delay-Aware Online Workload Allocation (DAOWA) Algorithm, Fuzzy Algorithm (FA), and Dynamic Task Scheduling (DTS), by an average of 143.75 ms with a 5.5 s system deadline. Additionally, it significantly reduces task transmission, computation latency, and overall job offloading time by 59%. Evaluated in an ENIGMA-based simulation environment, GTBTL-IoT demonstrates its ability to compute requests in real-time with optimal resource usage, ensuring efficient and balanced task execution in the IoT-MEC paradigm.
CONCLUSION: The Game Theory-Based Task Latency (GTBTL-IoT) algorithm presents a novel approach to optimize task offloading in IoT-MEC environments. By leveraging Game Matching Theory and Data Partitioning Theory, GTBTL-IoT effectively reduces latency, balances workloads, and optimizes resource usage. The algorithm's superior performance compared to existing methods underscores its potential to enhance the responsiveness and efficiency of IoT devices in real-world applications, ensuring seamless task execution in IoT-MEC systems.

Keywords: Mobile Edge Computing, Internet of Things, Task Computation Latency, Time Critical Responses, Increased Responsiveness.

Received on 26-03-2024, accepted on 30-10-2024, published on 11-11-2024

doi: 10.4108/eetiot.5556

*Corresponding author. Email: tasneemrke@gmail.com

1. Introduction

Mobile devices are becoming more powerful and can now run more complex applications, but their limited resources and battery life can make it difficult to meet the growing demand for computing power and provide real-time experiences. Overcoming these limitations is essential for achieving digital intelligence in various industries [1]. Current network topologies cannot handle the amount of

data generated by IoT devices, so the cloud is a better option for processing this data quickly and efficiently. Clouds can provide unlimited resources and computing services, but this comes at a cost: more energy is consumed because the data has to be transmitted to distant servers [2]. Mobile edge computing solutions improve compute capabilities for time- and computation-intensive IoT applications by addressing issues like location unawareness, mobility support, low latency, and rapid reaction times [3]. Shifting remote processing to close edge servers can improve computation speed, decision-making, latency reduction, and resource efficiency, despite adding computational burden since the distance between an edge server and the device plays an important role in remote computation for calculating the total task transmission and computation latency [4]. MEC servers are recommended for limited local resources, but this prolongs processing latency and delays. Task offloading in MEC computing is challenging due to factors like latency, energy consumption, workload distribution, and longer transmission durations, which impact model performance and resource allocation (RA) [5].

Nonetheless, Edge Computing (EC) has enhanced cloud capabilities near network edges, while optimal RA is still a crucial challenge for efficient decision-making and processing. The proposed resource allocation mechanism in IoT enhances Quality of Service (QoS) by reducing energy usage and utilizing all available resources within the fog network [6]. However, the deployment of EC in sparsely inhabited areas is another major challenge. To overcome this challenge, a satellite-based Internet of Things (SAT-IoT) was developed using deep reinforcement learning to optimally distribute resources [7]. It is essential to eliminate as much delay as possible to deliver time-critical responses. Thus, to minimize delays in IoT applications like healthcare, a combination of task offloading (TO) and task scheduling (TS) combined with effective resource allocation is proposed to reduce overall task generation costs [8]. Also, deploying edge servers cooperatively, where tasks are divided among MEC servers, can reduce task computation latency, which is more efficient than static offloading techniques [9]. Another significant problem in EC is identifying the most appropriate resource for job computation. A three-tier TO technique has been developed to optimize the allocation of cells as computing resources in EC to ensure efficient cell formation, selection, and offloading for real-time answers [10]. In an IoT-MEC environment, both the devices and MEC servers are resource- and power-constrained; therefore, there is always a scarcity of resources. Resource management and allocation in an optimal manner are thus critical requirements [11].

The carbon emissions produced during task computation (TC) operations are another significant difficulty for the MEC computing environment. The alternate direction method of multipliers (ADMM) technology reduces task traffic congestion in the MEC computing environment by reducing carbon emissions from TC, TO, and RA operations [12]. The inverse relationship between energy usage and latency suggests that evenly distributing workload among servers can reduce total energy and optimize latency [13–15]. Proactive caching and optimal work allocation among MEC servers is another suggested method to reduce TC latency using clustering and matching game theory, but challenges include task division, scheduling, and assignment in multi-server environments [16–17]. Long-term evolution (LTE) is one of several technologies that are utilized for task transfer in remote locations. However, OpenAirInterfaces with network slicing can be used instead of LTE for improved user quality of experience (QoE) in remote location task transfer [18]. The MEC platform can utilize location-based services for computational offloading to decrease latency for mobile users [19]. If emphasis is given to the caliber of network connectivity and resource usage, the overall TC time may be reduced by a substantial amount. This may be done by using sparse code multiple access (SCMA), a method that enhances network performance and throughput [20]. Unmanned aerial vehicles (UAVs) can create a perfect MEC environment with minimal latency, efficient energy usage, and high QoE based on key factors and RA approaches [21–24]. Optimal RA approaches are also required when data is aggregated at MEC servers and is required to be processed in a real-time manner. The use of smart grids in this case proves to be a better option [25].

Pre-allocation of resources can also be done before transmitting the data for remote computation. This significantly reduces the total uplink transmission latency. When resolved using a distributed antenna system (DAS), fewer resources with optimal utilization were achieved in a 5G environment [26–27]. With resource utilization, it is very necessary to optimize energy utilization and traffic congestion on the server. This has been achieved using an active queue management-based green cloud model (AGCM) under stringent deadline constraints [28–29]. Many 6G transmission techniques are being used nowadays with enhanced data transmission rates in Tb/s and ultra-latent responses, especially for collecting seismological and geophysical data [30–31]. Using multiple-offloading strategies with data portioning can also significantly reduce total TC time and latency overhead with balanced workloads and cell selection [32–33]. Another promising solution for a real-time IoT environment is IoT-Grids (IoT-G), with broad optimal spectrum resources for TC and scheduling purposes. These activities can also be done using a multi-server environment where tasks can be redirected for workload balancing [34–35].

Moreover, Mobile Edge Computing (MEC) architecture integrates with ultra-dense networks (UDNs) for 5G, employing a DQN-AC algorithm to optimize computation offloading and resource allocation [36]. Queuing time also adds up to the total latency in task computation in MEC; therefore, employing differential-difference equations to model IoT-based MEC systems and utilizing M/M/1 queue theory to compute performance can significantly reduce the

queuing latency [37]. A multi-task offloading scheme utilizing a hierarchical spatial-temporal monitoring module and fine-grained resource scheduling can be used for task awareness, abnormality inference, and offloading efficiency [38]. Addressing service dependencies through joint consideration of task scheduling and resource allocation utilizing a layered scheme and detailed algorithms minimizes latency and energy consumption [39]. Incorporating relay selection and adaptive bandwidth allocation, which minimize computation time by utilizing evolutionary algorithms, can resolve joint multi-task partial offloading issues in MEC servers [40].

In addressing the crucial challenge of minimizing task computation latency and making optimal task offloading decisions for improved resource utilization and heightened responsiveness, this paper employs two synergistic theories. The first, Game Matching Theory, is harnessed to determine the nearest Mobile Edge Computing (MEC) server, thereby reducing overall task transmission and computation time. The second theory, Data Partitioning Theory, is applied to optimize resource utilization through parallel task computation in both local and remote regions, resulting in enhanced responsiveness with minimal latency overhead. With the combination of the above two theories, an optimal task computation and offloading algorithm has been proposed namely the Game Theory-Based Task Latency (GTBTL-IoT) algorithm. The proposed task offloading and computation algorithm surpass comparable algorithms in efficiency, significantly reducing total task computation overhead and ensuring timely processing of tasks in critical scenarios.

