Machine Learning for Edge Computing and Task Placement in 5G/6G Architectures

Hosila Khushbokova[1], Shivansh Sahni[2]

[1]Presidential School, Termez, Uzbekistan
[2]Detroit Country Day School, Beverly Hills, Michigan, United States

Abstract. The emergence of 5G and the vision of 6G have shifted computation from centralized clouds to the network edge, enabling low-latency services for applications such as autonomous driving, augmented reality, and industrial IoT. However, the dynamic, heterogeneous, and resource-constrained nature of edge environments renders classical scheduling policies (e.g., Round Robin, Proportional Fair) and static optimization methods inadequate for joint optimization of latency, energy, and fairness. This paper presents a comprehensive review of machine learning approaches for task offloading and placement in multi-access edge computing (MEC) architectures for 5G/6G networks. The review synthesizes literature published between 2020 and 2025, complemented by foundational works on edge architectures and reinforcement learning. Key themes include: (i) a comparative analysis of optimization, queueing, heuristic, supervised learning, reinforcement learning (RL), and federated learning methods across assumptions, scalability, adaptability, and deployment practicality; (ii) a systematic decomposition of the latency-energy tradeoff, including Pareto front analysis, adaptive reward design, and scheduler overhead; (iii) an examination of fairness dimensions (starvation avoidance, throughput fairness, latency fairness, deadline fairness) that are rarely treated together; and (iv) an assessment of evaluation practices, highlighting the dominance of simulation-only, small-topology studies and the lack of reproducibility. The review concludes that while RL-based schedulers consistently outperform classical baselines under controlled simulation conditions, the field faces a critical generalization gap: few results have been validated on production MEC deployments with real user traffic, nonstationary workloads, or mobility. Closing this gap requires standardized benchmarks, shared workload traces, and norms for releasing code and hyperparameters. The paper concludes with open research directions, including multi-objective reward design, fairness-aware scheduling, and integration of non-terrestrial networks in 6G.

Keywords: Multi-access edge computing, task offloading, reinforcement learning, federated learning, 5G/6G networks, scheduling policies, latency-energy tradeoff, fairness, network orchestration

Introduction

The rapid expansion of Internet of Things (IoT) devices, autonomous systems, and immersive applications is not merely increasing network load, but fundamentally redefining the requirements of computing infrastructure itself. This shift demands low-latency, high-reliability processing at the network edge. While 5G and emerging 6G networks promise to deliver on this through enhanced mobile broadband (eMBB), ultra-reliable low latency communication (URLLC), and massive machine type communications (mMTC), the reality is that the underlying architecture often falls short. Traditional cloud-centric models, which centralize computation, incur delays of 100-500 milliseconds, violating the QoS requirements of latency-sensitive applications such as extended reality (XR) and industrial automation [1, 2].

This fundamental latency bottleneck has accelerated the paradigm shift toward edge computing, which processes data closer to its source to reduce delay, conserve bandwidth, and enhance data privacy [2]. However, a critical question emerges: how do we effectively orchestrate this new, decentralized infrastructure? Efficiently placing tasks across heterogeneous, dynamic edge environments remains a formidable challenge. Classical scheduling algorithms, like Round Robin (RR) and Proportional Fair (PF), which served well in less demanding contexts, are proving inadequate for the scale and complexity of 5G/6G networks. Their core weakness lies in an inability to adapt to real-time network fluctuations, diverse Quality of

Service (QoS) demands, and the sheer volume of connected devices, without the ability to jointly optimize latency, energy efficiency, and fairness across dynamically changing network conditions [1, 3].

Machine learning enables a different approach: instead of static rules, schedulers can learn from network conditions, user mobility, and application demands. The distinction between learning paradigms matters. RL-based schedulers, for instance, excel in stochastic environments by balancing throughput, latency, and energy efficiency through trial-and-error optimization, effectively learning what works where static formulas fail. Federated learning offers something arguably more valuable for distributed edge architectures: the ability to train collaborative models across nodes without centralizing sensitive data, preserving privacy while still benefiting from collective intelligence [1, 3, 4].

Yet for all the progress, the literature reveals a gap between promise and practice. Differences in device hardware capabilities continue to limit the ability of ML models to generalize across deployment environments. Real-time adaptability remains elusive when training overhead directly competes with latency budgets. And the rush to embrace emerging paradigms like digital twins, quantum-assisted scheduling, and joint sensing communication frameworks often outpaces clarity about how these pieces will actually integrate into working systems [1, 3, 4].

This paper contributes to the literature in four specific ways: (1) It organizes existing ML-based offloading methods into a unified taxonomy spanning RL, supervised learning, federated learning, and hybrid approaches, clarifying their assumptions and applicability. (2) It provides a comparative analysis of evaluation practices, identifying that the majority of studies use simulation-only, small-topology setups and do not report hyperparameters or code, severely limiting reproducibility. (3) It systematically decomposes the latency-energy tradeoff into Pareto front analysis, adaptive weighting, and scheduler overhead, showing where RL reward design consistently fails in practice. (4) It identifies the generalization gap between benchmark performance and production deployments as the field's most critical unsolved problem, and proposes specific infrastructure requirements (standardized traces, code release norms) to close it.

The discussion begins with the architectural foundations of edge networks and task offloading, then moves through the core tensions that define the field: latency versus energy consumption, fairness versus efficiency, and edge autonomy versus cloud coordination. The benchmarks and evaluation tools that separate genuine advances from academic exercises receive dedicated scrutiny. The final section maps where the field currently stands and where it must evolve to deliver on the promise of intelligent, scalable edge orchestration in 5G/6G systems.

This review primarily focuses on peer-reviewed papers published between 2020 and 2025 that address task offloading, scheduling, and resource orchestration in MEC-enabled 5G/6G networks using machine learning. We searched IEEE Xplore, ACM Digital Library, and arXiv using keywords: "edge computing task offloading", "reinforcement learning scheduling", "federated learning MEC", "latency-energy tradeoff edge", and "fairness scheduling edge". We excluded papers without quantitative evaluation, those focusing solely on cloud-only or device-only execution, and non-English publications. The final corpus includes the papers cited throughout this review, with emphasis on RL-based methods (Section 4), federated learning (Section 6), and latency-energy modeling (Section 5). Foundational works from earlier years (e.g., cloudlets, fog computing, early MEC standardization, and deep reinforcement learning foundations) are included for historical context but are not the primary focus of the review. Complementary taxonomies of task offloading organize strategies by system model (single MEC server, multiple MEC servers, distributed MEC), decision-making entities (cloud-assisted, UE-driven, SDN-based, collaborative), and algorithmic paradigms, such as greedy heuristics, integer programming, machine learning, branch and bound [5].

Background: Edge Computing in 5G/6G

Edge computing reduces cloud-to-device latency from hundreds of milliseconds to single digits by moving processing to servers co-located within the radio access network, but the scheduling problem this creates is

harder than it appears. Strict latency budgets, the competing energy costs of local versus remote execution, finite shared bandwidth, and task dependencies all interact under continuously shifting conditions, producing an optimization problem that is intractable at scale for exact methods, primarily due to the combinatorial explosion of task-to-resource assignments as the number of users and edge servers grows.

Historical Context

The evolution of edge computing and learning-based scheduling rests on two parallel lines of research: edge architectures and reinforcement learning algorithms.

