EDGEGO: ENABLING COST-EFFECTIVE 6G EDGE COMPUTING WITH MOBILE RESOURCE SHARING FOR IOT SYSTEMS

[1]Mrs. K. Rajakumari, [2]K. Sravya Sri, [3]G. Naga Surya Ramya Sri, [4]V. Padma Prannathi, [5]M. Mounika

[1]M. Tech, Assistant professor, [2,3,4,5]B. Tech, Student
Department of Electronics and Communication Engineering
Usha Rama College of Engineering and Technology
Andhra Pradesh, India.

Abstract- **With the incredible advancement of 5G technologies, 5G edge servers can now handle an increasing number of real-time and complicated computational jobs from Internet-of-Things (IoT) systems. While ultra-dense deployment is essential for 5G edge services, dense deployments are nearly impossible to achieve in the approaching era of 6G, which has an even shorter communication range. To solve this fundamental limitation, we present EdgeGO, a mobile resource-sharing architecture that uses mobile edge servers to deliver a cost-effective deployment of 6G edge computing, allowing edge resource sharing for huge IoT devices. EdgeGO, unlike standard mobile cloudlets, uses synchronization between request reception and result return to decouple the rigorous delay and resource needs of edge computing. As a result, server movement and job processing might be done in concurrently. In addition, EdgeGO has a two-layer iterative update technique that optimizes path planning and task scheduling simultaneously to improve overall task efficiency. Extensive simulation findings demonstrate that, by carefully regulating edge server mobility and task execution, EdgeGO can significantly boost resource utilization by 166.67% while decreasing the deployment cost of 6G edge computing by 25.58%.**

Index Terms: **6G edge computing, large IoT systems, mobile edge servers, resource sharing, job processing, maximizing network efficiency.**

I.INTRODUCTION

In the realm of telecommunications, the transition from 5G to 6G heralds not just an evolution but a revolution in connectivity. With the promise of unprecedented speed, capacity, and reliability, 6G technology is poised to redefine the landscape of edge computing. However, as we look forward to the era of 6G, we encounter a fundamental challenge: the limitations imposed by the communication range, which pose significant obstacles to achieving ubiquitous coverage through traditional deployment methods. In response to this challenge, we introduce EdgeGO, a pioneering framework designed to unlock the full potential of 6G edge computing by harnessing the power of mobile resource sharing. Built upon the foundation of mobile edge servers, EdgeGO offers a cost-effective and scalable solution for deploying edge computing in 6G networks, enabling efficient resource sharing for a multitude of IoT devices. One of EdgeGO's distinctive features is its ability to take use of the inherent a synchronicity between request acceptance and result delivery, effectively decoupling latency and resource allocation. EdgeGO embraces a synchronicity, allowing for parallel server mobility and responsiveness. EdgeGO also includes a powerful two- layer iterative update algorithm that was carefully designed to optimize both path planning and task scheduling. EdgeGO uses this iterative approach to dynamically adapt to changing network conditions and workload demands, ensuring that resources are always used optimally and tasks are efficient. In our pursuit of empirical validation, we conducted extensive simulations to assess EdgeGO's performance. The findings show that by carefully managing server mobility and job execution, EdgeGO delivers tremendous improvements in resource utilization, with an astonishing gain of 166.67%, while also lowering the deployment cost of 6G edge computing by 25.58%. In conclusion, EdgeGO marks a paradigm leap in the world of 6G edge computing, providing a versatile scalable framework that not only overcomes the inherent restrictions of communication range, but also realizes the full potential of resource sharing in vast IoT systems. As we embark on the path into the era of 6G, EdgeGO stands as a beacon of innovation, ready to revolutionize the way we perceive and harness the power.

