# Fair Joint Offloading and Consensus Optimization in Blockchain-Enabled Mobile Edge Computing

Libo Feng, Chenxi Wang, Zhenli He, *Senior Member, IEEE*, Mengzhuang Liu, Jixian Zhang, and Keqin Li, *Fellow, IEEE*

***Abstract*—Blockchain-enabled mobile edge computing (MEC) must jointly optimize *task offloading* and *consensus finality* under highly heterogeneous AIoT devices, where latency/energy constraints and fairness-sensitive incentives coexist with time-varying validator reliability. We propose *FE-CTDE*, a unified framework that couples (1) a Stackelberg pricing-and-allocation layer that reaches a unique equilibrium and reduces utility disparity, (2) a reliability-aware dynamic BFT committee and block-packing mechanism that stabilizes confirmation delay under intermittent connectivity, and (3) a centralized-training/decentralized-execution multi-agent policy that outputs a continuous offloading ratio while requiring only local observations at run time. Extensive simulations across diverse heterogeneity, workload burstiness, and link intermittency show that FE-CTDE consistently improves social welfare and fairness while reducing end-to-end latency/energy and sustaining higher *effective* consensus throughput, outperforming strong baselines by up to 22.23%. We further report protocol/learning overheads and provide reproducible implementation details.**

***Index Terms*—Mobile edge computing, blockchain, task offloading, consensus optimization, fair resource allocation.**

## I. Introduction

### A. Motivation

THE accelerating proliferation of AIoT devices across smart urban infrastructure, industrial automation lines, cooperative vehicular environments, and safetycritical monitoring is pushing computation, data fusion, trust management, and privacy enforcement ever closer to end devices [1], [2], [3]. MEC supplies proximity processing that trims roundtrip delay and alleviates core network pressure, while blockchain contributes auditability, tamper resistance, and decentralized trust in multistakeholder environments where parties neither fully trust nor wish to surrender control to a single coordinator [4], [5]. Their convergence, hereafter referred to as blockchainenabled MEC, offers a compelling paradigm for lowlatency, accountable, and privacypreserving AIoT services under heterogeneous wireless links and rapidly fluctuating AD capabilities [6], [7].

Within this paradigm two tightly coupled decision layers dominate endtoend performance. The first is *task offloading:* deciding how to partition finegrained computational demand between constrained devices and edge servers so as to balance latency, energy consumption, and Quality of Service (QoS) [8]. The second is *blockchain consensus:* determining which nodes validate, order, and commit offloaded task metadata and incentive transactions, thereby shaping system throughput, confirmation delay, and security level [9], [10]. Misalignment between these layers is costly. Aggressive offloading when consensus capacity is temporarily saturated creates backlog, elevated confirmation delay, and wasted transmission energy. Overly conservative offloading despite unused trustworthy consensus and edge compute capacity squanders potential latency and energy savings for delaysensitive perception or control loops.

A rich body of work has advanced individual components. Optimization and heuristic approaches address latency and energy tradeoffs for (often binary) offloading under simplified or quasistatic assumptions [11]. Blockchain has been integrated to replace centralized brokers, usually via fixed PBFT committees that provide deterministic finality at moderate scale [12]. Learningbased schedulers exploit experiencedriven adaptation, although many retain assumptions of global observability or centralized controllers for state aggregation [13]. Incentive or

TABLE I
Authentication Primitives in LDPBFT Phases

| Phase | Message Type | Auth Primitive |
|---|---|---|
| Request | Client → Primary | Signature (non-repudiation) |
| Pre-Prepare | Primary → Replicas | MAC (efficiency) |
| Prepare | Replica → Validators | MAC (efficiency) |
| Commit | Validator → Validators | MAC (efficiency) |
| Reply | Validator → ADs | MAC (efficiency) |

Received 21 August 2025; revised 29 March 2026; accepted 10 April 2026. Date of publication 14 April 2026; date of current version 6 August 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62362068, in part by the Yunnan Science and Technology Program under Grant 202303AC100001 and Grant 202403AP140021, in part by the Open Foundation of Yunnan Key Laboratory of Software Engineering under Grant 2023SE208, in part by the Yunnan Fundamental Research Projects under Grant 202501AS070076, and in part by the Program for Excellent Young Talents, Yunnan, China. Recommended for acceptance by O. Onireti. *(Corresponding author: Zhenli He.)*

Libo Feng, Chenxi Wang, Zhenli He, and Mengzhuang Liu are with the Engineering Research Center of Cyberspace, Yunnan Key Laboratory of Software Engineering, Yunnan University, Kunming 650500, China, and also with the School of Software, Yunnan University, Kunming 650500, China (e-mail: fenglibo@ynu.edu.cn; wangchenxi@stu.ynu.edu.cn; hezl@ynu.edu.cn; liumengzhuang@stu.ynu.edu.cn).

Jixian Zhang is with the School of Information Science and Engineering, Yunnan University, Kunming 650504, China (e-mail: zhangjixian@ynu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA.

This article has supplementary downloadable material available at https://doi.org/10.1109/TMC.2026.3683699, provided by the authors.

Digital Object Identifier 10.1109/TMC.2026.3683699

pricing heuristics have been introduced to encourage participation and cost sharing, yet frequently adopt uniform tariffs or coarse reward sharing [14]. These strands deliver foundational gains in responsiveness, trust establishment, adaptability, or incentive alignment in isolation. However, they remain fragmented, and holistic fairnessaware coordination of offloading and consensus under realistic heterogeneity is still immature.

As deployments scale and heterogeneity deepens, four interlocking challenges emerge:

1) *Centralized optimization versus decentralized operability:* Designs requiring central controllers or global state snapshots introduce control traffic bursts, decision staleness from propagation delay, and aggregation bottlenecks. In dynamic wireless channels with bursty arrivals, staleness triggers oscillatory offloading that inflates queueing delay and drains battery energy.
2) *Static consensus membership ignoring temporal heterogeneity:* Fixed PBFT committees or greedy resourcebased selection ignore transient CPU depletion, contention, or connectivity loss. Retaining underperforming validators elongates commit latency; purely resourcedriven selection narrows diversity and may weaken Byzantine fault tolerance.
3) *Absent incentive-compatible pricing undermining fairness:* Assuming altruistic servers or uniform tariffs neglects heterogeneous valuations of latency and energy savings. Without differentiated, strategy-proof compensation, low-value tasks may be over-subsidized while high-impact tasks starve, driving inefficiency, elevated utility disparity (high Gini), and strategic misreporting.
4) *Coarse offloading granularity constraining efficiency:* Treating tasks as indivisible (all-local or all-edge) forfeits intermediate partitioning (e.g., local feature extraction with edge inference) that amortizes transmission cost under moderate bandwidth. Coarse granularity causes congestion bursts when large tasks transfer during rate dips, swelling consensus backlog and confirmation delay.

If left unresolved, these challenges form a reinforcing degradation loop. Inequitable rewards (C3) drive validator attrition, shrinking the pool and weakening Byzantine fault tolerance. This exacerbates static membership limitations (C2), making committees less diverse and more vulnerable to collusion. Static consensus (C2) compounds confirmation delays under overload, nullifying offloading benefits and forcing energy-intensive local execution despite available edge capacity. Coarse partitioning (C4) creates bursty transmissions that congest channels during rate dips, saturating consensus backlogs. Centralized controllers (C1) rely on stale state snapshots, triggering oscillatory offloading that inflates queueing delay and wastes energy on retransmissions. Each inefficiency amplifies the others: validator loss from C3 worsens consensus delay from C2, which increases local execution from binary offloading in C4, while stale decisions from C1 exacerbate all effects.

Consequently there is a pressing need for a fair, resourceadaptive, and decentralizationconsistent joint decision paradigm. Such a paradigm must operate with limited local observability, dynamically tailor consensus committee composition and block packing to temporal heterogeneity, embed incentivecompatible pricing aligning individual and system welfare, and support refined continuous offloading granularity. Only by resolving these friction points in a coordinated manner can blockchainenabled MEC deliver resilient, secure, and highefficiency services at scale.

### B. Our Contributions

Targeting the four challenges (C1 centralized dependence, C2 static consensus membership, C3 absent incentivecompatible pricing, C4 coarse offloading granularity), we develop an algorithmic integration framework coordinating economic incentives, consensus adaptation, computational control, and decentralized learning. Our contribution lies in demonstrating how crosslayer codesign yields performance and fairness gains beyond componentwise optimization.

*Conceptual Contributions. Unified crosslayer modeling (addresses C1–C4 jointly):* We formulate a joint optimization problem (Section III) coupling MEC utility, heterogeneous AD valuations, and consensus metrics. Explicitly modeling interdependencies among pricing, resource allocation, consensus, and offloading exposes latencyenergythroughputfairness tradeoffs that siloed approaches miss.

*Methodological Contributions:*

1) *S-RAP mechanism (targets C3):* Derives closedform equilibrium allocations and payments aligning individual incentives with system welfare (Section IV), addressing gaps left by heuristic pricing in prior blockchainenabled MEC work.
2) *LD-PBFT protocol (targets C2):* Dynamically reselects validators and tunes block capacity based on realtime resource availability and reliability (Section III), reducing latency and sustaining throughput.
3) *Continuous offloading (targets C4):* Continuous partition ratio $x_n \in (0, 1)$ enables smooth exploration of intermediate collaboration points (Section III-D), yielding disproportionate gains in load smoothing and energy efficiency without prohibitive exploration costs.
4) *Fair CTDE coordination (addresses C1 with C2–C4):* Trains MATD3 centrally, deploys decentrally with SRAP providing realtime pricing guidance (Section V). Externalizing pricing to a gametheoretic mechanism achieves both provable equilibrium and runtime decentralization, avoiding global state aggregation bottlenecks.

*Performance-Related Contributions. Empirical validation:* Theoretical analysis (Section IV-B) and experiments (Section VI) demonstrate simultaneous gains: MEC utility (+352.43%), AD utility (+5.49%), fairness (Gini -95.99%), latency (-22.23%), energy (-19.06%), consensus throughput (+100.3%). Key insight: in blockchainenabled MEC, codesign yields performance ceilings unattainable through modular optimization. FECTDE serves as an integration template for fairnessaware, resourceadaptive edge intelligence.

The remainder of this paper is organized as follows. Section II surveys related work and situates our study within existing task offloading, blockchainenabled MEC, PBFT adaptation,

and joint optimization research. Section III presents the system model and unified problem formulation. Section IV details the S-RAP mechanism. Section V describes the learning framework, including the centralized training and decentralized execution (CTDE) MATD3 policy and refined continuous offloading action. Section VI reports extensive experimental evaluation. Section VII concludes the paper and outlines future research directions. **To facilitate reading, a comprehensive abbreviation reference table is provided in the supplementary material (Section I, Table I).**

## II. Related Work

We group prior studies into four coherent strata: (1) task offloading methods, (2) blockchain enabled MEC architectures with incentive and security enhancements, (3) PBFT based consensus within MEC, and (4) joint optimization of offloading and consensus.

### A. Task Offloading Methods

Foundational work adopted combinatorial optimization and mathematical programming to minimize latency and energy under deadline, mobility, or resource constraints in UAV assisted, vehicular, and heterogeneous edge settings [3], [12], [13], [14], [15], [16], [17], [18]. Metaheuristics and decomposition schemes improved tractability, yet most rely on quasi static parameter knowledge (channel gain distributions, task arrival statistics, residual battery) and require frequent re solving when conditions shift. In dense AIoT deployments with fast temporal variation this recalculation burden causes decision staleness and escalates control overhead, impeding scalable decentralized operation.

To enhance adaptivity, single agent deep reinforcement learning (DRL) approaches have been proposed for dynamic offloading, power control, and resource allocation [19], [20], [21], [22], [23], [24], [25], [26], [27]. Representative works include deep reinforcement learning with transfer-learning-initialized Q-tables for partial offloading under network dynamics [26], and lightweight Q-learning with CNN-based state compression deployed across IoT devices and edge platforms [27]. These methods reduce dependence on explicit analytical models and can track nonstationary environments. Nevertheless, many still aggregate rich global state at a central learner or parameter server. This introduces communication load, synchronization delay, and vulnerability to single point failure. Further, the offloading action is frequently encoded as a binary local versus edge decision, limiting the exploitation of partial task partitioning that could smooth traffic and energy usage under intermediate bandwidth or compute headroom conditions.

Multi-agent DRL (MADRL) methods advance decentralization by modeling interacting devices or hierarchical layers [28], [29], [30], [31], [32]. However, in numerous implementations either full global observability is retained during execution (through shared replay buffers or synchronized parameter broadcasts) or the action abstraction remains binary. Thus latent centralization persists and fine grained collaborative efficiency remains unrealized. Additionally, economic incentives and pricing fairness are usually exogenous or implicit, with edge servers assumed altruistic; this can encourage strategic behavior or under provisioning once heterogeneous valuations dominate.

### B. Blockchain Enabled MEC Systems

Blockchain integration has been explored to strengthen trust, integrity, provenance, and auditable resource usage in MEC ecosystems. Smart contracts enforce secure trading or access control, and distributed ledgers provide tamper evidence for offloaded computation and data sharing [33], [34], [35], [36], [37]. Recent work integrates blockchain-based access control with DRL offloading across IoT-fog-cloud tiers, supporting both online and offline operation modes [38]. These frameworks mitigate single point manipulation and support accountable collaboration. Yet most either fix or implicitly assume the set of resource providers and often treat MEC servers as unconditional contributors without explicit compensation models. Uniform or heuristic pricing can foster adverse selection where low benefit tasks dominate edge compute while high value latency sensitive tasks experience delay, increasing welfare loss and widening utility disparity. Moreover, offloading strategies embedded in these systems commonly remain coarse (all local or all edge), leaving intermediate partition opportunities unexploited. Several designs also retain centralized orchestration elements for resource registry updates or contract deployment, reintroducing coordination bottlenecks.