Early edge computing concepts emerged to overcome the latency limitations of centralized cloud. Cloudlets proposed VM-based offloading over Wi-Fi, placing small cloud servers at community locations, but they lacked integration with cellular networks and did not support user mobility. Fog computing extended edge computing to IoT and industrial environments using any network device (routers, switches, base stations) organized hierarchically, but it still lacked standardized APIs and deep cellular RAN integration. Multi-access Edge Computing (MEC), standardized by ETSI starting in 2014, addressed these gaps by embedding cloud platforms directly into the RAN, introducing integration with 3GPP network functions, standardized APIs for third-party applications, virtualization infrastructure, and mobility support through service migration [6].

Reinforcement learning provides a framework for sequential decision-making under uncertainty. Core concepts include Markov decision processes, Bellman equations, value functions, and policy iteration. Q-learning offers a model-free algorithm for learning optimal action-value functions. The combination of RL with deep neural networks, such as deep Q-networks, demonstrated that agents can learn control policies directly from high-dimensional inputs. Subsequent advances extended RL to continuous action spaces (DDPG) and improved sample efficiency with actor-critic methods (A3C, PPO). For multi-agent settings, algorithms such as COMA, QMIX, and CommNet address challenges of nonstationarity, partial observability, and scalability [7].

Edge scheduling problems exhibit precisely the properties that RL algorithms are designed to handle such as nonstationary environments, delayed rewards, and structured action spaces. This review builds on these foundations while focusing on recent 5G/6G developments.

Architecture of 5G/6G Edge Networks

Multi-access Edge Computing moves cloud-side intelligence to the radio access network, dramatically reducing the physical distance data must travel before processing. Traditional cloud architectures impose end-to-end delays of 100-500 ms; edge deployments bring this down to 5-20 ms, a reduction that is less about convenience than about making certain application classes viable at all, such as autonomous driving, remote surgery, and industrial automation, all of which operate under latency budgets that centralized cloud cannot meet [8]. ETSI has standardized MEC reference architectures since 2014, producing a three-tier model in which user devices sit at the base, edge servers offering moderate compute with low latency occupy the middle tier, and the cloud provides abundant resources at higher delay above them. Tasks are triaged across this hierarchy based on deadline sensitivity and computational demand [9].

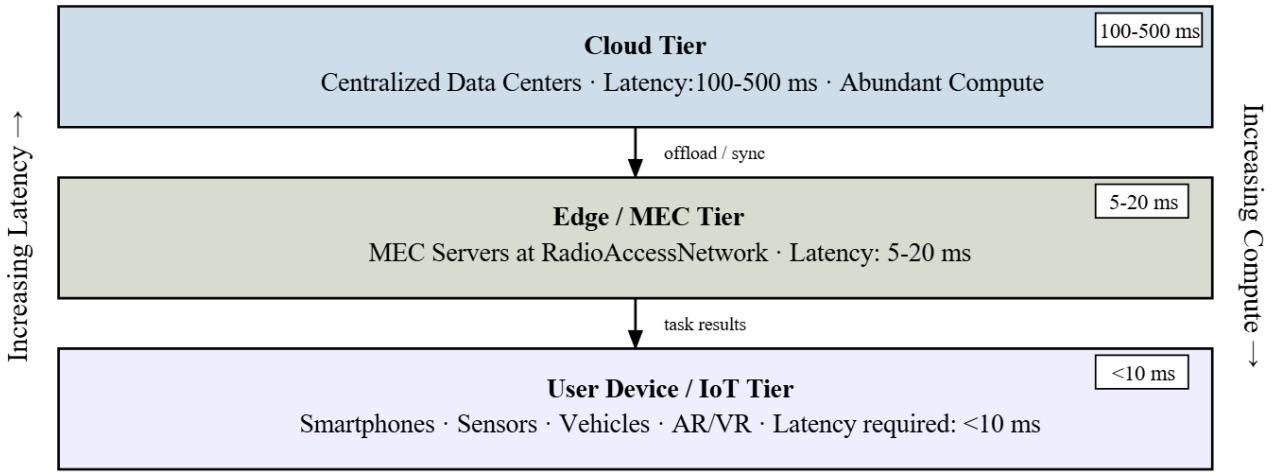

Figure 1. Three-tier architecture of Multi-access Edge Computing in 5G/6G networks.

For 6G, this model extends beyond terrestrial infrastructure. Satellite and non-terrestrial networks will integrate into the compute continuum, enabling offloading across aerial and ground domains as users move between coverage areas [10]. It is important to distinguish between 5G MEC, which is commercially deployed today (e.g., by Verizon, Ericsson, and other operators), and 6G, which remains a research vision.

Feature	5G	6G (projected / target)
Peak latency	1 ms	0.1 ms (target)
Peak data rate	20 Gbps	1 Tbps (projected)
Network architecture	Terrestrial MEC, base stations	Terrestrial + NTN: satellites, UAVs, air nodes
AI / ML role in scheduling	Supplementary; heuristics dominant	Core architectural requirement; AI-native design
MEC offloading complexity	Binary / partial; 2-tier decisions	Multi-domain DAG; cross-layer orchestration
Handover challenge	Terrestrial cell handoff	Cross-domain handoff across ground, aerial, satellite

Table 1. Comparison of architectural and performance parameters between 5G and emerging 6G networks.

This expansion increases architectural complexity considerably, since handoff across heterogeneous access technologies introduces new sources of latency and reliability uncertainty that current MEC orchestration frameworks were not designed to handle.

Multi-Access Edge Computing Fundamentals

MEC servers deliver three core capabilities at the network edge: computation, communication, and caching, commonly abbreviated as the 3Cs [8]. Through virtualization, they run containerized applications close to users, enabling rapid instantiation and teardown of services as demand shifts. This agility depends on underlying radio technologies including NOMA for efficient multi-user uplink and the flexible subcarrier spacing of 5G/6G waveforms, which together support dynamic adaptation to channel conditions [11, 12]. A central orchestrator manages server selection, application placement, and resource allocation across edge nodes, continuously balancing response time, throughput, and user fairness. MEC also enables local processing for privacy-sensitive applications, keeping data within administrative boundaries, a constraint that simplifies certain security concerns while complicating resource coordination across nodes [9].

Task Offloading Pipeline

The offloading pipeline involves four sequential decisions: whether to execute locally or remotely, which server to target, how to schedule transmission, and how to manage the return of results.

