



IJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: <https://doi.org/10.22214/ijraset.2026.80816>

www.ijraset.com

Call:  08813907089

E-mail ID: ijraset@gmail.com

Optimization of Mobile Edge Computing and Networking for Ultra-Low Latency and Green Internet of Things Applications

Prasidhee Koli, Darshan Bhangale, Dhrumi Karia, Dhruv Chawda, Mayur Hingu, Prithvi Bhadra, Neha Panadi, Dr. Asha Durafe

Electronics and Computer Science Shah & Anchor Kutchhi Engineering College Mumbai, India

Abstract: *IoT growth has quietly outpaced what cloud-centric network architectures were built to handle. When billions of sensors, actuators, and embedded controllers started producing continuous data streams, the assumption that round-trip latency to distant servers was an acceptable cost stopped holding. In industrial automation lines, hospital monitoring loops, and vehicle-to-vehicle coordination, tens of milliseconds separate a correctly functioning system from a failed one.*

This paper examines Mobile Edge Computing (MEC) as a structural fix: computation moves physically closer to where data originates, cutting propagation delay and relieving backhaul pressure at the same time. We look at how task offloading, energy budgeting, and shared resource allocation actually behave under field conditions — varying wireless channels, heterogeneous workloads, energy supplies that aren't always predictable. The optimization framework we propose uses a Deep Q-Network (DQN) that learns from what the system does rather than from a pre-built model of what it's supposed to do, which matters when operating conditions drift.

One finding we want to highlight upfront: minimizing latency and minimizing energy are not genuinely competing objectives. A well-trained edge policy can satisfy timing requirements while simultaneously cutting power draw across the device, network, and server layers.

Keywords: *Mobile Edge Computing (MEC), Internet of Things (IoT), Ultra-Low Latency, Green Computing, Resource Optimization, Task Offloading, Deep Q-Network, Edge Networking*

I. INTRODUCTION

Think about what a large hospital network actually looks like in practice. Infusion pumps, ventilators, wearable monitors, environmental sensors, and access control systems all share the same infrastructure, yet they have vastly different timing requirements. Routine telemetry can wait seconds or minutes. A closed-loop medication delivery system or an emergency alert cannot — the difference between a 15ms and a 150ms response time carries direct clinical consequences in those cases.

The mismatch between these requirements and conventional cloud architecture is fundamentally a physics problem, not a software one. Optical fiber carries signals at roughly 200km/ms. That places a hard floor on round-trip latency to a geographically distant data center. When platforms like AWS were originally designed, latency was an annoyance that affected user experience; it was not a safety-critical parameter. Sub-10ms application deadlines simply were not part of the design brief for centralized cloud infrastructure.

Multi-access Edge Computing (MEC) is the architectural response to that mismatch. Instead of sending computation to distant data centers, MEC co-locates processing with radio access infrastructure. Base stations already installed across urban areas become hosting sites for edge servers, cutting both propagation delay and the volume of data that needs to travel over backhaul links. Reducing that data movement also brings a secondary benefit: measurable reductions in network-level energy consumption.

The rest of this paper covers: MEC hardware and software architecture; the task offloading problem and its inherent complexity; energy efficiency across all system tiers; multi-tenant resource allocation; latency decomposition; emerging research directions; open challenges; and our DQN-based optimization approach.

II. MOBILE EDGE COMPUTING (MEC) ARCHITECTURE

A. Foundational Concepts

Every measurement an IoT device takes has to go somewhere: processed locally, offloaded to an edge server, or sent to the cloud. Local processing works when the device has enough compute capacity and battery headroom. Cloud transmission works when deadlines are soft enough to tolerate the round-trip. MEC sits between these two — a server physically near the data source, often sharing a cabinet with the serving base station, capable of handing back results before hard deadlines expire. The device effectively borrows server-grade compute without absorbing the latency penalty of a data center round-trip.

This idea has roots in content delivery networks, which have been moving computation geographically closer to users for decades. MEC's distinguishing feature is its coupling with mobile radio infrastructure and its explicit targeting of latency and energy constraints that CDNs were never designed around.

B. Hardware and Software Architecture

MEC systems are organized into three layers. The *edge server* is the physical foundation: compute hardware at a tower or small-cell site, equipped with CPUs, optional GPU accelerators for inference, local storage, and network interfaces. The *MEC host* refers to the complete physical node including radio access and backhaul connectivity. The *MEC platform* is the software layer that handles task reception, scheduling, resource allocation, and result delivery — essentially a miniaturized cloud controller operating at tower scale.