### C. PBFT in Blockchain Enabled MEC

Practical Byzantine Fault Tolerance (PBFT) and its derivatives are widely adopted in permissioned edge contexts for deterministic finality and bounded latency [39], [40], [41], [42]. Existing MEC oriented deployments typically preconfigure the committee (primary and replicas) and keep membership static over extended intervals. Although such designs ensure integrity against a bounded number of faulty nodes, they rarely incorporate temporally varying CPU availability, reliability scores, or intermittent connectivity into dynamic rotation policies. Persistently retaining under-performing validators elongates the prepare and commit phases, increasing confirmation delay and risking throughput collapse under task bursts. Conversely, naive greedy selection of only the highest resource nodes decreases diversity and may amplify collusion risk or perceived unfairness. Block packing (number of transactions per block) is similarly seldom adapted to short term workload or resource fluctuations, limiting elasticity and potentially creating mismatches between consensus throughput and offloading arrival patterns.

### D. Joint Task Offloading and Consensus Optimization

A growing line of research addresses the coupled latency and energy components of offloading and consensus together, often formulating composite objective functions solved via metaheuristics, reinforcement learning, or evolutionary search [43], [44], [45], [46], [47]. These studies recognize cross layer interactions: offloading surges influence block filling delay and

consensus backlog, while consensus bottlenecks delay confirmation of offloading related transactions. However, most joint formulations inherit one or more earlier simplifications. Many retain centralized state aggregation or global synchrony, limiting decentralized scalability. Committee membership is often fixed or selected by static heuristics without integrating dynamic reliability and resource metrics. Incentive compatible pricing is typically absent, so resource allocation can become inefficient or unfair under heterogeneous valuations. Offloading decisions remain binary, constraining the attainable Pareto frontier by excluding intermediate partition ratios that could simultaneously moderate consensus load and reduce AD energy consumption. As a result, performance and fairness improvements plateau before reaching their domain potential.

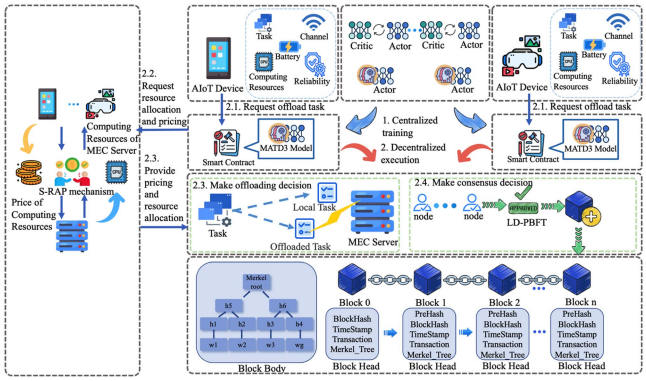



Fig. 1. FECTDE framework for a blockchainenabled MEC system.

### E. Comparison With Recent Representative Systems

While existing research has made progress across optimization methods (controlled latency-energy tradeoffs), DRL/MADRL (adaptivity), blockchain integration (trust), and PBFT (deterministic finality), their residual limitations are mutually reinforcing. Four key gaps persist: (1) centralized or globally synchronized decision logic creates bottlenecks in dynamic wireless environments; (2) static committee membership and fixed block packing elongate confirmation latency under resource variation; (3) absent incentive-compatible pricing distorts allocation and encourages misreporting; (4) binary offloading granularity prevents smooth load balancing, exacerbating transmission spikes during rate dips. These factors trigger compounding degradation where delayed consensus magnifies offloading variance, unfair compensation reduces provisioning, and coarse partitioning wastes capacity while amplifying failure risk.

*The Integration Imperative:* The key insight driving our work is that these mechanisms cannot be optimized independently without sacrificing system-wide efficiency and fairness. The challenge lies in creating a *unified integration framework* where economic equilibrium, consensus adaptation, and learning-based offloading operate in synergy. This necessity stems from three critical interdependencies:

*1) Economic Preconditioning of Learning Environments:* Traditional MADRL assumes altruistic servers or exogenous pricing, creating misaligned incentives between edge servers and devices. Our S-RAP mechanism embeds game-theoretic resource allocation *before* offloading decisions to establish fair pricing and resource allocation between edge servers and devices, ensuring mutual benefit and equitable utility distribution rather than relying on ad-hoc reward shaping or artificial utility functions.

*2) Consensus-Offloading Feedback Loops:* Existing joint optimization treats consensus capacity as fixed, ignoring how offloading bursts influence transaction backlogs while consensus bottlenecks affect offloading viability. Our LD-PBFT closes this loop by adaptively tuning validator selection and block packing based on real-time offloading patterns.

*3) Decentralization-Fairness Trade-Offs:* Prior frameworks sacrifice either decentralized execution (through global state dependencies) or fairness guarantees (through heuristic allocation). Our CTDE approach with equilibrium-guided policies achieves runtime decentralization while maintaining provable fairness properties.

This integrated perspective motivates the FE-CTDE architecture in Section III, where interdependencies among pricing, resource allocation, consensus dynamics, and adaptive offloading are explicitly addressed. **(A comprehensive synthesis of research streams and residual limitations across these dimensions is provided in Section II of the supplementary material (Table II)).**

### F. Positioning of Our Work

The primary contribution of our work lies not in the individual algorithmic components—Stackelberg games, PBFT variants, continuous action spaces, and multi-agent reinforcement learning are all established techniques—but in their *principled integration* to address the compounded inefficiencies that emerge when these mechanisms operate in isolation. Prior blockchain-enabled MEC systems treat economic incentives, consensus protocols, and offloading policies as separable optimization problems, leading to suboptimal equilibria where improvements in one layer are offset by degradations in others.

## III. FE-CTDE Framework for Blockchain-Enabled MEC

In blockchain-enabled MEC, task offloading and consensus finality are tightly coupled under heterogeneous compute, links, and validator reliability, so optimizing either layer alone can amplify delay, energy waste, and unfairness; FE-CTDE stabilizes the analytically tractable parts (pricing/allocation and consensus timing) while letting the adaptive part (continuous offloading) learn within these signals, then specifies the actors and workflow, derives computation/consensus cost models, and formulates a unified objective that makes the cross-layer trade-offs explicit.

### A. Network Model

Fig. 1 sketches the operational environment, which contains heterogeneous AIoT devices (ADs), MEC servers, and smart contracts that reside on a permission-ed blockchain. Let $\mathbb{N} = \{1, 2, \ldots, N\}$ denote the set of ADs. In each time slot $t$, AD

$n \in \mathbb{N}$ receives a job $L_n = (s_n, c_n, \tau_n)$, where $s_n$ is the input size in bits, $c_n$ is the required CPU cycles per bit, and $\tau_n$ is the latency deadline. All ADs participate in the ledger; those that meet the eligibility rules validate and commit taskoffloading transactions. The four core entities are described below.

- *AIoT Devices:* Each AD publishes a resource vector $Y_n = (f_n, v_n, b_n, r_n)$, which records its local CPU frequency $f_n$, current up-link rate $v_n$, residual battery budget $b_n$, and CPU cycles $r_n$ reserved for consensus. A reliability index $k_n \in [0, 1]$ summarizes longterm connectivity and fault history; higher values improve the chance of being elected as a validator. In production deployments, $k_n$ is maintained via an exponentially weighted moving average (EWMA) of successful consensus participations, managed by a reputation smart contract to ensure temporal continuity and tamper resistance.[1]
- *Smart Contracts:* Every AD hosts a selfexecuting contract that embeds a pretrained reinforcement learning agent. On task arrival the contract decides the offloading ratio using only local observations and settles payments at the prices broadcast by the MEC server.
- *MEC Servers:* Edge servers execute two duties. First, they train all reinforcement learning agents with global information and redistribute the converged policies to the ADs. Second, they process the offloaded workload they receive and return the results within each task's QoS requirement.
- *Blockchain Layer:* ADs and servers jointly maintain a permissioned ledger protected by the LDPBFT protocol. At every consensus round the protocol evaluates resource availability and reliability scores, elects one primary and $M-1$ replica validators, and adjusts block size so that consensus throughput tracks current traffic.

This hierarchical yet decentralized design aligns economic incentives, computation scheduling, and consensus formation without continued dependence on a global controller.

FE-CTDE cleanly separates roles while preseving cross-layer coupling. Next we summarize the per-slot on-chain workflow that coordinates these roles.

### B. Operational Workflow of FE-CTDE

Fig. 2 illustrates the end-to-end workflow. The FECTDE framework operates in two phases: an offline training phase and a runtime execution phase with four tightly coupled stages:

1) *Centralized Policy Training (Offline):* Using globally observed network snapshots, the MEC server trains an MATD3 model. After convergence, the distilled parameters are embedded into the smart contracts of all ADs.
2) *Task Arrival & Offloading Request (Runtime):* Upon receiving a job $L_n = (s_n, c_n, \tau_n)$, $\text{AD}_n$ publishes its resource state and workload characteristics, initiating the offloading decision process.

[1]For analytical tractability in our simulations, we approximate dynamic reliability fluctuations by uniformly sampling $k_n$ from $[0.0, 1.0]$ each epoch, isolating the impact of our core mechanisms without introducing additional reputation-tracking complexity. This randomized model preserves heterogeneity while ensuring reproducibility across experiments.

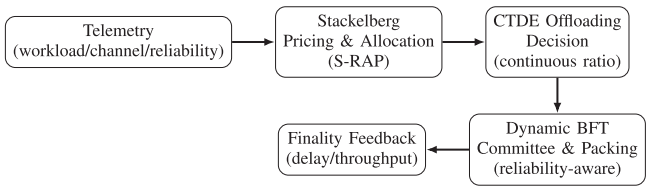



Fig. 2. End-to-end workflow of FE-CTDE. Centralized training is used offline; at run time, each device executes with local observations, while consensus finality feedback closes the loop.

3) *Price & Resource Negotiation via SRAP (Runtime):* At the start of every time slot, MEC servers and ADs execute the SRAP protocol. The MEC server computes equilibrium allocations $\{F_n^*\}$ and broadcasts prices $\{TK_n^*\}$ in a manner that is both incentivecompatible and provably fair.
4) *Decentralized Offloading Decision (Runtime):* With the SRAP pricing and resource allocation in hand, $\text{AD}_n$ invokes its local smart contract to compute an offloading ratio $x_n$ based solely on local observations. The fraction $x_n L_n$ is transmitted to the edge and billed automatically, while $(1 - x_n)L_n$ is executed locally.
5) *Dynamic Consensus Formation (Runtime):* At the end of each epoch $t$, the LDPBFT routine reelects one primary and $M-1$ replica validators using the latest reliability scores $k_n$ and available consensus cycles $r_n$. Transaction-packing size $\varphi$ is tuned adaptively, and the PBFT phases proceed until the block reaches finality.

Algorithm 1 presents the master control loop that orchestrates the interplay among economic negotiation (Algorithm 2), centralized policy training (Algorithm 3), decentralized offloading inference (Algorithm 4), and dynamic consensus adaptation (Algorithm 5). This highlevel abstraction clarifies the separation between offline training and runtime execution, and highlights how finality feedback stabilizes pricing and resource allocation in subsequent epochs.

**A comprehensive list of notation is provided in Section III-A of the supplementary material (Table III).**

FE-CTDE executes as a four-stage loop: S-RAP pricing/allocation, CTDE training, local offloading, and LD-PBFT consensus adaptation. Next we quantify LD-PBFT timing to connect consensus delay with offloading decisions.

### C. LD-PBFT Consensus Model

LDPBFT reconciles two often conflicting goals in edgehosted blockchains—high throughput and strict reliability—by electing validators on the fly and tuning block size on demand. In every consensus epoch, one *primary* AD forges the candidate block, while $M-1$ *replicas* validate it; the remaining ADs simply append the finalized block to their local ledgers.

Distinct from static PBFT variants, LDPBFT refreshes validator membership using realtime CPU availability $r_n$ and reliability scores $k_n$. It then computes a transactionpacking size $\varphi$ that maximizes confirmedtransactionspersecond without breaching latency guarantees. The full protocol unfolds in six phases.

**Algorithm 1:** FE-CTDE Master Control Loop.

**Input:** Network parameters $\{\alpha_n, p_n, f_n, \upsilon_n, b_n, r_n, k_n\}$, task arrivals $\{L_n\}$
**Output:** Offloading decisions $\{x_n\}$, consensus results with finality
**Phase I: Offline Training**
Execute **Algorithm 2** (MATD3 Training) with global state
Distribute trained policies $\{\psi_n^*\}$ to all AD smart contracts
**Phase II: Runtime Execution (per time slot** $t$**)**
**for** each epoch $t = 1, 2, \ldots$ **do**
  **Step 1: Economic Layer**
    Execute **Algorithm 1** (S-RAP) $\rightarrow$ obtain $\{F_n^*, TK_n^*\}$
  **Step 2: Offloading Layer**
  **for all** AD $n \in \mathbb{N}$ **do**
      Execute **Algorithm 3** (Task Offloading) with local $s_n(t)$
      $\rightarrow$ determine offloading ratio $x_n(t)$, compute $T_n, E_n$
  **end for**
  **Step 3: Consensus Layer**
  Execute **Algorithm 4** (Consensus Adaptation) with committee size $M$
  $\rightarrow$ select validators $\mathcal{N}(t)$, tune block size $\{\varphi_n(t)\}$
  Run LD-PBFT phases, achieve finality with delay $T^C$
**end for**
**return** Stabilized offloading-consensus equilibrium

*1) Cryptographic Primitives and Trust Model:* To ensure nonrepudiation of transactions while maintaining efficient intervalidator communication, LDPBFT employs a hybrid authentication scheme: (i) each transaction carries a digital signature from its originator, incurring $\beta_{\text{sig}}^{\text{gen}}$ CPU cycles to generate and $\beta_{\text{sig}}^{\text{ver}}$ cycles to verify; (ii) protocol messages exchanged among validators use symmetric Message Authentication Codes (MACs), requiring $\beta_{\text{mac}}^{\text{gen}}$ cycles to generate and $\beta_{\text{mac}}^{\text{ver}}$ cycles to verify. This design balances accountability (transactions remain auditable) with throughput (validator messages avoid expensive publickey operations). Table I summarizes the authentication primitives used in each phase.