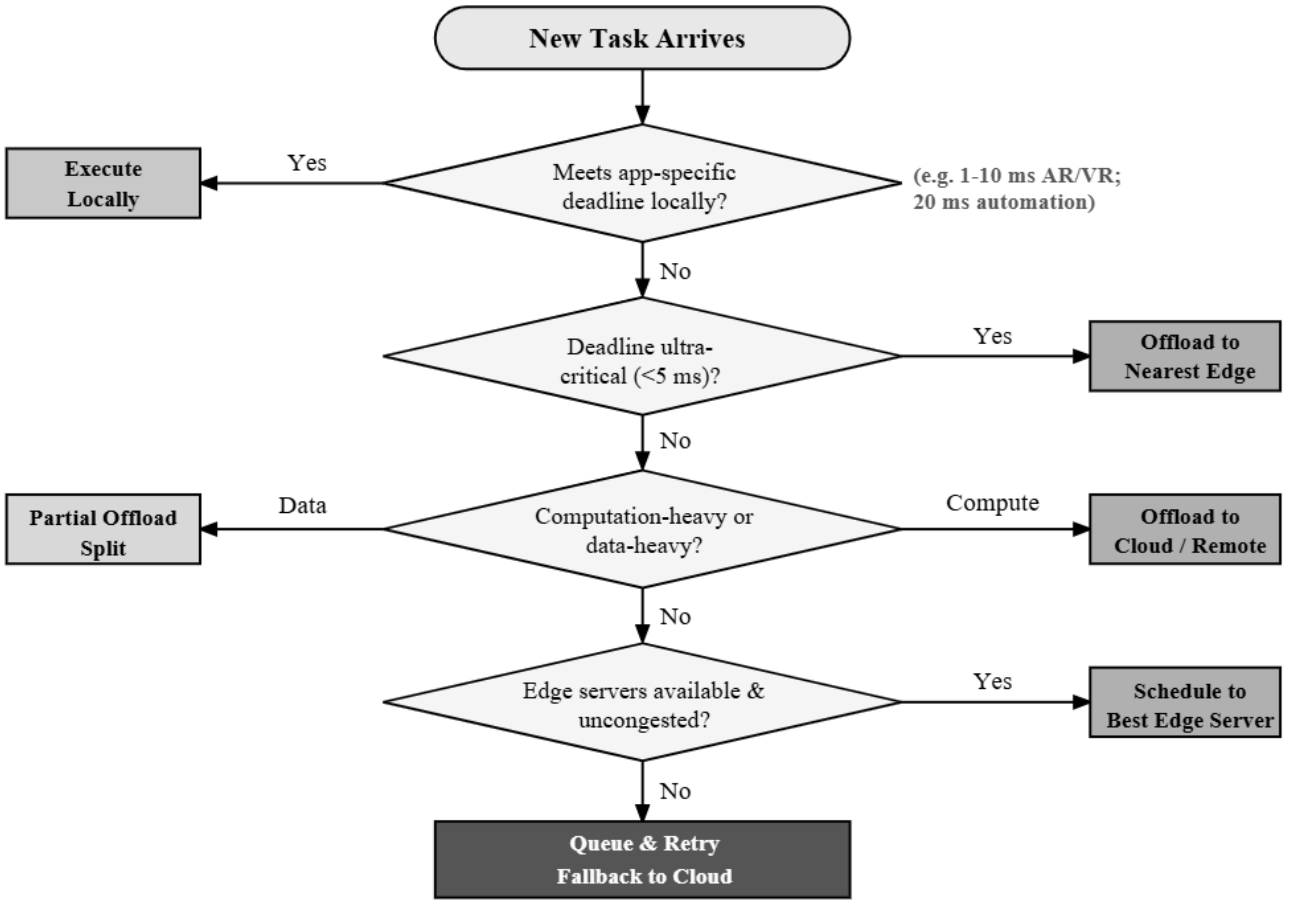

Figure 2. Decision flowchart for task offloading in MEC-based 5G/6G networks.

Each step consumes time and energy, and the interaction between them creates non-obvious trade-offs. Partial offloading, where a fraction of a task is sent to an edge server while the remainder executes locally, adds a continuous split variable to what would otherwise be a binary assignment problem, transforming the formulation from a mixed-integer linear program to a mixed-integer nonlinear optimization problem, substantially increasing its complexity [8].

Task dependencies introduce further structure. When tasks require each other's outputs, the scheduler must respect precedence constraints, typically modeled as directed acyclic graphs. Resource contention compounds this: multiple users sharing an edge server partition its capacity, so scheduling decisions for one user affect experienced latency for all others. The challenge is that congestion affects latency and energy simultaneously, making it difficult to optimize either in isolation [13].

Constraints and Limitations

Edge systems operate under four hard constraints that interact in ways traditional schedulers handle poorly. Latency requirements for applications such as autonomous driving and AR/VR fall in the 1-10 ms range, which the cloud cannot satisfy and the edge can only meet under favorable conditions, and queueing delays and handovers can still breach these budgets [9].

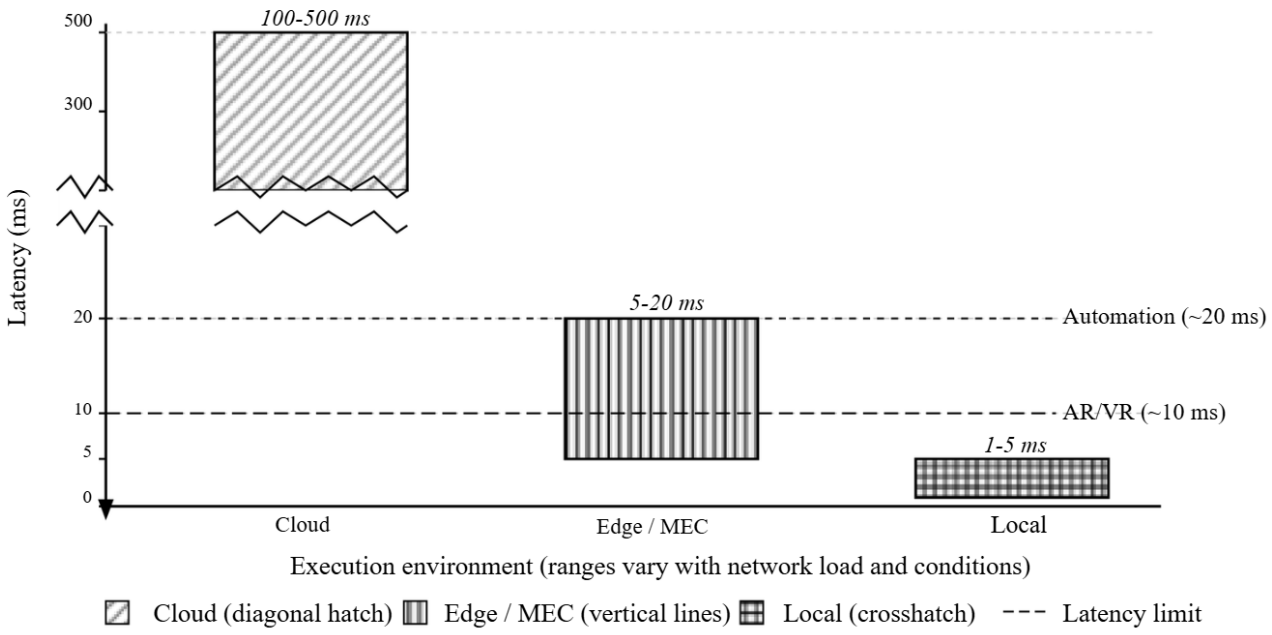

Figure 3. The reinforcement learning decision loop for edge task scheduling.

Energy presents a structural tension: radio transmission frequently costs more power than local computation, so offloading reduces compute energy while increasing communication energy, and the net balance depends on task size and channel conditions in ways that are hard to predict without real-time information [13]. Bandwidth is finite and shared, meaning that high offloading demand by some users degrades throughput for others as each local offloading decision increases delay for other users sharing the same edge server [11]. Finally, reliability requirements for mission-critical applications demand redundancy mechanisms that consume resources and introduce coordination overhead.

Resilience against failures can be improved by assigning primary and backup network slices to different MEC cloud facilities using multi-connectivity, which enables service continuity under VNF, server, link, or entire MEC facility failures while minimizing deployment cost via a genetic algorithm [14].

Classical schedulers such as Round Robin and Proportional Fair manage some of these constraints in isolation but cannot reason about their interactions under dynamic conditions. Faster transmission trades energy for latency; task consolidation improves throughput but increases queueing delay; handovers preserve signal quality at the cost of service interruption. In mobile environments with moving users and variable workloads, these trade-offs shift continuously in ways that fixed-rule policies cannot track [8].

