I. J. Computer Network and Information Security, **2026, 2, 19-37**
Published Online on April 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2026.02.02

Three-dimensional Decision Task Offloading Model in Mobile Edge Computing

Van Long Nguyen Huu*
Can Tho University, Can Tho, Vietnam
E-mail: nhvlong@ctu.edu.vn
ORCID iD: https://orcid.org/0000-0001-8566-4137
*Corresponding author

An. Cong Tran
Can Tho University, Can Tho, Vietnam
E-mail: tcan@ctu.edu.vn
ORCID iD: https://orcid.org/0000-0001-6184-909X

Received: 09 October 2025; Revised: 18 December 2025; Accepted: 20 January 2026; Published: 08 April 2026

Abstract: Mobile Edge Computing (MEC) mitigates cloud computing systems' latency and limited responsiveness by offloading computationally intensive tasks from user devices to nearby Edge Servers (ESs). However, achieving efficient offloading under dynamic mobility, fluctuating link quality, and constrained resources remains a significant challenge. To address this, we propose MSQ, a lightweight and adaptive three-dimensional decision offloading model that jointly incorporates *Mobility*, *Sociality*, and *QoS awareness*. MSQ employs Kalman filtering for mobility prediction, Rényi entropy to quantify social affinity among mobile users, and Affinity Propagation (AP) clustering to reduce redundant ES candidates while balancing computational load. Comprehensive experiments across small and medium-scale MEC networks demonstrate that MSQ reduces average task delay by up to 78%, energy consumption by 66%, and load imbalance by 64% compared with a random offloading strategy while having decision latency below one millisecond. Moreover, MSQ lowers the 95th-99th percentile tail delays by 35-45%, ensuring smoother and more reliable user experience in real- time applications. These results confirm that MSQ offers a scalable, low-latency, and energy-efficient offloading decision suitable for dynamic and intelligent edge systems.

Index Terms: Mobile Edge Computing, Task Offloading, Mobility Prediction, Social Scoring, QoS Optimization.

1. Introduction

In recent years, the rapid growth of 5G has increased data and computation-intensive applications, including ultra-high definition video streaming, cloud gaming, and augmented/virtual reality experiences [1]. These applications require high computational resources, which often exceed the capabilities of mobile devices [2]. To alleviate these limitations, cloud computing enables remote server offloading that improves the performance experience and reduces the energy consumption of mobile devices [3]. However, when the number of users or tasks is large, offloading tasks to distant cloud servers can result in high transmission latency and poor Quality of Service (QoS) [4]. Mobile Edge Computing (MEC) addresses these limitations by placing computing resources closer to users at the network edge. MEC allows mobile users (MUs) to offload tasks to nearby Edge Servers (ES), thus reducing latency, minimizing communication overhead with the cloud server, and improving service responsiveness [2]. However, efficient task offloading in the MEC context is not a trivial problem since choosing the ES with the strongest signal or the most available resources at a given time is insufficient in real-world environments [4]. Thus, task offloading from MUs to ES should consider several factors, such as MUs' mobility, ES's capability, and the history of connection status. We address these factors through their dimensions: mobility, sociality, and QoS awareness.

Mobility aware offloading. When an MU moves unpredictably, it may trigger frequent handovers between ESs, resulting in service interruptions, degraded latency performance, and even task failures [3,5]. Thus, the objective is to achieve accurate and timely mobility prediction. Light-weight techniques frequently operate in real time, provide sufficient accuracy, and remain robust under noisy or incomplete trajectory data. For example, Kalman Filtering (KF) offers a computationally efficient approach for short-term trajectory estimation, which is critical in task offloading [2].

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

In contrast, complex learning-based approaches that rely on large-scale data and higher computational requirements (i.e., time and resources) often limit their practicality in MEC even though they can capture long-term dependencies in MEC [4,6]. Therefore, we are motivated to leverage such a light-weight and adaptive approach, the KF, which can operate effectively under real-world data and system constraints, as the first dimension of offloading decision.

Sociality aware offloading. MEC offloading also depends on the social relationships among MUs, since the stability of their interactions often shapes cooperation and clustering. Strong social ties facilitate more reliable collaboration and resource sharing, while weak or irregular ties increase the risk of task disruption. Entropy-based methods have been widely applied to quantify such stability by measuring the uncertainty of associations between MUs in real time. For instance, Rényi entropy provides a flexible and interpretable metric that distinguishes between predictable and unstable contacts [7]. The main advantage of Rényi entropy is that it does not require extensive interaction data and introduces significant processing overhead, unlike learning-based approaches. Therefore, we are motivated to integrate this light-weight entropy-based sociality awareness as a second dimension into the offloading decision.

QoS aware of ES. MEC offloading decisions must also guarantee Quality of Service (QoS) requirements, such as latency, energy consumption, reliability, and service continuity [5,8]. An offloading strategy that fails to meet QoS constraints may result in poor user experience or system inefficiency (e.g., tasks missing deadlines, out of battery, or edge servers overloaded) [8]. Thus, we require a light-weight decision method to quickly evaluate trade-offs (e.g., latency vs. load, or energy vs. reliability). Such a decision component should be adaptive to changing network loads, user requirements, and resource availability. Moreover, to ensure scalability, we need an adaptive clustering (i.e., Affinity Propagation AP) step that narrows down the candidate set using prior measurements, thereby reducing signaling costs while preserving accuracy. To conclude, we are motivated to include a QoS-aware module as the third dimension of our model to ensure that offloading decisions remain aligned with service requirements under real-world constraints.

Light-weight three-dimensional decision model. Based on the above motivations, we propose MSQ, a light-weight three-dimensional offloading decision model which consists of *mobility*, *sociality*, and *QoS*. We describe the contributions as follows:

- Mobility prediction. Based on communication scenarios, we develop a KF module to predict the user's trajectory (position, velocity) in real time to decide whether to offload, migrate, or hold tasks.
- Social strength scoring. We compute the Rényi entropy to score the social strength based on past interactions to ensure MU offloads only the tasks to the preferred ES.
- Affinity Propagation AP. We implement the AP clustering algorithm to reduce the number of candidates to a handful of high-potential ESs.
- QoS aware offloading. We inject the QoS-aware utility, which consists of service ratio and channel conditions, to balance reliability, delay, and load.

The experimental results show that with a small-scale network, MSQ can cut average task completion time up to 78%, reduce energy consumption by up to 66%, and improve load balancing 64% compared with a random offloading strategy. Moreover, its 95th and 99th percentile task completion times are 35-45% faster than the baseline, reflecting fewer service interruptions and higher user perceived reliability under real-time workloads. Finally, its offloading decisions incur only millisecond-scale overhead, demonstrating low-latency performance and practical deployability in dynamic MEC scenarios.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 gives the preliminaries of each MSQ's dimension. Section 4 describes the MSQ decision model. Section 5 presents the numerical simulation results. The final section concludes this paper.

2. Related Works

2.1. Mobility Aware Strategies

Kalman Filtering (KF) has been adopted in multiple lightweight strategies for mobility prediction. Ojima and Fujii [9] employed KF as a trajectory estimator to support resource management decisions at the edge, enabling smoother handover-aware offloading. Rui et al. [10] combined KF-based mobility correction with Lyapunov optimization to reduce service discontinuities under mobility. In Vehicular Edge Computing (VEC), Deng et al. [11] proposed a hybrid method with a Cubature KF (CKF) to predict vehicular trajectories under acceleration variations better. Mehmood et al. [12] advanced this approach by embedding road-aware rectification into the KF, reducing error significantly in edge tracking scenarios. Zaki et al. [13] used KF for estimating vehicle movements and combined these estimates with quality metrics (e.g., Value of Information) to guide offloading decisions.

Other lightweight strategies are addressed in favor of optimization, online policies, and Markov-based heuristics. Jahandar et al. [14] developed online schemes for base station selection that explicitly incorporate handover cost, delay,

and energy metrics into their decision function. Xia et al. [15] adopted a game-theoretic framework (MAORAS) for joint-offloading and resource allocation in MEC. In the UAV-assisted domain, Chen et al. [16] proposed the MAVTO algorithm, which predicts vehicle positions and supports hybrid offloading modes to improve task completion rates under mobility constraints. Shokouhi et al. [17] presented a hierarchical MEC design that mitigates Virtual Machine (VM) migration via a Markov chain–based strategy. Huang and Yu [18] proposed an offloading scheme that combines mobility awareness with lightweight heuristics to balance delay and energy consumption. Wang et al. [19] designed a fault-tolerant offloading scheme using Markov chain mobility models to inform load balancing under dynamic network conditions.