II. RELATED WORK

1. Mobile Edge Computing (MEC) is a promising method for improving wireless network performance by moving processing and storage resources closer to end users. Researchers have looked into numerous aspects of MEC, such as resource allocation, workload offloading, and mobility management [1].
2. Edge computing for 5G networks: Edge computing integration with 5G networks has been extensively studied to provide low-latency, high-throughput applications. Previous research focused on optimizing edge server placement, workload scheduling, and network slicing to improve the performance of 5G edge computing systems [2]
3. Resource Sharing for Edge Computing: Effective resource sharing among edge servers can increase resource utilization while reducing operational costs. A previous study proposed solutions for dynamic resource allocation, load balancing, and collaborative processing among edge servers [3].
4. 6G Edge Computing Challenges: New difficulties arise with 6G technology, including limited communication range and scalability of edge computing deployments. Recent research has highlighted the importance of creative approaches to resolving these challenges, such as leveraging mobility and dynamic resource allocation [4].
5. Asynchronous Task Processing: Asynchronous task processing improves efficiency and scalability in distributed computing systems. Previous research has investigated approaches for separating task execution and communication latency, such as speculative execution, pipelining, and parallelism [5].
6. Resource Allocation in Edge Computing: Dynamic resource allocation methodologies can be used to manage resources more effectively. Previous research investigated approaches such as reinforcement learning, game theory, and optimization algorithms to dynamically distribute resources in response to changing workload needs and network conditions [6].
7. Mobility Management in Edge Computing: Edge computing systems containing mobile servers or devices require effective mobility management. Research in this subject has focused on solutions for seamless handovers, anticipatory mobility, and context-aware resource allocation to provide continuous service delivery in dynamic mobile situations [7].
8. The Importance of Task Offloading in IoT Systems: Offloading computing processes between edge devices and cloud servers boosts system performance. Previous research has investigated offloading algorithms that use task characteristics, network conditions, and energy constraints to reduce latency and energy consumption [8].
9. 6G Network Architecture and Technologies: A new study investigates architectural design and technological achievements for 6G networks. Terahertz transmission, AI- driven networking, and holographic data storage have all been studied as answers to the demanding requirements of future wireless communication systems [9].
10. Edge Computing Security and Privacy: The distributed nature of data processing and storage presents security and privacy challenges. Previous research has focused on data integrity, access control, and privacy-preserving computation as ways to mitigate security risks and safeguard user privacy in edge computing systems [10].
11. Federated Learning in Edge Computing: Federated learning is a promising technique for training machine learning models across several devices while protecting data privacy. Previous research has investigated federated learning strategies for edge computing environments, focusing on challenges such as model aggregation, communication efficiency, and privacy protection [11].
12. Energy-Efficient Computing in IoT: Energy efficiency is critical for IoT devices in resource-constrained environments. Previous research has focused on energy-efficient computing strategies such as dynamic voltage scaling, work scheduling, and and low-power communication protocols to increase the battery life of IoT devices and reduce energy usage [12].
13. Block chain for Edge Computing: Decentralized and tamper-resistant data management methods are excellent for edge computing settings. Block chain integration with edge computing has been studied as a way to improve data integrity, provenance tracking, and trustworthiness in scattered edge networks.
14. QoS provisioning guarantees that edge computing systems meet performance requirements. Previous research has created QoS-aware resource allocation algorithms, admission control approaches, and traffic management strategies to meet the diverse QoS requirements of edge applications [14].
15. Edge Computing for Real-time Applications: Real-time applications such as augmented reality, autonomous cars, and industrial automation necessitate low latency processing and high reliability, making edge computing an appealing computing model. Researchers in edge computing have investigated strategies for real-time work scheduling, latency optimization, fault tolerance [15].

III. PROPOSED METHODOLOGY

EdgeGO, the suggested methodology, is made up of numerous important components and strategies that aim to enable cost-effective deployment and efficient resource sharing in 6G edge computing environments. Below, I discuss the

major parts of the EdgeGO framework:

EdgeGO uses mobile edge servers with compute and storage capabilities to deliver edge computing services to IoT devices. These servers are deliberately positioned to cover areas with a high concentration of IoT devices, while also being mobile enough to react to changing network circumstances and workload requirements. EdgeGO leverages the asynchronous nature of request receiving and result delivery in edge computing environments. EdgeGO optimizes resource utilization and responsiveness by separating task processing from communication delay. This section introduces the system concept and formulates the cooperative path planning and task scheduling problem in the EdgeGO framework. The notations used throughout the paper are summarized in Table 1. As a result, the overall delay is substantially lower than with movable cloudlet techniques. Although such an approach is predicted to drastically lower deployment costs while maintaining comparable performance, it confronts the following concerns.

1) Request collecting and result delivery will be combined, making path planning for mobile edge servers a non-trivial issue.

2) The path plan of mobile edge servers must incorporate task processing time, which is influenced by the moving delay of the planned paths.

As a result, mobility management and task scheduling cannot be handled separately. In the next part, we will demonstrate EdgeGO's main design and explain how we overcome the aforementioned difficulties while improving resource utilization and lowering deployment costs.

TABLE I NOTATIONS

Notations Descriptions

D Set of IoT devices.
qi Task requests from device i.
ci The required CPU cycles to accomplish task qi.
ui The size of computation input of task qi.
si The required storage space of task qi.
li The location of task qi.
α The average moving speed of the mobile edge server k.
C Storage capacity of the mobile edge server.
F Computation capability of the mobile edge server.
γi Data transmission rate from device di to the mobile edge server.
P Movement path of the mobile server.
pi The i − th stop point in the server movement path. tmove i,j Server moving time from i to j.
dis(i, j) Distance between device di and dj .
R Results of task scheduling ri Priority of task qi.
higher(i) The set of tasks that have higher priority than qi.
tarrive i The time when the mobile server arrive at the point pi.
tleave i The time when the mobile server leave the point pi.
Ti The stay duration at pi.

t finish higher(k) The time when qi's higher priority tasks are completed.

Two-Layer Iterative Update Algorithm:

EdgeGO uses a powerful two-layer iterative update method that optimizes both path planning and task scheduling. This algorithm dynamically adapts server mobility patterns and task assignment algorithms in response to real-time network conditions and workload factors, providing optimal resource utilization and task efficiency. EdgeGO enables resource sharing among mobile edge servers, leading to better resource utilization and lower implementation costs. EdgeGO ensures optimal utilization of compute and storage resources across the edge computing infrastructure by implementing processes like as load balancing, collaborative processing, and dynamic resource allocation. EdgeGO's mobility management features provide consistent service delivery to IoT devices, even when servers move around. These approaches use anticipatory mobility, context-aware resource allocation, and seamless handover techniques to minimize the impact of server mobility on task execution and network performance.