User equipment — phones, sensors, cameras, vehicle units — sits at the far edge generating the workloads that the MEC platform arbitrates.

Interoperability across vendor components remains a genuine problem in practice. A device certified for one MEC platform does not automatically work with another, and standards bodies are still working through this gap.

C. Task Execution Flow

When a task arrives, the MEC platform checks local resource availability. If capacity permits, the edge server processes it and returns a result — typically under 5ms. Heavier jobs that can't be handled locally, such as large-scale model inference, multi-site analytics queries, or requests that need archival data, get pushed up to cloud infrastructure.

There's an energy angle here that matters for battery-operated devices. Moving a computationally expensive task off the device lets its processor coast at a lower utilization level, which cuts power draw significantly. Across a deployment lifetime measured in months or years between battery swaps, these per-task savings accumulate into a real extension of maintenance intervals.

D. Comparison with Cloud Computing

Cloud infrastructure makes sense for a specific class of workloads: training large models, running batch analytics across sensor populations distributed across geography, archival processing where timing flexibility exists. The hard limit is propagation delay — a request traveling to a metropolitan-area data center and back cannot fit inside a 10ms window under normal network conditions.

Edge servers are smaller and more constrained, but locally handled tasks come back in roughly 1–5ms. That's the range hard-deadline IoT applications actually need. The engineering challenge is building a classifier that correctly routes each arriving task to the right tier, quickly, and with low overhead.

III. GREEN COMPUTING: ENERGY EFFICIENCY ACROSS SYSTEM TIERS

In much of the MEC literature, energy efficiency appears as a secondary constraint — something to optimize once latency targets have been met. At IoT scale, this framing gets the priority order wrong, and the numbers make clear why.

Take a network of one million sensors, each sending 1KB of raw data to cloud servers every 10 seconds. That generates 100MB/s of sustained backhaul traffic. Replace raw transmission with local preprocessing and 50-byte summaries, and that traffic drops by a factor of roughly 20. Every node along the path — routers, switches, optical amplifiers — draws power in proportion to the traffic it carries. Across the size of IoT infrastructure today, the gap between these two approaches represents a meaningful share of total data center energy budgets.

A. Device-Level Power Consumption

On most battery-powered IoT devices, the radio transceiver is the dominant power consumer. Modern low-power microcontrollers handle simple computation efficiently; transmission, especially over extended wireless paths, is expensive by comparison.

Adaptive offloading policy — choosing between raw transmission and local preprocessing based on current channel quality — can meaningfully extend battery life across deployment timescales when preprocessing costs less than what transmission would consume.

B. Network-Level Power Consumption

Traffic processed at the edge never touches backhaul infrastructure. For high-volume IoT applications this is a substantial saving. Core network power consumption scales with load, so reducing upstream traffic reduces energy expenditure all the way up the chain to cloud data centers. Operational cost and environmental cost arguments point in the same direction: process at the edge when doing so is feasible.

C. Server-Level Power Management

Edge servers need their own power management, particularly at sites running on renewable or battery backup power. Dynamic voltage and frequency scaling lets server trade clock rate for power draw during quiet periods. Workload consolidation deactivates cores or entire machines when aggregate demand is low. Where solar generation is available at tower sites, the scheduler has to incorporate energy forecasts into allocation decisions, because power availability is no longer guaranteed.

D. Energy Harvesting Devices

Some IoT deployments use devices that extract energy from the environment — photovoltaic cells, piezoelectric transducers, thermoelectric generators, or RF scavenging. These devices can run for long periods without direct battery replacement, but the energy they collect is not constant or predictable. An offloading policy that doesn't account for this uncertainty — one that behaves the same way on a sunny afternoon as during an overcast morning — will drain devices faster than necessary or defer too much work. Coupling energy forecasting with offloading policy decisions in a principled way is an open problem that has attracted attention but lacks a fully satisfying solution.

IV. LATENCY DECOMPOSITION

Reaching sub-10ms end-to-end requires knowing which components are actually consuming the latency budget, because the effective interventions differ by component. Applying the wrong fix to the wrong source wastes both engineering effort and compute resources.

A. Air Interface Delay

The first delay sits between the device and the base station. In 4G LTE networks, the 1ms subframe structure combined with the multi-subframe scheduling pipeline typically contributes 15–20ms of radio round-trip — a number that surprises many application developers who assumed it was negligible. 5G NR reduces this substantially; under clean channel conditions it can fall below 1ms. But cell-edge devices, physically obstructed links, and congested spectrum can push the effective air interface delay back up toward LTE levels regardless of the radio generation. This component is set by physics and antenna geometry, not by application software — there's no way to engineer around it in code.