*2) Consensus Node Selection:* Before each round, the contract nominates one primary and $M-1$ replicas from the AD pool $\mathbb{N}$ according to their spare cycles $r_n$. Owing to PBFT's fault tolerance, the system endures up to $f = \lfloor (M-1)/3 \rfloor$ Byzantine failures. By maintaining a fixed committee size $M$, LDPBFT preserves this guarantee even though membership changes dynamically.

*3) Request Phase:* The primary assembles $\varphi$ unverified transactions, sorts them by timestamp, and verifies each transaction's digital signature (costing $\beta_{\text{sig}}^{\text{ver}}$ cycles). The computational cost is thus

$$c_1^{\text{prim}} = \varphi \beta_{\text{sig}}^{\text{ver}}. \tag{1}$$

*4) Pre-Prepare Phase:* The primary broadcasts the block of $\varphi$ transactions plus a preprepare protocol message, attaching one MAC per replica (total $M-1$ MACs at $\beta_{\text{mac}}^{\text{gen}}$ each). Each replica then verifies the header MAC and all $\varphi$ transaction signatures:

$$c_2^{\text{prim}} = (M-1)\beta_{\text{mac}}^{\text{gen}}, \tag{2}$$

$$c_1^{\text{rep}} = \beta_{\text{mac}}^{\text{ver}} + \varphi \beta_{\text{sig}}^{\text{ver}}. \tag{3}$$

*5) Prepare Phase:* Upon successful verification, each replica multi-casts a prepare message with MACs to all other validators. Progression to the next stage requires $2f$ matching prepares. The corresponding costs are

$$c_3^{\text{prim}} = 2f\beta_{\text{mac}}^{\text{ver}}, \tag{4}$$

$$c_2^{\text{rep}} = (M-1)\beta_{\text{mac}}^{\text{gen}} + 2f\beta_{\text{mac}}^{\text{ver}}. \tag{5}$$

*6) Commit Phase:* All validators issue a commit message after collecting $2f$ matching prepares. Both primary and replicas must verify $2f$ incoming MACs and generate $M-1$ outgoing MACs:

$$c_4^{\text{prim}} = c_3^{\text{rep}} = (M-1)\beta_{\text{mac}}^{\text{gen}} + 2f\beta_{\text{mac}}^{\text{ver}}. \tag{6}$$

*7) Reply Phase:* Once a validator receives $2f$ matching commits, the block attains finality. Each validator then sends a reply with an attached MAC to ordinary nodes. ADs update their ledgers after gathering $f+1$ valid replies:

$$c_5^{\text{prim}} = c_4^{\text{rep}} = \beta_{\text{mac}}^{\text{gen}}. \tag{7}$$

*Validator Latency* Let $r_n$ denote the CPU cycles reserved by $\text{AD}_n$ for consensus. The compute latency for the primary and replicas is

$$T_{\text{prim}}^C = \frac{c_1^{\text{prim}} + c_2^{\text{prim}} + c_3^{\text{prim}} + c_4^{\text{prim}} + c_5^{\text{prim}}}{r_n}, \tag{8}$$

$$T_{\text{rep}}^C = \frac{c_1^{\text{rep}} + c_2^{\text{rep}} + c_3^{\text{rep}} + c_4^{\text{rep}}}{r_n}, \tag{9}$$

and the epoch's consensus delay is governed by the slower role:

$$T^C = \max\{T_{\text{prim}}^C, T_{\text{rep}}^C\}. \tag{10}$$

*Interpretation:* The round delay is governed by the slowest validator role; dynamic selection prioritizes high available consensus cycles $r_n$ and stable reliability $k_n$ to reduce this bottleneck.)

By coupling adaptive validator rotation with onthefly block sizing, LDPBFT achieves high transaction throughput while keeping latency within stringent MEC budgets.

LD-PBFT turns validator heterogeneity into an explicit latency bound $T^C$. Next we model partial task offloading so the policy can trade off local cost against edge and consensus load.

### D. Task Offloading Model

Given the quality of service tuple $L_n = (s_n, c_n, \tau_n)$ that specifies the input size $s_n$, the required CPU cycles per bit $c_n$, and the strict deadline $\tau_n$, $\text{AD}_n$ must determine what portion of the task should be processed at the edge. Let $x_n \in (0,1)$ denote this decision, where the fraction $x_n$ is offloaded to the MEC server and the remaining $1-x_n$ is executed locally. The following analysis derives the latency and energy associated with each part of the task.

*1) Edge Execution:* When $\mathrm{AD}_n$ forwards $x_n L_n$ to the edge, two delays dominate:

- *Wireless transmission:* The transmission delay includes both uplink and downlink components. Assuming the computation result is compressed to 1% of the original input size, the total transmission delay is
$$T_n^{\text{trans}} = \frac{x_n s_n}{v_n} + \frac{x_n s_n/100}{v_n}, \tag{11}$$
where $v_n$ is the instantaneous data rate, the first term represents the uplink delay for transmitting the offloaded task, and the second term accounts for the downlink delay of receiving the compressed result.
- *Edge processing:* The MEC server allocates $F_n$ CPU cycles to the offloaded slice, leading to
$$T_n^{\text{MEC}} = \frac{x_n s_n c_n}{F_n}. \tag{12}$$

The endtoend edge latency combines both transmission and processing delays:
$$T_n^{\text{off}} = T_n^{\text{trans}} + T_n^{\text{MEC}}. \tag{13}$$

*Interpretation:* $x_n$ scales both transmission and edge execution, so continuous offloading can smooth load rather than creating bursty all-or-nothing traffic.

The radio energy consumed includes both uplink and downlink transmissions:
$$E_n^{\text{off}} = T_n^{\text{trans}} p_n, \tag{14}$$
where $p_n$ is the transmit power of the AD.

*2) Local Execution:* The remaining $(1 - x_n)L_n$ is processed on the AD's own CPU at frequency $f_n$. The corresponding delay is
$$T_n^{\text{local}} = \frac{(1 - x_n)s_n c_n}{f_n}. \tag{15}$$

The dynamic power model $p = k_C f^{\varsigma}$ (with $\varsigma = 3$) gives the local energy draw:
$$p_n^{\text{local}} = k_C f_n^{\varsigma}, \tag{16}$$
$$E_n^{\text{local}} = p_n^{\text{local}} T_n^{\text{local}} = (1 - x_n) s_n c_n k_C f_n^2. \tag{17}$$

*Interpretation:* Local energy decreases with $x_n$ but grows quadratically with $f_n$, motivating offloading when battery is tight or local CPU runs fast.

*Remark 1:* This continuous formulation lets the learning agent search a finegrained space of collaboration strategies, unlike binary offloading models that force a myopic alllocal or alledge choice. By exposing smooth latencyenergy tradeoffs, the model provides richer gradients for MATD3 training and, as shown later, delivers significant improvements in both user experience and system fairness.

Continuous $x_n$ exposes a controllable latency–energy surface instead of a discrete cliff. Next we unify task and consensus utilities into one optimization target for FE-CTDE.

### E. Problem Formulation

Since every job $L_n$ is divided between local execution and edge execution, the end to end latency equals the slower of the two branches:
$$T_n = \max\left(T_n^{\text{local}}, T_n^{\text{off}}\right). \tag{18}$$

The total energy spent by $\mathrm{AD}_n$ is the sum of local and radio consumption,
$$E_n = E_n^{\text{local}} + E_n^{\text{off}}. \tag{19}$$

*Interpretation:* Latency is a two-branch critical path (max), while energy is additive; partial offloading can therefore reduce energy without necessarily reducing latency (and vice versa).

*Task utility:* We map latency and energy to dimensionless scores that rise when performance improves. Let $T_n^{\max}$ and $E_n^{\max}$ denote the cost of fully local computation ($x_n = 0$). The latency score is
$$J_n^{\text{time}}(x_n) = \frac{1}{1 + e^{-\left(1 - \frac{T_n}{T_n^{\max}}\right)}}, \tag{20}$$
and the energy score is
$$J_n^{\text{energy}}(x_n) = \frac{1}{1 + e^{-\left(1 - \frac{E_n}{E_n^{max}}\right)}}. \tag{21}$$

To discourage performance degradation, we introduce smooth penalty terms based on sigmoid functions. For latency, we define
$$P_n^{\text{time}} = -\frac{2}{1 + e^{-\left(\frac{T_n}{\tau_n} - 1\right)}}, \tag{22}$$
Similarly, for energy consumption
$$P_n^{\text{energy}} = -\frac{2}{1 + e^{-\left(\frac{E_n}{b_n} - 1\right)}}, \tag{23}$$
Both penalties increase smoothly when the actual metric ($T_n$ or $E_n$) exceeds its baseline threshold ($\tau_n$ or $b_n$).

The overall task performance metric combines both scores and penalties
$$J_n^{\text{task}}(x_n) = J_n^{\text{time}}(x_n) + J_n^{\text{energy}}(x_n) + P_n^{\text{time}} + P_n^{\text{energy}}. \tag{24}$$

*Interpretation:* $J_n^{\text{task}}$ rewards relative improvements over full-local baselines and adds smooth constraint-aware penalties, which stabilizes learning signals under occasional violations.

*Consensus utility:* Validator utility depends on reliability $k_n$, block size $\varphi_n$, and the duration of the consensus round $T^{\text{block}}$. Let $k_{\max} = 1$ be the highest reliability and $\varphi_{\max}$ the largest allowable block. To encourage fast confirmation while penalizing timeout violations, we introduce a consensus time penalty term:
$$P_n^{\text{con}} = -\frac{2}{1 + e^{-\left(\frac{T^{\text{block}}}{\tau} - 1\right)}}, \tag{25}$$
where $\tau$ is the consensus timeout threshold. This penalty increases smoothly when $T^{\text{block}}$ exceeds $\tau$, mirroring the task-layer time penalty structure. The composite validator utility integrates reliability, block size, time reward, and time penalty:
$$J_n^{\text{block}}(z_n, \varphi_n) = z_n \frac{k_n}{k_{\max}} \frac{\varphi_n}{\varphi_{\max}} \left( \frac{1}{1 + e^{-\left(1 - \frac{T^{\text{block}}}{\tau}\right)}} \right) + P_n^{\text{con}}, \tag{26}$$
where $z_n \in \{0, 1\}$ indicates whether $\mathrm{AD}_n$ serves as a validator. The first sigmoid term rewards faster confirmation, while $P_n^{\text{con}}$ explicitly penalizes delays beyond $\tau$.

*Interpretation:* Consensus utility couples reliability ($k_n$), packing ($\varphi_n$), and timing ($T^{\text{block}}$) so throughput gains are only credited when finality remains fast.

*System objective:* The goal is to choose the offloading ratio $x_n$, the validator flag $z_n$, and the block size $\varphi_n$ that maximize

the average utility of computation and consensus:

$$\max_{x_n, z_n, \varphi_n} \quad \frac{1}{N}\sum_{n=1}^{N} J_n^{\text{task}} + \frac{1}{M}\sum_{n=1}^{M} J_n^{\text{block}} \tag{27}$$

$$\text{subject to} \quad 0 \le x_n \le 1, \qquad \forall n \in \mathbb{N}, \tag{27a}$$

$$z_n \in \{0,1\}, \qquad \forall n \in \mathbb{N}, \tag{27b}$$

$$\varphi_{\min} \le \varphi_n \le \varphi_{\max}, \qquad \forall n \in \mathbb{N}, \tag{27c}$$

$$0 < T_n \le \tau_n, \qquad \forall n \in \mathbb{N}, \tag{27d}$$

$$0 < E_n \le b_n, \qquad \forall n \in \mathbb{N}, \tag{27e}$$

$$\sum_{n=1}^{N} z_n = M. \tag{27f}$$

*Interpretation:* Variables split naturally across layers: $x_n$ is learned per AD, while $(z_n, \varphi_n)$ are adapted by LD-PBFT under committee and budget constraints. Constraint (27a) bounds the offloading ratio, (27b) encodes validator selection, and (27c) limits block size. Constraints (27d) and (27e) enforce latency and energy budgets, while (27f) fixes the committee size for the consensus round. **(For a detailed discussion of scale normalization in the objective function, see Remark 2 in Section III-B of the supplementary material).**

The unified formulation makes computation utility and consensus finality comparable in one objective. Next we instantiate the economic layer (S-RAP) that provides equilibrium prices and allocations for the subsequent modules.

## IV. Stackelberg Resource Allocation and Pricing Mechanism

Heterogeneous valuations and selfish behavior destabilize naive pricing, causing unfair allocation that distorts offloading and consensus. We embed a Stackelberg economic layer where the MEC server leads with CPU allocations and ADs respond with payments. This leader–follower structure yields analytically tractable equilibria computable each epoch. We formalize the game, prove equilibrium uniqueness, and summarize online execution with negligible overhead.

### A. Stackelberg Game Model

*Leader move:* The MEC server publishes its allocation decision

$$\mathbf{F} = \{F_1, F_2, \ldots, F_N\}, \tag{28}$$

where $F_n > 0$ denotes the CPU cycles reserved for $\text{AD}_n$.

*Follower response:* After observing $\mathbf{F}$, each AD chooses a payment

$$\mathbf{TK} = \{TK_1, TK_2, \ldots, TK_N\} \tag{29}$$

that maximizes its own utility.

*AD utility:* $\text{AD}_n$ derives diminishing returns from the cycles it receives and pays a direct monetary cost:

$$U_{AD}^n = \alpha_n \ln\Big(1 + \frac{F_n}{p_n}\Big) - TK_n, \tag{30}$$

where $\alpha_n > 0$ captures marginal benefit and $p_n > 0$ reflects sensitivity to computation resources.