A. System Model

As seen in Figure 1, we envision a 6G edge computing network with large IoT devices and a mobile edge server. The collection of n IoT devices, indicated as D = {d1, d2,..., dn}, are deployed over a two-dimensional service area and can create multiple compute-massive and delay-tolerant task requests (e.g., HD cameras upload monitoring data every hour). The device di's task request can be expressed as a three-term qi = {ci, ui, si}. Here, ci represents the neededother tasks, as seen in Figure 3. The server moving time t move i,j can be obtained as:

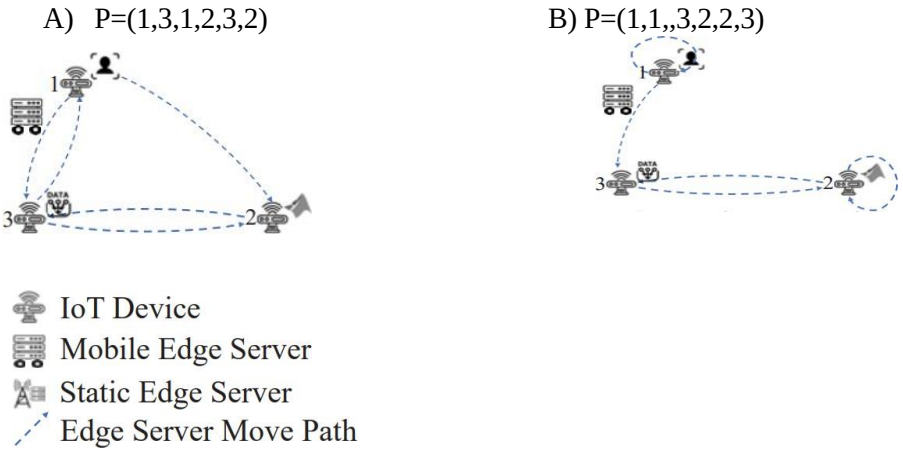


Fig. 1. The representation of server moving paths.

The total number of CPU cycles required to complete this work is denoted by ui, which represents the size of the computation input data, and si, which represents the storage space required for this task. We ignore the time it takes the edge server to deliver the compute results back to IoT devices, as did many other research [15]. Because, in many cases, calculation output data is significantly smaller than input data. The gadget di's location is stated using two-dimensional coordinates (xi, yi). Mounting the mobile edge server S on a mobile robot or unmanned aerial vehicle allows for α-speed movement. The server also contains a limited storage resource C for storing computing task data, as well as a CPU with a maximum frequency F (cycles per second) for processing computing jobs that have been offloaded from IoT devices. It's worth mentioning that storage capacity has a direct impact on the number of tasks offloaded during transportation.

Path Planning and Task Scheduling

Before a server S starts moving and collecting task requests from the IoT devices, it should make two decisions:

1) Path Planning: In contrast to the path planning difficulty in classic mobile cloudlets [10], [12], and [14], cloudlets will only pass each IoT device once. The EdgeGO framework allows the mobile edge server to pass one IoT device once (the server computes the task in place) or twice (the server computes the task while moving). This implies that the path planning problem in this scenario is not a traditional Travelling Salesman Problem (TSP). To clearly express these two possibilities for the mobile edge server for one IoT device, we use a 1 × 2n dimensional vector P = (p1, p2,..., p2n). Let pi ∈ {1,..., n} represent the i-th stop point on the server movement track. Assuming pi = k and pi+1 = m, we will have two situations:

1) k = m, which means that the mobile edge server S will stop at lk until it has completed the responsibilities of qk.

2) k = m, which means S would leave lk after receiving qk's input data and compute this task while moving or uploading

$$t_{i,j}^{move} = \frac{dis(i,j)}{\alpha}$$

Eq.1

where dis(i, j) denotes the Euclidean distance between li and lj

, which can be calculated by

dis(i, j) = (xi − xj)2 + (yi − yj)2.

The EdgeGO framework's mobile edge server can use parallel server moving and task computing, which means it can handle numerous jobs. To completely use the server's processing and storage capabilities while also reducing the completion time for serving all IoT devices in the vicinity, an efficient job scheduling algorithm must be developed. In this paper, we utilise R = (r1, r2,..., rn) to describe the task scheduling outcome, with ri denoting the priority of job qi from device di. The lower the value, the greater the priority. If the mobile edge server is handling multiple tasks at the same time, the job with the highest priority will be processed first. And the set of tasks, which have higher priority than qi, is denoted as higher(i). higher(i) can be acquired as

$$higher(i) = \{q_j \,|\, r_j < r_i, \forall j \in N\}$$

Eq.2

Let t arrive i represent the time the mobile edge server arrives at the point pi. And let t leave i represent the time when the mobile edge server leaves the point pi. It is important to note that each device's position will appear twice in the server's moving path. The first is for uploading duties, while the second is for calculating tasks. The time when the edge server gets at pi can be calculated using the server mobility model in Eq. (1) as follows:

$$t_i^{arrive} = t_{i-1}^{leave} + t_{i-1,i}^{move}$$

Eq.3

Specifically, t arrive 1 equals 0. And t depart 2n is the total delay after the mobile edge server has served all IoT devices in this area. Ti describes the duration of stay at the place pi. There is a link between the time the edge server arrives at point pi and when it leaves.