Deep Learning DL has emerged as a new paradigm for mobility prediction in MEC offloading due to its ability to capture nonlinear spatio-temporal dynamics. Cui et al. [20] leveraged LSTM networks to handle noise in mobility traces, enabling more robust availability prediction for offloading decisions. Zaman et al. [21] proposed LiMPO, a neural-network-based framework that integrates mobility prediction with task offloading, demonstrating improved latency and energy efficiency. Extending this idea, they introduced COME-UP [22], which employs a Long Short Term Memory LSTM-based direction prediction module to reduce handover failures effectively. Nishad et al. [23] developed an adaptive BiLSTM with Federated Learning FDL and Deep Reinforcement Learning DRL to support energy-efficient, privacy- preserving prediction in large-scale systems. Meanwhile, Huang et al. [24] proposed a Transformer, named DDPG hybrid framework, for long-term offloading optimization in innovative city environments. Nevertheless, their reliance on large training datasets and high inference cost raises deployment challenges on resource-constrained edge devices.

Unlike the related work focusing only on mobility, our MSQ jointly integrates prediction, social context, and QoS metrics. As the first dimension of the decision model, our proposal maintains low complexity while explicitly incorporating uncertainty estimation by applying a lightweight KF for different communication scenarios. In this way, MSQ meets the requirements of short-term mobility prediction in MEC.

2.2. Socially Aware Strategies

We have seen many studies rely on encounter frequency or social graph metrics. For instance, Yang et al. [25] proposed a Top-K algorithm for task offloading in Device-to-Device D2D mobile social networks, leveraging social relationships to improve load balancing and efficiency. However, their model relies solely on encounter frequency, which may not capture deeper trust or behavioral similarities critical for reliable task offloading in sensitive applications. Wu et al. [26] proposed a Trust-Aware Task Offloading Framework (TATOF) in which they implemented Social Collaborative MEC (SCMEC) to map the trust relationships from online social networks (e.g., Weibo) to the device layer. However, since using Support Vector Machines (SVM) based trust evaluation, their proposal requires significant training effort that may not scale well on resource-constrained devices. By considering users' willingness to share devices, Gong et al. [7] developed a dependent task offloading strategy based on Bipartite Graph Matching. However, it lacks explicit social trust modeling and relies only on unspecified social relationship metrics. Teng et al. [27] proposed a user multi-factor social relationship graph-based task scheduling model for Smart City Library. However, their reliance on historical ratings and static Euclidean distance for D2D connectivity limits adaptability to dynamic social interactions. Li et al. [28] augmented offloading policies with an entropy term integrated with Reinforcement Learning RL to improve exploration and adaptivity to capture uncertainty in associations. They unfortunately did not discuss tunable sensitivity in their entropy definitions.

Nevertheless, existing social-aware schemes typically assume dense or static social graphs and rely on simple frequency counts. They also lack a mechanism to adapt sensitivity when interactions are sparse or evolving. We fill this gap by employing Rényi entropy into MSQ as a second dimension to score tie strength, which can magnify dominant interactions or emphasize diversity through its order parameter.

2.3. Joint Mobility-sociality Offloading

Few studies explicitly integrate both mobility prediction and social context. Belli et al. [29] proposed a mobility and sociality framework to enhance task offloading through Mobile Fog Edge Computing (MFEc) nodes. This framework focuses on social coverage via community detection. However, it lacks predictive accuracy in the mobility model and does not have QoS optimization while relying on computationally intensive community detection. Xu et al. [5] developed a joint selection algorithm (JSMCO) that used a directional social weight matrix and device mobility to minimize energy consumption. In particular, this approach considers each user's social relationships, location correlations, and movement activity in the context of D2D communication. While effective in reducing energy consumption, JSMCO lacks explicit consideration of QoS metrics, such as latency and reliability, which are critical for user-centric applications. Additionally, its mobility handling does not employ predictive techniques for precise trajectory estimation. For Vehicular Fog Computing (VFC), Raju et al. [30] proposed a dependency and mobility-aware intelligent task scheduling (DMITS) mechanism. This approach leverages Markov renewal processes (MRP) for vehicle mobility and social relationships based on communication patterns to optimize task success rates and energy consumption. Since it is a mixed integer nonlinear programming (MINLP) formulation and Federated Deep Reinforcement Learning (FDRL), it exhibits high computational costs and overlooks comprehensive QoS metrics. Thus, no prior work provides a lightweight unified offloading decision framework that concurrently leverages short-term mobility forecasts, adaptive social-tie scoring, and real-time QoS utilities.

While DRL-based frameworks [24,31] achieve near-optimal performance in large-scale simulations, they typically require extensive online exploration, convergence time, and centralized model coordination, making them impractical for real-time MEC environments with limited device power and intermittent connectivity. On the other hand, federated offloading (FO) [32] mitigates privacy and communication bottlenecks by training local models collaboratively. However, significant synchronization and update overhead are still incurred when user mobility is high. In contrast, we aim to adopt a lightweight prediction mechanism that integrates mobility and sociality without continuous retraining. Thus, our model can achieve faster inference and better scalability for real-time edge offloading scenarios.

3. Preliminaries

3.1. Mobility Predictions

Regarding the current position and velocity of the user, mobility prediction can estimate, at each scheduling instant, where the user will be in the next time slot. Kalman Filtering (KF) fits this role as an online predictor of short-term user motion since it requires only a few matrix-vector multiplies and one slight matrix inversion per step. More precisely, KF allows smoothing noisy position and velocity readings while predicting the user's next state in a linear Gaussian model. At each time step, it combines a prediction (i.e., Time update) and correction (i.e., Measurement update) so that the estimation remains optimal under Gaussian noise. The filter maintains a state vector (i.e., position and velocity) and an error-covariance matrix, updated as follows:

Prediction step.

$$\begin{aligned} \hat{x}_{k|k-1} &= A\hat{x}_{k-1|k-1} \\ P_{k|k-1} &= AP_{k-1|k-1}A^T + Q \end{aligned} \tag{1}$$

Before incorporating the new measurement, we present $\hat{x}_{k|k-1}$ as the predicted state vector consisting of position and velocity. The uncertainty $P_{k|k-1}$ is modeled through predicted estimate covariance. To tackle the user movement, which can be randomly varied, we use Q as the process noise convenience matrix.

Update step. We need to compute the Kalman gain matrix K at time k, which balances the trust between the prediction and measurement.

$$K_k = P_{k|k-1}H^T\left(HP_{k|k-1}H^T + R\right)^{-1} \tag{2}$$

In such computation, we use the observation matrix H to map the actual state into the measurement space. We must also model the measurement covariance matrix R to capture the sensor inaccuracies. We update or filter the state vector $\hat{x}_{k|k}$ using the measurement vector z_k based on the computed Kalman gain. Note that the difference between actual and predicted measurement, innovation or residual, is modeled as $z_k - H\hat{x}_{k|k-1}$.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\left(z_k - H\hat{x}_{k|k-1}\right) \tag{3}$$

3.2. Social Tie Strength

In MSQ, social tie strength scoring with Rényi entropy should quantify how willing a MU delegate is to work with an appropriate ES based on their historical interactions, shared context, and network structure. In detail, Rényi entropy yields values in the range $0 \le H_\alpha(P) \le \log n$ where the upper bound occurs for the uniform distribution $p_i = \frac{1}{n}$ and the lower bound of 0 when one $p_i = 1$. We use Rényi entropy because it directly measures the unpredictability of interactions between MU and ES. In fact, it introduces an order parameter α that tunes sensitivity between uniform diversity and dominant events. For a discrete distribution $P = (p_1, ..., p_n)$ with $\sum_i p_i = 1$, the Rényi entropy $H_\alpha(P)$ of order α is given by

$$H_\alpha(P) = \frac{1}{1-\alpha}\log\left(\sum_{i=1}^{n} p_i^\alpha\right), \qquad \alpha > 0, \alpha \neq 1 \tag{4}$$

Because $H_\alpha(P)$ decreases with α, choosing $\alpha > 1$ emphasizes the most significant probabilities (i.e., most substantial ties). Meanwhile, $\alpha > 1$ emphasizes diversity across many small p_i. Thus, setting the value of α allows for magnifying each user's dominant past associations with ESs. For example, if an MU spreads offloads evenly across many ESs, the entropy $H_\alpha(P)$ is high, resulting in an uncertain choice. In contrast, if MU concentrates a single ES, the entropy $H_\alpha(P)$ is low, reflecting a strong preference. In this way, we can filter out the weak candidate ESs for further investigation. Note that the probability vector P is formed by normalizing historical offload counts w_i through $p_i = \frac{w_i}{\sum_j w_j}$ and computing $H_\alpha(P)$, which is negligible compared to KF and clustering.