2. System Model Development

An IoT-MEC task offloading solution has been proposed to address latency issues using IoT and MEC. It assigns workloads to local and remote locations for parallel computing due to IoT devices' limited storage and calculation capabilities. The structure of the proposed IoT-MEC model is shown in Figure 1.

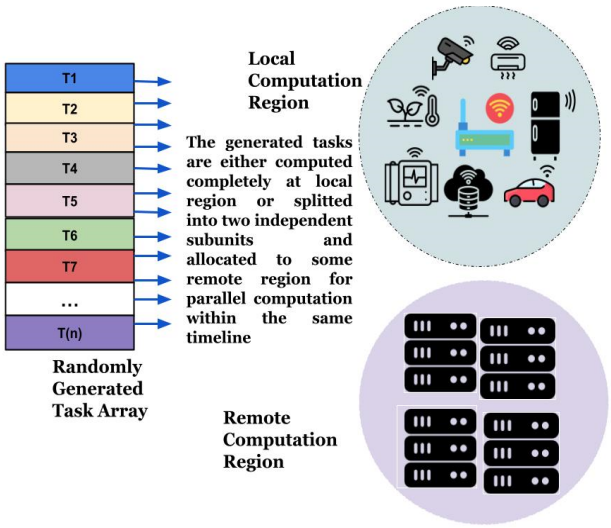

Figure 1. Proposed System Model (GTBTL-IoT)

The proposed system paradigm generates various tasks randomly in the IoT zone, each with varying data size and computational power. Due to this variability in the nature of tasks as well as computing devices, each task as well as the device and server should be modeled based on its unique ID, computation facilities and requirements, and the amount of memory required and utilized through task modeling, device modeling, and server modeling.

2.1. Task Modeling

IoT devices perform various tasks like sensor analysis, streaming, and document uploading, each acting as a standalone operation or part of a larger procedure. In general, any generated task may be categorized as a vector of five possible attributes, namely Task ID (Λ_{ID}), Task Size (d_Λ),Computational Intensity i.e., number of CPU Cycles in bits (c_Λ), Task Deadline (Λ_{td}) and Task Workload (WL_Λ). It may be modeled as Λ ($\Lambda_{ID}, d_\Lambda, c_\Lambda, \Lambda_{td}, WL_\Lambda$). According to the proposed model, a task Λ is computed within a time which is the sum of time to compute at the local region $\left(T^i_{cmp}\right)$, time to compute at the remote region $\left(T^j_{cmp}\right)$, and total time to transmit a task at a remote region ($tTrans^t_\Lambda$). T^j_{cmp} and $tTrans^t_\Lambda$ is 0 in the case of complete local task computation (TC).
The device must allocate work units for both local and remote computing before the deadline to offload work when local computing is insufficient. The device makes the internal choice to offload or divide tasks, which may be calculated as given in equation (1).

$$t_{off} = \Lambda_{TD} - t$$
$$t_{split} = \Lambda_{TD} - \left(t_{off}/2\right) \quad (1)$$

where, t_{off} is the total time to offload the task and t_{split} is the time to break the task into smaller components. The task component is assigned to the MEC server using the available bandwidth BW_i after being split into two halves. Next, $tTrans_{\Lambda}^{t}$ may be calculated as given in the equation (2).

$$tTrans_{\Lambda}^{t} = \frac{d_{\Lambda}^{remote}}{BW_i log_2(1 + SINR_i)} \quad (2)$$

where $SINR_i$ is signal-to-interference plus noise ratio and can be calculated as given in the equation (3).

$$SINR_i = \alpha_{ij} \frac{tTrans_{\Lambda}^{t} Chg_{ij}}{\sum_{i \in N, j \in U} Trans_{ij} Chg_{ij} + \partial} \quad (3)$$

where, BW_i is the bandwidth allocated to the local device, d_{Λ}^{remote} is the size of data for the task sub-component, Chg_{ij} is the channel gain between device i and server j where $i \in N$ and $j \in U$ and N, U are the total numbers of IoT devices and MEC servers present in a specified deployment area. Next, ∂ is the Gaussian white noise factor. Equations (2) and (3) are used to calculate the total transmission time of the task as well as the signal-to-interference ratio. The SINR is calculated since there are millions of IoT devices connected to a single server. Therefore, there is a high chance of packet loss and channel fading. Therefore, the transmission time will be calculated using the beamforming theory which calculates SINR as a crucial component.

IoT devices connected to a single MEC server require noise computation to prevent data transmission impacts. Parallel remote and local computing uses parametric comparisons and assessments for task offloading, with random split points as P. The following equations (4), (5), and (6) calculate the task splitting procedure for parallel computation. Firstly, the whole task data size is divided into $n_{packets}$ as the total number of task components is given in equation (4).

$$n_{packets} = {}^{d_{\Lambda}}\!/_{10} \quad (4)$$

Next, the point at which these components should be split for local and remote execution is calculated by split point P, whose value depends on the job, device, and server state, using the pivot point setting formula given in equation (5).

$$P = {}^{\mathrm{n_{packets}}}\!/_{\Lambda_{TD}} * 2 \quad (5)$$

Using Equations (4) and (5), the total task is split into two halves. Next, the total computation time T_{cmp}^{j} in which the task will get computed remotely can be represented as given in equation (6).

$$T_{cmp}^{j} = t_{off} + t_{split} + tTrans_{\Lambda}^{t} \quad (6)$$

Similarly, if the task gets computed locally without any offloading, then it may be computed as follows:

$$T_{cmp}^{i} = \frac{(d_{\Lambda} * Toff)}{Cap_{\mathrm{i}}} \quad (7)$$

where, Cap_{i} is the computational capability of device *i.* If the task-splitting process is carried out then the total computation time can be calculated as:

$$Tcmp = \left(T_{cmp}^{i} + T_{cmp}^{j}\right).(\alpha_{ij}) \quad (8)$$

Here α_{ij} will be 0 if no remote computation takes place. In that case $Tcmp = T_{cmp}^{i}$ respectively. The task, when it gets processed over a specific resource, will impose some workload on it. Therefore, the total workload imposed (WL_{Λ}) by the task may be calculated as given in equation (9).

$$\mathrm{WL}_{\Lambda} = d_{\Lambda} * Toff * \left(\frac{d_{\Lambda}}{t}\right) \quad (9)$$

The workload is measured in WLU and has a significant impact on the processing time of the task in the IoT region. The workload is the single independent factor that decides the total number of CPU cycles that are to be utilized for TC and, hence, how many tasks can be executed in a single instance of time.

2.2. Device Modeling

IoT devices generate computing jobs by representing U IoT nodes, each with potential properties like Workload (WL_i), total allotted bandwidth (Bw_{i}), Device ID (i_{ID}), and computational capability (Cap_{i}). Each device is individually recognized by ID, determining its state in the IoT-MEC ecosystem, where device capacity is determined by processor speed, memory health, queue size, and network usage. Server workload is important for offloading activities to prevent resource depletion and wasteful use and can be calculated as given in equation (10).

$$\mathrm{WL}_i = \sum_{\Lambda_n \in \Lambda} \mathrm{WL}_{\Lambda_n} \quad (10)$$

Each device has a threshold capacity for processing offloaded tasks and workloads. WL_{max}^{i} represents the maximum permissible workload, and the mathematical formulation given in equation (11) determines the available processing capacity for arriving loads.

$$WL_{rem}^{i} = WL_{max}^{i} - \mathrm{WL}_i \quad (11)$$

where, WL_{rem}^{i} is the remaining workload capacity left for the device. If the incoming tasks do not fit with WL_{rem}^{i}, then it may cause workload overflow and thus it is rejected and sent back for waiting until the current time instant is complete.