$$t_i^{leave} = t_i^{arrive} + T_i$$

Eq.4

Assuming that the mobile edge server stop at location lk in the path point pi and pj , that is pi = pj = lk, the stop duration at pi can be computed as

$$T_i = \begin{cases} u_k/\gamma_k & \text{if i} < \text{j} \\ c_k/F + t_{higher(k)}^{finish} - t_i^{arrive} & \text{if i} > \text{j} \end{cases}$$

γk denotes the transmission rate between the mobile edge server S and device dk. As shown in Eq. (6), γk can be obtained by Shannon formulation with 6G channel multiplexing (orbital angular momentum multiplexing).

$$\gamma_k = Blog_2\left(1 + \frac{G_{k,s}}{\bar{\omega_0} + \sum_{i \in D \setminus d:_{a_i = a_j}} G_{i,j}}\right)$$

Eq.6

Where B denotes the wireless bandwidth, Gi,s denotes the channel gain from device i to the server and ω¯0 is the background noise power. Eq. (5) implies that there are two situations when the edge server stops at one point.

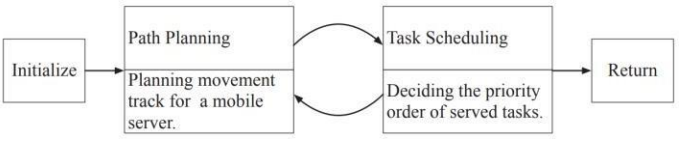


Fig.2. Overview of the proposed algorithm.

1. For the path point pi, this is the first time the server has stopped at lk. In this case, the edge server spends time receiving the input data for task qi. Therefore, the stay duration Ti can be represented as uk/γk.
2. For the path point pi, this is the second time the server has stopped at lk. In this case, the edge server spends time processing jobs and communicating compute results to devices. According to the system model mentioned above, we ignore the time it takes the server to communicate calculation results back to the IoT device. The likelihood that the higher priority jobs, higher(k), that were offloaded to the server, have not yet been completed before the

server arrives at pi. There are two sub-situations.

Problem Formulation

Our goal is to reduce the overall latency for all IoT devices in this region, denoted as t finish 2n, by optimizing the server moving path and task scheduling simultaneously while meeting the storage capacity limitation for the mobile edge server. We optimize this aim since a short overall delay allows the mobile edge server to service other huge IoT devices in the next area as quickly as possible. The problem is stated as follows:

$$\min_{\mathbf{P},\mathbf{R}} \quad t_{2n}^{finish}$$

Eq.8

$$\text{s.t.} \quad S(p_i) \leq C \qquad \forall p_i \in \mathbf{P}$$

Eq.9

$$S(p_i) = \sum_{j=1}^{2n} \sum_{i=1}^{j} \sum_{p_i=p_j=l_k}^{n} s_k$$

Eq.10

Here, S(pi) represents the edge server's occupied storage space at point pi, which can be determined using Eq. (10). Eq. (9) ensures that at no time do the offloaded processes surpass the edge server's storage capacity. The more computationally intensive jobs offloaded to the server, the shorter the system's overall delay will be. However, the server's limited storage capacity limits the amount of tasks that may be transferred to the edge server. This optimization issue is NP-hard because the path planning sub problem is a knapsack problem [15]. In the following part, we will present a two-layer iterative updating approach for efficiently solving our offloading problem.

A. Algorithm Design:

As stated in the preceding section, we jointly optimize path planning and task scheduling to reduce the total delay of huge IoT jobs. The difficulties of this problem stems from the fact that the two sub-problems are coupled, which means that solving one sub-problem necessitates solving the other. We present an Iterative Path Planning and Task Scheduling Update (IPTU) technique to address the coupled challenges. We use IPTU to optimise one of the decisions iteratively while fixing the other variables until the method converges. Figure 4 depicts a high-level link between the two sub-algorithms.

B. Path Planning in IPTU Algorithm

In the outer layer of IPTU, the server moving path is iteratively updated to achieve the smallest overall latency with the acceptance probability ρ, as illustrated in Algorithm 1. First, we choose the server's moving path using the 2- OPT algorithm (Steps 3–4), which is a local TSP searching technique. It is

important to note that our scenario is not a classic TSP; the mobile edge server may stop at one node twice, once to upload data and again to return results. As a result, we are unable to employ existing TSP algorithms directly.

Second, we'll determine whether the path is inside the edge server's storage capacity limit. If it does not reach the limit, remove it and create a new one (Steps 5–8). In Step 11, we build a new path P' based on P∗, indicating that the edge server will stop at one node until the task is completed. We compute the path P' since the solution to the server going through one node twice is unsuitable for tasks with low computational demands. As a result, we evaluate the overall latency between these two pathways to select which option to choose (Steps 9–14). Finally, based on the moving path P∗, we may create the task scheduling policy and further reduce the overall delay through the Algorithm 2 (Step 15).