3.3. Candidate Clustering Methods

Clustering the set of edge servers (ESs) into a few promising candidates is essential. To achieve this goal, unsupervised clustering with Affinity Propagation (AP) is preferable since we do not need label "good" or "bad" ES groups for every possible user situation. AP starts with a similarity matrix $s(i,k)$ measuring how well point k can serve as an exemplar for point i. We form $s(i,k)$ as a weighted sum of the mobility score $p_{(i,k)}$ and social strength $SC_{i,k}$. Then, we set each diagonal preference $s(k,k)$ to the median of all off-diagonal $s(i,k)$ to present a moderate number of clusters.

Two $N \times N$ matrices for responsibilities $r(i,k)$ and availabilities $a(i,k)$ respectively, are initialized to zero and then updated iteratively. Final clusters form by each point selecting the exemplar k maximizing $r(i,k)+a(i,k)$.

$$\begin{aligned} r(i,k) &\leftarrow s(i,k) - \max_{k' \neq k} \{a(i,k') + s(i,k')\} \\ a(i,k) &\leftarrow \begin{cases} \sum_{i' \neq k} \max\{0, r(i',k)\}, & i = k \\ \min\{0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k))\}, & i \neq k \end{cases} \end{aligned} \tag{5}$$

These updates incorporate a damping factor to ensure numerical stability and iterate until convergence. We apply damping by.

$$\begin{aligned} r_{new} &= \lambda r_{old} + (1-\lambda) r_{update} \\ a_{new} &= \lambda a_{old} + (1-\lambda) a_{update} \end{aligned} \tag{6}$$

with $\lambda \in [0.5, 0.9]$. In this way, AP requires $O(N^2)$ memory for the similarity matrix and $O(N^2 T)$ time over T iterations, making it suitable for moderate numbers (tens to a few hundred) of ESs.

3.4. QoS Utility Metrics

QoS metrics are frequently presented as estimated latency, device energy cost, outage probability, link reliability, and service acceptance ratio. In MSQ, we focus on three important metrics: server load, service ratio, and connection reliability [27]. First, server load $L_j \in [0,1]$ is the fraction of j 's CPU already in use, where lower L_j implies greater spare capacity. Second, the service ratio represents ES's completed offloaded tasks n_j out of a total of N user requests.

$$\text{ServiceRatio}_j = \frac{n_j}{N} \in [0,1] \tag{7}$$

Third, wireless channel impairments can cause an offload attempt to fail when the Signal-to-Interference-plus-Noise Ratio (SINR) falls below a minimum threshold γ_{th}. The outage probability at link distance d is defined as

$$P_{\text{out}}(d) = \Pr\{\text{SINR}(d) < \gamma_{\text{th}}\} \tag{8}$$

accounting for path loss, multipath fading, and noise. Thus, we can compute the connection reliability as

$$P_{\text{conn}}(d) = 1 - P_{\text{out}}(d) \in [0,1] \tag{9}$$

By maintaining counters for n_j and N while computing $P_{\text{out}}(d)$, we note that the scaled delay or energy incurs only $O(1)$ per update. As a result, QoS metric updates are negligible compared to the prediction and clustering.

4. Model Evaluation

4.1. System and Execution Models

In our network, each MU $i \in \{1,...,N\}$ generates a sequence of independent tasks $a_{i,1},...,a_{i,K}$. Each task $a_{i,K}$ is described by three parameters: input data A_{in}, required CPU cycles A_{com}, and output data A_{out}. To satisfy the latency expectations and conserve device battery, we model delay and energy for every task using three execution modes: local, offload, and offload with migration if MU moves out of range (as can be seen in Fig. 1). Then, we use binary matrices:

$$\begin{aligned} \Lambda &= (\lambda_{i,j,k}) \in \{0,1\}^{N\times M\times K} \\ \Gamma &= (\mu_{i,j,j',k}) \in \{0,1\} \end{aligned} \tag{10}$$

to indicate offload decisions. $\lambda_{i,j,k} = 1$ if task $a_{i,k}$ of MU i goes to ES j and migration $\mu_{i,j,j',k} = 1$ if results of task $a_{i,k}$ move from ES j to ES j'. To clarify, $i = 1,...,N$ indexes mobile users, $j, j' = 1,...,M$ represents ESs, and $k = 1,...,K$ are tasks per user. Note that these indices are consistent across the exhibition of our proposals and contributions.

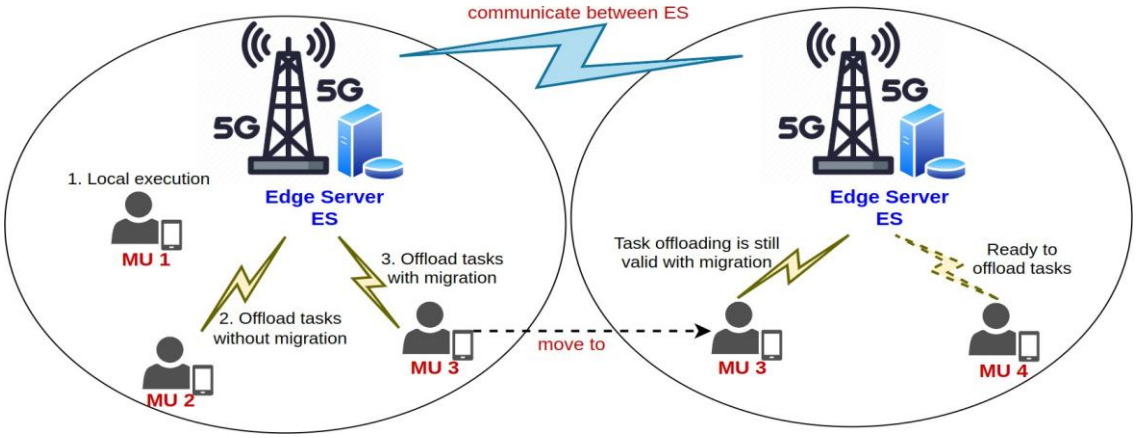


Fig.1. Three offloading paradigms: (1) Local execution, (2) One hop offload, and (3) Two or more hops offload

Feasibility constraints. Since our primary goal is to guarantee timely completion with minimal extra delay and control complexity, we thus provide two rules for our execution models:

$$\begin{aligned} \sum_{j} \lambda_{i,j,k} &\le 1, \\ \sum_{j'} \mu_{i,j,j',k} &\le 1 \end{aligned} \tag{11}$$

The former ensures each task is offloaded to at most one ES. In other words, we never allow double offloading of the same task to multiple servers. Meanwhile, the latter enforces at most one task result forwarding per offload.

Local execution. If MU i executes task $a_{i,k}$ on its CPU with the computation capability f_i^{loc} (e.g., CPU cycles per second) and working power p_i^{cpu} (active CPU power in watts), the delay $T_{i,k}^{loc}$ and energy $E_{i,k}^{loc}$ can be given:

$$\begin{aligned} T_{i,k}^{\text{loc}} &= \frac{A_{i,k}^{\text{com}}}{f_i^{\text{loc}}} \\ E_{i,k}^{\text{loc}} &= p_i^{\text{cpu}} T_{i,k}^{\text{loc}} \end{aligned} \tag{12}$$

Offload without migration. When MU i offloads its tasks to ES j, the three main phases, such as uploading, computing, and downloading, are realized through uplink and downlink between MU and ES. Such an offload decision is saved as $\lambda_{i,j,k}=1$ in the indicator matrix Λ. Moreover, we quantify the measurements which concern to uplink and downlink rate $R_{i,j}$, such as bandwidth W, channel gain $h_{i,j}$, path-loss $d_{i,j}^{\theta}$ (where θ is the path-loss exponent), noise N_0, and MU transmit power $p_{i,j}$:

$$R_{i,j}=W\log_2\left(1+\frac{p_{i,j}h_{i,j}^2}{d_{i,j}^{\theta}N_0}\right) \tag{13}$$