B. Server Queue Delay

Once a task reaches the edge server, it joins a queue. Under light load this wait is near zero. Under burst arrivals — say, a factory floor anomaly triggers simultaneous alerts from a dozen sensors — tasks arrive faster than the server can process them and queue depth climbs. At that point, queuing delay dominates the latency budget. Importantly, allocating more CPU doesn't fix this: when arrival rate exceeds service rate, the queue grows regardless of per-task execution speed. The right intervention is admission control — shedding or deferring low-priority tasks — not raw clock speed.

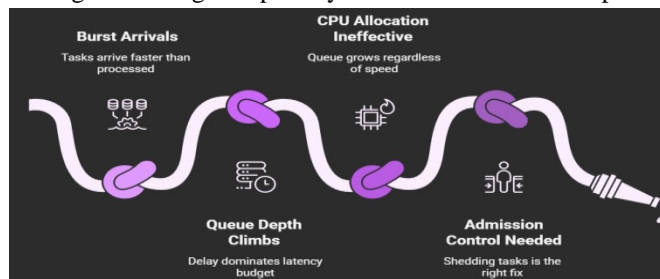


Fig. 1. Latency Bottleneck

The time to actually execute a task varies by orders of magnitude. A simple threshold comparison or running average takes microseconds. A convolutional network processing a high-resolution image might need 30–50 ms even on hardware with dedicated inference accelerators. Higher CPU frequency shortens execution time but raises power draw super-linearly, so maximizing frequency for every task isn't a viable strategy; the scheduler must weigh execution speed against the power budget on a per-task basis.

C. Downlink Transmission Delay

Returning the result to the requesting device is usually the smallest contributor. Computation tends to compress information: the output of an image classifier is a label, not a feature tensor; the output of a control loop is a setpoint, not a raw sensor frame. Downlink delay only becomes significant when outputs are large—reconstructed video streams, model weight updates for federated learning, high-fidelity AR rendering. For most IoT tasks it's negligible relative to the other components. One system-level failure mode deserves explicit mention: optimizing one component in isolation can worsen others. Maximizing CPU frequency across all tasks to shrink computation delay will exhaust the server's thermal envelope, triggering throttling that causes queuing delay to grow beyond what a more measured allocation would have produced. Effective optimization has to treat the system as an integrated whole, not a set of independent sub-problems.

V. EMERGING RESEARCH DIRECTIONS

A. Split Inference Architectures

Running a complete neural network on an IoT device is constrained by memory, compute, and the latency of local inference on battery-powered hardware. Sending raw sensor data to the cloud for inference incurs bandwidth costs and round-trip latency that hard-deadline applications cannot absorb. Split inference partitions the network across the device-edge boundary: initial layers run on the device, producing intermediate feature representations that are smaller than the raw input, which are then transmitted to the edge for the remainder of the computation. The result comes back as a compact output.

The optimal partition point depends on current channel conditions, server load, and the specific network architecture. Determining this in real time without itself adding significant latency remains an open problem.

B. Federated Learning at the Edge

Growing regulatory pressure around health records, location data, and behavioral profiles has made centralized data collection for model training harder to justify legally and operationally. Federated learning offers a practical alternative: devices keep their data local, run training locally, and contribute only parameter gradients—small numerical vectors to an edge aggregation server. The server merges these contributions and redistributes an updated shared model.

At edge scale, however, several difficulties emerge that don't appear in textbook federated learning setups. Devices dropout mid-round due to battery depletion or connectivity loss. Devices training at different speeds produce stale gradients that can destabilize aggregation. And devices observing fundamentally different data distributions—a scenario common when sensors are deployed across varied physical environments make it harder for a single shared model to generalize well across the population. Working through these practical complications, particularly in high-mobility 6G scenarios, is an active area.

C. UAV-Assisted Edge Coverage

Fixed base station infrastructure leaves gaps in disaster response, temporary high-density events, remote agriculture, and maritime scenarios. Mounting edge compute hardware on UAVs allows mobile edge capacity to be deployed where fixed infrastructure is absent or damaged.

This makes the optimization problem harder. UAV position affects channel quality to served devices, coupling trajectory planning with offloading and resource allocation decisions. Flight competes with compute provisioning for onboard energy. The joint three-dimensional optimization of trajectory, offloading, and resource allocation is drawing significant research attention.