Considering that the greedy algorithm may fall into a local optimal solution, we design the probability of accepting current

ALGORITHM 1	ALGORITHM 2
Path Planning in IPTU Algorithm Input: Disn×n, D1×n, Q1×n, α, F, C Output: P1×2n = {pi}, R1×n = {ri}, t f inish 2n . 1: Initialize P ← (l1, l1, ..., ln, ln), R ← (1, ..., n), t f inish 2n ← Inf. 2: for iteration = 1, 2, ... do 3: Randomly select two nodes pi, pj ∈ P. 4: P∗ ← Invert the order of nodes between pi and pj . 5: Compute S(pi) for each path point. 6: if ∃S(pi) > C then 7: goto line 2. 8: end if 9: Generate the path P′, with server computing in place. 10: Based on the path P and P′ , separately compute the overall delay t f inish 2n and ′ t f inish 2n . 11: if ′ t f inish 2n < tf inish 2n then 12: P∗ ← P′, t f inish∗ 2n ← ′ t f inish 2n . 13: goto line 16. 14: end if 15: Based on the path P∗ and time t f inish 2n , make task scheduling policy R∗ and compute the overall delay t f inish∗ 2n , by Algorithm 2. 16: Let P ← P∗, R ← R∗, with the probability ρ. 17: end for	Task Scheduling in IPTU Algorithm Input: P1×2n, t f inish 2n , Disn×n, D1×n, Q1×n, α, F, C Output: R1×n, t f inish∗ 2n 1: Initialize R ← (1, ..., n), t f inish∗ 2n = t f inish 2n , List = {}. 2: Compute the deadline time for each device t DDL k ← t arrive i , {∀i | pi = pj = lk and i < j}. 3: if t f inish k > tDDL k then 4: Add device dk in List. 5: end if 6: for all di ∈ List do 7: while t f inish i > tDDL i and ri > 1 do 8: ri ← ri − 1. Update t f inish i and t DDL i . 9: end while 10: if t f inish i ≤ t DDL i then 11: Delete device di in List. 12: end if 13: end for 14: Compute the overall delay t f inish∗ 2n .

Considering that the greedy algorithm may fall into a local optimal solution, we design the probability of accepting current

$\rho = \frac{1}{1+exp\left[\left(t_{2n}^{finish*} - t_{2n}^{finish}\right)/\omega\right]}$, which associates the moving path decision with the objective value (Step 16). And ω is a smooth parameter that can influence the convergence of IPTU algorithm. When ω ← 0 , the proposed IPTU algorithm can converge to the optimal moving path with high probability.

C. Task Scheduling in IPTU Algorithm

The important element of Algorithm 2 is that the server's second halting time at the device can be lowered by altering the priority of the associated tasks. It indicates that if the movement path is known, task scheduling can lower the overall completion time. In Algorithm 2, we first calculate the deadline for each device, which is indicated by dDDL k. The deadline time for device dk is the time at which the edge server switches to this device for the second time. If the job from qk cannot be completed before dDDL k, dk will be added to the list of devices loaded after the deadline (Steps 2–5). priority of tasks in this list to reduce the overall delay (Step 6 to 14).

IV. SIMULATION AND EVALUATION

EdgeGO's effectiveness is evaluated using extensive simulations of realistic 6G edge computing scenarios. Performance indicators such as resource utilization, deployment cost, latency, and throughput are used to evaluate EdgeGO's effectiveness in enhancing the efficiency and scalability of 6G edge computing deployments. EdgeGO provides a comprehensive framework for cost-effective deployment and efficient resource sharing in 6G edge computing settings, enabling huge IoT systems to realize their full potential.

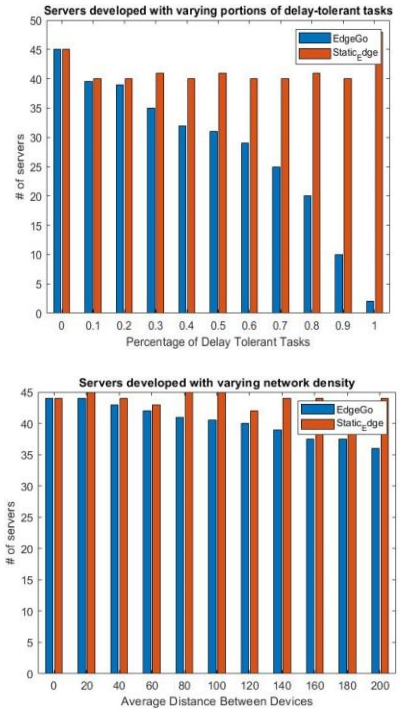


Fig. 3. Deployment cost of 6G edge system for massive IoT with different frameworks.

We employ three main criteria to assess the performance of the aforementioned works.

1) The number of standardized 6G edge servers required to cover the entire target area with large IoT devices. The term "fully cover" refers to the 6G edge network's ability to perform all created IoT workloads.
2) Computational efficiency through path planning and task scheduling.
3) Total resource utilization in terms of computing duration over time.