Based on this rate $R_{i,j}$, we can give the corresponding delays and the total energy consumption for MU i. Note that the working power p is formulated for three main phases. In particular, $p_{i,0}$ represents the local power consumed when the device CPU is idle during remote task processing at ES. Additionally, $p_{i,j}$ allows for modeling the cost of using power for transmission between MU i and ES j. To receive the results from ES, MU consumes an amount of power measured through $p_{i,R}$. Interestingly, when an ES j forwards the task output to ES j over the fiber link, the edge network carries that migration traffic and does not consume any additional radio frequency transmit or receive energy at the MU. In short, it is not necessary to consider the powers of participatory ES in our network model. Besides the powers, we also model $T_{i,j,k}^{\text{com}}$ as ES capability in which $f_{i,j,k}^{\text{sev}}$ is the CPU cycle rate at server j' for MU 's task k rather than the MU capability.

$$\begin{aligned}&T_{i,j,k}^{\text{in}}=\frac{A_{i,k}^{\text{in}}}{R_{i,j}},\quad T_{i,j,k}^{\text{com}}=\frac{A_{i,k}^{\text{com}}}{f_{i,j,k}^{\text{sev}}},\quad T_{i,j,k}^{\text{out}}=\frac{A_{i,k}^{\text{out}}}{R_{i,j}}\\&E_{i,j,k}^{\text{sev}}=p_{i,j}T_{i,j,k}^{\text{in}}+p_{i,0}T_{i,j,k}^{\text{com}}+p_{i,R}T_{i,j,k}^{\text{out}}\end{aligned} \tag{14}$$

Offload with migration. If MU i moves out of ES j 's coverage before results return, ES j forwards the output to neighbor ES j' (with the ES to ES optical fiber communication rate R_S). Thus, ES j' has the responsibility to deliver results to MU i. In this case, migration is saved as $\mu_{i,j,j',k}=1$ in the indicator matrix Γ. Finally, the extra delay and energy consumption can be calculated:

$$\begin{aligned}&T_{i,j,j',k}^{mig}=\frac{A_{i,k}^{out}}{R_s}+\frac{A_{i,k}^{out}}{R_{i,j'}}\\&E_{i,j,j',k}^{mig}=p_{i,0}\frac{A_{i,k}^{out}}{R_s}+p_{i,R}\frac{A_{i,k}^{out}}{R_{i,j'}}\end{aligned} \tag{15}$$

Total delay and energy consumption. Putting the three execution modes together, the total completion time for task $a_{i,k}$ is

$$T_{i,k}^{tot}=(1-\sum_j\lambda_{i,j,k})T_{i,k}^{loc}+\sum_j\lambda_{i,j,k}\left[(1-\sum_j\mu_{i,j,j',k})T_{i,j,k}^{sec}+\sum_{j'}\mu_{i,j,j',k}T_{i,j,j',k}^{mig+out}\right] \tag{16}$$

where

$$\begin{aligned}&T_{i,j,k}^{sec}=T_{i,j,k}^{in}+T_{i,j,k}^{com}+T_{i,j,k}^{out}\\&T_{i,j,j',k}^{mig+out}=T_{i,j,k}^{in}+T_{i,j,k}^{com}+T_{i,k}^{mig}+T_{i,j',k}^{out}\end{aligned} \tag{17}$$

Similarly, the total energy consumed by MU i for task k is

$$E_{i,k}^{tot}=(1-\sum_j\lambda_{i,j,k})E_{i,k}^{loc}+\sum_j\lambda_{i,j,k}\left[(1-\sum_j\mu_{i,j,j',k})E_{i,j,k}^{sec}+\sum_{j'}\mu_{i,j,j',k}E_{i,j,j',k}^{mig+out}\right] \tag{18}$$

with

$$E_{i,j,k}^{sec} = p_{i,j}T_{i,j,k}^{in} + p_{i,0}T_{i,j,k}^{com} + p_{i,R}T_{i,j,k}^{out}$$
$$E_{i,j,j',k}^{mig+out} = p_{i,0}T_{i,k}^{mig} + p_{i,R}T_{i,j',k}^{out} + E_{i,j,k}^{\sec} \tag{19}$$

Based on these execution modes, we can propose the joint optimization over Λ, Γ, and server allocations.

4.2. Problem Formulation

Regarding to total delay $T_{i,k}^{tot}$ and energy $E_{i,k}^{tot}$, we formulate a joint optimization problem.

$$\min_{\{\lambda,\mu,p,f\}} \sum_{i=1}^{N}\sum_{k=1}^{K} T_{i,k}^{tot} \tag{20}$$

For each MU i and task k, the objective is to decide whether to compute locally, offload to ES j, or offload with migration while minimizing the total completion time across all tasks concerning energy, resource, and capacity constraints. Such optimization could be achieved through the following investigations for all i, j, j', and k:

Energy budget. Ensuring that each MU i does not exceed its per-task energy budget $E_i^{\max}$. It precisely represents the device's battery limitations when choosing between local execution and various offload modes.

$$E_{i,k}^{tot} \le E_i^{max} \tag{21}$$

Offload choice. Ensuring each task is offloaded to at most one ES (or computed locally). Such an approach prevents duplicating a task across multiple servers and reflects the binary decision encoded in Λ.

$$\sum_{j} \lambda_{i,j,k} \le 1 \tag{22}$$

Migration consistency. Ensuring that migration of task k between ES j and j' does not happen unless task was first offloaded from MU i.

$$\mu_{i,j,j',k} \le \lambda_{i,j,k} \tag{23}$$

Server capacity. Limiting the total CPU cycle allocations at each ES j to its maximum capacity F_j^{sev}. Thus, it can prevent overloading servers and ensure that remote computations respect available resources.

$$\sum_{i,k} f_{i,j,k}^{sev} \le F_j^{sev} \tag{24}$$

Power limits. Bounding each MU's transmit power $p_{i,j}$ by its hardware limit $p_i^{\max}$. This way captures radio-frequency regulatory or hardware constraints on uplink power.

$$0 \le p_{i,j} \le p_i^{\max} \tag{25}$$

To conclude, since choosing both λ and μ jointly across all tasks and servers grows combinatorically, solving such a mixed integer program is an NP-hard problem that needs an appropriate approach to reduce the complexity while achieving near-optimal offload decisions within a small time constraint.

4.3. MSQ Driven Decision Offloading Strategy

To tackle the presented NP-hard problem, we filter the set of ESs to get a few promising candidates by using mobility prediction and social strength scoring. Finally, regarding the QoS utility of ESs, we select the best one that strikes the balance between the delay and energy consumption

A. Mobility Prediction

For each MU i and ES j, we have to estimate the probability that i will remain in j's radio range long enough to complete an offloaded task. We first build a motion model for the user's trajectory, which consists of location and velocity over time with a linear Gaussian

$$x_k = \begin{bmatrix} p_x(k) \\ p_y(k) \\ v_x(k) \\ v_y(k) \end{bmatrix} \tag{26}$$

where (p_x, p_y) is 2D location and (v_x, v_y) is 2D velocity at time step k. The state dimension is *4*, so all covariance and gain matrices are *4 × 4*.

Next, we apply a Kalman filter to smooth noisy velocity readings (e.g., GPS) and predict the next position of the MU. Under a constant-velocity assumption over interval Δt, we set:

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{27}$$
$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

where A is the motion model matrix and H the observation matrix. These choices ensure that the filter propagates position by velocity and observes only position.

Then, we model the process noise covariance Q assuming that the user takes random accelerations with variance σ_a^2. Similarly, we exhibit the measurement noise covariance R based on independent GPS position measurement variance σ_p^2.

$$Q = \sigma_a^2 diag\left(\frac{\Delta t^4}{4}, \frac{\Delta t^4}{4}, \Delta t^2, \Delta t^2\right) \tag{28}$$
$$R = \sigma_p^2 I_2$$

In Q computation, the entries, Δt^4 and Δt^2 come from discretizing a continuous-time model where acceleration is a zero-mean Gaussian with variance σ_a^2. Such an approach aligns with our goal for a lightweight mobility predictor since it has only four nonzero entries. Meanwhile, in R computation, since we observe only $\left(p_x, p_y\right)$, we consider that R lives in *2 × 2* observation space. The corresponding identity matrix I says two position measurements share the same noise variance σ_p^2 and are uncorrelated.

Internally, the Kalman filter maintains a state estimate $\hat{x}$ and an error covariance P. For each time step, it performs the time update (predict) and measurement update (correct) as shown in Section 3.1. Each prediction requires only $O\left(n^2\right)$ operations for $n = 4$; thus, it is lightweight enough for real-time edge decisions.