Remaining Challenges

The offloading problem can be stated formally as assigning each task to either local execution or a remote edge or cloud server so as to minimize a weighted sum of delay and energy consumption, subject to deadline and capacity constraints [11]. This is a mixed-integer optimization problem, reducible to a generalization of the multidimensional assignment problem, and therefore NP-hard even for moderate numbers of users and servers. Binary assignment decisions create combinatorial explosion, and extending to continuous variables such as transmission power levels and bandwidth fractions produces a mixed-integer nonlinear program in the general case. Dynamic formulations model the problem as a Markov Decision Process in which the system state captures current channel quality, queue lengths, and user locations, and actions correspond to offloading destinations and power settings. While dynamic programming admits theoretically optimal solutions, the state space grows exponentially with the number of users and available server choices, rendering exact methods computationally infeasible at scale [13]. The NP-hardness of the offloading problem

has been demonstrated in various MEC formulations, including a mixed-integer nonlinear program for software-defined ultra-dense networks that decomposes into convex resource allocation and integer task placement subproblems [15]. This gap between theoretical optimality and practical tractability provides the primary motivation for machine learning approaches, which learn near-optimal policies through experience rather than exhaustive search.

Computation Offloading Models

Classical approaches to computation offloading, whether static policies, constrained optimization, or queueing models, share a common weakness: they require the problem to be fully specified in advance and have no mechanism for adapting when conditions change. Optimization methods assume a stationary environment, queueing models assume arrival distributions that real workloads rarely match, and heuristics have no feedback loop through which to improve. In simulation-based evaluations, Deep RL schedulers have demonstrated improved performance over these baselines under high load and dynamic conditions, though results vary across workload assumptions, network topologies, and scale.

Static vs. Dynamic Offloading

Offloading strategies divide along one fundamental axis: are placement decisions fixed at design time, or made continuously in response to live conditions? Static policies predefine tasks-to-resource mappings and work reasonably well in stable environments, but they have no mechanism for response when workloads spike or channel quality shifts. Dynamic offloading re-evaluates at runtime, which is essential in mobile edge environments where conditions change on millisecond timescales. The cost is instability risk: naive implementations can oscillate between targets under transient spikes, a failure mode that may be underrepresented in the literature, where evaluations often assume stationary arrival distributions rather than the bursty traffic patterns more commonly observed in real deployments [16]. Partial offloading adds a third mode, splitting tasks between local and remote execution via a continuous ratio variable rather than a binary decision, and becomes significantly more complex when tasks carry interdependencies modeled as directed acyclic graphs [17].

Optimization-Based Formulations

The classical approach frames offloading as a constrained optimization: assign tasks to local, edge, or cloud resources to minimize combined latency and energy, subject to server capacity and per-task deadlines. This is reducible to a generalization of the multidimensional assignment problem, and therefore NP-hard. The number of possible assignments grows exponentially with the number of tasks and servers, and incorporating continuous variables like transmission power makes it even harder to solve exactly. Decomposition methods such as Lyapunov optimization and ADMM break the problem into smaller per-timeslot subproblems that are individually tractable, but their optimality guarantees rest on the assumption that the environment is stationary, an assumption real networks routinely violate [18, 17]. In multi-operator settings where edge resources are owned by competing providers, game-theoretic approaches like Stackelberg pricing games and double auctions offer a different path, producing decentralized decisions without a central coordinator. A survey of blockchain-enabled MEC found these to be the dominant frameworks for cross-operator scenarios [18].

Queueing and Resource Allocation Models

Queueing models offer a simpler analytical lens: treat an edge server as a queue, characterize arrivals and service times statistically, and derive mean latency as a function of load. As the arrival rate approaches the server's capacity, waiting times grow without bound, a clean result that motivates keeping servers comfortably below saturation. The limitation is that these models assume Poisson arrivals and exponentially distributed task sizes, neither of which holds in practice. Real edge workloads show strong long-range dependence, with

measured Hurst exponents near 0.78, and task sizes that follow heavy-tailed rather than exponential distributions [16]. Queueing theory remains useful for rough capacity planning, but it tends to underestimate tail latencies and overestimate how much headroom a system actually has, primarily because real workloads exhibit heavy-tailed task sizes that violate the exponential assumptions underpinning standard queueing models.

Limitations of Analytical Approaches

The deeper problem shared by both optimization and queueing frameworks is structural: they require the problem to be fully specified before any solution can be computed. When the environment drifts, no self-correction is possible. Heuristics such as genetic algorithms and ant colony optimization trade provable optimality for adaptability and have shown competitive results on MEC benchmarks, but they remain offline methods [18]. When a scheduling decision proves wrong, a heuristic policy has no feedback mechanism through which to learn.

Transition from Heuristics to ML-Driven Policies

The shift to machine learning is a direct response to this limitation.

Attribute	**Optimization (MILP/MINLP)**	**Queueing-based**	**Heuristics (greedy, GA, ACO)**
Core assumption	Stationary environment; full system state known at solve time	Poisson arrivals; exponential service times	Problem structure fixed and fully known offline
Optimization objective	Minimize exact latency or energy under deadline and capacity constraints	Minimize mean latency as a function of server load	Approximate feasible objective; trades optimality for tractability
Adaptability	None; computed once for a fixed instance	None; no runtime adjustment mechanism	Offline only; no feedback loop to improve from past decisions
Data needs	Full state: channel quality, queue lengths, deadlines, server capacities	Aggregate arrival rates and service distributions	Fully specified problem instance at execution time
Scalability	Poor; NP-hard via multidimensional assignment; exponential growth with users and servers	Moderate; closed-form results degrade when distributional assumptions are violated	Moderate; solution quality degrades with scale and heterogeneity
Key limitation	Optimality guarantees fail when the environment drifts; Lyapunov and ADMM decompositions assume stationarity	Underestimates tail latency; real workloads exhibit heavy-tailed sizes and long-range dependence	No mechanism to correct wrong decisions; cannot improve without rerunning offline
Deployment practicality	Low; requires centralized solver with full visibility; infeasible at production scale	Medium; suitable for capacity planning, not real-time scheduling	Medium; tuned per scenario; does not generalize across topologies

Table 1a. Comparison of classical offloading method families. All three require the problem to be fully specified in advance and cannot adapt when conditions change (Rashid et al., 2025; Tran-Dang & Kim, 2025; Wang & Yang, 2025).

Table 2A. Comparison of classical offloading method families.

Attribute	**Supervised Learning**	**Reinforcement Learning**	**Federated Learning**
Core assumption	Labeled optimal decisions available for training prior to deployment	Markov property holds; reward encodes delay and energy at each step	Data distributed across nodes under privacy constraints; only model updates shared
Optimization objective	Imitate a pre-computed optimal policy by minimizing prediction error	Learn state-to-action policy via trial-and-error environment interaction	Train a global model collaboratively without centralizing raw data
Adaptability	None after training; cannot update without full retraining	Continuous online adaptation under changing channel, queue, and mobility conditions	Adaptive across nodes; local updates aggregated globally without a central data store
Data needs	Large labeled dataset mapping system states to optimal decisions	Reward signal and environment interaction; no labeled data required	Local data per node; periodic model update exchanges only
Scalability	Good; fast inference once trained	Good with function approximation; deep networks handle large state spaces	Good; decentralized by design; avoids centralized training bottleneck
Key limitation	Labeling optimal decisions in dynamic edge environments is rarely feasible in practice	Weighted-sum rewards can collapse to a single Pareto point; sensitive to hyperparameters; nearly all results are simulation-only	Convergence sensitive to scheduling policy and channel conditions; communication overhead grows with node count
Deployment practicality	Low; sufficient labeled data under real conditions is rarely obtainable	Low to medium; training cost and instability limit live deployment	Medium; communication overhead must be managed; preferable under privacy constraints

Table 2B. Comparison of learning-based offloading method families.