*MEC server utility:* The MEC server collects revenue from payments but incurs a quadratic energy cost:

$$U_{MEC} = \sum_{n=1}^{N} TK_n - \rho k_{MEC} \sum_{n=1}^{N} F_n^2, \tag{31}$$

with $\rho > 0$ and $k_{MEC} > 0$ denoting energyrelated coefficients.

*Interpretation:* Equation (30) captures diminishing marginal benefit of edge CPU cycles and a direct payment cost, while (31) prices the MEC server's capacity through a convex (quadratic) energy penalty, which discourages over-allocation and yields a well-behaved equilibrium.

*Equilibrium definition:* A Stackelberg equilibrium is a pair $(\mathbf{F}^*, \mathbf{TK}^*)$ such that

$$F_n^* = \arg\max_{F_n} U_{\text{AD}}^n\big(F_n, TK_n^*\big), \ \forall n \in \{1, 2, \ldots, N\} \tag{32}$$

$$\mathbf{TK}^* = \arg\max_{\mathbf{TK}} U_{MEC}\big(\mathbf{F}^*, \mathbf{TK}\big), \tag{33}$$

and no participant can improve its utility by deviating unilaterally. This equilibrium balances the server's profit motive with each AD's demand for affordable computation, ensuring a stable and fair operating point for the FECTDE system.

*Addressing Strategic Misreporting:* S-RAP assumes truthful reporting of $\{\alpha_n, p_n\}$, justified by **(i)** limited information in permissioned networks and **(ii)** EWMA-based reputation systems (Section III-E) that punish misreporting via reduced future resource priority. While sophisticated adversaries may manipulate parameters, three countermeasures mitigate this risk: **1.** Historical consistency audits cross-validate reports against usage patterns; **2.** Deposit-based commitment forfeits stakes when consumption deviates from declarations; **3.** VCG-like mechanisms charge ADs based on marginal externalities. Fully strategyproof variants trade computational overhead for incentive compatibility—a direction for future work.

The Stackelberg form turns strategic interaction into a computable operating point $(\mathbf{F}^*, \mathbf{TK}^*)$. Next we discuss how misreporting can be discouraged so the equilibrium analysis remains meaningful in practice.

### B. Rigorous Game–Theoretic Analysis

This section establishes the analytical backbone of the SRAP mechanism. We proceed in three steps:

1) Derive the closedform *best response* of every AD;
2) Determine the MEC server's *optimal allocation* given these responses;
3) Prove that the resulting leader–follower strategies constitute a *unique* Stackelberg equilibrium.

Throughout, all parameters $\{\alpha_n, p_n, \rho, k_{MEC}\}$ are strictly positive, and the notation introduced in the previous subsection is retained.

*1) Optimal Payment Strategy of an AD:* Recall the utility of $\text{AD}_n$,

$$U_{AD}^n(F_n, TK_n) = \alpha_n \ln\Big(1 + \tfrac{F_n}{p_n}\Big) - TK_n, \ F_n > 0, \ TK_n \ge 0. \tag{34}$$

*Theorem 1 (AD Best Response):* For any resource allocation $F_n$ announced by the MEC server, the unique payment that

maximizes $U_{AD}^n$ is

$$TK_n^* = \frac{\alpha_n F_n}{p_n + F_n}. \tag{35}$$

*Proof: Step 1 (monotonicity):* With $F_n$ fixed, $\partial U_{AD}^n / \partial TK_n = -1 < 0$; hence $U_{AD}^n$ is strictly decreasing in the payment.

*Step 2 (individual rationality):* Rational participation requires $U_{AD}^n \geq 0$, i.e., $TK_n \leq \alpha_n \ln(1 + F_n/p_n)$.

*Step 3 (server's reservation price):* The MEC server will never accept $TK_n$ below its marginal cost. In equilibrium this minimum acceptable price coincides with the righthand side of (35) (shown later in Theorem 2). Because any larger payment would strictly reduce $U_{AD}^n$ by Step 1 and any smaller payment would be rejected by the server, (35) is the unique maximizer.□

*2) Optimal Allocation of the MEC Server:* Substituting (35) into the server's utility gives the singlevariable concave program

$$\max_{F_n > 0} U_{MEC}(F_n) = \frac{\alpha_n F_n}{p_n + F_n} - \rho k_{MEC} F_n^2. \tag{36}$$

*Theorem 2 (Unique Optimal Allocation):* The objective in (36) is strictly concave on $(0, \infty)$ and attains its unique global maximizer $F_n^* > 0$, which is the sole positive root of

$$g(F_n) \;=\; F_n^3 + 2p_n F_n^2 + p_n^2 F_n - \frac{\alpha_n p_n}{2\rho k_{MEC}} = 0. \tag{37}$$

*Proof:* Compute the first and second derivatives:

$$U'_{MEC}(F_n) = \frac{\alpha_n p_n}{(p_n + F_n)^2} - 2\rho k_{MEC} F_n,$$

$$U''_{MEC}(F_n) = -\frac{2\alpha_n p_n}{(p_n + F_n)^3} - 2\rho k_{MEC}. \tag{38}$$

Because every term in $U''_{MEC}$ is negative for $F_n > 0$, $U_{MEC}$ is strictly concave and therefore has at most one maximizer.

Setting $U'_{MEC}(F_n) = 0$ yields (37). Observe that $g(0) < 0$ and $\lim_{F_n \to \infty} g(F_n) = \infty$, while $g'(F_n) = 3F_n^2 + 4p_n F_n + p_n^2 > 0$ for $F_n > 0$. Hence $g$ is strictly increasing and crosses zero exactly once on $(0, \infty)$. This unique root is the global maximizer $F_n^*$.□

*3) Verification of KKT Conditions: Theorem 3 (Global Optimality via KKT):* The allocation $F_n^*$ obtained in Theorem 2 satisfies the Karush–Kuhn–Tucker conditions and is therefore globally optimal for (36).

*Proof:* Introduce the nonnegativity constraint $F_n \geq 0$ with multiplier $\lambda_n \geq 0$ and form the Lagrangian $\mathcal{L}(F_n, \lambda_n) = U_{MEC}(F_n) + \lambda_n F_n$. Stationarity implies $\lambda_n = 2\rho k_{MEC} F_n - \alpha_n p_n / (p_n + F_n)^2$. At $F_n = F_n^*$ the right-hand side equals 0 because $U'_{MEC}(F_n^*) = 0$; hence $\lambda_n = 0$. Complementary slackness $\lambda_n F_n^* = 0$ and dual feasibility $\lambda_n \geq 0$ follow immediately. Primal feasibility ($F_n^* > 0$) is obvious. Because $U_{MEC}$ is strictly concave, these KKT conditions are sufficient for global optimality.□

*4) Existence and Uniqueness of the Stackelberg Equilibrium:*

*Theorem 4 (Uniqueness under Feasible Domain):* Let the leader's decision variable satisfy $F_n \geq 0$ (and optionally $\sum_n F_n \leq F^{\text{tot}}$). If $\alpha_n > 0$, $p_n > 0$, $\rho > 0$, and $k_{\text{MEC}} > 0$, then the leader's objective is strictly concave over the convex feasible set, and hence admits a unique maximizer.

**Algorithm 2:** Stackelberg–Based Resource Allocation and Pricing (S-RAP).

**Input:** $\{\alpha_n, p_n\}_{n=1}^N$, $\rho$, $k_{MEC}$, tolerance $\delta = 10^{-6}$, `maxIter`$= 100$
**Output:** Equilibrium allocation $\{F_n^*\}$ and payments $\{TK_n^*\}$
**for all** $n \in \{1, \ldots, N\}$ **do**
  $F_n \leftarrow p_n$ ▷ warm-start at the dependency scale
  **for** $k = 1$ **to** `maxIter` **do**
    $g \leftarrow F_n^3 + 2p_n F_n^2 + p_n^2 F_n - \dfrac{\alpha_n p_n}{2\rho k_{MEC}}$
    $g' \leftarrow 3F_n^2 + 4p_n F_n + p_n^2$
    $F_{\text{new}} \leftarrow F_n - \dfrac{g}{g'}$
    **if** $|F_{\text{new}} - F_n| < \delta$ **then**
      **break**
    **end if**
    $F_n \leftarrow F_{\text{new}}$
  **end for**
  $F_n^* \leftarrow F_n$
  $TK_n^* \leftarrow \dfrac{\alpha_n F_n^*}{p_n + F_n^*}$
**end for**
**return** $\{F_n^*\}$, $\{TK_n^*\}$

*Proof:* For each $n$, the second derivative satisfies

$$U''(F_n) = -\frac{2\alpha_n p_n}{(p_n + F_n)^3} - 2\rho k_{\text{MEC}} < 0, \quad \forall F_n \geq 0. \tag{39}$$

Thus each term is strictly concave and the sum is strictly concave. Since the feasible set defined by $F_n \geq 0$ (and $\sum_n F_n \leq F^{\text{tot}}$ if present) is convex, the strictly concave maximization problem has a unique maximizer.□

*Theorem 5 (Unique Stackelberg Equilibrium):* The strategy pair $\left(F_n^*, TK_n^*\right)$ obtained from Theorem 2 and Theorem 1 constitutes the one and only Stackelberg equilibrium of the SRAP game.

*Proof: Existence:* For any allocation $\mathbf{F}$, each AD has a singlevalued best response $\mathbf{TK}^*(\mathbf{F})$ (Theorem 1). The MEC server then maximizes its concave objective with respect to $\mathbf{F}$ and obtains the unique vector $\mathbf{F}^*$ (Theorem 2). Therefore an equilibrium exists.

*Uniqueness*. The follower best response is unique, and the leader's optimal allocation is unique; hence their composition admits exactly one fixed point. No agent can gain by unilateral deviation, so the equilibrium is unique in the sense of Nash refinements for Stackelberg games.□

*Interpretation:* The equilibrium allocation $F_n^*$ strikes a delicate balance: ADs receive just enough edge capacity to maximize their diminishingreturn utilities, and the MEC server is compensated precisely at its marginal energy cost. The resulting pricing rule (35) is *incentive compatible* (truthful reporting is optimal) and *Pareto efficient* (no further bilateral trades can improve both parties), thereby furnishing a sound economic anchor for the learningbased layers that follow.

The analysis yields a unique equilibrium that can be computed efficiently and reused as a stable price/allocation signal. Next we

describe the online execution flow that reaches this equilibrium each epoch and interfaces cleanly with the learning-based control in the next section.

### C. End-to-End Execution Flow of the S-RAP Mechanism

Having proven that a unique Stackelberg equilibrium $\left(F_n^*, TK_n^*\right)$ exists (Theorem 5), we now articulate *how* the MEC server and the ADs reach this operating point online. The procedure unfolds in two tightly coupled phases and is summarized in Algorithm 2. All symbols follow the notation in Section IV-B; the cubic polynomial $g(F_n)$ is defined in (37).

*Phase 1 — Leader Optimization:* At the beginning of every pricing epoch the MEC server **(1)** collects the latest $\{\alpha_n, p_n\}$ reported by the ADs, **(2)** solves $g(F_n) = 0$ for each $n$ via a Newton–Raphson iteration, and **(3)** broadcasts the allocation vector $\mathbf{F}^* = \{F_1^*, \ldots, F_N^*\}$. Because $g(\cdot)$ is strictly increasing (Theorem 2), a single real root lies in $(0, \infty)$; the iteration therefore converges quadratically once it is sufficiently close to the solution. In practice, setting the initial guess to $p_n$ delivers convergence within 10–15 iterations under the tolerance $\delta = 10^{-6}$.

*Phase 2 — Follower Adjustment:* After observing $\mathbf{F}^*$, every AD computes its payment using the closed-form rule $TK_n^* = \alpha_n F_n^*/(p_n + F_n^*)$ (Theorem 1) and settles the fee through a lightweight smart contract. No further negotiation is needed because this payment is both individually rational and Pareto efficient. **(For detailed analysis of convergence, complexity, and communication overhead, see Section IV-A of the supplementary material).**

*Guarantees:* By construction, the two-phase routine in Algorithm 2 terminates at the unique Stackelberg equilibrium of the S-RAP game. Consequently, no AD can lower its payment without violating the MEC server's minimum acceptable price, the MEC server cannot unilaterally increase its profit by reallocating CPU cycles, and social welfare is maximized subject to the quadratic energy cost of the edge infrastructure. These properties furnish a firm economic bedrock for the learning and consensus layers introduced in subsequent sections.

S-RAP therefore provides fast, low-overhead, and equilibrium-grounded pricing/allocation at each epoch. Next we move to the adaptive control layer (MATD3 under CTDE) that learns continuous offloading and consensus-related actions on top of these economic signals.

## V. MATD3–A RefinedAction Policy for Joint Offloading–Consensus Control

SRAP fixes equilibrium prices before offloading decisions. Each AD must then learn, using only local observations, how to split workload between local and edge execution while coordinating LDPBFT participation. We address this with **M**ulti**A**gent **T**win**D**elayed **D**eep **D**eterministic policy gradient (**MATD3**), a continuouscontrol algorithm combining TD3's low-bias value estimation with CTDE scalability.

### A. Why MATD3?

Classic valuebased schemes (e.g., DQN) discretize actions and thus implode when the action dimension grows. In our setting each agent must output *three* continuous variables $\left[x_n,\ z_n,\ \varphi_n\right]$ per step: the offloading ratio, the validator flag, and the (normalized) block size. Policygradient methods cope well with such continuous domains, but singlecritic DDPG suffers from overestimation. MATD3 inherits TD3's dualcritic architecture and delayed policy updates, thereby suppressing bias while still learning a deterministic actor that is amenable to onchain execution.

Overall, MATD3 enables highdimensional continuous control while curbing critic overestimation. Next, we refine the action head so offloading and consensus choices remain feasible and executable.