In order to evaluate coverage over a short look-ahead horizon, we repeat the predict step L times by generating $\hat{x}_{k+1|k}, \hat{x}_{k+2|k}, \ldots, \hat{x}_{k+L|k}$. We must choose an appropriate L regarding the AP clustering window to balance the responsiveness and forecasting horizon. Such a work allows us to check whether MU i remains in ES j's radio range for the next L slots. Once we have the predicted trajectory $\hat{x}_{k|k}$, we compute the mobility score by comparing predicted positions to ES j's coverage radius. Higher $p_{i,j}$ means greater reliability (i.e., lower chance of disconnection) for offloading to ES j:

$$p_{i,j} = \Pr\{\text{user } i \text{ stays within range of server } j\} \tag{29}$$

B. Sociality Strength Scoring

We need to quantify such scoring based on the historical connection frequencies to do that. In practice, to handle new users with no prior interactions ($\Omega_i = 0$), our policy considers that $p_{i,j} = 1/M$ and $SC_{i,j} = 0$ to treat all ESs equally. To start with, $w_{i,j}$ is the number of times user i offloaded tasks to ES j, and $\Omega_i = \sum_{j=1}^{M} w_{i,j}$ presents the total number of offloading interactions for user i. To get the probability vector

$$P_i = \left(p_{i,1},...,p_{i,M}\right), \qquad p_{i,j} = \frac{w_{i,j}}{\Omega_i} \tag{30}$$

we convert the raw counts Ω_i into normalized probabilities. Computing all $p_{i,j}$ for a given user i costs only $O(m)$ time, negligible compared to clustering.

Using such a valid probability vector, we compute the Rényi entropy $H_\alpha(P_i)$ to identify the uncertainty or diversity in the user's offloading behavior. Here we set $\alpha = 2$ (Table 1.) to emphasize dominant past associations. In this way, the entropy computation grows linearly with the number of ESs,

$$H_\alpha(P_i) = \frac{1}{1-\alpha}\log\left(\sum_{j=1}^{M} p_{i,j}^{\alpha}\right), \qquad \alpha > 0, \alpha \neq 1 \tag{31}$$

Finally, we convert entropy into a social strength score where $\log M$ is the maximum possible entropy (i.e., uniform distribution). Thus, larger $SC_{i,j}$ means user i prefers ES j:

$$SC_{i,j} = 1 - \frac{H_\alpha(P_i)}{\log M} \tag{32}$$

This normalized score $SC_{i,j} \in [0,1]$ dynamically filters out less-trusted ESs before the clustering stage, improving reliability without manual thresholds

C. AP Clustering

AP clustering efficiently filters out the complete ES set to a minor candidate subset to reduce signaling overhead and decision latency without exhaustive probing. Thus, we employ a similarity metric that combines social trust $SC_{i,j'}$ and mobility reliability $p_{i,j}$ into a full similarity matrix. We first build the full similarity matrix $S_i(j,j')$ for MU i:

$$s_i(j,j') = \omega SC_{i,j'} + (1-\omega)p_{i,j'} \tag{33}$$

where $\omega \in [0,1]$ balances social strength $SC_{i,j'}$ and mobility reliability $p_{i,j'}$. We then set each "preference" value (called data-driven choice) and leverage the automation of AP to select a moderate number of clusters without manual thresholds.

$$s_i(j,j') = median\left(\left\{s_i(u,v) \mid u \neq v\right\}\right) \tag{34}$$

where u and v each range over the ES index set $1,...,M$. In detail, AP maintains two $M \times M$ matrices, responsibilities $r_i(j,k)$ and availabilities $a_i(j,k)$ for user i, that are initialized to zero and updated each iteration as:

$$\begin{aligned} r_i(j,k) &\leftarrow s_i(j,k) - \max_{k' \neq k}\left\{a_i(j,k') + s_i(j,k')\right\} \\ a_i(j,k) &\leftarrow \begin{cases} \sum_{j' \neq k} \max\{0, r_i(j',k)\}, & j = k \\ \min\left\{0, r_i(k,k) + \sum_{j' \notin \{j,k\}} \max(0, r_i(j',k))\right\}, & j \neq k \end{cases} \end{aligned} \tag{35}$$

To ensure the achievement of stable convergence, we apply damping with factor $\lambda = 0.5$

$$\begin{aligned} r_i &= \lambda r_i + (1-\lambda) r_i^{update} \\ a_i &= \lambda a_i + (1-\lambda) a_i^{update} \end{aligned} \tag{36}$$

We thus iterate until convergence or for at most $T = 100$ iterations. Interestingly, we find out that each iteration of AP updates the $M \times M$ matrices in time proportional to M^2 and uses M^2 memory overall, which remains practical when the number of ES M is moderate. Once converged, each ES j selects exemplar

$$\arg\max_k \left\{r_i(j,k) + a_i(j,k)\right\} \tag{37}$$

while the set of unique exemplars forms the candidate set C_i. By clustering on combined mobility and social signals, MSQ reduces $|C_i| \ \square M$, cutting signaling overhead and decision latency.

D. QoS Aware Offloading Decision

Among the set of candidates C_i as a result of clustering, we have to choose a single ES $j*$. Thus, for each candidate ES $j \in C_i$, we compute: (1) connection reliability $P_{con}(d_{i,j})$, (2) total offload completion $T_{i,j}$, and (3) server load L_j. Recall that (3) server load $L_j \in [0,1]$ is the fraction of j' s CPU already in use, where a lower L_j implies greater spare capacity. Meanwhile, (1) connection reliability is already detailed in section 3.4. The last metric, (2) total offload completion, is computed based on the per-task delays $T_{i,k}^{tot}$:

$$T_{i,j} = \sum_{k=1}^{K} T_{i,k}^{tot} \mid \lambda_{i,j,k} = 1 \tag{38}$$

We then normalize $T_{i,j}$ by the maximum observed delay $T_{\max}$ over all $j \in C_i$, yielding $T_{i,j}' \in [0,1]$. Interestingly, evaluating the metrics $P_{con}(d_{i,j})$, $T_{i,j}'$, and L_j for each $j \in C_i$ costs $O(|Ci|)$ total time. Thus, such evaluation is negligible compared to clustering. Finally, we combine them into one scalar utility

$$U_{i,j} = w_1 P_{con}(d_{i,j}) - w_2 T_{i,j}' - w_3 L_j \tag{39}$$

with weights $\{w_1, w_2, w_3\}$ can be defined as shown in Table 1. As a result, the chosen server is

$$j^* = arg\ \max_{j \in C_i} U_{i,j} \tag{40}$$

This way, we obtain a weighted sum trade-off between the three normalized QoS metrics. More precisely, our MSQ framework can retain the flexibility and adaptability concerning diverse application requirements via the weights $\{w_1, w_2, w_3\}$. To conclude, the decision process is decentralized across clustered ESs, which inherently improves scalability because each cluster makes local offloading decisions with bounded computation cost $O(k)$ proportional to its members.

5. Experiments

5.1. Experimental Setups

Environments and Datasets. We use the social network "Usense" [33] to employ the distances between users, encounter duration, and other parameters. All experiments are conducted in MATLAB R2019b on Ubuntu 22.04 LTS with a 2 CPU Xeon Gold 6150 (36 cores, 2.7GHz) and 64GB DDR4 RAM. From Usense, we extract timestamped user positions and encounter logs, such as peer identifier, received signal strength, and contact duration. These traces are preprocessed into decision step samples at $\Delta t = 1s$ intervals. Each sample includes (1) mobility history over the last $T = 10$ positions, (2) aggregated social contact features, such as number of distinct peers, cumulative contact duration, and a Rényi entropy–based score, and (3) distances to candidate ESs.

Since Usense does not provide direct latency measurements, task transmission delay is synthesized via a standard radio model. In other words, we model processing congestion at ES by queuing delay with an M/M/1 queue to approximate the dynamic load. In fact, M/M/1 is reasonable for providing an analytically tractable and realistic model of MEC task processing behavior where waiting time grows non-linearly. Parameter values for communication, mobility, and processing models are summarized in Table 1.

Since realistic MEC deployments are often small and edge-focused, the configuration with $M = 5$ is reasonable as prior works ([34,35]). Moreover, we generate from *40* to *120* MUs to stress the system and reflect realistic MEC workloads at scale. Indeed, these settings also ensure that the evaluation environment is scalable, for example, $M = 10$ or coverage radius $1000m \times 1000m$. To illustrate the distribution of ESs and MUs and their possible communications, we present a snapshot within $500m \times 500m$ area at time *t* for $M = 5$ and $N = 50$ in Fig. 2.