2.3. Sever Modeling

This section covers the modeling of *j* edge servers, using j (j_{ID}, Cap_{j}, Bw_{j}, WL_j). Here j_{ID} represents the unique ID of

the server, Cap_j is the computational capability of the server measured as the total number of CPU cycles it affords, Bw_j is the total allocated bandwidth and WL_j is the total workload imposed upon the server. The recommended model consists of U IoT nodes and portable N MEC servers, with dynamic devices that can be remote or near the server. The chosen region has the geographical distribution of N MEC servers, allowing the server's workload to include the total number of devices managed and tasks created. Therefore, WL_j may be calculated as follows:

$$\mathrm{WL}_j = \sum_{i \epsilon N} \mathrm{WL}_i \tag{12}$$

To process tasks on a server, it's vital to determine the available workload to avoid issues like request overload or resource exhaustion, which could disrupt real-time job computation. Thus, to calculate the remaining workload capacity, equation (13) can be used:

$$WL^j_{rem} = WL^j_{max} - \mathrm{WL}_j \tag{13}$$

The same is done in the case of the server where if the incoming task exceeds WL^j_{rem}, it is rejected and sent back for waiting until the current time instant is complete.

3. Task Offloading Strategies

This section discusses various offloading strategies for task offloading, focusing on optimizing resources, minimizing latency, and enhancing response times, as described in the subsections.

3.1. Distance-Based Task Offloading

Multiple MEC servers are located at different locations. Therefore, it is necessary to know the distance between the device and these servers. Also, IoT devices are dynamic and always moving. Because of this, the distance value is variable at every instance of time. Let x and y represent the coordinates of each device and server. The distance between each pair can be calculated using equation (14).

$$t_{dist} = \sqrt{(j_{x1} - i_{x1})^2 + \left(j_{y1} - i_{y1}\right)^2}, t_{dist} < t^{max}_{dist} \tag{14}$$

where (i_{x1}, i_{y1}) and (j_{x1}, j_{y1}) are the co-ordinate of server j and device i *and* t^{max}_{dist} is the maximum acceptable distance between the associated pair *i-j*. Minimal uplink transmission delay is achieved by implementing restrictions on device connections to the closest server, reducing task offloading time, and ensuring efficient computing. It is therefore important to check the minimum distance association between *i-j*. This can be accomplished using equations (15) and (16).

$$Cons\ A: min \sum_{u \epsilon U,\ j \epsilon N} (t_{dist}) < t^{max}_{dist} \tag{15}$$

$$Cons\ B = \sum_{j \epsilon N} \alpha_{ij} \epsilon \{0,1\} \tag{16}$$

The minimum distance is selected based on the availability of an array of MEC servers. These servers are first discovered, and the nearest server is selected before the final task is offloaded. The nearest-distance resource discovery algorithm proposes the discovery and selection of the nearest computing resource to minimize total transmission and computation latency and provide real-time responses. This approach has been used for task offloading to cut down the total transmission delay and resource allocation within the given system deadline.

Algorithm 1: Nearest Distance Resource Discovery Algorithm

INPUT: Set of IoT Nodes $i \epsilon U$, Set of MEC Servers $j \epsilon N$.
$i = \{i_1, i_2, i_3, \ldots\ldots, i_n,\}$ and $j = \{j_1, j_2, j_3, \ldots\ldots, j_n,\}$
OUTPUT: Association pair *i-j* with shortest distance
START
Discover j*// possible *i-j* pairs
For i[th] device make a distance-based set of server *j** discovered for association as $t^j_{dist} = \{d_1, d_2, d_3, \ldots\ldots, d_n,\}$
For k=0 to n-1
    Search for $min(t^j_{dist})$ using eq 15.
Calculate SINR by eq 3
Verify $Cons\ A\ and\ Cons\ B$ by eq 15 and 16
IF Yes **THEN**
  Set $\alpha_{ij} = 1$
  Proceed To **Algorithm 2**
ELSE
  GOTO Step 1
STOP

As discussed in the Resource Discovery Algorithm, initially all the available MEC servers will be discovered, and the nearest server will be assigned for TO and TC operations. With the help of the algorithm, TC is achieved with optimal latency in a time-critical manner.

3.2. Device-Based Task Offloading

Task offloading splits tasks locally, remotely, at edge servers, or in the cloud, adjusting resource requirements and affecting execution times significantly. The requirements of task execution include the same model of the task, i.e. $\Lambda(\Lambda_{ID}, d_\Lambda, c_\Lambda, \Lambda_{TD}, \Lambda_{Pr}, WL_\Lambda)$. As discussed in sub-section 3.1. If the device where the task is initially generated matches the requirements, then it may get executed locally without offloading. Otherwise, it has to be offloaded to either an edge server or a remote server for execution. Table 1 summarizes variables and comparisons for task requirement-based offloading in one-to-one parametric comparisons, aiding decision-making.

Table 1. Parametric Comparison for Resource Assignment

Considered Parameters	Split Decision	Offloading Type	Remarks
$d_{\Lambda} > M_{free}^{l}$	Yes	Partial	Offload to Server
$d_{\Lambda} > M_{free}^{R}$	Yes	Partial	Offload to Cloud
$\mathrm{WL}_{\Lambda} > WL_{max}^{i}$	Yes	Partial	Offload to Server
$\mathrm{WL}_{\Lambda} > WL_{max}^{j}$	Yes	Partial	Offload to Cloud
$Tcmp > T_{cmp}^{i}$	Yes	Partial	Offload to Server
$Tcmp > T_{cmp}^{j}$	Yes	Partial	Offload to Cloud

Now, each device i has limited memory and storage capacity to process the tasks and give computed results. Let M_{free} be the total free memory available unused and M_{alloc} be total memory allocated for the computation of some task; currently allocated memory will be calculated by dividing the total memory given by the used memory. Thus, total memory M_{total} maybe given as $\mathrm{M}_{total} = \mathrm{M}_{free} + \mathrm{M}_{alloc}$. First of all, the status of computing resources, that is, the devices and servers, will be updated for that instant of time. Secondly, their status will be mapped to task computation requirements as described in Table 1, and a final TO decision will be made. If the task computation requirements are not accomplished either by the device or by the server, the task will be finally rejected for that instant of time, as described in Algorithm 2. The rejected tasks will be checked for recomputation for a different time window after some wait time.