D. Reconfigurable Intelligent Surfaces

Devices physically obstructed from their base stations suffer channel degradation that limits offloading throughput regardless of server availability. Reconfigurable Intelligent Surfaces (RIS) are arrays of passive reflective elements with software-controlled phase responses, positioned to redirect wireless signals around obstacles. No additional transmit power is required.

For MEC, improved channel quality between device and base station directly increases achievable offloading rates and reduces device transmission energy. Joint optimization of surface configuration, offloading decisions, and resource allocation is an active research area.

VI. OPEN RESEARCH CHALLENGES

A. Tractable Heterogeneity Modeling

A consistent simplification in published MEC literature is treating all devices as equivalent — same data rate classes, compute profiles, battery characteristics, deadline distributions. Real deployments are far more varied. A smart building network might simultaneously serve passive HVAC sensors generating bytes per minute and 4K video analytics nodes saturating gigabit links. Existing optimization frameworks handle either device class well in isolation; they tend to degrade when both are present in the same system. Building frameworks that accommodate genuine heterogeneity without collapsing tractability is an unresolved challenge.

B. Privacy in Multi-Tenant Deployments

Most MEC offloading models assume that the edge server is a trusted party for all tenants using it. In practice, commercial edge deployments are multi-tenant: a single server handles requests from device populations belonging to different organizations, operating under different data governance constraints. A wearable health device that offloads biometric processing to such a server has no hardware-enforced guarantee that its data won't be accessed by co-located workloads. Homomorphic encryption is the theoretically clean answer — it enables computation on encrypted data without decryption — but the overhead it imposes today is two to three orders of magnitude beyond what MEC latency budgets can absorb. Hardware-based trusted execution environments are substantially faster, but they come with their own vulnerability surface and depend on silicon features that aren't present in all deployed edge hardware. Neither option fully closes the gap between privacy requirements and practical constraints.

C. Long-Horizon Planning Under Non-Stationarity

Most deployed offloading algorithms are reactive: observe current battery level, channel quality, and server load, then pick an action. This works reasonably well when system statistics are stationary — when current observations are representative of near-future conditions. IoT environments often are not stationary. Energy-harvesting devices are the clearest example: a device that anticipates strong solar input in two hours should make different offloading decisions now compared to one facing an overcast afternoon. Lyapunov optimization and model-predictive control address temporal coupling under specific assumptions about future statistical behavior that field conditions routinely violate. Robust long-horizon planning under non-stationary uncertainty in resource-constrained systems is still an open problem.

D. Standardization Fragmentation

ETSI MEC, the OpenFog Consortium, and AKRAINO describe overlapping but mutually incompatible visions of edge infrastructure. Offloading frameworks built for one platform require substantial rework to deploy on another. For practitioners, this means research results — which are almost always platform-specific — cannot be straightforwardly compared or translated to production deployments. The widening gap between research capability and practical deployability is a real impediment to the field's progress.

VII. METHODOLOGY

A. System Model

We consider a network of *M* IoT devices and *M* edge servers communicating through shared base station infrastructure. Device *i* generates tasks continuously. Each task is characterized by a data volume d_i (bits), a computational requirement c_i (CPU cycles), and a completion deadline T_i . Tasks that do not complete within their deadlines are reclassified as failures. The optimization objective therefore addresses both average latency minimization and deadline miss rate reduction.

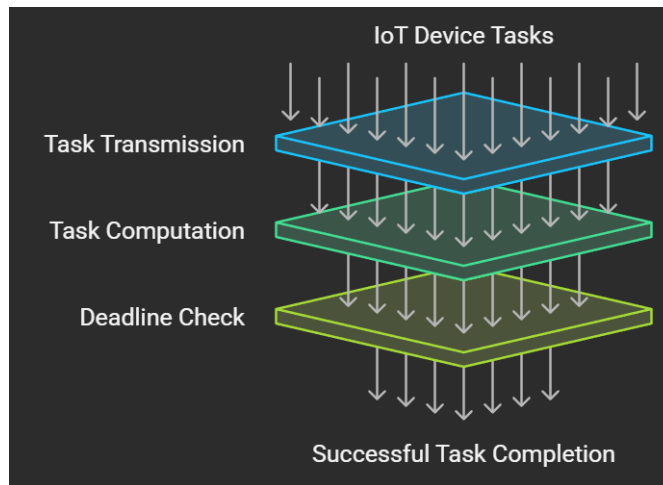


Fig.2. Task Processing Funnel

B. Problem Formulation

We seek a joint policy that reduces total latency and energy expenditure across all tasks in a given time window. Four constraint classes bound the feasible solution space:

- **Deadline constraints:** Task k must finish execution and return its result within time T_k from submission.
- **Server capacity constraints:** At no time step may the aggregate CPU demand assigned to any single edge server exceed that server's physical processing capacity.
- **Transmission power constraints:** Radio transmit power on each device must stay within the hardware-specified limit throughout the offloading period.
- **Assignment constraints:** Every task runs on exactly one resource—the local device or one designated edge server—with no mid-task migration permitted.