Table 1. Simulation parameter settings

Parameter	Value(s)	Notes
Network scale	$N \in \{40, 60, 80, 100, 120\}$ users	Each issues $K = 15$ tasks
Edge servers	$M = 5, 10$	Placed in $500m \times 500m$ or $1000m \times 1000m$
Task profile	$A_i^{in} = 300KB$, $A_i^{com} = 7.5 \times 10^7 cycles$, $A_i^{out} = 30KB$	Fixed per task: CPU cycles and result size
Computation speeds	$f^{loc} = 1GHz$, $F^{sev} = 10GHz$	Local vs. edge-server CPU; 50% server utilization
Wireless link	$W = 10MHz$, $N_0 = -95dBm$, $P_{tx} = -15dBm$	Rayleigh fading, path-loss exponent 4
Mobility model	Speed $\in \{0, 10, 20\}$ m/s	Random waypoint; Kalman window $L = 5$ slots
Process-noise	$\sigma_a = 1m/s^2$	Variance of user acceleration (for Q)
Measurement-noise	$\sigma_p = 5m$	GPS position noise (for R)
Social scoring	Rényi order $\alpha = 2$	Balances recent vs. historical ties
Scoring weight	$\omega = 0.5$	From pure social to pure mobility
QoS utility weights	$w_1 = 0.4, w_2 = 0.3, w_3 = 0.3$	Balances connection, time, and load in MSQ
Decision step interval	$\Delta t = 1s$	Sampling interval for traces
Mobility history length	$T = 10$ positions	KF or LSTM input window
Queueing model	*M/M/*1	Waiting delay at ES
LSTM-based settings	*64* hidden units, Adam Optimizer, 50 epochs	Training configurations

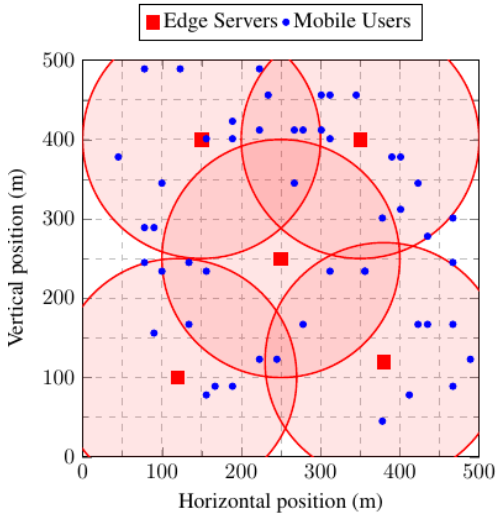


Fig.2. A representative of five edge servers and 50 mobile users in a 500 ×500 m area. The coverage radius of each edge server is 150 m

Baselines. We compare MSQ with these baselines: (1) Random Offloading (RO), where MU selects an ES randomly, (2) Mobility aware with KF (MKF) as in [10], and (3) MLSTM, which uses an LSTM model as an instance of a DL technique. We did the training of MLSTM in PyTorch 1.12 with Python 3.9. We configure one recurrent layer and *64* hidden units for the model. The learning rate is *0.0001*, batch size is *64*, and maximum training epoch is *50*. The Usense dataset is split *80:20* into training and testing sets. It is worth noting that the training phase of MLSTM is negligible compared to inference costs, which is more critical to be evaluated for realistic overhead. Specific metrics of MLSTM, such as accuracy, F1-score, or Recall, are presented in Table 2.

Table 2. Performance metrics of MLSTM over Usense dataset

Model	Train size	Test size	Accuracy(%)	F1-score
MLSTM	144,000	36,000	83.68	0.86

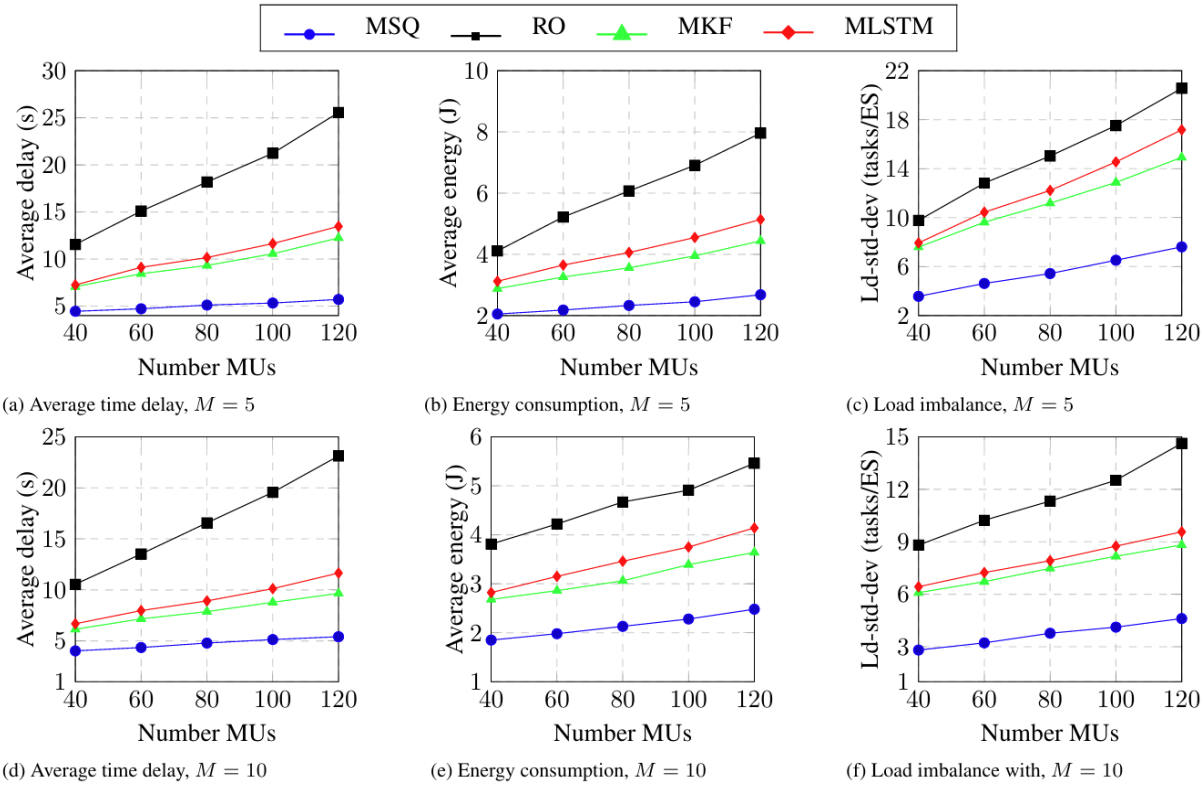
Scenarios and metrics. Each simulation is repeated *100* times with different random seeds. In detail, MATLAB random seed ensures all baselines face identical conditions, while PyTorch seeding controls only the offline training of MLSTM. We measure the key performance indicators, such as average completion delay, energy consumption, task distribution ratio (load imbalance), and connection reliability.

5.2. Experimental Setups

A. Overall Performances

To exhibit the stability of MSQ performance in small and large scaled MEC, we present two cases of simulations: $M=5$ with $500m\times500m$ and $M=10$ with $1000m\times1000m$ with variety of MU distributions (e.g., from *40* to *120*) in Fig. 3. For average time delay, as shown in Fig. 3a and 3d, MSQ consistently achieves the lowest delay, showing its strong scalability under different infrastructure densities. Quantitatively, MSQ outperforms MKF and MLSTM by roughly 36-40% and RO by over 70% when $M=10$ in $1000m\times1000m$. It leverages one dimension, as in MKF and MLSTM, and the social affinity and QoS utility to avoid both distant and overloaded ESs. RO is not an appropriate approach because it relies on uninformed, random ES selection, where many tasks are routed to overloaded ES, causing severe queueing and transmission penalties. Interestingly, MLSTM performs worse than MKF since the LSTM classifier attains only ≈84% correct server predictions in a small and noisy dataset. More precisely, remaining misroutes of MLSTM still induce extra transmission and queuing.

Second, we investigate the energy consumption (in Joules) in Fig. 3b and Fig. 3e. MSQ consistently maintains the lowest cost by selecting proximal and moderately loaded servers while its QoS function suppresses dispatches to congested ESs. In detail, with $M=10$, MSQ's energy consumption remains between 1.85-2.48J, showing that it effectively balances transmission and computation loads, yielding 26-32% lower energy than MKF and up to 68% less than RO. Meanwhile, MLSTM occasionally wrong offloads. It typically adds an extra transmission and an additional queuing contribution, thus the average penalty is nontrivial even with moderate load. Conversely, MKF is a low variance, short-term predictor that produces minor errors for Δt =1s decisions, translating into fewer costly misroutes.