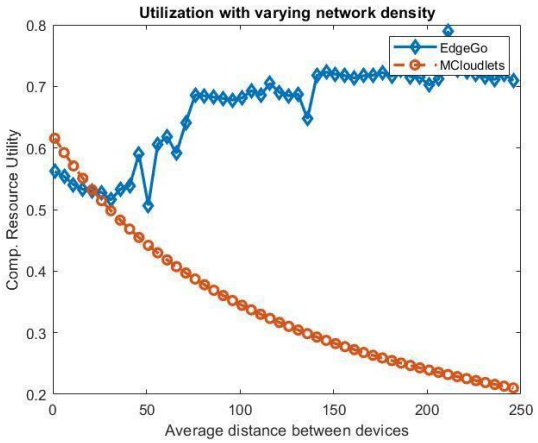
We run simulations with EdgeGO, StaticEdge, and MCloudlets and compare their performance to gain system insights. Furthermore, we investigate many actual scenarios by adjusting factors such as network scale, network density, workloads from IoT devices, edge server storage, and so on.

Figure 3 displays the implementation costs of 6G edge computing for huge IoT across several frameworks. Figure 3(a) depicts the number of servers deployed at various network densities. It is worth remembering that we are focusing on the same target area, thus a dense network suggests that more IoT devices will be covered in that area. With the density increasing, more edge servers are required for both frameworks.

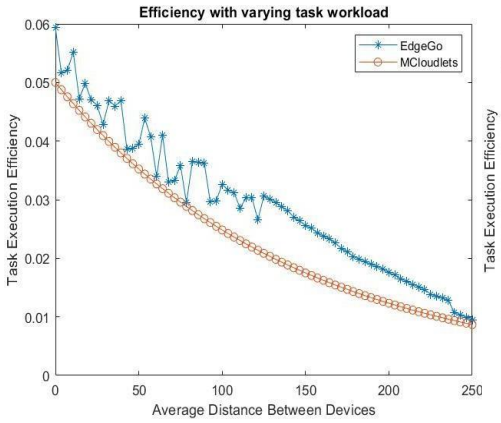
EdgeGO requires fewer servers than StaticEdge, and the reduction grows with higher density. The number of servers remains consistent with variable inter-device distance because in static deployment, the number of edge servers is dictated by task demands and is unaffected by distance, whereas EdgeGO employs considerably fewer edge servers when the network is sparse. Figure 3(b) depicts the deployed servers with various amounts of delay-tolerant IoT jobs. Remember that EdgeGO uses mobile edge servers to cover delay-tolerant operations; as the portion increases, the number of servers dramatically falls with EdgeGO, while remaining identical to StaticEdge. Specifically, with 50% IoT devices performing delay-tolerant functions, the deployment cost is reduced by 25.58%. Figure 6 compares EdgeGO's performance to the most recent work with mobile edge servers, MCloudlets.

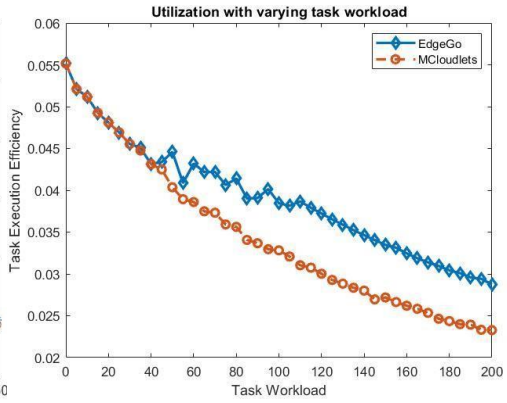
The presented methodology uses a MATLAB script to run simulations and generate various charts to compare the performance of the proposed EdgeGO framework to a baseline approach (referred to as "Static_Edge" or "MCloudlets"). The script has several portions, each concentrating on a distinct component of the evaluation. Below, I'll describe the results and graphs created by each section of the script.

The number of standardized 6G edge servers required to cover the entire target area with large IoT devices. The term "fully cover" refers to the 6G edge network's ability to perform all created IoT workloads.

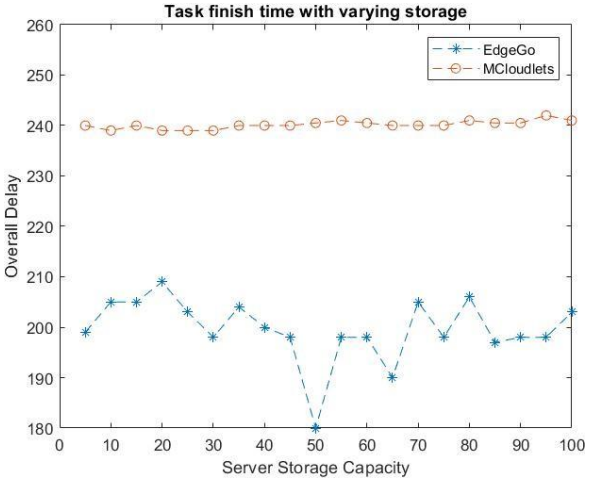


a) Utilization with varying network density.