### B. Refined Continuous Offloading Action

Binary "alllocal vs. alledge" control cannot smooth bursty traffic and often wastes residual edge capacity. We therefore *refine* the offloading component into a continuous ratio $x_n \in (0, 1)$. The actor's raw output $a_{n,1} \in \mathbb{R}$ is converted through a squashing function:

$$x_n = \frac{\tanh(a_{n,1}) + 1}{2}, \tag{40}$$

which guarantees feasibility of constraint (27a) without additional projection.

*Handling the discrete validator flag:* The validator selection $z_n \in \{0, 1\}$ poses a fundamental challenge: discrete decisions are non-differentiable, precluding gradient-based policy optimization. To preserve end-to-end differentiability during training, we adopt a *stochastic relaxation* approach. The actor outputs a raw logit $a_{n,2} \in \mathbb{R}$, which is transformed into a probability via a temperature-scaled sigmoid:

$$q_n = \sigma\left(\frac{a_{n,2}}{T}\right), \tag{41}$$

where $T$ is a temperature parameter that anneals from 1.0 to 0.1 during training to gradually sharpen the probability distribution. During each training step, we sample $z_n \sim \text{Bernoulli}(q_n)$, enabling the policy gradient to flow through the expected value $\mathbb{E}[z_n] = q_n$.

At deployment, blockchain execution requires deterministic decisions. We therefore switch to a threshold-based policy: $z_n = \mathbf{1}\{a_{n,2} > 0\}$. This transition from stochastic to deterministic mode mirrors common practices in deep learning (e.g., dropout, exploration noise): the network learns robust logit representations during training, which naturally converge to confident binary decisions as temperature annealing progresses. The temperature schedule ensures that by the end of training, $q_n$ is near 0 or 1, minimizing the train-test discrepancy.

Overall, $x_n$ supports smooth load splitting and the relaxed $z_n$ keeps training differentiable while remaining deterministic at deployment. Next, we describe CTDE training and execution so only local observations are needed online.

### C. Centralized Training, Decentralized Execution

During each training episode the MEC server aggregates *global* experience tuples $\left(s, a, r, s'\right)$ from every AD into a shared replay buffer. Both critics $Q^1, Q^2$ observe the *joint* state–action pair $(s, a)$, whereas each actor $\pi_n$ sees only its own state slice $s_n$. Once the networks converge they are embedded inside the ADs'

smart contracts and operate *fully locally:* no runtime model parameters, gradients, or global states are ever exchanged, thereby eliminating the synchronization bottleneck that plagues many MADRL systems.

A detailed MDP specification, network architecture, and the pseudocode of the learning algorithm follow in next subsections; here we merely highlight the highlevel workflow:

1) *Offline (edge cloud):* Train actor–critic pairs with MATD3 until the joint reward stabilizes; perform soft-updates of the target networks every $d$ steps and apply targetpolicy smoothing noise to mitigate overfit.
2) *Deployment (onchain):* Distribute frozen actor weights $\{\psi_n^*\}$ to all ADs. At each time slot $t$, $\text{AD}_n$ observes $s_n(t)$, evaluates $a_n(t) = \pi_n(s_n(t); \psi_n^*)$, applies (40), and executes the resulting control locally.

Overall, centralized critics learn from joint interactions while decentralized actors stay lightweight at run time. Next, we formalize the problem as an MDP by specifying states, actions, and rewards.

### D. MDP Formulation

The joint offloading–consensus control problem in (27) can be cast as a finitehorizon, discretetime *multiagent Markov decision process* (MDP) $\langle \mathbb{N}, \mathbb{S}, \mathbb{A}, \mathbb{R} \rangle$, where

- $\mathbb{N} = \{1, \ldots, N\}$ is the set of ADs (agents);
- $\mathbb{S}$ is the global state space;
- $\mathbb{A}$ is the joint action space;
- $\mathbb{R}$ maps state–action pairs to a scalar reward that mirrors the system utility.

Below we detail *observable* state variables, the *refined* continuous action vector, and a reward design that aligns exactly with the objective in (27) while embedding hard latency/energy constraints as soft penalties.

*1) State Space:* At the beginning of slot $t$, each $\text{AD}_n$ observes seven local features:

- task descriptor $s_n^{\text{task}}(t) = [s_n(t),\, c_n(t),\, \tau_n(t)]$,
- instantaneous uplink rate $s_n^{\text{chan}}(t)$,
- residual battery budget $s_n^{\text{res}}(t)$,
- edgeside CPU share $s_n^{\text{mec}}(t)$ reserved by the MEC server for $n$,
- candidate committee size $s_n^{\text{node}}(t)$ for the upcoming LDPBFT round,
- local CPU cycles $s_n^{\text{blo}}(t)$ earmarked for consensus,
- reliability score $s_n^{\text{rel}}(t) \in [0, 1]$.

The *local* state is therefore

$$s_n(t) = \left\{ s_n^{\text{task}}(t),\, s_n^{\text{chan}}(t),\, s_n^{\text{res}}(t),\, s_n^{\text{mec}}(t),\, s_n^{\text{node}}(t),\, s_n^{\text{blo}}(t),\, s_n^{\text{rel}}(t) \right\}, \tag{42}$$

and the *joint* state is $s(t) = \big(s_1(t), \ldots, s_N(t)\big) \in \mathbb{S}$.

*2) Refined Action Space:* Upon observing $s_n(t)$, agent $n$ produces a *continuous* triple

$$a_n(t) \;=\; \big[x_n(t),\, z_n(t),\, \varphi_n(t)\big], \tag{43}$$

where

- $x_n(t) \in (0, 1)$ is the finegrained offloading ratio,
- $z_n(t) \in \{0, 1\}$ indicates validator candidacy (1 if the agent volunteers, 0 otherwise),

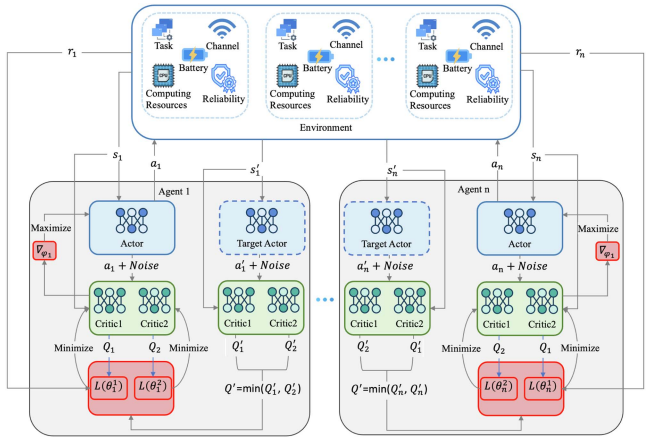



Fig. 3. MATD3 with refined action: dual critics mitigate overestimation; the offloading–consensus action head outputs $(x_n, z_n, \varphi_n)$.

- $\varphi_n(t) \in [\varphi_{\min}, \varphi_{\max}]$ is the number of transactions the agent proposes to pack when acting as a validator.

This continuous design enables smooth load shaping and quickly adapts to channel or compute fluctuations, unlike rigid binary offloading schemes.

*3) Reward Function:* To convert the socialutility maximization in (27) into individual learning signals, we define a *timestep* reward that aggregates (i) task utility for *all* ADs and (ii) consensus utility for the selected validators, then subtracts shaped penalties whenever latency or energy budgets are breached:

$$\bar{r}\big(s(t), a(t)\big) = \frac{1}{N} \sum_{n=1}^{N} J_n^{\text{task}} + \frac{1}{M} \sum_{n=1}^{N} z_n(t)\, J_n^{\text{block}}, \tag{44}$$

where the task utility $J_n^{\text{task}}$ already incorporates deadline and energy constraints through the utility shaping in Section III, ensuring monotone alignment with the original optimization objective. Maximizing the expected discounted return of (44) is therefore equivalent to maximizing social utility subject to constraints (27d) and (27e).

Overall, this reward provides a faithful perstep signal for socialutility maximization under latency/energy limits. Next, we detail centralized MATD3 training within the FECTDE framework.

### E. Centralized Training in the FECTDE Framework

During the *offline* learning phase, all ADs upload their interaction trajectories to a **shared replay buffer**. The MEC server—equipped with global observability and ample compute capacity—samples minibatches from this buffer to update *identicalarchitecture yet agentspecific* actor–critic pairs. Such centralized training lets every agent reason about the latent impact of *others'* offloading and consensus behavior, which would be impossible under purely decentralized data collection.

*1) MATD3 With Refined Action:* As illustrated in Fig. 3, we instantiate policy learning with *MultiAgent Twin Delayed Deep Deterministic Policy Gradient* (MATD3). Compared to vanilla TD3, MATD3 augments each agent with (i) a dualcritic ensemble $\big(Q_n^1, Q_n^2\big)$ to curb Qvalue bias, and (ii) a parametersharing backbone so that gradient information is propagated efficiently

**Algorithm 3:** Centralized MATD3 Training.

**Input:** replay buffer $RB$, max episodes $ME$, steps per episode $T$
**Output:** actor parameters $\psi_n^*$, hence policies $\pi_n^*$
Initialize $\theta_n^1, \theta_n^2, \psi_n$ ▷ random weights
$\theta_n^{1'} \leftarrow \theta_n^1,\ \theta_n^{2'} \leftarrow \theta_n^2,\ \psi_n' \leftarrow \psi_n$ ▷ target nets
Set $(\gamma, \sigma, c, \Gamma, d)$ and clear $RB$
**for** $e = 1$ **to** $ME$ **do**
  reset environment, obtain $s(0)$
  **for** $t = 1$ **to** $T$ **do**
    $a_n(t) \leftarrow \pi_n\big(s_n(t); \psi_n\big), \forall n$
    Execute $\{a_n(t)\}$, observe $r(t), s'(t)$
    Store $\big(s(t), a(t), r(t), s'(t)\big)$ into $RB$
    Sample minibatch $B \subset RB$
    **for all** agents $n$ **do**
      Compute $y_n$ via (45)
      Update $\theta_n^1, \theta_n^2$ by minimizing (46)
      **if** $t \bmod d = 0$ **then**
        Update $\psi_n$ using (47)
        Softupdate targets $(\theta_n^{j'}, \psi_n')$
      **end if**
    **end for**
    $s(t) \leftarrow s'(t)$
  **end for**
**end for**

across agents while still allowing individualized finetuning. The continuous triplet $x_n, z_n, \varphi_n$ from (43) serves as the *sole* output of the actor head, enabling nuanced tradeoffs between latency, energy, and consensus throughput.

*1) Critic update:* For a sampled transition $\big(s, a, r, s'\big)$, the target critic estimate is

$$y_n = r_n + \gamma \min_{j\in\{1,2\}} Q_n^{j'}\big(s', a_1', \ldots, a_N'; \theta_n^{j'}\big), \tag{45}$$

with $a_k' = \pi_k'(s_k') + \epsilon$ and $\epsilon \sim \text{clip}\big(\mathcal{N}(0,\sigma), -c, c\big)$. Each critic minimizes the meansquared Bellman error

$$\mathcal{L}\big(\theta_n^j\big) = \mathbb{E}\Big[\big(Q_n^j(s, a; \theta_n^j) - y_n\big)^2\Big], \qquad j \in \{1, 2\}. \tag{46}$$

*1) Actor update:* The deterministic policy is improved by ascending the gradient

$$\nabla_{\psi_n} J = \mathbb{E}_s\Big[\nabla_{a_n} Q_n^1\big(s, a; \theta_n^1\big)\big|_{a_n=\pi_n(s_n)} \nabla_{\psi_n} \pi_n(s_n)\Big], \tag{47}$$

where $\psi_n$ collects the actor parameters of agent $n$.

*1) Targetpolicy smoothing & delayed updates:* Target noise $\epsilon$ prevents premature overfitting to narrow deterministic actions, while *soft* Polyak averaging $\theta_n^{j'} \leftarrow \Gamma\theta_n^j + (1-\Gamma)\theta_n^{j'}$ and $\psi_n' \leftarrow \Gamma\psi_n + (1-\Gamma)\psi_n'$ with $\Gamma \in (0,1)$ stabilize learning. Actors are updated only every $d$ critic steps to mitigate destructive feedback loops.

*2) Centralized Training Algorithm:* Algorithm 3 summarizes the entire pipeline. Lines 1–6 initialize networks and the replay memory; the outer loop iterates over episodes, and the inner loop (Lines 9–28) executes environment interactions, critic fitting, policy improvement, and target synchronization. The centralized training phase exhibits computational complexity $\mathcal{O}\big(E \cdot T \cdot N \cdot B \cdot d_{\text{net}}\big)$, where $E$ denotes the number of training episodes, $T$ the steps per episode, $N$ the number of agents, $B$ the minibatch size sampled from the replay buffer, and $d_{\text{net}}$ the total network parameters (actor plus twin critics). Convergence is empirically robust provided that (i) the replay buffer is sufficiently large to break temporal correlations and (ii) the delay factor $d$ is chosen such that critics approximately track the moving target defined by actors.

**Algorithm 4:** PerSlot TaskOffloading Inference.

**Input:** local state $s_n(t)$, actor weights $\psi_n^*$
$a_n(t) \leftarrow \pi_n^*\big(s_n(t); \psi_n^*\big)$
Compute $x_n(t)$ via (48)
Execute $\big(1 - x_n(t)\big)L_n$ locally;offload $x_n(t)L_n$
Evaluate $T_n$ and $E_n$ for accountability
**Return** $x_n(t)$

Upon completion, the MEC server seals $\{\psi_n^*\}$ into onchain smart contracts. Thereafter, *no* global coordination is needed—the execution phase relies solely on each agent's private observation, guaranteeing scalability and resilience desired by blockchainenabled MEC deployments.

Overall, centralized training yields deployable actors without any runtime coordination. Next, we show how these frozen policies execute locally each slot for offloading and consensus adaptation.