(a) Average time delay, $M=5$ (b) Energy consumption, $M=5$ (c) Load imbalance, $M=5$

(d) Average time delay, $M=10$ (e) Energy consumption, $M=10$ (f) Load imbalance with, $M=10$

Fig.3. Overall performance comparison

Third, we quantify how evenly tasks are spread across the ESs in Fig. 3c and 3f. The objective is to observe the potential bottlenecks that can increase queuing delays and waste energy on overloaded ES, while the others sit idle or underutilized. More precisely, a low standard deviation of ES loads implies better resource utilization and leads to more

predictable QoS. To measure such an imbalance, we present:

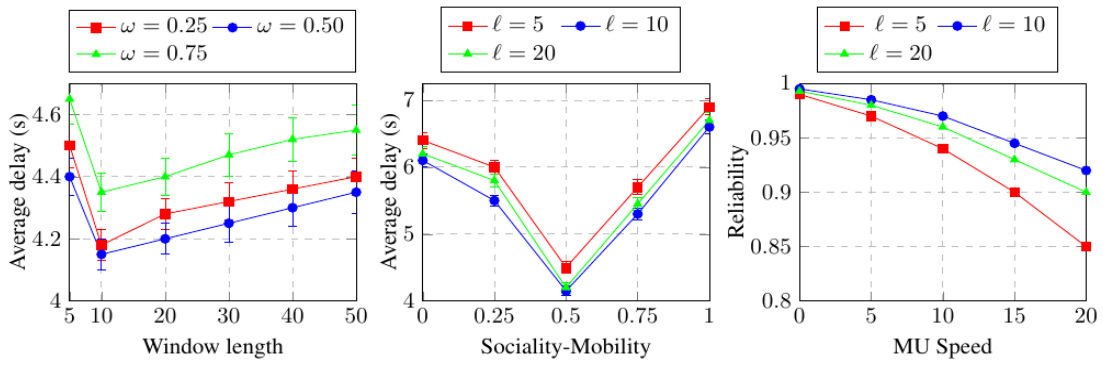
$$\bar{q} = \frac{1}{M}\sum_{j=1}^{M} q_j$$
$$\sigma_{load} = \sqrt{\frac{1}{M}\sum_{j=1}^{M}\left(q_j - \bar{q}\right)^2} \tag{41}$$

where q_j is the tasks assigned to ES j. Task assignments of RO yield a moderate growth since such randomness still spreads the load statistically evenly. MSQ maintains the most balanced task distribution across environments thanks to its QoS-driven load-aware, with standard deviations increasing gradually from 3.6 to 7.6 tasks $M = 5$ and from 2.8 to 4.6 tasks $M = 10$. Meanwhile, with MKF, some high-contact ESs accumulate tasks, creating spatial congestion hot spots, thus limiting the load balance capability. MLSTM suffers more severely from this effect since the imperfect prediction accuracy yields clustered misassignments, aggravating variance across ESs.

To summarize, we demonstrated the scalability of MSQ across deployment densities. When the network expands from $M = 5$ to $M = 10$, MSQ consistently maintains lower delay and energy overhead, confirming its adaptability in even larger topologies. If the system scales up in *N* and *M*, MSQ still exhibits near-linear scalability by jointly leveraging mobility and social prediction. Therefore, we confirm that MSQ scales gracefully under denser and larger MEC environments.

B. Sensitive Analysis

To further validate the robustness of MSQ, we perform a sensitivity analysis under the representative setting of $M = 5$ and $N = 80$. The experiments in Fig. 4 aim to assess how critical design parameters, such as KF window length ℓ, social–physical weight ω, and MU speed v, influence stability and adaptability of MSQ.



(a) Window length ℓ (b) Sociality-mobility ω (c) Connection reliability

Fig.4. MSQ sensitive analysis over critical design parameters.

Fig. 4a exhibits that the average completion delay follows a U-shaped curve. For small windows ($\ell = 5$ slots), the KF does not have historical data, leading to poor ES choices and high delay for noisy predictions. Increasing ℓ to 10-20 slots smooths out such noise since the filter benefits from additional data. This way, it reduces prediction noise faster than accumulating stale bias. This balance minimizes decision errors and yields the lowest observed delay at $\ell = 10$ slots.

However, when ℓ increases beyond 20 slots, the filter cannot keep up with user movements, resulting in a gradual delay growth due to offload miss emerging opportunities. This behavior is consistent for all ω, making $\ell = 10$ the optimal trade-off between responsiveness and smoothing. Fig. 4b reveals a similar balance when varying ω. As $\omega = 0$, the delay is relatively high because social-strength gains are ignored. Meanwhile, at $\omega = 1$, it again climbs due to physically suboptimal ES choices. Between these extremes lies $\omega = 0.5$, which consistently minimizes delay by blending reliability and proximity. We achieve the most significant reduction for latency (4.2 seconds) with $\ell = 10$. We also reveal that MSQ's average delay is minimized when ℓ and ω are set to their mid-range values ($\ell \approx 10$, $\omega \approx 0.5$). Consequently, the intersection of small window and low weight ($(\ell, \omega) = (10, 0.5)$) shows a balance between physical and social awareness.

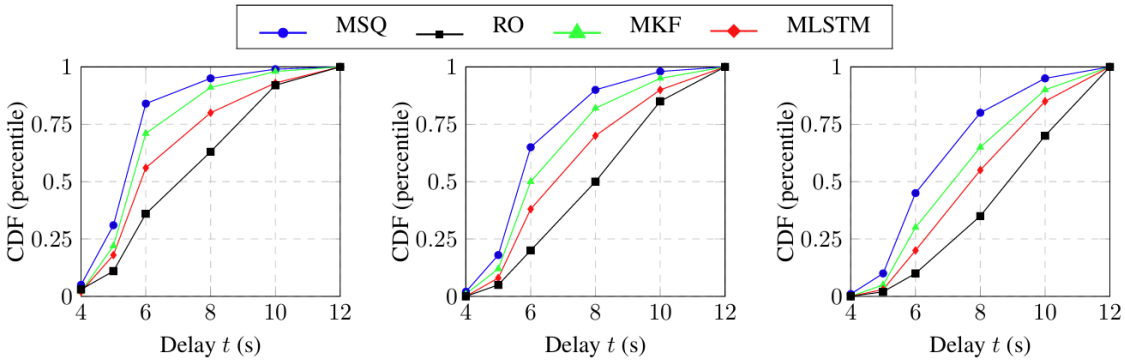
Fig. 4c shows a decline in connection reliability P_{con} as MS's speed v increases. In other words, faster users trigger more frequent handovers and prediction errors. The best case is that when the user does not move ($v = 0$), we achieve near perfect reliability ($P_{con} \approx 0.99$). As speed rises to 20 m/s, handovers become more frequent but Kalman predictions

are slow, causing misses in tasks offloading (P_{con} 0.85). Indeed, we can maintain higher reliability at moderate speeds using longer windows (e.g. $\ell = 10$ or 20) since their smoother estimates better anticipate movement. The best reliability at $v \le 10$ m/s is achieved with ($\ell = 10$), which smooths noise without becoming stale. To conclude, we state that MSQ maintains stable performance with only marginal degradation under extreme configurations.

C. Delay Distribution

To further assess the responsiveness of MSQ in real-time conditions, Fig. 5 plots the cumulative distribution function (CDF) of task-completion delay (the slowest performing packets or connections at the 95th to 99th percentile of latency values). This metric exposes the tail behavior that directly impacts user experience.

We exhibit the delay CDFs for a variety of numbers of MUs N = 40, 80, and 120. Across all configurations, we state that our strategy, MSQ, completes 95% of tasks in the shortest time. The reason is that MSQ leverages three dimensions to avoid both unstable links and overloaded servers while keeping worst cases under control. For example, at $N = 80$, MSQ completes 95% Single-dimension methods like MKF and MLSTM show wider tails, while RO exhibits the heaviest tail and the slowest rise. In detail, MLSTM shows slower convergence to unity than MKF due to its limited prediction accuracy, which leads to significant location errors and misoffloading. Meanwhile, MKF provides a more accurate short- term mobility prediction, although it cannot capture social context or dynamic server congestion. MSQ improves average latency and guarantees stricter delay bounds at the 95th–99th percentiles.