C. DQN-Based Offloading Framework

We use a Deep Q-Network to learn the offloading policy from experience rather than from a pre-specified system model. At each scheduling epoch, the agent observes three quantities from the current environment: the battery charge level of the requesting device, the estimated channel quality toward each available edge server, and the current queue depth at each of those servers. These form the state vectors s_t .

Given s_t , the agent computes a Q-value for each candidate action—run the task locally, or offload to server $k \in \{1, \dots, M\}$ —and takes the action with the highest value. After execution, the observed outcomes determine the reward:

$$r_t = -(\alpha \cdot L_t + \beta \cdot E_t) \quad (1)$$

where L_t is the measured task latency, E_t is the measured energy consumption, and α, β are weights that set the relative importance of the two objectives. The negative sign means the agent is rewarded for minimizing both simultaneously.

The case for using RL here rests on one practical observation: any analytically derived offloading policy encodes assumptions about channel statistics, arrival rates, and server behavior that hold at calibration time and erode afterward. A DQN policy doesn't freeze those assumptions—it keeps updating from what the deployed system actually does, retaining effectiveness as operating conditions evolve.

D. Algorithm

Algorithm 1 DQN-Based Task Offloading

- 1: Initialize DQN network parameters θ and edge server capacities
- 2: **repeat**
- 3: IoT device generates task with parameters (d_i, c_i, T_i)
- 4: Observe states s_t ; battery level, channel quality, server queue depths
- 5: Select action $a_t = \arg \max_a Q(s_t, a; \theta)$

```

6:   if  $a_i = \text{offload to server } k$  then
7:     Transmittask to server  $k$ 
8:     Executetask at edge server  $k$ 
9:     Transmitresult to device
10:  else
11:    Executetask locally on device
12:  endif
13:  Measure resulting latency  $L_i$  and energy  $E_i$ 
14:  Computereward  $r_i = -(\alpha L_i + \beta E_i)$ 
15:  Update DQN parameters using experience
( $s_t, a_t, r_t, s_{t+1}$ )
16: until convergence or deployment termination

```

VIII. RELATED WORK

Table I summarizes relevant prior work, the problems addressed, and identified limitations.

TABLE I
SUMMARY OF RELATED WORK: PROBLEMS ADDRESSED AND LIMITATIONS

Sr.	Author(s)	Problem Addressed	Limitations
1	Ke Zhang et al. (2018)	IoT requires low-latency distributed processing	Limited deployment scope; scalability under large device populations not validated
2	Zhihao Ren et al. (2025)	AR/VR applications demand microsecond-scale latency guarantees	High infrastructure cost; integration complexity with existing deployments
3	Seung-Que Lee et al. (2020)	Routing overhead causes unacceptable latency; URLLC requires sub-1ms response	Validation limited to 5G scenarios; scalability to real deployments unconfirmed
4	Raghu Dhumapati et al. (2025)	Static resource allocation produces latency spikes under load variation	ML inference overhead; privacy concerns in shared infrastructure
5	Mustafa Ergen et al. (2024)	5G edge capacity insufficient for anticipated future communication demands	Early-stage; standardization pending across vendor platforms
6	Tien V. Thai et al.	MIMO-MEC integration for 6G scenarios inadequately studied	Survey only; no experimental validation provided
7	Abdulmohsen Almalawiet al.	Inefficient edge caching causes unnecessary latency at access nodes	Algorithm validated in limited scenarios; real-world deployment not verified
8	Shamim Taimoor et al.	Fixed edge servers cannot cover remote or disaster-affected areas	High UAV energy consumption; trajectory planning is computationally intensive
9	Zouheir Trabelsiet al.	Latency-sensitive IoT applications require intelligent offloading	Fuzzy rule sets require manual tuning; generalization across IoV scenarios
10	Laszlo Toka	Deploying URLLC applications at edge lacks standard orchestration	Unverified Kubernetes overhead may conflict with strict latency targets
11	Luciano Baresiet al.	Traditional edge architectures lack flexibility for dynamic workloads	Cold-start latency in serverless functions; limited support for stateful task processing
12	Sreenu Banoth et al.	IoT data processing requires low-latency architectures proximate to data sources	Survey-based; lacks performance benchmarking across architectural variants
13	Amit Kumar Mishra	Applications simultaneously demand low latency and high throughput	Theoretical analysis only; no simulation or experimental validation provided