Framing offloading as a sequential decision problem allows a scheduler to improve through experience rather than requiring a pre-specified environment model. The empirical evidence is consistent: under high load, a Double Deep Q-Network scheduler reduced total processing time by 18-21% and improved resource utilization by 7-12% against priority-based and load-balancing baselines, with the gap widening precisely where static approaches degrade most [16]. This is not an isolated result. Across blockchain-integrated MEC frameworks, over 60% of recent studies now incorporate AI-based scheduling, and federated meta-learning approaches that combine MAML with federated averaging cut convergence time by up to 48% while keeping data local [19, 18]. Taken together, these results suggest that the field has largely moved past debating whether ML belongs in edge scheduling, and toward the harder question of which approaches generalise reliably across deployment conditions.

Reinforcement Learning for Task Placement

Framing offloading as a Markov Decision Process unlocks a different class of solutions. Rather than solving a fixed optimization problem, a reinforcement learning agent learns a policy through interaction, improving over time without requiring a pre-specified model of the environment. Three properties of edge networks make this framing particularly apt: conditions are nonstationary, rewards are delayed, and the action space involves interdependent decisions that cannot be optimized in isolation.

From Optimization to Sequential Decision-Making

The previous section established why exact optimization methods fail at scale: the offloading problem is NP-hard, and real networks change faster than any solver can respond. Reinforcement learning addresses this not by finding a better solver but by changing what is being solved. Rather than computing an optimal assignment for a fixed problem instance, an RL agent learns a policy, a mapping from observed system state to placement decisions, through repeated interaction with the environment. The policy improves over time as the agent observes which decisions led to good outcomes and adjusts accordingly, without requiring an explicit model of the system dynamics it is navigating [20]).

The natural mathematical framework here is the Markov Decision Process. At each decision point, the agent observes a state encoding channel conditions, queue backlogs, user locations, and remaining task sizes; selects an action determining offload destination and, where applicable, transmission power; receives a cost signal reflecting the delay and energy incurred; and transitions to a new state.

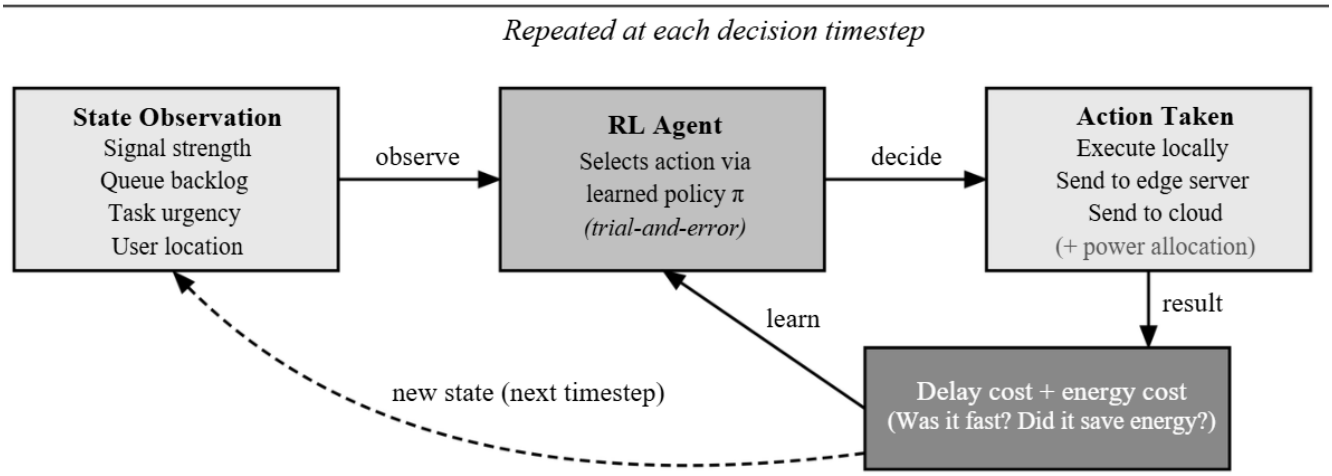

Figure 4. The reinforcement learning decision loop for edge task scheduling.

Dynamic programming guarantees optimal solutions to MDPs in principle, but only when the state space is small enough to enumerate, a condition that fails almost immediately as user count and time horizon grow. RL-based approaches sidestep this by learning compressed representations of the state-action mapping through experience, trading the guarantee of optimality for policies that are tractable and adaptive [21].

Why Reinforcement Learning Fits This Problem

Several properties of the edge offloading problem make it particularly well-suited to reinforcement learning. The environment is nonstationary: channel quality, user mobility, and workload intensity shift continuously, so a policy trained on yesterday's conditions must generalize to today's. Rewards are delayed: the cost of a placement decision may not materialize until downstream tasks execute or a deadline expires. And the action space is structured: offload destination, resource allocation, and scheduling priority interact in ways that make independent optimization of each dimension suboptimal. Standard supervised learning cannot directly optimize sequential, long-term control objectives without a generative model of the environment to simulate future outcomes, and classical optimization cannot handle the interdependent action structure without solving a problem that is computationally intractable. RL, by contrast, is designed precisely for settings where an agent must make sequential decisions under uncertainty and learn from the consequences [22, 21]. When multiple edge nodes or devices must coordinate, multi-agent reinforcement learning enables the learning of communication protocols; differentiable inter-agent learning passes gradients between agents during centralized training, allowing end-to-end optimization of discrete messages for cooperative tasks [23].

Limitations of RL in Edge Scheduling

Despite the advantages outlined above, reinforcement learning faces several well-documented limitations that are often understated in the edge scheduling literature.

Sample inefficiency: Deep RL typically requires hundreds of thousands of interaction steps to learn a stable policy, which is impractical in real edge environments where each decision carries energy and latency costs. Most studies rely on simulators that do not reflect real-world variability; the number of simulation steps is rarely reported, making it impossible to assess practical training costs.

Training instability: Q-learning variants, including DQN and Double DQN, are sensitive to hyperparameters (learning rate, replay buffer size, target network update frequency, exploration schedule). Minor changes can produce widely divergent outcomes, and the literature rarely reports sensitivity analyses. This instability is exacerbated in multi-agent settings due to nonstationarity [7].

Reward design is an open problem: As noted in Section 5.3, simple weighted-sum rewards often collapse to a single Pareto point. Poorly specified rewards can lead to unintended behaviors, such as starving low-priority tasks to minimize average latency. The problem is compounded by delayed rewards and partial observability.

Sim-to-real transfer: Policies trained on synthetic traffic models (e.g., Poisson arrivals, stationary channel conditions) fail when deployed on real networks with bursty, nonstationary, and correlated dynamics. The gap between simulation assumptions and real-world variability is rarely quantified.

Weak real-world validation. To date, very few peer-reviewed studies have demonstrated a live RL-based edge scheduler on a production 5G MEC testbed with real user traffic. Nearly all results are simulation-only, often on small topologies with few edge nodes.

These limitations do not invalidate RL as a direction, but they place the current state of the art firmly in the research prototype category.

Latency-Energy Tradeoff Modeling

Even with a well-designed RL policy, the question of what to optimize remains open. Latency and energy pull in opposite directions, and the weights assigned to each are a design choice with real consequences. Fixed tradeoffs fail in practice because the right balance shifts with device state, application type, and network conditions, and no static formulation accounts for all three simultaneously.