### F. Decentralized Execution in the FECTDE Runtime

After centralized training (Algorithm 3) converges, the MEC server embeds the frozen actor parameters $\{\psi_n^*\}$ into onchain smart contracts. At run time, **each** AD makes decisions *locally*—no further parameter exchange or synchronization is required. This section details the two decision channels executed every time slot $t$: (i) continuous taskoffloading and (ii) lightweight consensuslayer adaptation.

*1) TaskOffloading Inference:* Each AD observes its private state $s_n(t)$ from (42) and queries its onchain actor $\pi_n^*(\cdot; \psi_n^*)$ for the latent action vector $a_n(t) = [a_{n,1}, a_{n,2}, a_{n,3}]$. The first component is mapped to a legal offloading ratio via

$$x_n(t) = \frac{\tanh\big(a_{n,1}\big) + 1}{2} \in [0, 1], \tag{48}$$

guaranteeing compliance with constraint (27a). The smart contract then splits the incoming job $L_n$ into a local part $\big(1 - x_n(t)\big)L_n$ and an edge part $x_n(t)L_n$, and computes the resulting latency $T_n$ and energy $E_n$ following (13)–(19). Because *all* environment factors (channel rate, workload size, residual battery, etc.) refresh each slot, this fully distributed policy adapts $x_n(t)$ onthefly so that the joint latency–energy utility remains high while meeting (27d) and (27e).

*2) ConsensusLayer Adaptation:* The second and third actor outputs, $a_{n,2}$ and $a_{n,3}$, govern validator selection and block size, respectively (Algorithm 5).

*Validator ranking:* A provisional score is derived from

$$b_n(t) = \tanh\big(a_{n,2}\big) \in [-1, 1], \tag{49}$$

**Algorithm 5:** PerSlot Consensus Adaptation.

1: **Input:** local state $s_n(t)$, actor weights $\psi_n^*$, desired committee size $M$
2: $a_n(t) \leftarrow \pi_n^*(s_n(t); \psi_n^*)$
3: Compute $b_n(t)$ via (49)
4: Broadcast $b_n(t)$ to elected contract, aggregate (50)
5: Derive $\varphi_n(t)$ via (51)
6: Run LDPBFT with $\mathcal{N}(t)$ and $\{\varphi_n(t)\}$
7: Verify $T^C < \tau$; otherwise trigger failsafe rollback

and the smart contract deterministically chooses the $M$ ADs with the highest $b_n(t)$:

$$\mathcal{N}(t) = \mathrm{Top}_M\{b_1(t), \ldots, b_N(t)\}, \tag{50}$$

satisfying the committeesize requirement (27f). The highestranked member becomes the primary; the remainder serve as replicas in the ensuing LDPBFT round.

*Adaptive block packing:* The pervalidator transaction budget is

$$\varphi_n(t) = \varphi_{\min} + \frac{\tanh(a_{n,3}) + 1}{2}(\varphi_{\max} - \varphi_{\min}), \tag{51}$$

which respects (27c) by construction. The selected committee then executes the LDPBFT phases ((1)–(10)); the round is deemed valid only if the observed consensus delay $T^C$ remains below the runtime limit $\tau$.

*Runtime properties:* Because every control variable $\{x_n, z_n, \varphi_n\}$ is computed from a bounded TANH projection, feasibility is *guaranteed by design*. Moreover, decisions are made in $\mathcal{O}(1)$ local time, so computational overhead on resourceconstrained ADs is negligible. Together with the proven convergence of SRAP and the stability of MATD3, this decentralized execution layer delivers scalable, fair, and resourceadaptive coordination for blockchainenabled MEC.

Overall, execution is fully local, feasible by design, and computationally lightweight on ADs. Next, we validate these properties empirically in the experimental evaluation.

## VI. Experimental Evaluation

This section addresses three core questions: how fair and incentivecompatible SRAP is, how effectively MATD3 with the refined offloading action reduces the latency–energy cost, and how much LDPBFT improves consensus throughput and utility. To that end, we build a cycleaccurate eventdriven simulator that faithfully instantiates every analytical component described in Sections III–V. Unless otherwise noted, each numerical result averages 10 independent task arrivals under identical random seeds to guarantee statistical reproducibility.

Table II summarizes the key distinguishing features of FECTDE's three core components against their respective baselines, providing a roadmap for the subsequent experimental analysis.

### A. Experimental Setup

**Due to space constraints, detailed experimental configuration—including network, workload, channel parameters, heterogeneous device profiles, consensus protocol settings, and MATD3 hyperparameters—is provided in the supplementary material (see Section V-A and Section V-B).**

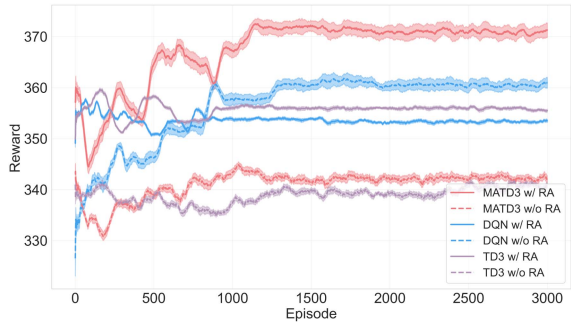



Fig. 4. Episode-averaged reward over 3,000 training episodes. Solid lines denote agents with refined continuous actions; dashed lines use binary offload actions.

### B. Effectiveness of the S-RAP Mechanism

We compare S-RAP against four baselines spanning non-strategic allocation paradigms: **Equal Allocation** (uniform distribution), **Random Allocation** (stochastic assignment), and **Fixed Payment (TK=1.0/2.0)** (fixed unit pricing). These baselines are intentionally chosen to represent common, simple-to-implement alternatives prevalent in practical MEC deployments due to their low computational overhead and ease of integration. While more advanced schemes (e.g., proportional fairness-based allocations, sophisticated heuristic methods) exist in the literature, our objective is to demonstrate the fundamental advantage of game-theoretic, incentive-compatible mechanisms over widely-adopted non-strategic methods. The selected baselines serve this purpose effectively by isolating the value of strategic pricing mechanisms: they establish a clear boundary between strategic resource management (S-RAP) and fundamental non-strategic paradigms (uniformity, randomness, fixed pricing), thereby highlighting the core benefits of incorporating game-theoretic principles without confounding the comparison with algorithm-specific engineering optimizations. We evaluate across two scenarios spanning the $(\alpha_n, p_n)$ parameter space (Section IV):

- *Scenario I (High-$\alpha_n$ [3.5, 5.0], Low-$p_n$ [0.5, 1.0])*: Emergency medical imaging—portable CT scanners offload brain scans for stroke detection where delays cause irreversible damage (high $\alpha_n$), but optimized inference models require few CPU cycles (low $p_n$).
- *Scenario II (Low-$\alpha_n$ [1.0, 2.5], High-$p_n$ [1.5, 2.0])*: Smart agriculture monitoring—sensors offload environmental data where delays minimally affect decisions (low $\alpha_n$), but high-dimensional processing and database joins demand substantial resources (high $p_n$).

Tables III and IV compare S-RAP against four baselines across efficiency and fairness metrics in both scenarios.

*1) Efficiency:* S-RAP achieves superior efficiency across heterogeneous regimes. In Scenario I, it improves server utility by up to 352.43% over Fixed(1.0) while maintaining AD utility

TABLE II
KEY ADVANTAGES OF PROPOSED METHODS OVER BASELINES

| Component | Baselines | Key Advantages of Proposed Method |
|---|---|---|
| **S-RAP** (Resource Allocation) | Equal, Random, Fixed (TK=1.0/2.0) | • Game-theoretic incentive-compatible pricing with provable Nash equilibrium<br>• Substantially lower inequality (Gini coefficient) across heterogeneous scenarios<br>• Simultaneously sustains server profitability and user participation incentives |
| **MATD3 w/ RA** (Task Offloading) | MATD3/TD3/DDPG (w/ & w/o RA) | • Multi-agent credit assignment with refined continuous action space<br>• Significant latency and energy reductions, especially under harsh regimes<br>• Consistently higher utility across diverse operating conditions |
| **LD-PBFT** (Consensus) | PBFT, GNS, HotStuff, Raft, PoA | • Agent-driven dynamic committee selection with reliability awareness<br>• Superior consensus utility while maintaining competitive throughput<br>• Completely avoids unreliable nodes that greedy methods systematically select |

TABLE III
PERFORMANCE COMPARISON FOR SCENARIO I (HIGH-$\alpha_n$ LOW-$p_n$): CRITICAL EDGE AI INFERENCE TASKS

| Metric | S-RAP | Equal | Random | Fixed (1.0) | Fixed (2.0) |
|---|---|---|---|---|---|
| MEC utility | 267.21 | 266.89 | 221.67 | 59.05 | 159.05 |
| Average AD utility | 2.50 | 2.50 | 2.37 | 4.58 | 3.58 |
| Social welfare | 517.18 | 516.89 | 459.15 | 517.17 | 517.17 |
| Gini coefficient | 0.0122 | 0.0200 | 0.3045 | 0.0121 | 0.0121 |
| Payment std. dev. | 0.39 | 0.37 | 0.94 | 0.00 | 0.00 |
| Jain fairness index | 0.96 | 0.96 | 0.73 | 0.96 | 0.94 |
| AD utility std. dev. | 0.52 | 0.53 | 1.45 | 0.89 | 0.89 |

TABLE IV
PERFORMANCE COMPARISON FOR SCENARIO II (LOW-$\alpha_n$ HIGH-$p_n$): NON-CRITICAL BATCH DATA PROCESSING

| Metric | S-RAP | Equal | Random | Fixed (1.0) | Fixed (2.0) |
|---|---|---|---|---|---|
| MEC utility | 58.21 | 57.48 | 42.31 | 78.55 | 178.55 |
| Average AD utility | 0.27 | 0.26 | 0.27 | 0.07 | -0.93 |
| Social welfare | 85.52 | 83.61 | 68.82 | 85.52 | 85.52 |
| Gini coefficient | 0.0581 | 0.0200 | 0.3044 | 0.0583 | 0.0583 |
| Payment std. dev. | 0.25 | 0.20 | 0.33 | 0.00 | 0.00 |
| Jain fairness index | 0.85 | 0.94 | 0.64 | 0.04 | 0.87 |
| AD utility std. dev. | 0.11 | 0.07 | 0.20 | 0.36 | 0.36 |

(2.50) competitive with Equal. In Scenario II, S-RAP increases server profit by 37.59% over Random while preserving positive AD utility (0.27), unlike Fixed(2.0) which drives users to negative returns ($-0.93$). Social welfare remains near-optimal in both scenarios (517.18 and 85.52), outperforming Equal and Random without parametric retuning. Critically, S-RAP is the only mechanism sustaining server profitability and user rationality simultaneously.

*2) Fairness:* S-RAP demonstrates exceptional fairness under diverse configurations. In Scenario I, Gini drops to 0.0122 (95.99% reduction vs. Random), while Jain index reaches 0.96. In Scenario II, Gini remains 0.0581 (80.92% reduction vs. Random) with Jain=0.85, preventing Fixed(1.0)'s catastrophic collapse (Jain=0.04). Payment standard deviation (0.39 in Scenario I, 0.25 in Scenario II) confirms differentiated pricing that adapts to heterogeneous valuations without triggering attrition. These metrics translate to tangible benefits: in Scenario I (Emergency Medical Imaging), low Gini (0.0122) ensures equitable stroke-detection service across all ambulances; in Scenario II (Smart Agriculture), progressive pricing prevents resource monopolization by corporate farms while maintaining affordable rates for small cooperatives.

*Takeaway:* SRAP delivers nearoptimal social welfare and superior fairness across both critical high-$\alpha_n$ tasks and non-critical low-$\alpha_n$ workloads, *without requiring scenario-specific recalibration*. It is the only mechanism retaining server profitability, user rationality, and equitable resource distribution simultaneously under parameter heterogeneity—properties essential for real-world blockchain-enabled MEC deployments.

### C. Robustness and Scalability Analysis of S-RAP

To rigorously evaluate the robustness of the S-RAP mechanism, we conduct three comprehensive experiments: Distributional Fairness analysis, Scalability With AD Count evaluation, and Sensitivity to Economic Parameters assessment. **Due to space constraints, detailed experimental results are presented in the supplementary material (see Section V-C).**

### D. Convergence Behavior of the Learning Agents

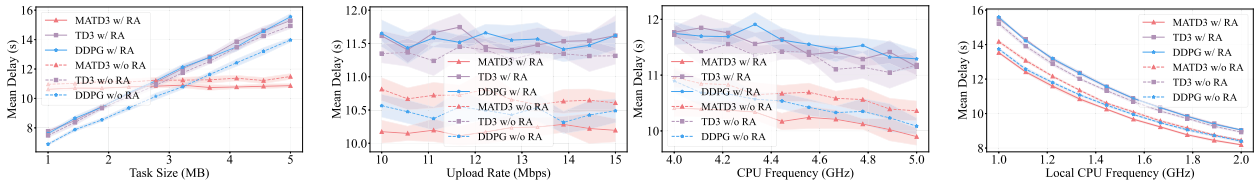
We now scrutinize the learning dynamics of the six candidate agents: MATD3, TD3, and DQN, each *with* and *without* the refined continuousaction design introduced in Section V-D. All variants share an identical threelayer multilayer perceptron backbone.

*Baselines:*

1) *Deep Deterministic Policy Gradient (DDPG)*—a deterministic policygradient agent that directly maps states to actions.
2) *Twin Delayed DDPG (TD3)*—a deterministic policygradient agent with the same twincritic safety net as MATD3 but without the multiagent credit assignment.

Each baseline is evaluated with the original binaryoffload action set *and* with the proposed refined action space for a fair contrast.