IX. CONCLUSION

MEC works. The latency and energy reductions it produces are not theoretical projections — they follow from moving computation physically close to where data is generated, and the mechanism is well understood. What's harder to close is the gap between what MEC can do in a well-configured deployment and what it consistently delivers under the messy conditions of real infrastructure: dynamic traffic, heterogeneous devices, shifting channel statistics, and energy supplies that don't hold to schedule.

Three problems sit at the center of that gap. First, offloading policies that perform well in controlled settings tend to degrade as operating conditions drift, because most of them embed assumptions about the environment that stop holding over time. Second, multi-tenant resource scheduling on shared edge servers requires handling genuine network load heterogeneity—not averaged abstractions — to avoid systematic unfairness or priority inversions at peak load. Third, energy management across device, network, and server tiers needs to be treated as a coupled problem, not a per-tier optimization.

The DQN-based framework described here targets the offloading piece. It learns from experience rather than from a pre-calibrated model, which gives it the ability to track shifting conditions without manual re-tuning. On its own it doesn't solve the multi-tenant scheduling or cross-tier energy coupling problems, but it provides a foundation that can be extended. Planned next steps include multi-agent coordination to handle competition among concurrent offloading decisions, integration of energy harvesting forecasts into the reward signal, and evaluation against mobility traces that include realistic handover sequences.

X. ACKNOWLEDGMENT

We are grateful to Shah & Anchor Kutchhi Engineering College for providing the research environment and facilities that supported this work. We also thank the anonymous reviewers whose comments strengthened the paper.

REFERENCES

- [1] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Zhang, X. Peng, L. Pan,
- [2] S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.
- [3] Z. Ren, Z. Liu, and Y. Chen, "Ultra-low latency edge computing for immersive AR/VR applications," *IEEE Trans. Mobile Comput.*, 2025.
- [4] S.-Q. Lee, D.-H. Han, and J.-H. Kim, "URLLC-aware routing for industrial IoT in 5G networks," in *Proc. IEEE VTC*, 2020.
- [5] R. Dhumpati, A. Sharma, and K. Rao, "Adaptive resource allocation for heterogeneous MEC environments," *IEEE Internet Things J.*, 2025.
- [6] M. Ergen, B. Aydin, and T. Yilmaz, "Beyond 5G edge computing: Architectural considerations for 6G," *IEEE Commun. Mag.*, 2024.
- [7] T. V. Thai, N. H. Tran, and C. S. Hong, "MIMO-assisted mobile edge computing: A survey for 6G systems," *IEEE Commun. Surv. Tut.*, 2023.
- [8] A. Almalawi, X. Yu, Z. Tari, A. Fahad, and I. Khalil, "Intelligent edge caching for latency-sensitive IoT services," *IEEE Trans. Parallel Distrib. Syst.*, 2023.
- [9] S. Taimoor, M. Ali, and A. Khan, "UAV-assisted mobile edge computing: Energy and coverage optimization," *IEEE Wireless Commun. Lett.*, 2023.
- [10] Z. Trabelsi, H. Afifi, and F. Zarai, "Fuzzy logic-based task offloading for Internet of Vehicles," in *Proc. IEEE ICC*, 2023.
- [11] L. Toka, "Kubernetes-based orchestration for URLLC edge applications," in *Proc. IEEE INFOCOM*, 2022.
- [12] L. Baresi, D. Filgueira, and G. Quattrocchi, "Serverless edge computing: Challenges and opportunities," *IEEE Pervasive Comput.*, vol. 21, no. 2, pp. 48–56, 2022.
- [13] S. Banoth and R. Thakur, "Survey of edge computing architectures for real-time IoT data processing," *IEEE Access*, 2023.
- [14] A. K. Mishra, "Joint latency and throughput optimization in multi-tier edge computing networks," *IEEE Trans. Netw. Sci. Eng.*, 2023.
- [15] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, 2017.
- [16] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, 2019.



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)