Algorithm 2: Task Execution Decision Algorithm

INPUT: Set of tasks Λ_i=[$\Lambda_1, \Lambda_2, \Lambda_3, \ldots\ldots, \Lambda_n$] $n \epsilon \Lambda$.
OUTPUT: Assignment of each unit of task Λ to a local device or edge server or cloud for computation and delivery of computed results
START
Let TS(t)= Λ_i(t)
WHILE TS(t) ≠ NULL **DO**
Check task status by $\Lambda(\Lambda_{ID}, d_{\Lambda}, c_{\Lambda}, \Lambda_{TD}, \Lambda_{Pr}, \mathrm{WL}_{\Lambda})$.
Check device status by $\mathrm{i}(\mathrm{i}_{ID}, Cap_{\mathrm{i}}, Bw_{\mathrm{i}}, \mathrm{E}_i, \mathrm{WL}_i)$
Check server status by $\mathrm{j}(\mathrm{j}_{ID}, Cap_{\mathrm{j}}, Bw_{\mathrm{j}}, \mathrm{E}_j, \mathrm{WL}_j)$
FOR k=1 to n-1
CHK 1: IF $Tcmp < \Lambda_{TD}$ AND $\mathrm{WL}_{\Lambda} < Cap_i$ AND $d_{\Lambda} < M_{free}^{l}$
Initially Λ_i to be allocated to *i* for local processing
CHK 2: IF $Tcmp > \Lambda_{TD}$ AND $\mathrm{WL}_{\Lambda} > Cap_i$ AND $d_{\Lambda} > M_{free}^{l}$
IF $\mathrm{WL}_{\Lambda} < WL_{max}^{j}$ AND $\Lambda_{size} < M_{free}^{R}$
Initially $\Lambda_i(t)$ to be allocated to server *j* for remote processing at server
ELSE
CHK 3: Reject Task Computation
ENDIF
ENDIF
ENDFOR
ENDWHILE
Proceed To Algorithm 3
STOP

The proposed Task Execution Decision Algorithm maps the current status of both the device and server with that of the task computation requirements before the final TO decision. This will help to elevate the system's performance by optimally assigning computing resources to the tasks and computing them in a real-time manner for the proposed MEC-IoT model. Next, the task will be split into two subcomponents: one will be computed locally, and the other will be a computer in a remote region in parallel. This remote region is the same to which the device has been connected using Algorithm 1. On this server, the split task component will be finally offloaded for remote computation.

3.3. Split-Based Task Offloading

The goal of carrying out Task Splitting is to minimize the total task computation latency with optimal RA. For this purpose, two sets have been designed namely $\Lambda = \{\Lambda_1^l, \Lambda_2^l, \ldots\ldots, \Lambda_n^l\}$, $\mathrm{R}_y = \{\Lambda_1^R, \Lambda_2^R, \ldots\ldots, \Lambda_n^R\}$, and containing those subunits of tasks Λ assigned for local and remote-based computation. The size of the total task may be represented as $\mathrm{L}_x + \mathrm{R}_x$ in the case of the server association. It may be formulated as given in equation (17).

$$\Lambda_{size} = (\mathrm{L}_x + \mathrm{R}_x).(\alpha_{ij}) \quad (17)$$

It is a crucial requirement to optimize offloading decisions by assigning the best resources for each task subunit's execution and delivery, minimizing energy and latency limitations. This is accomplished by Algorithm 3 after successful task splitting using Algorithm 2.

Algorithm 3: Resource Assignment Decision Algorithm (Final Task Offloading Decision)

INPUT: Set of local processing subunit $\mathrm{L}_x = \{\Lambda_1^l, \Lambda_2^l, \ldots\ldots, \Lambda_n^l\}$, Set of remote processing subunits $\mathrm{R}_y = \{\Lambda_1^R, \Lambda_2^R, \ldots\ldots, \Lambda_n^R\}$ and Set of Cloud processing subunits $\mathrm{DC}_z = \{\Lambda_1^C, \Lambda_2^C, \ldots\ldots, \Lambda_n^C\}$
OUTPUT: Optimal task-splitting and resource allocation for Task Offloading.
START
GOTO CHK1
Allocate task subunits to L_x until *P*
GOTO CHK2
IF $\mathrm{WL}_{\Lambda} < Cap_j$ AND $d_{\Lambda} < M_{free}^{l}$ AND $\Lambda_{TD} < T_{th}^{j}$
Offload to Edge server *j*
$results = \mathrm{L}_x \cup \mathrm{R}_y$
ELSE
GOTO CHK3
Reject
STOP

Algorithm 3 uses local computing mode and never broadcasts tasks to distant regions, gathering all calculated task components for decision-making. If an offloading strategy is used, results are transmitted back to the device, combining components for further decision-making. It should be noted that the proposed GTBTL-IoT algorithm only discusses uplink transmission delay. No discussion is

done on the back propagation of results to the requesting IoT device.

4. Performance Evaluation

GTBTL-IoT's efficacy is confirmed in terms of optimal task computation latency and balanced task offloading decision. The suggested approach is validated by comparing it to similar proposed algorithms such as dynamic task scheduling (DTS) [41], Fuzzy Algorithm (FA) [42], and Delay-Aware Online Workload Allocation (DAOWA) algorithm [43]. The DTS [41] algorithm tends to reduce overall latency and energy usage in the fog zone while maintaining optimal RA. FA [42] reduces task service time by taking into account variables such as CPU utilization and resource needs while DAOWA [43] intends to lower the long-term average task service latency. ENIGMA [44] is used to simulate an Edge Environment to assess the performance of GTBTL-IoT. It is a scalable simulator for the fog, edge, and cloud computing paradigms.

Dataset. A dynamic and intelligent task offloading-based strategy dataset [45] was used to simulate the proposed GTBTL-IoT method. This database contains information on latency, resource usage, user activity, and network parameters and is used as input by GTBTL-IoT for the sake of measuring its performance and its efficacy over other algorithms.

Simulation Setup. For simulation, 150 used nodes are distributed over an area of 1000 X 1000 m. The capacity of each use node Cap_i as well as the server Cap_j is randomly selected from Table II. The simulator parameter settings for the proposed model have been summarized in Table 2.

Table 2. Simulator Settings

Simulation Parameter	Setting Value
Simulation duration	150-250 secs
Device Status Updating	10 sec
Server Status Updating	10 sec
IoT to Edge Delay	200 ms
IoT to Edge Jitter	50 ms
Edge To Cloud Delay	350 ms
Edge To Cloud Jitter	100 ms
SNR	100 dB
Subchannel Bandwidth	200 kHz
Packet Size	100-1000 bytes
Deployment Area Radius	1000 m
Max. Association Delay	15 ms
Fading	Rayleigh Flat Fading
Path Loss Exponent	4
Power Spectral Density of Noise	-174 dBm/Hz
Data Size of Task	[0.1-1] MBits
No. of Required CPU Cycles	[0.1-1] GHz
Computational Capacity of Device	[0.7-1] GHz
Computational Capacity of Server	20 GHz
Max. Transmission Power	300 mW
No. of Iterations	5
No. of IoT Devices	100-1000
No. Of Edge Servers	5
Max. Acceptable Latency	[1.0-5.5] secs

Data Preprocessing and Visualization. Preprocessing the data is essential before executing the code through the database. Prior to the final implementation, this preprocessing stage is crucial for guaranteeing the accuracy of classifiers and removing any inconsistent or untrustworthy data. In order to prepare the data for testing and training, preprocessing steps include data normalization and scaling. The dataset is first split into training and testing sets. To be more precise, 90% of the data is used to train the suggested model, while the remaining 10% is set aside for testing. The task offloading approach is made more understandable by the analysis done in [45]. Plot axis and Pyplot have been used to show the findings using the suggested model. The model's results may be properly presented and interpreted with the help of these visualization approaches.

In the modeled environment, it has been assumed that there are three computing regions: IoT, edge, and cloud. Every IoT node is heterogeneous in terms of resource capacity and application execution. The value of simulation parameters within a specific range is determined by the minimum and maximum values of the dataset collected, which is again tuned through the pseudocode random number generator as shown in Figure. 2.