Joint Optimization Objectives

Latency and energy do not trade off cleanly. Offloading a task to a faster server cuts computation time but adds transmission delay and radio energy; keeping it local avoids the network cost but risks missing the deadline if the device is underpowered. The standard response is a weighted cost function that combines both objectives, with weights set to reflect application priorities. A video analytics pipeline tolerates almost no latency; a background sensor reading can wait. What the formulation tends to obscure is that the weights are a judgment call, and fixing them a priori locks the system into a solution space that reward design could otherwise navigate dynamically [24, 25]. A cloud-MEC collaborative offloading scheme with service orchestration processes metadata at the edge into high-quality services, reducing transmission load by an order of magnitude and achieving approximately 74% latency reduction compared to non-caching baselines [26].

Energy-Aware Scheduling

Scheduling heuristics like Earliest Idle Time First and Best-Fit were designed for throughput, and it shows. In IoT deployments with battery-powered edge nodes, optimizing for throughput drains nodes unevenly,

triggering thermal throttling on overloaded devices while underutilized ones sit idle. The result is availability loss that has nothing to do with network conditions and everything to do with the scheduler ignoring energy state. Switching the scheduling policy based on remaining energy, running an aggressive policy when reserves are high, offloading to a backup node when they fall critically low, reduces energy waste by around 1.5% and improves network availability by 6.4% over fixed-policy baselines [27]. The numbers are not dramatic, but the point is not the numbers: it is that a scheduler unaware of energy state will keep making the same mistake regardless of how much the situation has changed.

Pareto Front Analysis

When two objectives genuinely conflict, the honest answer is that there is no best solution, only tradeoffs. The Pareto front captures this: the set of operating points where improving latency necessarily worsens energy efficiency, and vice versa. In Massive MIMO systems, this front is non-convex, which creates a specific problem for standard multi-criteria methods: they exhibit a geometric bias toward the extremes of the front, selecting solutions that are optimal by one measure and poor by the other, unless the front's shape is corrected beforehand [24]. The same bias can emerge in RL training. An agent rewarded with a simple weighted sum will learn to maximize that sum, which may correspond to a single point on the Pareto front rather than a policy that navigates it intelligently. This is not a failure of reinforcement learning as a method, but it is a failure of reward design.

ML for Adaptive Tradeoff Control

The deeper issue with fixed weights is situational: the right balance between latency and energy depends on conditions that change continuously. A device with a full battery handling a real-time request should behave differently from the same device at 10% charge with a background task, and any policy that treats both situations identically is leaving performance on the table. An adaptive deep RL framework with confidence-aware reward adjustment, which scales the reward signal based on how certain the agent is about its current estimate, outperforms both scheduling heuristics and standard DRL baselines including Double DQN and DDPG across response time, task success rate, and load variance [28]. The same logic applies to inference workloads: in edge-based object detection, dynamically switching between a lightweight and a heavy model depending on context consistently outperforms committing to either model alone, because no single model is the right answer across all conditions [25].

Real-Time Decision Constraints

There is an irony in deploying a sophisticated scheduler at the edge: the scheduler itself consumes the resources it is trying to allocate, adding inference latency and energy overhead that must be accounted for in the system budget. If a placement decision takes longer to compute than the task's deadline allows, the decision is useless regardless of its quality. This is not a hypothetical concern. As model complexity grows, inference latency grows with it, and in millisecond-deadline environments the margin is thin. Hybrid CNN-LSTM and Transformer architectures with adaptive pruning have been shown to cut energy consumption by 27% while maintaining predictive accuracy, which suggests that model compression is more than a deployment convenience, a functional requirement for schedulers operating under tight time budgets to be accurate [29]. The efficiency of the decision-making process is as much a design variable as the decisions it produces.

Fairness, Coordination, and Evaluation

Performance is only part of the picture. Edge networks serve many users simultaneously, and a scheduler that is efficient in aggregate can still be deeply unfair to individual users. Coordinating decisions across edge and cloud tiers adds further complexity, and evaluating whether any of this actually works requires simulation

tools and metrics that the field has not yet standardized.

Fairness in Multi-User Scheduling

In edge networks serving many simultaneous users, fairness splits into at least four distinct concerns that are rarely treated together in current schedulers: (1) Starvation avoidance ensures that no task is indefinitely postponed. This is typically addressed via aging mechanisms that gradually raise the effective priority of waiting tasks. (2) Throughput fairness equalizes the average data rate across users, often measured by Jain's fairness index. (3) Latency fairness ensures that task completion times do not systematically disadvantage certain users or traffic classes. (4) Deadline fairness meets per-task deadlines proportionally across users, rather than only optimizing the average.

Periodic scheduling policies that guarantee stability for multiple competing plants sharing a communication channel can be constructed using T-contractive cycles on a weighted directed graph, where each vertex represents a set of active plants and multiple Lyapunov functions ensure global asymptotic stability [30].

Current ML-based schedulers almost exclusively target starvation avoidance or Jain's index. Latency fairness and deadline fairness are rarely explicitly optimized, and no study to our knowledge has compared tradeoffs among all four dimensions. This gap is significant because a scheduler that achieves high Jain's index can still exhibit severe deadline misses for a subset of users. Future work should treat fairness as a multi-objective problem with explicit metrics and reward shaping for each dimension [31].

The more immediate problem is starvation. Under heavy load, static allocation policies tend to serve whoever is highest priority and leave low-priority tasks waiting indefinitely. Aging mechanisms that gradually raise the effective priority of waiting tasks address this without abandoning responsiveness to genuinely urgent requests [32]. Fairness and efficiency conflict in the same way latency and energy do: serving all users equitably costs something in aggregate throughput, and the right balance depends on the application context. ML-based schedulers that treat fairness as an explicit optimization objective alongside performance metrics handle this more gracefully than heuristics designed for a single objective, but they require careful reward design to avoid trading one form of fairness, such as equity in resource distribution, for another [31].

Edge-Cloud Coordination

Coordinating computation across edge and cloud tiers requires more than routing decisions. Each tier has different latency, capacity, and privacy properties, and the right split depends on the task. Time-sensitive inference belongs at the edge; long-term model training and historical aggregation belong in the cloud. Federated learning formalizes this division by keeping raw data local while aggregating model updates centrally, preserving privacy without sacrificing the benefits of learning from distributed experience [31]. The convergence of federated learning under wireless scheduling is sensitive to both the scheduling policy and the signal-to-interference-plus-noise ratio (SINR); proportional fair scheduling outperforms random and round robin under high SINR, while round robin is preferable under low SINR, and a fundamental trade-off exists between the number of scheduled users and subchannel bandwidth [33]. The coordination challenge that the literature underemphasizes is scale. Algorithms that perform well with a handful of edge nodes tend to degrade as node count grows, because communication overhead and synchronization costs rise faster than the compute gains. Many published results evaluate on small topologies, often fewer than ten edge nodes, and report numbers that do not generalize to production deployments [34]. Cross-layer optimization compounds this: user experience depends on interactions between application behavior, network conditions, and compute availability simultaneously, and frameworks that optimize each layer independently tend to miss the interactions that matter most in practice [35].

Evaluation Frameworks and Benchmarks

Choosing the right simulation tool matters more than the literature sometimes acknowledges.