*Reward trajectory:* Fig. 4 plots the episodeaveraged cumulative reward versus training iterations, where solid lines represent the mean over 10 independent runs, shaded regions indicate 95% confidence intervals, and curves are smoothed using a sliding window filter to enhance visual clarity. MATD3 w/ RA achieves the highest asymptotic reward among all evaluated algorithms, demonstrating superior learning performance. To quantify convergence rigorously, we define an agent as converged when the standard deviation of average rewards within a 500-episode sliding window falls below threshold 1.0. Under this criterion,



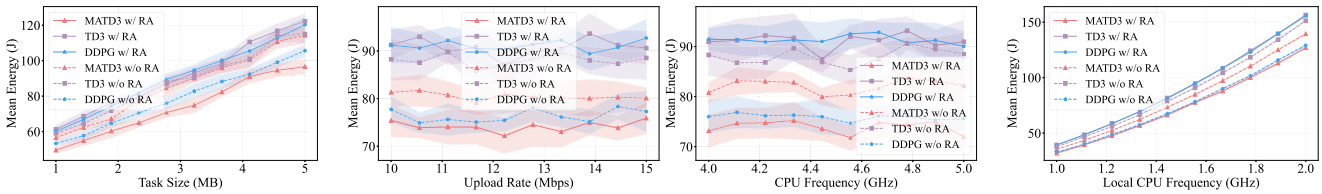

(a) The average latency under task sizes of 1-5MB

(b) The average latency under transmission rates of 1-5MB/s

(c) The average latency under MEC computing resources of 3-5GHz

(d) The average latency under ADs computing resources of 2-4GHz

Fig. 5. Average Latency Comparison with Increasing Task Size, Transmission Rate, MEC Computing Resources, and ADs Computing Resources.




(a) The average energy consumption under task sizes of 1-5MB

(b) The average energy consumption under transmission rates of 1-5MB/s

(c) The average energy consumption under MEC computing resources of 3-5GHz

(d) The average energy consumption under ADs computing resources of 2-4GHz

Fig. 6. Average Energy Consumption Comparison with Increasing Task Size, Transmission Rate, MEC Computing Resources, and ADs Computing Resources.

the convergence episodes are: MATD3 w/ RA (1558 episodes), MATD3 w/o RA (1241 episodes), TD3 w/ RA (1331 episodes), TD3 w/o RA (1500 episodes), DDPG w/ RA (1068 episodes), and DDPG w/o RA (1468 episodes). Notably, while DDPG w/ RA converges fastest, MATD3 w/ RA achieves substantially higher final reward, validating the benefit of multi-agent credit assignment despite longer training duration.

*Training Time Analysis:* The wall-clock training times on an Intel i9-13900 K workstation are: MATD3 w/ RA (43 min), MATD3 w/o RA (39 min), TD3 w/ RA (27 min), TD3 w/o RA (26 min), DDPG w/ RA (24 min), and DDPG w/o RA (20 min). While MATD3 w/ RA requires approximately 2.15× longer training than the fastest baseline (DDPG w/o RA), this overhead is acceptable given that training occurs only once offline during initial system configuration. The one-time 43-minute training cost is negligible when amortized across millions of operational inference decisions over the system's deployment lifetime.

*Takeaway:* The asymptotic reward of MATD3 + refined establish it as the most promising candidate for deployment in FECTDE. Subsequent experiments therefore adopt this agent configuration for all taskoffloading and consensus evaluations.

### *E. Task–Offloading Performance Under Varying Operating Conditions*

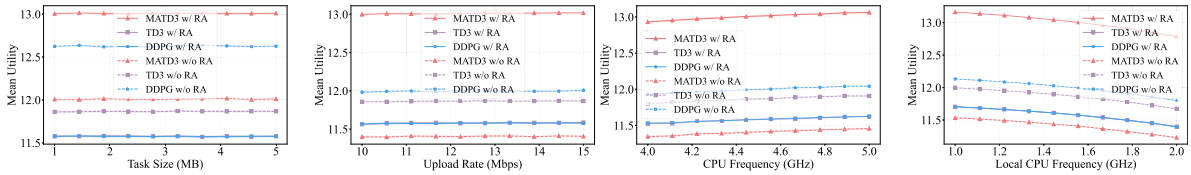
To gauge the realtime efficacy of the **MATD3** *with* **Refined Actions** (denoted "MATD3 w/RA"), we sweep four key environmental factors—task size, uplink rate, MEC capacity, and AD CPU budget—and contrast the resulting *latency*, *energy*, and *system utility* against five strong baselines: MATD3 without refined actions, TD3 with/without refined actions, and DDPG with/without refined actions. Each experiment uses the converged weights from Section VI-D; every data point averages 10 MonteCarlo runs. The lines represent the mean values, while the shaded areas indicate 95% confidence intervals.

*1) EndtoEnd Latency:* Fig. 5 reports mean job completion delay across four operating conditions. MATD3 w/ RA achieves **7.61%–22.23%** latency reduction versus singleagent baselines when task size exceeds 3.5 MB (Fig. 5(a)), **2.53%–11.67%** reduction across uplink rates of 1–5 MB/s (Fig. 5(b)), **2.45%–11.79%** reduction under varying MEC capacity (Fig. 5(c)), and **2.53%–11.67%** reduction across AD CPU budgets (Fig. 5(d)). Removing refined actions increases latency by **3.05%–5.92%**. Stability analysis confirms robust performance with CV ranges of **0.57%–3.11%** across all conditions.

*2) Radio and Compute Energy:* Fig. 6 shows energy consumption under four conditions. MATD3 w/RA achieves **6.44%–19.02%** (task size), **2.85%–18.59%** (uplink rate), **2.61%–19.06%** (MEC capacity), and **1.81%–18.73%** (AD CPU) energy savings versus baselines, with **8.01%–16.99%** reductions over MATD3 w/o RA. CV ranges of **0.66%–8.94%** confirm robust energy efficiency across diverse operational regimes. **Detailed device-specific energy analysis is provided in Section V-D1 of the supplementary material.**

*3) Overall Offloading Utility:* The composite utility in Fig. 7 is computed with (24); higher values are better. MATD3 w/RA achieves utility gains of **3.02%–114.07%** (task size), **8.46%–12.38%** (uplink rate), **8.44%–12.33%** (MEC capacity), and **8.42%–14.02%** (AD CPU) versus baselines, with **4.08%–14.11%** improvements over MATD3 w/o RA. Exceptionally low CVs (**0.01%–0.17%**) confirm consistent utility across all operational regimes.

To comprehensively evaluate the task offloading performance, we further supplement the analysis with **System Overhead Analysis**, **Computational Complexity Analysis**, **Communication Overhead for Model Deployment**, **Offloading Success Rate Validation**, and **Stress Tests** under extreme operational conditions. **Due to space constraints, these detailed evaluations are provided in the supplementary material (see Section V-D3, V-D4, V-D5, V-E).**




(a) The average system utility under task sizes of 1-5MB

(b) The average system utility under transmission rates of 1-5MB/s

(c) The average system utility under MEC computing resources of 3-5GHz

(d) The average system utility under ADs computing resources of 2-4GHz

Fig. 7. System utility comparison with increasing task size, transmission rate, MEC computing resources, and ADs computing resources.

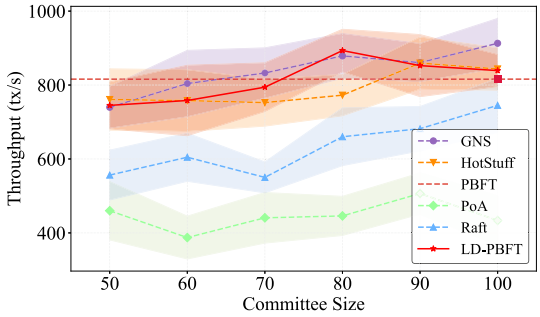



Fig. 8. Consensus throughput vs. committee size ($M = 50$–$100$).

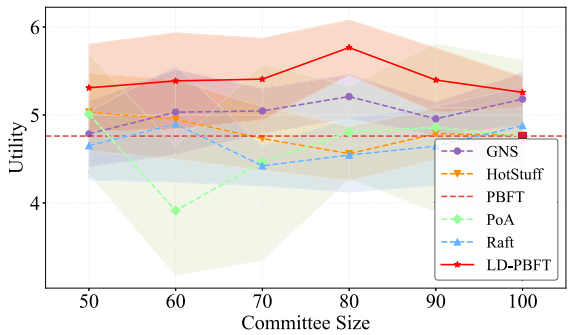



Fig. 9. Consensus utility vs. committee size ($M = 50$–$100$).

*Takeaway:* Across all four knobs, refined continuous actions and the multicritic, multiagent architecture jointly underpin the consistent wins of MATD3 w/ RA. Notably, its gains are most pronounced in harsh regimes—large tasks, congested channels, or limited edge compute— where intelligent partial offloading is essential.

### F. ConsensusLayer Performance

We next examine the effectiveness of the proposed LD-PBFT under varying committee sizes $M$. We benchmark LD-PBFT against five representative consensus mechanisms: **Greedy Node Selection (GNS)**, **vanilla PBFT**, **Proof of Authority (PoA)**, **Raft**, and **HotStuff**, which collectively span resource-greedy heuristics, classical Byzantine Fault Tolerance (BFT) protocols, and advanced consensus variants.

*1) Confirmed–Transaction Throughput:* Fig. 8 shows throughput across committee sizes (50–100 nodes). While GNS achieves peak throughput via greedy resource selection, LD-PBFT's adaptive packing delivers **15.62%** gains over HotStuff, **25.02%–44.36%** over Raft, and **61.95%–100.30%** over PoA once committee size exceeds 70. LD-PBFT exhibits CV of **11.89%**, confirming superior stability versus PBFT (10.36%), GNS (11.03%), HotStuff (12.29%), PoA (20.04%), and Raft (14.23%).

*TPS–Latency Trade-off Analysis:* Larger block size $\varphi$ boosts TPS but elevates verification overhead (proportional to $\varphi\beta_{\text{sig}}^{\text{ver}}$ per (1)), prolonging consensus delay $T^C$. Static PBFT enforces fixed $\varphi$: oversized blocks violate timeout $\tau$ (triggering penalty $P_n^{\text{con}}$ in (25)), while undersized blocks waste capacity. LD-PBFT dynamically tunes $\varphi$ via (26), balancing reward ($\varphi_n/\varphi_{\max}$) and time-penalty terms by adapting to real-time $r_n$, $k_n$, and network conditions. This state-dependent policy achieves competitive throughput (Fig. 8) and superior utility (Fig. 9), whereas GNS sacrifices reliability for peak TPS and vanilla PBFT cannot exploit resource heterogeneity.

*2) Consensus Utility Analysis:* Fig. 9 shows consensus utility across committee sizes (50–100 nodes). LD-PBFT's agent-driven selection and adaptive packing achieve **5.49%–37.70%** utility gains over all baselines while ensuring reliability. Stability analysis reveals CVs of 10.17% (LD-PBFT), 12.31% (PBFT), 8.65% (GNS), 10.43% (HotStuff), 24.70% (PoA), and 12.83% (Raft), confirming LD-PBFT's robustness. **Supplementary evaluations (validator convergence, fault injection) are in Sections F-1, F-2 of the supplementary material.**

*Takeaway:* LD-PBFT achieves superior consensus performance via agent-driven committee adaptation and block packing, delivering highest utility and substantial throughput gains over baselines.

### G. Ablation Study

**Due to space constraints, detailed ablation-study validations are provided in Section V-G of the supplementary material.**

## VII. Conclusion and Future Directions

This paper presents **FE-CTDE**, a unified framework integrating game-theoretic pricing (S-RAP), multi-agent reinforcement learning (MATD3), and adaptive consensus (LD-PBFT) for blockchain-enabled MEC. Experiments validate: *(i)* economic fairness with Gini $<$0.06; *(ii)* up to 22% latency and 19% energy reductions; *(iii)* 16%–100% consensus throughput gains. Lightweight deployment (67 $\mu$s inference, 140 KB model) confirms practical viability. By decoupling centralized training from

decentralized execution, FE-CTDE enables scalable, incentive-compatible coordination for heterogeneous edge environments.

Future extensions include three directions:

*Security Hardening:* Validator election remains vulnerable to resource fabrication and Sybil attacks. Future work should integrate TEE-based attestation, reputation tracking, and stake-weighted identity binding. S-RAP's truthfulness can be strengthened via Bayesian mechanism design and cryptographic verification.

*Scalability:* Large-scale deployment requires hierarchical MATD3 across regional clusters, model compression for IoT gateways, and validation on 5G URLLC testbeds under realistic mobility scenarios, complemented by adaptive mechanisms to handle explicit packet loss models (e.g., Gilbert-Elliott channels) in both learning and consensus layers.

*Expressiveness:* Extensions to DAG-structured workflows demand GNN-based critics. The current framework adopts a simplified linear scalarization of task and consensus utilities with equal weights—a pragmatic choice for initial deployment but inherently limited to a single operating point on the Pareto frontier. Multi-objective RL techniques (e.g., Pareto-conditioned value networks) represent a promising future direction to expose the full efficiency–throughput trade-off surface, enabling operators to dynamically shift priorities without retraining. Adaptive committee sizing based on workload intensity would further enhance flexibility.

## REFERENCES


[1] A. R. M. Forkan et al., "AIoT-CitySense: AI and IoT-driven city-scale sensing for roadside infrastructure maintenance," *Data Sci. Eng.*, vol. 9, no. 1, pp. 26–40, Mar. 2024.

[2] P. N. Thanh and M.-Y. Cho, "Advanced aiot for failure classification of industrial diesel generators based hybrid deep learning CNN-BiLSTM algorithm," *Adv. Eng. Inf.*, vol. 62, 2024, Art. no. 102644.

[3] P. Chen, Y. Li, H. Wu, and J. Zhang, "A deep learning-based reverse auction mechanism for semantic communication in iov crowdsensing services," *Comput. Netw.*, vol. 271, 2025, Art. no. 111643.