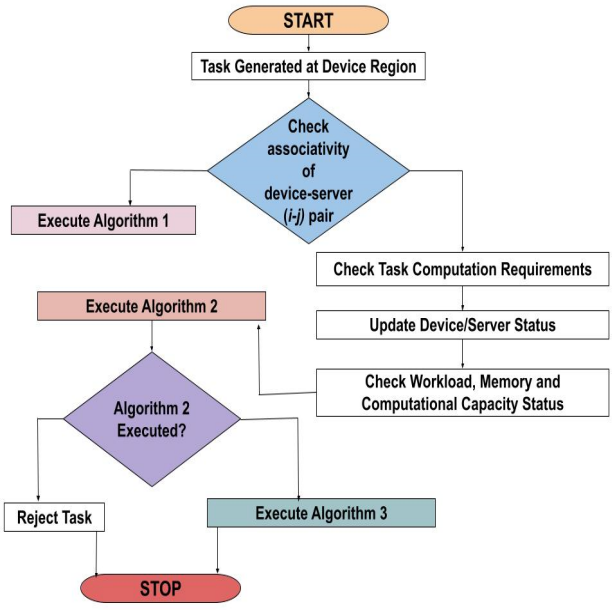

Figure 2. Workflow diagram of GTBTL-IoT

As illustrated in Figure 2, the TC method involves randomly creating tasks on IoT devices, verifying connectivity with the closest server, reviewing task requirements, and updating device and server status. If task specifications match available computational resources, work is offloaded locally, or task splitting occurs for distant processing. The task's sub-components will be gathered, and a real-time choice will be made if Algorithm 2 is successfully applied; otherwise, the procedure ends.

4.1 Complexity Analysis of GTBTL-IoT

The suggested Task Offloading and TC Latency minimizing algorithm's complexity depends on the following factors: (a) the number of resources needed for TC, (b) the system deadline, (c) location awareness, and d) the amount of workload being updated. Since the number of tasks continues to grow exponentially from the initial generation stage as the method approaches its queue size threshold with fixed time constraints, the complexity of tasks created is $O(2\hat{}n)$. Since it is a linear activity, updating the device and server in terms of their remaining computational resources and location awareness has O(n) complexity. The greatest iterative tolerance for Algorithm 1 is based on φ_1 with complexity $O(log_2 Z)$, where Z is an n-digit complex integer with a $log_{10}(\varphi_1^{-1})$ digit. Algorithms 2 and 3 are based on linear feasibility with maximum iterative $\varphi_2$and complexity $O\left(log_2 \frac{WL_{max}-WL_{rem}}{\varphi_2}\right)$, where m is the number of inequality constraints and n is the number of optimization variables. The total complexity is $O\left[(2\hat{}n) \times (log_2 Z) \times \left(log_2 \frac{WL_{max}-WL_{rem}}{\varphi_2}\right)\right]$ what moves with polynomial time.

5. Results and Discussion

The results of the proposed GTBTL-IoT algorithm and its usefulness in lowering overall task computation latency and optimizing workload allocation between local devices and MEC servers are shown in this section. It also illustrates how the algorithm is capable of making an optimal task-offloading decision to maximize time and resources.

5.1 Simulation Period Analysis

The simulation time of each approach has been investigated in this section. Within the simulation time, tasks are received, their status is updated, the locations of servers and devices are updated, server workloads are computed, tasks are split and offloaded, and the results are evaluated. The task generation rate influences the duration of the simulation. Table 3 shows the results of the simulation after four iterations namely #I1, #I2, #I3, and #I4 respectively.

Table 3. Simulation Period Analysis

Duration (in Secs)					**250**
Warm-Up Period (in Secs)					90
Number of Active Nodes					150
	#I1	**#I2**	**#I3**	**#I4**	**Average**
DTS	12	14	15	16	14.52
FA	10	12	13	14	12.25
DAOWA	9	11	12	13	11
GTBTL-IoT	8	9.5	9.8	10	**9.32**

The simulation time was set to 250 seconds, with a warm-up period of 90 seconds. Table 3 and Figure 4 show that the DTS method occupied the majority of the simulation time, 14.52 seconds on average. FA and DAOWA took 12.25 and 11 seconds, respectively. The suggested GTBTL-IoT method, on the other hand, outperformed the other three by requiring just 9.32 seconds on average for simulation over all four iterations, demonstrating its efficiency in terms of time and resource utilization.

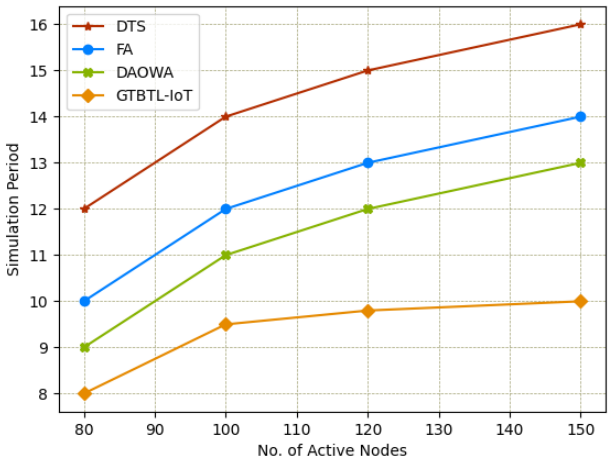

Figure 3. Simulated Performance Vs Number of Active Nodes

5.2 Latency Evaluation

The packet sizes of the incoming tasks vary from 100 to 1000 bytes. As a result, the task computation latency of these variable-sized tasks varies. The total latency is determined by a variety of factors, including the time it takes to record the available resources, the deployment area for task computing, the waiting or queue time, task transmission time (if transferred for remote computation), server queue time, and other critical variables. Table 4 summarizes the results of the latency evaluation after simulating for four iterations respectively.

Table 4. Latency Analysis

Duration (in Secs)					**250**
Warm-Up Period (in Secs)					90
Number of Active Nodes					150
	#I1	**#I2**	**#I3**	**#I4**	**Average**
DTS	148	150	165	180	160.75
FA	143	145	160	175	155.75
DAOWA	137	140	156	170	150.75
GTBTL-IoT	129	132	150	164	**143.75**

The goal is to develop a task-computing approach that will undoubtedly lower total work computation time. Based on Table 4 and Figure 5, it can be stated that GTBTL-IoT outperforms DTS, FA, and DAOWA in terms of minimizing total job computation delay. The overall latency induced by the request from the IoT node until the final results were delivered was simulated for all four algorithms across four iterations. Finally, when GTBTL-IoT caused the least latency overhead which is 143.75 seconds, DTS, FA, and DAOWA required 160.75 seconds, 155.75 seconds, and 150.75 seconds, respectively. Packet sizes of up to 1000 bytes were progressively and steadily raised to test performance.