Tool	Layer	Models well	Does not model	Safe / unsafe conclusions
NS-3	Packet / PHY– MAC	Packet queuing; channel fading; PHY & MAC protocols; handover latency; NOMA/OFDM	Task scheduling logic; VM/container lifecycle; application-layer QoS	*Safe:* link-layer delay, protocol overhead, radio throughput. *Unsafe:* end-to-end task latency, offloading policy gains
OMNeT++	Packet / PHY– MAC	Modular protocol stacks; large topologies; custom channel models; event-driven simulation	Resource orchestration; energy- aware scheduling; ML policy integration	*Safe:* network-layer delay distributions, protocol overhead at scale. *Unsafe:* scheduler fairness, energy per task
CloudSim	Application / Cloud	VM provisioning; task queuing; datacenter energy; broker scheduling policies	Radio access dynamics; user mobility; wireless channel variation	*Safe:* compute utilization, provisioning delay, datacenter-level energy. *Unsafe:* wireless latency, MEC-specific gains
iFogSim	Application / Edge	Edge-cloud task placement; hop-level latency; network cost; IoT workloads	PHY/MAC interference; real radio channel; dynamic channel variation; bursty traffic	*Safe:* placement policy comparisons, edge vs. cloud cost. *Unsafe:* wireless latency variance, results under fading

Table 3A. Comparison of simulation tools for edge computing.

Commonly reported in the literature	Should also be reported for deployment validity
Average throughput: optimizable without improving user QoS	Task completion rate under deadline: directly measures QoS compliance
Mean task latency: hides tail; deadline violations undetected	95th / 99th percentile latency: exposes tail behavior critical for real-time systems
Average energy consumption: conflates efficiency with underloading	Energy per successful task: normalizes cost by outcome quality
RL convergence speed (training): training metric; reveals nothing about generalization	Policy performance under distribution shift: tests robustness to deployment conditions
Resource utilization (%): can be high while fairness and deadlines are violated	Jain's fairness index: quantifies distributional equity across concurrent users
Topology size rarely disclosed: most studies use 3-10 nodes	Scheduler inference overhead: model latency and memory footprint at decision frequency

Table 3B. Evaluation metrics: commonly reported versus deployment-relevant.

Packet-level simulators such as NS-3 and OMNeT++ model network behavior in fine detail but abstract away application-level resource management; application-level simulators such as CloudSim and iFogSim do the opposite. No single tool covers both, and results from one class do not straightforwardly transfer to the other [34]. Metrics present a similar problem. Throughput is easy to measure and easy to game; user-perceived quality, task completion rate under deadline, and energy efficiency per successful task are harder to measure but closer to what actually matters. Fairness requires its own metrics, with Jain's fairness index being the most widely used quantitative measure of distributional equity across users [36]. The reproducibility situation in this field is poor: hyperparameters are frequently undisclosed and code is rarely published, leaving reported improvements difficult to verify or build upon. Without standardized benchmarks, comparing results across papers involves more assumptions than the field tends to admit. This means a substantial fraction of reported improvements cannot be verified or built upon, which slows genuine progress considerably.

Conclusion and Future Directions

The environments that 5G and 6G create are too dynamic, too heterogeneous, and too constrained for the scheduling methods that preceded them. Classical optimization offers theoretical guarantees that evaporate under real network conditions. Heuristics are adaptable in structure, for instance, by tuning to queue depth or load thresholds, but they are not updated based on observed outcomes over time. Reinforcement learning and its variants address this at a level that incremental improvements to existing methods cannot, by treating the scheduling problem as something to be discovered through experience rather than solved in advance.

The empirical record supports this shift. Deep RL schedulers reduce processing time and improve resource utilization consistently across workload scales, with the advantage growing under exactly the conditions where static methods are weakest. Federated learning enables collaborative adaptation across nodes without centralizing data, which matters increasingly as privacy constraints tighten and edge deployments grow. Adaptive tradeoff control, adjusting the balance between latency and energy in real time based on device and network state, outperforms any fixed-weight policy across the range of conditions deployments actually encounter.

What the literature has not resolved is the generalization problem. Most results are produced on small, controlled topologies under workload distributions that are stationary enough to train on. Production edge deployments are neither small nor stationary, and the gap between benchmark performance and deployment performance is rarely quantified rigorously, particularly under the user mobility and bursty task arrivals that define real environments. This remains the field's most significant open problem, and it does not reduce to a question of algorithm design. It requires better evaluation infrastructure: standardized benchmarks, shared workload traces, and a norm of releasing code and hyperparameters that currently does not exist.

Several technical directions remain open. Reward design for multi-objective RL, particularly in settings where the Pareto front is non-convex, is not a solved problem. Fairness under resource contention requires schedulers that treat equity as a first-class constraint rather than a secondary correction. The compute cost of the scheduler itself becomes a real concern at millisecond-deadline scales, and lightweight architectures that preserve decision quality under tight inference budgets are an active area with practical consequences. Looking ahead to 6G, integrating non-terrestrial networks into the offloading hierarchy introduces handoff dynamics and coverage discontinuities that current orchestration frameworks were not built to handle.

The broader picture is that edge intelligence is no longer a research prototype. It is a deployment requirement, and the distance between what the literature demonstrates and what production systems can rely on is where the field should focus. Closing that gap requires not just better algorithms but clearer reporting of the conditions under which existing methods work.

References

[1] Jaradat, A. M., Alayedi, M., & Arslan, H. (2025). A Survey of Radio Resource Scheduling for 6G and Future Wireless Networks. IEEE Open Journal of the Communications Society, 6, 10191-10218. https://doi.org/10.1109/OJCOMS.2025.3640408

[2] Konakanchi, S. (2024). Edge Computing for Latency-Sensitive AI Applications. International Journal of Computer Engineering and Technology, 15, 2036-2045. https://doi.org/10.34218/IJCET_15_06_174

[3] Kumari, M., Singh, M. P., & Singh, A. K. (2025). A latency sensitive and agile IIoT architecture with optimized edge node selection and task scheduling. Digital Communications and Networks, S2352864825000483. https://doi.org/10.1016/j.dcan.2025.04.012

[4] Zheng, Y., Chen, Y., Qian, B., Shi, X., Shu, Y., & Chen, J. (2025). A Review on Edge Large Language Models: Design, Execution, and Applications. ACM Computing Surveys, 57(8), 209:1-209:35. https://doi.org/10.1145/3719664

[5] Islam, A., Debnath, A., Ghose, M., & Chakraborty, S. (2021). A Survey on Task Offloading in Multi-access Edge Computing. Journal of Systems Architecture, 118, 102225. https://doi.org/10.1016/j.sysarc.2021.102225

[6] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., & Sabella, D. (2017). On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. IEEE Communications Surveys & Tutorials, 19(3), 1657-1681. https://doi.org/10.1109/COMST.2017.2705720

[7] Canese, L., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., & Spanò, S. (2021). Multi-Agent Reinforcement Learning: A Review of Challenges and Applications. Applied Sciences, 11(11), 4948. https://doi.org/10.3390/app11114948

[8] Ismail, A. A., Khalifa, N. E., & El-Khoribi, R. A. (2025). A survey on resource scheduling approaches in multi-access edge computing environment: A deep reinforcement learning study. Cluster Computing, 28(3), 184. https://doi.org/10.1007/s10586-024-04893-7

[9] Jahandar, S., Shayea, I., Gures, E., El-Saleh, A. A., Ergen, M., & Alnakhli, M. (2025). Handover decision with multi-access edge computing in 6G networks: A survey. Results in Engineering, 25, 103934. https://doi.org/10.1016/j.rineng.2025.103934