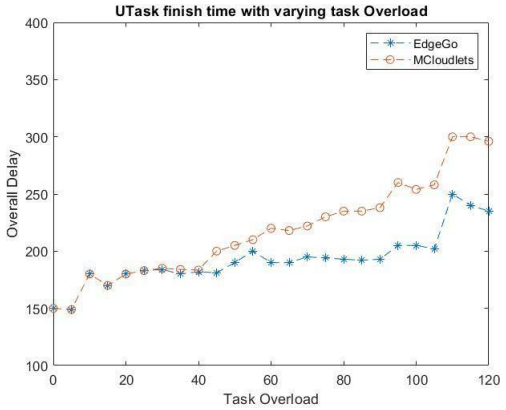


(b) Efficiency with varying task workload. (c) Task finish time with varying storage.



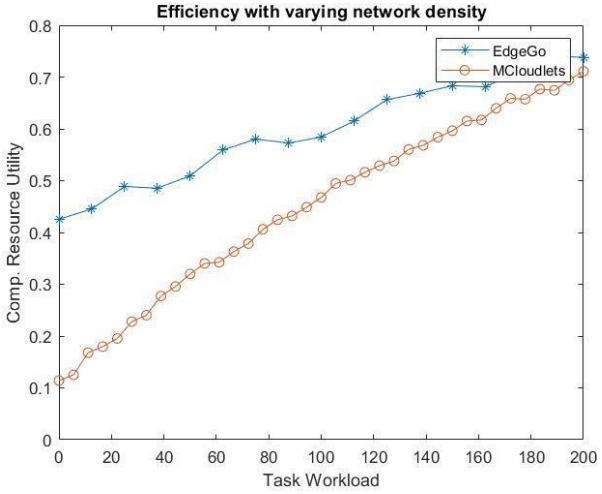
(d) Efficiency with varying network density

(e) Utilization with varying task workload



(f) Task finish time with varying task workload.

Fig. 6. Performance comparison in terms of resource utilization, task execution efficiency and overall completion time.



Network Density Analysis:

This replicates different network densities by altering the average distance between the devices. The variable 'avg_dis_bt_dev' indicates the average distance between devices. The 'no_of_servers_dis' variable represents the number of servers created at various network densities. The first graph should display the number of EdgeGO servers and the baseline method at various network densities. Interpretation: This graph shows how EdgeGO and the baseline approach adapt server installations to varying network density. It demonstrates which method is more effective in terms of server utilization and deployment.

Task Delay Tolerance Analysis:

This simulates scenarios with different percentages of delay-tolerant tasks. The 'per_del_tol_task' variable indicates the percentage of delay-tolerant tasks. The 'no_of_servers_del' variable specifies the number of servers created for various percentages of delay-tolerant jobs.The second graph should indicate EdgeGO's server count and the baseline technique for various percentages of delay-tolerant workloads. Interpretation:

This graph shows how EdgeGO and the baseline approach adapt server deployments based on the proportion of delay-tolerant activities. It aids comprehension of the effect of task factors on server deployment techniques.

Task Workload Analysis:

This simulates scenarios with varying task workloads. The `Task_workloads` variable represents the task workload.The `Task_exec_eff_edgo` and `Task_exec_eff_mc` variables represent the task execution efficiency for EdgeGO and the baseline approach, respectively. The third generated graph should show the task execution efficiency of EdgeGO and the baseline approach under different task workloads. Interpretation: This graph compares the efficiency of EdgeGO and the baseline approach in executing tasks under different workloads. It provides insights into how each approach handles varying task demands.

Network Density and Task Workload Analysis:

This simulates scenarios with varying network densities and task workloads simultaneously. The `avg_dis_bt_devutl` variable represents the average distance between devices. The `Comp_res_utl_edgo` and

`Comp_res_utl_mc` variables represent the computational resource utility for EdgeGO and the baseline approach, respectively. The fourth generated graph should show the computational resource utility of EdgeGO and the baseline approach under different combinations of network densities and task workloads. Interpretation: This graph illustrates the efficiency of EdgeGO and the baseline approach in utilizing computational resources under varying network densities and task workloads.

Storage Capacity Analysis:

This simulates scenarios involving changing server storage capabilities. The'serv_stor_cap' variable represents the server's storage capacity. The variables 'ovr_del_edg_st' and 'ovr_del_mc_st' denote the overall job completion time for EdgeGO and the baseline method, respectively. The fifth generated graph should indicate EdgeGO's overall task completion time and the baseline method for varied server storage capacity. Interpretation: This graph compares EdgeGO's performance to the baseline technique for finishing tasks with varied server storage capacities. It helps to understand how storage capacity affects task completion time. **Task Overload Analysis:** This simulates scenarios with varying task overload. The `task_workl` variable represents the task overload. The sixth generated graph should show the overall task finish time of EdgeGO and the baseline approach under different levels of task overload. This graph evaluates how EdgeGO and the baseline approach handle task overload situations, providing insights into their scalability and robustness.

Distance Between Devices Analysis: This simulates scenarios involving different distances between devices.The variable 'avg_dis_bet_dev' indicates the average distance between devices. The 'task_exe_eff_edgo' and 'task_exe_eff_mc' variables denote the task execution efficiency of EdgeGO and the baseline approach, respectively. The seventh generated graph should compare the task execution efficiency of EdgeGO versus the baseline technique at various distances between devices. This graph compares EdgeGO's performance to the baseline technique in efficiently performing jobs at various distances between devices.