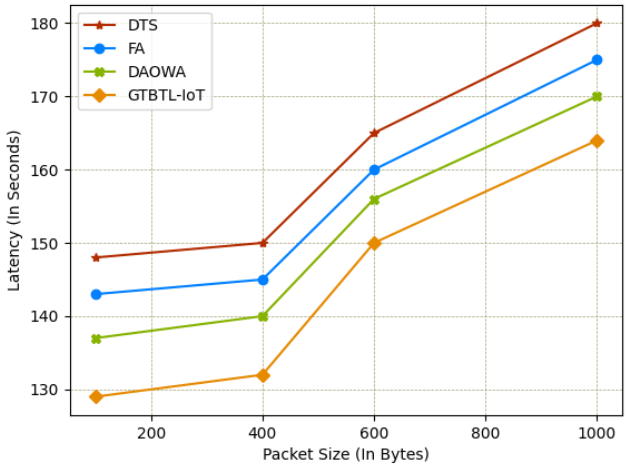

Figure 4. Latency Analysis Vs Packet Size

5.3 Task Offloading Decision

The choice to offload a task is dependent on the task requirements, memory requirements, and constraints such as workload overflow, device-server distance, allotted server and device memory, and maximum resource computing capabilities. Offloading a task as per the GTBTL-IoT algorithm, includes breaking it into two parts, calculating one locally and sending the other to the closest server available, and then computing both in parallel inside the same timeframe.

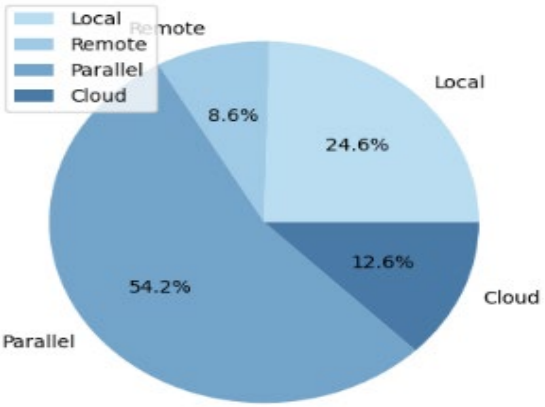

Figure 5. Task Offloading Decision

After Simulation, it was found that the GTBTL-IoT algorithm chose parallel computing for 54.2% of the incoming jobs, local computation for 24.6%, totally remote computation for 8.6%, and total cloud offloading for 12.6% of the workloads, as shown in Figure 6. The above comparison analysis demonstrates how effectively the proposed technique works. Also, an analysis of optimal task offloading decisions was carried out in the same way for DTS, FA, and DAOWA algorithms, whose results are illustrated in Figure 6.

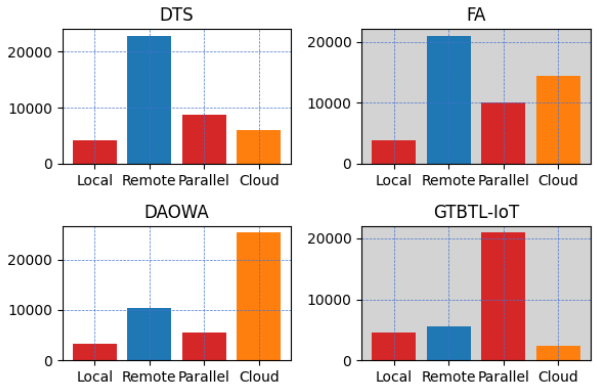

Figure 6. Task Offloading Efficiency

From Figure 6, it can be seen that DTS offloads its maximum generated task to the remote server fully, FA uses both parallel as well as full remote offloading approaches while DAOWA offloads its task fully to the cloud, which is again a complex process. However, as compared to these three, GTBTL-IoT uses parallel offloading and task computation approach which proves to be optimal for time and resource utilization.

5.4 Execution Time Analysis

The task's execution once it is delegated to a resource is the subject of concern, but its interpretation in a real-time

situation results are useful if they are received in a timely way. The task generation variable is continuously provided different values at various points in time t' for its examination across various algorithms, as shown in Figure. 8, to assess the effectiveness of the GTBTL-IoT algorithm. The task arrival rate in ms is given as rate = 0.1, 0.5, 1.0, 1.5, 1.7, 2.0, 2.5, 3.0, 3.5, and 3.8. The average number of tasks created in the IoT region with regard to 't' is used to calculate the task execution time. This analysis is summarized using Table 5.

Table 5. Execution Table Analysis

Duration (in Secs)					**250**
Warm-Up Period (in Secs)					90
Number of Active Nodes					150
	#I1	**#I2**	**#I3**	**#I4**	**Average**
DTS	98	212	654	802	441.5
FA	86	203	623	792	426
DAOWA	82	198	601	779	415
GTBTL-IoT	78	164	588	765	**398.75**

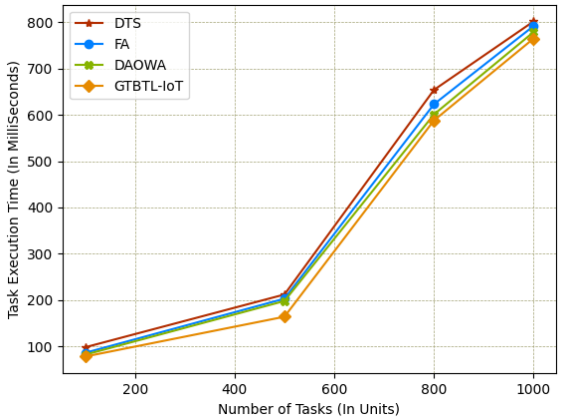

Figure 7. Task Execution Efficiency Vs Incoming Tasks

Based on Table 5 and Figure 7, it can be stated that GTBTL-IoT outperforms DTS, FA, and DAOWA in terms of task execution. The execution time can be determined based on available memory, wait times in queues, allotted resource blocks, and transmission times. After simulation for all four algorithms across four iterations, it was found that GTBTL-IoT took the least execution time that is 398.75 milliseconds seconds, as compared to DTS, FA, and DAOWA which took 441.5 milliseconds, 426 milliseconds, and 415 milliseconds, respectively. It should be remembered that $\boldsymbol{tTrans}_{\Lambda}^{t} = \mathbf{0}$ in the case of local TC. Therefore, a smaller number for the local computation timeframe is received. The proposed algorithm works better than any previous TO or computing technique.

5.5 Workload Distribution Analysis

The Poisson Distribution approach has been used to analyze the load placed on various local and remote resources for an array of tasks that are growing exponentially. It would be easier to analyze the workload imposed and the job computation efficiency when the task is spread among the various available resource blocks in accordance with the particular approach used. The task distribution has a significant influence on the computation and timely delivery of the results.

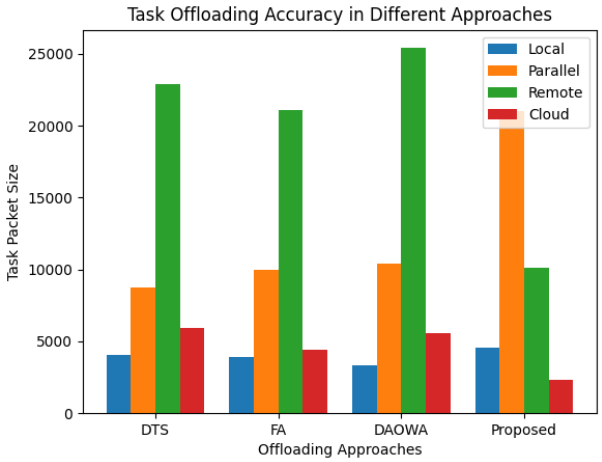

Figure 8. Workload Distribution

It is observed that the DTS spreads its maximum workload evenly between cloud and local resources, whereas the FA prefers to divide its workload over local resources. DAOWA places a remote edge server under its maximum workload, which may result in data packet congestion and an uneven distribution of workload calculations. However, the proposed method minimizes TC delay by making the most use of both local and remote computing resources while moving the smallest possible tasks to the cloud.