[10] Liebhart, R., Shafi, M., Tataria, H., Shivanandan, G., & Chandramouli, D. (2025). Perspectives on 6G Architectures. IEEE Wireless Communications, 32(1), 108-114. https://doi.org/10.1109/MWC.010.2400065

[11] Saeed, M., Saeed, R., & Ahmed, Z. (2025). Optimizing Computation Offloading in 6G Multi-Access Edge Computing Using Deep Reinforcement Learning. International Journal of Electrical and Computer Engineering Systems, 16. https://doi.org/10.32985/ijeces.16.8.1

[12] Salih, S., Abdulameer, S., Abbas, H., Fadhil, J., Sekhar, R., Shah, P., Kingsly, S., Dhasagounder, M., & Radhi, A. (2024). An Investigating of 5G Multi-Access Edge Computing (MEC) and Subcarrier Spacing to Improve Wireless Communications. International Journal of Intelligent Engineering and Systems, 18, 2025. https://doi.org/10.22266/ijies2025.0229.33

[13] Ray, K., & Banerjee, A. (2025). Computation Offloading and Band Selection for IoT Devices in Multi-Access Edge Computing. ACM Transactions on Modeling and Computer Simulation, 35(2), 11:1-11:28. https://doi.org/10.1145/3670400

[14] Thiruvasagam, P. K., Chakraborty, A., & Murthy, C. S. R. (2021). Resilient and Latency-aware Orchestration of Network Slices Using Multi-connectivity in MEC-enabled 5G Networks. IEEE Transactions on Network and Service Management, 18(3), 2502-2514. https://doi.org/10.1109/TNSM.2021.3091053

[15] Chen, M., & Hao, Y. (2018). Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network. IEEE Journal on Selected Areas in Communications, 36(3), 587-597. https://doi.org/10.1109/JSAC.2018.2815360

[16] Wang, Y., & Yang, X. (2025, February 26). Research on Edge Computing and Cloud Collaborative Resource Scheduling Optimization Based on Deep Reinforcement Learning. arXiv.org. https://arxiv.org/abs/2502.18773v2

[17] Tran-Dang, H., & Kim, D.-S. (2025). Digital Twin-empowered intelligent computation offloading for edge computing in the era of 5G and beyond: A state-of-the-art survey. ICT Express, 11(1), 167-180. https://doi.org/10.1016/j.icte.2025.01.002

[18] Rashid, S. M., Aliyu, I., Isah, A., Hahn, M., & Kim, J. (2025). Blockchain-Based Task Placement and Resource Management in Edge Computing: A Survey. Electronics, 14(17). https://doi.org/10.3390/electronics14173398

[19] Marmat, A., & Thankachan, D. (2025). ML-driven latency optimization for mobile edge computing in fiber-wireless access networks. MethodsX, 15, 103594. https://doi.org/10.1016/j.mex.2025.103594

[20] Balhara, S., Gupta, N., Alkhayyat, A., Bharti, I., Malik, R. Q., Mahmood, S. N., & Abedi, F. (2025). A survey on deep reinforcement learning architectures, applications and emerging trends. IET Communications, 19(1), e12447. https://doi.org/10.1049/cmu2.12447

[21] V, J. K., & Elumalai, V. K. (2025). A proximal policy optimization based deep reinforcement learning framework for tracking control of a flexible robotic manipulator. Results in Engineering, 25, 104178. https://doi.org/10.1016/j.rineng.2025.104178

[22] Terven, J. (2025). Deep Reinforcement Learning: A Chronological Overview and Methods. AI, 6(3). https://doi.org/10.3390/ai6030046

[23] Foerster, J., Assael, I. A., de Freitas, N., & Whiteson, S. (2016). Learning to Communicate with Deep Multi-Agent Reinforcement Learning. Advances in Neural Information Processing Systems, 29. https://proceedings.neurips.cc/paper/2016/hash/c7635bfd99248a2cdef8249ef7bfbef4-Abstract.html

[24] Haxhiraj, E., Shahu, D., & Agastra, E. (2025). Pareto Front Transformation in the Decision-Making Process for Spectral and Energy Efficiency Trade-Off in Massive MIMO Systems. Sensors, 25(5). https://doi.org/10.3390/s25051451

[25] Joshua, C., Karkala, S., Hossain, S., Krishnapatnam, M., Aggarwal, A., Zahir, Z., Pandhare, V., & Shah, V. (2025). Latency-Accuracy Trade-off Analysis in Edge-Based Object Detection Pipelines. Advances in Engineering Software.

[26] Huang, M., Liu, W., Wang, T., Liu, A., & Zhang, S. (2020). A Cloud-MEC Collaborative Task Offloading Scheme With Service Orchestration. IEEE Internet of Things Journal, 7(7), 5792-5805. https://doi.org/10.1109/JIOT.2019.2952767

[27] Ghorbian, M., Ghobaei-Arani, M., & Esmaeili, L. (2025). An energy-conscious scheduling framework for serverless edge computing in IoT. Journal of Cloud Computing, 14(1), 52. https://doi.org/10.1186/s13677-025-00780-7

[28] Anand, J., & Karthikeyan, B. (2025). Efficiency-aware adaptive deep reinforcement learning for dynamic task scheduling in edge-cloud environments. Results in Engineering, 27, 105890. https://doi.org/10.1016/j.rineng.2025.105890

[29] Ahmed, M. A. (2025). Energy-Aware Machine Learning Frameworks for Sustainable Intelligent Computing in Large-Scale Systems. International Journal on Smart & Sustainable Intelligent Computing, 2(3), 13-24. https://doi.org/10.63503/j.ijssic.2025.173

[30] Kundu, A., & Quevedo, D. E. (2020). Stabilizing Scheduling Policies for Networked Control Systems. IEEE Transactions on Control of Network Systems, 7(1), 163-175. https://doi.org/10.1109/TCNS.2019.2913566

[31] Liu, J., Huang, J., Zhou, Y., Li, X., Ji, S., Xiong, H., & Dou, D. (2022). From distributed machine learning to federated learning: A survey. Knowledge and Information Systems, 64, 1-33. https://doi.org/10.1007/s10115-022-01664-x

[32] Ali, A., Ullah, I., Singh, S. K., Sharafian, A., Jiang, W., I. Sherazi, H., & Bai, X. (2025). Energy-Efficient Resource Allocation for Urban Traffic Flow Prediction in Edge-Cloud Computing. International Journal of Intelligent Systems, 2025(1), 1863025. https://doi.org/10.1155/int/1863025

[33] Yang, H. H., Liu, Z., Quek, T. Q. S., & Poor, H. V. (2020). Scheduling Policies for Federated Learning in Wireless Networks. IEEE Transactions on Communications, 68(1), 317-333. https://doi.org/10.1109/TCOMM.2019.2944169

[34] Qi, J., Liu, C., Zhang, X., Wang, L., Wang, R., Dong, J., & Yu, Y. (2025). A Survey on Open-Source Edge Computing Simulators and Emulators: The Computing and Networking Convergence Perspective (arXiv:2505.09995). arXiv. https://doi.org/10.48550/arXiv.2505.09995

[35] Osee, K. M. T., Nkemeni, V., & Sone, M. E. (2025). Quality of Experience Management in Mobile Networks: Techniques, Constraints, and Emerging Trends for Value-Added Services. Computer Networks and Communications, 21-58. https://doi.org/10.37256/cnc.3220256766

[36] Yu, G., Ma, L., Wang, X., Du, W., Du, W., & Jin, Y. (2025). Towards fairness-aware multi-objective optimization. Complex & Intelligent Systems, 11(1), 50. https://doi.org/10.1007/s40747-024-01668-w