Task Workload and Network Density Analysis: This simulates scenarios with different task demands and network densities. The 'task_work_utl2' variable refers to the task workload. The 'com_res_util_edgo' and 'com_res_util_mc' variables denote the computational resource utility of EdgeGO and the baseline approach, respectively. The eighth generated graph should highlight EdgeGO's computational resource usefulness vs the baseline technique for various task workloads and network densities. Interpretation: This graph illustrates how EdgeGO and the baseline approach manage computing resources under different task workloads and network density. These analysis and info graphics give a detailed evaluation of the proposed EdgeGO architecture against the baseline method across many scenarios and parameters. They help identify the strengths and drawbacks of EdgeGO in various deployment scenarios and shine light on its potential benefits in real-world edge computing environments.

V. CONCLUSION

In conclusion, EdgeGO emerges as a pioneering solution to the pressing challenges of deploying cost-effective 6G edge computing infrastructures, particularly within the expansive landscape of massive IoT systems. Capitalizing on the advancements in 5G technologies, EdgeGO introduces a paradigm shift by introducing mobile edge servers and innovative resource-sharing mechanisms. This approach effectively circumvents the inherent limitations posed by the ultra-dense deployment requirements of 5G edge services, thereby opening new avenues for scalable and efficient 6G edge computing. Through the strategic utilization of asynchronization and a meticulously crafted two-layer iterative update algorithm, EdgeGO demonstrates remarkable improvements in resource utilization and deployment cost reduction. Extensive simulations substantiate EdgeGO's efficacy, showcasing a staggering 166.67% boost in resource utilization coupled with a substantial 25.58% reduction in the deployment cost of 6G edge computing. These findings underscore the transformative potential of EdgeGO in reshaping the landscape of edge computing infrastructure deployment, poised to seamlessly integrate with massive IoT ecosystems and spearhead the realization of efficient and scalable 6G networks.2

VI. FUTURESCOPE

In future, We Firstly, transitioning from simulations to real-world implementations will be paramount in validating EdgeGO's performance and efficacy under diverse operational conditions. Field trials and pilot projects offer invaluable insights into the practical challenges and opportunities inherent in deploying EdgeGO across varied environments. Secondly, continuous refinement of EdgeGO's optimization techniques, particularly in path planning and task scheduling algorithms, holds the potential to unlock even greater efficiency and scalability. Exploring advanced optimization methodologies, such as machine learning and reinforcement learning, can further elevate EdgeGO's performance in dynamic edge computing environments. Thirdly, bolstering EdgeGO's security and privacy features remains imperative in safeguarding sensitive data and fostering trust in edge computing operations. Developing robust encryption, authentication, and access control mechanisms tailored to EdgeGO deployments is essential in this regard. Additionally, exploring the integration of EdgeGO with emerging technologies like blockchain, federated learning, and quantum computing presents exciting avenues for innovation and expansion of

EdgeGO's capabilities. Lastly, advocating for standardization and interoperability efforts to facilitate seamless integration and collaboration across diverse edge computing frameworks will be pivotal in EdgeGO's journey towards widespread adoption and industry impact. By embracing these future directions, EdgeGO is poised to shape the future of edge computing, unlocking new frontiers in IoT applications, smart cities, autonomous systems, and beyond.

REFERENCES:

1. Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for VM-based cloudlets in mobile computing. IEEE Pervasive Computing, 8(4), 14-23.
2. Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 19(4), 2322-2358.
3. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637-646.
4. Liu, W., Chen, H. H., & Wu, J. (2021). Future generation networks: 6G challenges and opportunities. IEEE Network, 35(6), 12-19.
5. Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
6. Chen, M., Zhang, Y., Li, Y., & Mao, S. (2019). Smart edge computing and caching for 5G/6G wireless networks:Potentials, approaches, and research challenges. IEEE Communications Magazine, 57(3), 22-28.
7. Chen, Y., Zhang, Z., Xu, C., & Mao, S. (2019). A survey of mobility management in mobile edge computing. IEEE Access, 7, 103090-103107.
8. Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., & Niakanlahiji, A. (2019). A survey on mobile edge computing: Approaches, challenges, and future directions. IEEE Communications Surveys & Tutorials, 21(3), 2429-2456.
9. Akyildiz, I. F., Pierobon, M., Balasubramanian, V., & Kountouris, M. (2020). The age of terahertz wireless. IEEE Wireless Communications, 27(2), 162-169.
10. Wang, Q., Ren, K., & Wang, J. (2019). Security and privacy for mobile edge computing: A survey. IEEE Communications Surveys & Tutorials, 21(1), 867-904.
11. Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492.
12. Gia, T. N., Jiang, M., & Rahmani, A. M. (2017). Energy efficient mobile cloud computing: A review. IEEE Access, 5, 20584-20605.
13. Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). An overview of blockchain technology: Architecture, consensus, and future trends. IEEE International Congress on Big Data, 557-564.
14. Sonmez, C., Ozgovde, A., Ersoy, C., & Alagoz, F. (2018). QoS-aware resource allocation for edge computing. IEEE Transactions on Cloud Computing, 8(4), 1102-1115.
15. Shi, W., Dustdar, S., & Sun, Y. (2016). The promise of edge computing. Computer, 49(5), 78-81.