5.6 Resource Allocation

The term "resource block allocation time" essentially refers to the period of time during which a resource block will be assigned to the execution of a certain job. RBAlloc time affects how quickly randomly generated jobs are completed by the system deadline. This is crucial for preserving a real-time environment for real-time responses. The summary of resource allocation time of DTS, FA, DAOWA, and GTBTL-IoT is shown in Table 6.

Table 6. Resource Allocation time

Duration (in Secs)					250
Warm-Up Period (in Secs)					90
Number of Active Nodes					150
	#I1	#I2	#I3	#I4	Average
DTS	14	26	34	50	31
FA	12	21	30	45	27
DAOWA	10	18	27	40	23.75
GTBTL-IoT	8	15	23	36	**20.5**

The packet size taken for simulation and resource allocation of DTS, FA, DAOWA, and GTBTL-IoT algorithms is plotted together and shown in Figure 9.

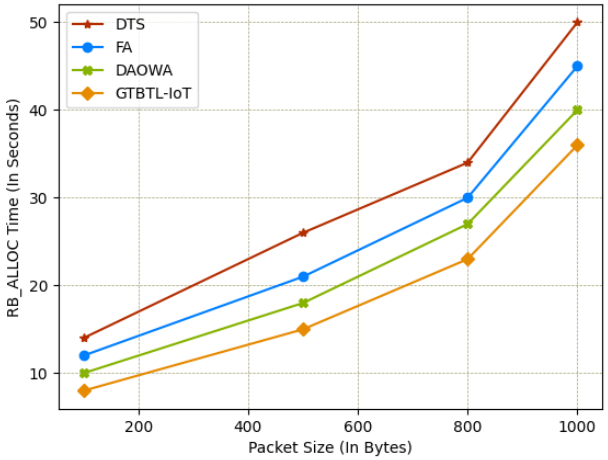

Figure 9. Resource Allocation Vs Packet Size

Based on Table 6 and Figure 9, it can be concluded that GTBTL-IoT has the shortest RBAlloc time when compared to the DTS, FA, and DAOWA Task Offloading Algorithms. For a 10-unit task queue size, GTBTL-IoT required 20.5 seconds on average of four iterations. DTS, FA, and DAOWA took 31, 27, and 23.75 seconds, respectively. As a result, it can be said that the suggested method performed better than the other three.

6. Conclusion

The study suggests that a partial TO strategy is more effective than sending the task to a remote server, and that distance significantly reduces offloading overhead and TC latency. Allocating tasks to local and remote regions in parallel for computing reduces task processing delays and improves system performance by enhancing responses. The simulation findings show that the GTBTL-IoT algorithm was effective in reducing the overall TC latency at 143.75 ms as compared to DTS at 160.75 ms, FA at 155.75 ms, and DAOWA at 150.75 ms. In the future, this work may be extended to enhance real-time responses in IoT environments with better decision-making using deep learning (DL) techniques. The DL models may be used to classify and prioritize the tasks for better resource utilization and their computation with a time-critical approach.

Data Availability.

The data used for test setup and experimental analysis may be made available by the authors upon request.

Conflicts of Interest.

The authors declare that they have no conflicts of interest regarding the publication of the research work.

Author Contributions.

- Research Idea Formulation, Research Content Collection, Designing of Algorithms, Experimental Setup-**Ms. Eram Fatima Siddiqui**
- Literature Survey, Result Interpretation, Research Writing, and Proofreading- **Dr. Tasneem Ahmed.**

Acknowledgements.

The authors are thankful to the Advanced Computing Research Laboratory, Department of Computer Application, Integral University, Lucknow for providing the necessary support to carry out the research work. The manuscript communication number issued by Integral University is IU/R&D/2023-MCN0002229.

References

[1] C. Swain *et al.*, "METO: Matching-Theory-Based Efficient task offloading in IoT-FOG interconnection networks," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12705–12715, Aug. 2021, doi: 10.1109/jiot.2020.3025631.

[2] M. Ali, N. Riaz, M. I. Ashraf, S. Qaisar, and M. Naeem, "Joint cloudlet selection and latency minimization in FoG networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4055–4063, Sep. 2018, doi: 10.1109/tii.2018.2829751.

[3] J. Xue and Y. An, "Joint task offloading and resource allocation for Multi-Task Multi-Server NOMA-MEC networks," *IEEE Access*, vol. 9, pp. 16152–16163, Jan. 2021, doi: 10.1109/access.2021.3049883.

[4] A. Rafiq, P. Wang, M. Wei, S. H. Hong, and N. N. Josbert, "Optimizing Energy consumption and Latency based on computation offloading and cell association in MEC enabled Industrial IoT environment," *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, Apr. 2021, doi: 10.1109/icsp51882.2021.9408693.

[5] S. Xia, X. Wen, Z. Yao, Y. Li, and G. Wang, "Dynamic Task Offloading and Resource Allocation for Heterogeneous MEC-enable IoT," *2020 IEEE/CIC International Conference on Communications in China (ICCC), 2020*, Aug. 2020, doi: 10.1109/iccc49849.2020.9238863.

[6] S. K. T, "EFFICIENT RESOURCE ALLOCATION AND QOS ENHANCEMENTS OF IOT WITH FOG NETWORK," *Journal of ISMAC the Journal of IoT in Social, Mobile, Analytics, and Cloud*, Sep. 2019, doi: 10.36548/jismac.2019.2.003.

[7] G. Cui, X. Li, L. Xu, and W. Wang, "Latency and energy optimization for MEC enhanced SAT-IoT networks," *IEEE*

Access, vol. 8, pp. 55915–55926, Jan. 2020, doi: 10.1109/access.2020.2982356.

[8] 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, Mar. 2019, doi: 10.1109/jsac.2019.2894306.

[9] J. Liu and Q. Zhang, "Adaptive Task Partitioning at Local Device or Remote Edge Server for Offloading in MEC," *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, May 2020, doi: 10.1109/wcnc45663.2020.9120484.

[10] W. Almughalles, R. Chai, J. Lin, and A. Zubair, "Task Execution Latency Minimization-based Joint Computation Offloading and Cell Selection for MEC-Enabled HetNets," *2019 28th Wireless and Optical Communications Conference (WOCC)*, May 2019, doi: 10.1109/wocc.2019.8770582.

[11] Y. Gu, W. Saad, M. Bennis, M. Debbah, and Z. Han, "Matching theory for future wireless networks: fundamentals and applications," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 52–59, May 2015, doi: 10.1109/mcom.2015.7105641.

[12] T. Cuong, N. H. Tran, C. Pham, Md. G. R. Alam, J. H. Son, and C. S. Hong, "A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing," *2015 International Conference on Information Networking (ICOIN)*, Jan. 2015, doi: 10.1109/icoin.2015.7057905.

[13] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in FOG-Cloud computing towards balanced delay and power consumption," *IEEE Internet of Things Journal*, p. 1, Jan. 2016, doi: 10.1109/jiot.2016.2565516.

[14] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in FOG Computing supported Software-Defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016, doi: 10.1109/tc.2016.2536019.

[15] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, "Computing Resource Allocation in Three-Tier IoT FOG Networks: A joint optimization approach combining Stackelberg game and matching," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1204–1215, Oct. 2017, doi: 10.1109/jiot.2017.2688925.

[16] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," *2017 European Conference on Networks and Communications (EuCNC)*, Jun. 2017, doi: 10.1109/eucnc.2017.7980678.

[17] T. Yang, R. Chai, and L. Zhang, "Latency Optimization-based Joint Task Offloading and Scheduling for Multi-user MEC System," *2020 29th Wireless and Optical Communications Conference (WOCC)*, May 2020, doi: 10.1109/wocc48579.2020.9114942.