[4] E. Maleki, W. Ma, L. Mashayekhy, and H. La Roche, "QoS-aware advanced engineering informaticscontent delivery in 5G-enabled edge computing: Learning-based approaches," *IEEE Trans. Mobile Comput.*, vol. 23, no. 10, pp. 9324–9336, Oct. 2024.

[5] W. Lan et al., "Security-sensitive task offloading in integrated satellite-terrestrial networks," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 2220–2233, Mar. 2025.

[6] B. Liu, H. Tian, Z. Shen, Y. Xu, and W. Dou, "A consortium blockchain-based edge task offloading method for connected autonomous vehicles," *ACM Trans. Auton. Adapt. Syst.*, vol. 20, no. 3, Sep. 2025, Art. no. 27.

[7] Z. Zhou, H. Liao, B. Gu, S. Mumtaz, and J. Rodriguez, "Resource sharing and task offloading in IoT fog computing: A contract-learning approach," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 3, pp. 227–240, Jun. 2020.

[8] A. Heidari, M. A. Jabraeil Jamali, N. Jafari Navimipour, and S. Akbarpour, "Internet of Things offloading: Ongoing issues, opportunities, and future challenges," *Int. J. Commun. Syst.*, vol. 33, no. 14, 2020, Art. no. e4474.

[9] S. Wu, N. Chen, G. Wen, L. Xu, P. Zhang, and H. Zhu, "Virtual network embedding for task offloading in IIoT: A DRL-assisted federated learning scheme," *IEEE Trans. Ind. Inf.*, vol. 20, no. 4, pp. 6814–6824, Apr. 2024.

[10] G. Sun et al., "Joint task offloading and resource allocation in aerial-terrestrial UAV networks with edge and fog computing for post-disaster rescue," *IEEE Trans. Mobile Comput.*, vol. 23, no. 9, pp. 8582–8600, Sep. 2024.

[11] J. Zhang, P. Yang, J. Cui, L. Wei, and H. Zhong, "Improved PBFT consensus based on reputation system in vehicle network," in *Proc. IEEE/ACM Int. Conf. Big Data Comput., Appl. Technol.*, 2024, pp. 366–371.

[12] Y. Wang, J. Zhu, H. Huang, and F. Xiao, "Bi-objective ant colony optimization for trajectory planning and task offloading in UAV-assisted MEC systems," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 12360–12377, Dec. 2024.

[13] Y. Sun, Z. Wu, K. Meng, and Y. Zheng, "Vehicular task offloading and job scheduling method based on cloud-edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 14651–14662, Dec. 2023.

[14] M. Guo, X. Hu, Y. Chen, Y. Yang, L. Zhang, and L. Chen, "Joint scheduling and offloading schemes for multiple interdependent computation tasks in mobile edge computing," *IEEE Internet Things J.*, vol. 11, no. 4, pp. 5718–5730, Feb. 2024.

[15] Z. Zhang and F. Zeng, "Efficient task allocation for computation offloading in vehicular edge computing," *IEEE Internet Things J.*, vol. 10, no. 6, pp. 5595–5606, Mar. 2023.

[16] K. Tan, L. Feng, G. Dán, and M. Törngren, "Decentralized convex optimization for joint task offloading and resource allocation of vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 71, no. 12, pp. 13226–13241, Dec. 2022.

[17] J. Zhang, P. Chen, X. Yang, H. Wu, and W. Li, "An optimal reverse affine maximizer auction mechanism for task allocation in mobile crowdsensing," *IEEE Trans. Mobile Comput.*, vol. 24, no. 8, pp. 7475–7488, Aug. 2025.

[18] J. Zhang, Y. Zhang, H. Wu, and W. Li, "An ordered submodularity-based budget-feasible mechanism for opportunistic mobile crowdsensing task allocation and pricing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1278–1294, Feb. 2024.

[19] M. Tang and V. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, 2022.

[20] A. Gao et al., "Task offloading and energy optimization in hybrid UAV-assisted mobile edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 73, no. 8, pp. 12052–12066, Aug. 2024.

[21] X. Jiao et al., "Sic-enabled intelligent online task concurrent offloading for wireless powered MEC," *IEEE Internet Things J.*, vol. 11, no. 12, pp. 22684–22696, Nov. 2024.

[22] J. Chi, X. Zhou, F. Xiao, Y. Lim, and T. Qiu, "Task offloading via prioritized experience-based double dueling DQN in edge-assisted IIoT," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 14575–14591, Dec. 2024.

[23] X. Chen, S. Hu, C. Yu, Z. Chen, and G. Min, "Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 35, no. 3, pp. 391–404, Mar. 2024.

[24] W. Fan, "Blockchain-secured task offloading and resource allocation for cloud-edge-end cooperative networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 8, pp. 8092–8110, Aug. 2024.

[25] X. Chen, J. Cao, Y. Sahni, M. Zhang, Z. Liang, and L. Yang, "Mobility-aware dependent task offloading in edge computing: A digital twin-assisted reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 24, no. 4, pp. 2979–2994, Apr. 2025.

[26] A. Heidari, M. A. J. Jamali, N. J. Navimipour, and S. Akbarpour, "A QoS-aware technique for computation offloading in IoT-edge platforms using a convolutional neural network and Markov decision process," *It Prof.*, vol. 25, no. 1, pp. 24–39, 2023.

[27] A. Heidari, N. J. Navimipour, M. A. J. Jamali, and S. Akbarpour, "A hybrid approach for latency and battery lifetime optimization in IoT devices through offloading and CNN learning," *Sustain. Comput.: Inform. Syst.*, vol. 39, 2023, Art. no. 100899.

[28] L. Qin, H. Lu, Y. Chen, B. Chong, and F. Wu, "Toward decentralized task offloading and resource allocation in user-centric MEC," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11807–11823, Dec. 2024.

[29] D. Wei, J. Zhang, M. Shojafar, S. Kumari, N. Xi, and J. Ma, "Privacy-aware multiagent deep reinforcement learning for task offloading in VANET," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 11, pp. 13108–13122, Nov. 2023.

[30] J. Cai, H. Fu, and Y. Liu, "Multitask multiobjective deep reinforcement learning-based computation offloading method for industrial Internet of Things," *IEEE Internet Things J.*, vol. 10, no. 2, pp. 1848–1859, Feb. 2023.

[31] A. Suzuki, M. Kobayashi, and E. Oki, "Multi-agent deep reinforcement learning for cooperative computing offloading and route optimization in multi cloud-edge networks," *IEEE Trans. Netw. Serv. Manage.*, vol. 20, no. 4, pp. 4416–4434, Dec. 2023.

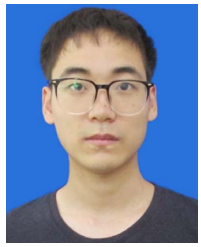
[32] A. M. Seid, J. Lu, H. N. Abishu, and T. A. Ayall, "Blockchain-enabled task offloading with energy harvesting in multi-UAV-assisted IoT networks: A multi-agent DRL approach," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 12, pp. 3517–3532, Dec. 2022.

[33] B. Lin, X. Chen, X. Chen, Y. Ma, and N. Xiong, "SGCS: An intelligent Stackelberg-game-based computation offloading and resource pricing scheme in blockchain-enabled MEC for IIoT," *IEEE Internet Things J.*, vol. 11, no. 16, pp. 26727–26740, Aug. 2024.
[34] A. Samy, I. Elgendy, H. Yu, W. Zhang, and H. Zhang, "Secure task offloading in blockchain-enabled mobile edge computing with deep reinforcement learning," *IEEE Trans. Netw. Serv. Manage.*, vol. 19, no. 4, pp. 4872–4887, Dec. 2022.
[35] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.
[36] Y. Zuo, S. Jin, S. Zhang, Y. Han, and K.-K. Wong, "Delay-limited computation offloading for MEC-assisted mobile blockchain networks," *IEEE Trans. Commun.*, vol. 69, no. 12, pp. 8569–8584, Dec. 2021.
[37] D. Wang, Y. Jia, M. Dong, K. Ota, and L. Liang, "Blockchain-integrated UAV-assisted mobile edge computing: Trajectory planning and resource allocation," *IEEE Trans. Veh. Technol.*, vol. 73, no. 1, pp. 1263–1275, Jan. 2024.
[38] A. Heidari, N. Jafari Navimipour, M. A. Jabraeil Jamali, and S. Akbarpour, "Securing and optimizing IoT offloading with blockchain and deep reinforcement learning in multi-user environments," *Wireless Netw.*, vol. 31, no. 4, pp. 3255–3276, 2025.
[39] S. Wang, H. Sheng, Y. Zhang, D. Yang, J. Shen, and R. Chen, "Blockchain-empowered distributed multicamera multitarget tracking in edge computing," *IEEE Trans. Ind. Inf.*, vol. 20, no. 1, pp. 369–379, Jan. 2024.
[40] Y. Li, J. Shen, S. Ji, and Y.-H. Lai, "Blockchain-based data integrity verification scheme in AIoT cloud–edge computing environment," *IEEE Trans. Eng. Manage.*, vol. 71, pp. 12556–12565, 2024.
[41] X. Han, D. Tian, J. Zhou, X. Duan, Z. Sheng, and V. Leung, "Privacy-preserving proxy re-encryption with decentralized trust management for MEC-empowered VANETs," *IEEE Trans. Intell. Veh.*, vol. 8, no. 8, pp. 4105–4119, Aug. 2023.
[42] X. Lan, X. Tang, R. Zhang, W. Lin, and Z. Han, "UAV-assisted computation offloading toward energy-efficient blockchain operations in Internet of Things," *IEEE Wireless Commun. Lett.*, vol. 12, no. 8, pp. 1469–1473, Aug. 2023.
[43] J. Li, M. Zhu, J. Liu, W. Liu, B. Huang, and R. Liu, "Blockchain-based reliable task offloading framework for edge-cloud cooperative workflows in IoMT," *Inf. Sci.*, vol. 668, 2024, Art. no. 120530.
[44] B. Sellami, A. Hakiri, and S. B. Yahia, "Deep reinforcement learning for energy-aware task offloading in join SDN-blockchain 5G massive IoT edge network," *Future Gener. Comput. Syst.*, vol. 137, pp. 363–379, 2022.
[45] S. Tian, Y. Zhang, Y. Bi, and T. Yuan, "Blockchain-based 6G task offloading and cooperative computing resource allocation study," *J. Cloud Comput.*, vol. 13, no. 1, 2024, Art. no. 95.
[46] J. Du, W. Cheng, and S. Li, "Joint task offloading and resource allocation in mixed edge/cloud computing and blockchain empowered device-free sensing systems," *Comput. Commun.*, vol. 209, pp. 38–46, 2023.
[47] C. Li et al., "Blockchain enabled task offloading based on edge cooperation in the digital twin vehicular edge network," *J. Cloud Comput.*, vol. 12, no. 1, 2023, Art. no. 120.

**Libo Feng** received the PhD degree from the School of Computer Science and Engineering, Beihang University, China, in 2019. He is currently a assistant professor with the School of Software, Yunnan University. His research interests include blockchain technology and application, network security, and the Internet of Things technology.

**Chenxi Wang** received the BS degree from the Henan University of Technology, zhengzhou, China, in 2023. He is currently working toward the MS degree in software engineering in Yunnan University. His current research interests include blockchain technology and information security.

**Zhenli He** (Senior Member, IEEE) received the BS degree in software engineering from Yunnan University, Kunming, China, in 2010, and the MS and PhD degrees in software engineering and systems analysis and integration, respectively, from the same university, in 2012 and 2015. From 2019 to 2021, he was a postdoctoral researcher with Hunan University, Changsha, China. He is currently an associate professor and head of the Department of Software Engineering, School of Software, Yunnan University. Dr. He was selected as a *Young Talent of the Yunnan Provincial "Xingdian Talent Support Program"* and is a recipient of the *Hongyun Gardener Award* for teaching excellence. His research interests include edge computing, energy-efficient computing, heterogeneous computing, and edge-AI techniques.



**Mengzhuang Liu** received the BS degree in software engineering from Henan Polytechnic University, in 2023. He is currently working toward the MS degree in electronic information engineering with the School of Software, Yunnan University. His research focuses on blockchain-related technologies.

**Jixian Zhang** received the MS and the PhD degrees in computer science from the University of Electronic Science and Technology of China, in 2006 and 2010, respectively. Currently, he is an associate professor in the School of Computer Science and Engineering at Yunnan University. He has published 50+ articles in peer-reviewed journals and conference proceedings, including *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Network and Service Management*, *IEEE Transactions on Network Science and Engineering*, FGCS, JOGC, Cluster Computing and Cloud Computing. His research interests include cloud computing, edge computing, and mechanism design.

**Keqin Li** (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, in 1985 and the PhD degree in computer science from the University of Houston, in 1990. He is a SUNY distinguished professor with the State University of New York and a National Distinguished Professor with Hunan University (China). He has authored or co-authored more than 1280 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top few most influential scientists in parallel and distributed computing, regarding single-year impact (ranked #2) and career-long impact (ranked #3) based on a composite indicator of the Scopus citation database. He is listed in *Scilit Top Cited Scholars* and is among the top 0.02% out of more than 20 million scholars worldwide based on top-cited publications in the last ten years. He is listed in *ScholarGPS Highly Ranked Scholars* and is among the top 0.002% out of more than 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He is a member of Academia Europaea (MAE), a Member of the European Academy of Sciences and Arts (EASA), a Fellow of the american association for the Advancement of Science (AAAS), a Member of the USA National Academy of Artificial Intelligence (NAAI), a Fellow of the International Academy of Artificial Intelligence Sciences (AAIS), a Fellow of the International Artificial Intelligence Industry Alliance (AIIA), a Member of the World Academy of Sciences (WAS), a Fellow of the World Academy of Engineering & Technology (WAET), a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), a Fellow of the Asia Computational Intelligence Society (ACIS), and a Member of the SUNY Distinguished Academy.