[18] N. Nikaein, X. Vasilakos, and A. Huang, "LL-MEC: Enabling Low Latency Edge Applications," *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, Oct. 2018, doi: 10.1109/cloudnet.2018.8549500.

[19] B. Brik, P. A. Frangoudis, and A. Ksentini, "Service-Oriented MEC Applications Placement in a Federated Edge Cloud Architecture," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Jun. 2020, doi: 10.1109/icc40277.2020.9148814.

[20] A. Alnoman, S. Erkucuk, and A. Anpalagan, "Sparse code multiple Access-Based edge computing for IoT systems," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 7152–7161, Aug. 2019, doi: 10.1109/jiot.2019.2914570.

[21] R. Han, Y. Wen, L. Bai, J. Liu, and J. Choi, "Rate splitting on mobile edge computing for UAV-Aided IoT systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 4, pp. 1193–1203, Dec. 2020, doi: 10.1109/tccn.2020.3012680.

[22] R. Gu, L. Yu, and J. Zhang, "MeFILL: A Multi-edged Framework for Intelligent and Low Latency Mobile IoT Services," *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, May 2020, doi: 10.1109/wcnc45663.2020.9120786.

[23] I. Kovacevic, E. Harjula, S. Glisic, B. Lorenzo, and M. Ylianttila, "Cloud and edge computation offloading for latency limited services," *IEEE Access*, vol. 9, pp. 55764–55776, Jan. 2021, doi: 10.1109/access.2021.3071848.

[24] K. Chen, Y. Wang, Z. Fei, and X. Wang, "Power Limited Ultra-Reliable and Low-Latency Communication in UAV-Enabled IoT Networks," *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, May 2020, doi: 10.1109/wcnc45663.2020.9120565.

[25] H. Yoshino, K. Ota, and T. Hiraguri, "Adaptive Control of Statistical Data Aggregation to Minimize Latency in IoT Gateway," *2018 Global Information Infrastructure and Networking Symposium (GIIS)*, Oct. 2018, doi: 10.1109/giis.2018.8635712.

[26] J.-P. Hong, J. Park, W. Shin, and S. Beak, "Distributed antenna system design for Ultra-Reliable Low-Latency Uplink communications," *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, Jan. 2019, doi: 10.23919/elinfocom.2019.8706492.

[27] M. Yang, S.-Y. Lim, S.-M. Oh, and J. G. Shin, "An Uplink Transmission Scheme for TSN Service in 5G Industrial IoT," *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct. 2020, doi: 10.1109/ictc49870.2020.9289303.

[28] A. H. Ismail, T. A. Soliman, G. M. Salama, N. A. El-Bahnasawy, and H. F. A. Hamed, "Congestion-Aware and Energy-Efficient MEC Model with Low Latency for 5G," *2019 7th International Japan-Africa Conference on Electronics, Communications, and Computations, (JAC-ECC)*, Dec. 2019, doi: 10.1109/jac-ecc48896.2019.9051312.

[29] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Mobile edge computing and networking for Green and Low-Latency Internet of Things," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 39–45, May 2018, doi: 10.1109/mcom.2018.1700882.

[30] Z. Zhang *et al.*, "6G Wireless Networks: vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, Sep. 2019, doi: 10.1109/mvt.2019.2921208.

[31] N. Germenis, P. Fountas, and C. Koulamas, "Low Latency and Low Cost Smart Embedded Seismograph for Early Warning IoT Applications," *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, Jun. 2020, doi: 10.1109/meco49872.2020.9134088.

[32] G. Calice, A. Mtibaa, R. Beraldi, and H. Alnuweiri, "Mobile-to-mobile opportunistic task splitting and offloading," *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2015, doi: 10.1109/wimob.2015.7348012.

[33] N. Kherraf, S. Sharafeddine, C. Assi, and A. Ghrayeb, "Latency and Reliability-Aware workload assignment in IoT networks with mobile edge clouds," *IEEE Transactions*

on Network and Service Management, vol. 16, no. 4, pp. 1435–1449, Dec. 2019, doi: 10.1109/tnsm.2019.2946467.

[34] H. Hao, Y. Wang, Y. Shi, Z. Li, Y. Wu, and C. Li, "IoT-G: A Low-Latency and High-Reliability Private Power Wireless Communication Architecture for Smart Grid," *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Oct. 2019, doi: 10.1109/smartgridcomm.2019.8909773.

[35] J. Park and Y. Lim, "Balancing Loads among MEC Servers by Task Redirection to Enhance the Resource Efficiency of MEC Systems," *Applied Sciences*, vol. 11, no. 16, p. 7589, Aug. 2021, doi: 10.3390/app11167589.

[36] C. Duo, D. Jia, Q. Gao, B. Li, and Y. Li, "MEC Computation Offloading-Based Learning Strategy in Ultra-Dense Networks," Information, vol. 13, no. 6, p. 271, May 2022, doi: 10.3390/info13060271.

[37] S. Bebortta and D. Senapati, "Toward Cost-Aware computation offloading in IoT-Based MEC systems," National Academy Science Letters, vol. 46, no. 6, pp. 531–534, May 2023, doi: 10.1007/s40009-023-01260-9.

[38] Z. Ai, W. Zhang, M. Li, P. Li, and L. Shi, "A smart collaborative framework for dynamic multi-task offloading in IIoT-MEC networks," Peer-to-Peer Networking and Applications, vol. 16, no. 2, pp. 749–764, Jan. 2023, doi: 10.1007/s12083-022-01441-1.

[39] J. Chen, Y. Leng, and J. Huang, "An intelligent approach of task offloading for dependent services in Mobile Edge Computing," Journal of Cloud Computing, vol. 12, no. 1, Jul. 2023, doi: 10.1186/s13677-023-00477-9.

[40] H. H. Imtiaz and S. Tang, "Multi-Task Partial Offloading with Relay and Adaptive Bandwidth Allocation for the MEC-Assisted IoT," Sensors, vol. 23, no. 1, p. 190, Dec. 2022, doi: 10.3390/s23010190.

[41] F. Alenizi and O. F. Rana, "Minimising delay and energy in online dynamic fog systems," *arXiv (Cornell University)*, Dec. 2020, doi: 10.48550/arxiv.2012.12745.

[42] J. Almutairi and M. Aldossary, "A novel approach for IoT tasks offloading in edge-cloud environments," *Journal of Cloud Computing*, vol. 10, no. 1, Apr. 2021, doi: 10.1186/s13677-021-00243-9.

[43] L. Li, M. Guo, L. Ma, H. Mao, and Q. Guan, "Online workload allocation via FOG-FOG-Cloud cooperation to reduce IoT task service delay," *Sensors*, vol. 19, no. 18, p. 3830, Sep. 2019, doi: 10.3390/s19183830.

[44] E. Del-Pozo-Puñal, F. Garcia-Carballeira, and D. Camarmas-Alonso, "A scalable simulator for cloud, fog and edge computing platforms with mobility support," *Future Generation Computer Systems*, vol. 144, pp. 117–130, Jul. 2023, doi: 10.1016/j.future.2023.02.010.

[45] Zumnan, "GitHub - Zumnan/Dynamic-Intelligent-Edge-Task-Offloading-in-MEC-Network," GitHub. https://github.com/Zumnan/Dynamic-Intelligent-Edge-Task-Offloading-in-MEC-Network/tree/main