(a) $N = 40$ (b) $N = 80$ (c) $N = 120$

Fig.5. Tail latencies analysis

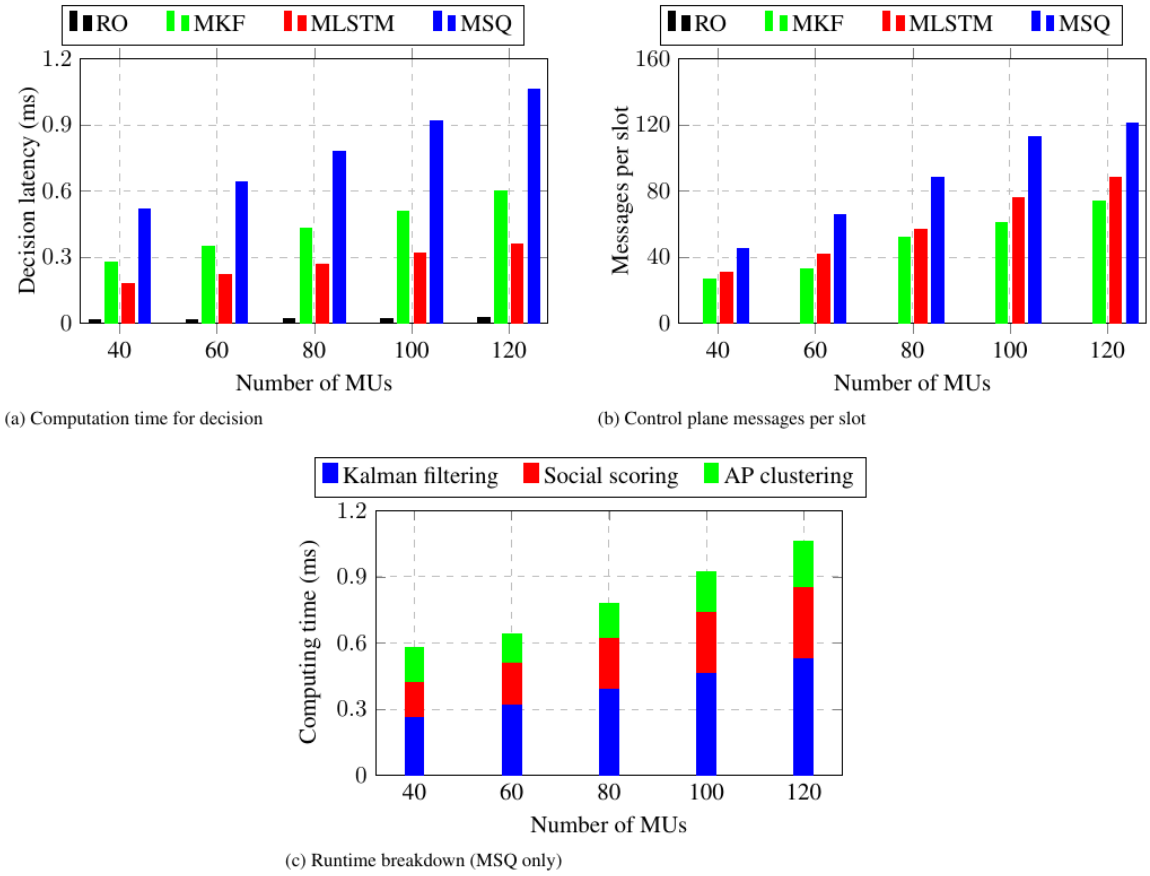
D. Overhead Analysis

To ensure that MSQ is practical for different scaled deployments, we examine its runtime and signaling overheads under increasing MUs. Thus, in Fig. 6, we exhibit the decision latency, control plane signaling overhead, and a detailed runtime breakdown for each dimension of MSQ.

In Fig. 6a, the decision latency of MSQ grows moderately from 0.5 to 1.0 *ms*. This latency is higher than single dimension approaches like KF and MLSTM. However, it remains within the sub-milliseconds, which is negligible compared with the average transmission in the overall performance evaluation. In detail, this overhead comes from the sequential execution of three dimensions, where each adds incremental computational effort based on the underlying algorithmic complexity. MLSTM is computationally heavy; it is thus executed as an inference-only model with fixed weights. Mean- while, MKF requires continuous state updates and covariance re-estimations per mobility events. As a result, MLSTM is slightly faster than MKF. We also find out that RO provides near-zero decision latency because it does not include either prediction or optimization.

To achieve the results shown in Fig. 6b, we need to construct a control plane messaging model with specific parameters in Table 3 for predicting and scoring. First, using a single position beacon per slot gives the KF enough fresh data to track user motion at slot granularity. More frequent reports (e.g., multiple per slot) could improve prediction at the cost of extra signaling. Second, since social ties change slowly based on user interactions, updating one vector per slot is enough to keep the Rényi entropy scores up-to-date. Third, scheduling exactly small responsibility broadcasts M per slot ensures that cluster assignments can be refreshed once every slot without flooding the network. Doing so ensures the clustering typically converges in a handful of iterations. In contrast, sending more messages per slot would duplicate work and offer negligible clustering improvement.

As expected, MSQ generates the most significant number of messages, particularly, 45-121 with respect to $N = 40 \rightarrow 120$. In detail, the MSQ message consists of beacons, social updates, and extra small AP clustering broadcasts, per server per slot. Such load remains a few hundred bytes per slot (e.g., 20 Bytes of message or ≈ 2 KBs), which is negligible compared to typical MEC control channel budgets, avoiding network congestion or device battery drain. MLSTM sends more messages per slot than MKF since its messages are larger and must be synchronized periodically among related ESs. More precisely, even the updates of MSLTM are less frequent, each carries more data, resulting in higher total messages per slot than MKF when averaged over time.



(a) Computation time for decision

(b) Control plane messages per slot

(c) Runtime breakdown (MSQ only)

Fig.6. Overhead analysis of strategies and runtime breakdown of MSQ

Table 3. Control plane messaging model per slot of time (Δt = 1 second)

Parameter	Value	Description
MU location report rate	1 report/slot	Each MU sends its current position for Kalman update.
Social-score update rate	1 update/slot	Each MU (or its serving ES) sends social-strength vector.
Social-score update rate	M broadcasts/slot	Each of the M ESs broadcasts its responsibility values once per slot

Breaking down the runtime of each procedure or component in MSQ can let us identify the bottleneck and the optimization chance (as seen in Fig. 6c). The results show that MF constitutes the dominant computation time (≈45−55%) because it continuously updates the mobility state of each MU within every decision window. More precisely, one matrix vector multiply is done per user, with each slot having $O(N)$ cost with a relatively significant constant factor. The social strength scoring module contributes roughly 25−30% to compute entropic sums per user, in which each operation is just a handful of arithmetic steps. Indeed, such per-user cost is smaller than that of KF. Finally, AP clustering contributes about 15-20% with its modest $O(M^2)$ complexity. Note that its fraction of computation could become comparable or even exceed social strength scoring and KF if M gets larger (e.g., M grows beyond 15). These fractions reflect the frequency and granularity of updates per ES cluster. Even with the highest number of MUs $N = 120$, the sum of fractions stays below one ms, confirming that MSQ incurs minimal runtime overhead while preserving the responsiveness for real-time edge decision making [35].

Breaking down the runtime of each procedure or component in MSQ can let us identify the bottleneck and the optimization chance (as seen in Fig. 6c). The results show that MF constitutes the dominant computation time (≈45−55%) because it continuously updates the mobility state of each MU within every decision window. More precisely, one matrix vector multiply is done per user, with each slot having $O(N)$ cost with a relatively significant constant factor. The social strength scoring module contributes roughly 25−30% to compute entropic sums per user, in which each operation is just a handful of arithmetic steps. Indeed, such per-user cost is smaller than that of KF. Finally, AP clustering contributes about 15-20% with its modest $O(M^2)$ complexity. Note that its fraction of computation could become comparable or even exceed social strength scoring and KF if M gets larger (e.g., M grows beyond 15). These fractions reflect the frequency and granularity of updates per ES cluster. Even with the highest number of MUs $N = 120$, the sum of

fractions stays below one *ms*, confirming that MSQ incurs minimal runtime overhead while preserving the responsiveness for real-time edge decision making [35].

To conclude, the total coordination and signaling latency of MSQ remains minimal (i.e., $\leq$ 1ms), since decision exchanges occur only within local edge domains. Even when scaled to multi-domain MEC architectures, the control-plane messaging of MSQ accounts for less than 0.1% of the overall traffic load, which is negligible compared to typical data-plane volumes. These numerical results ensure that MSQ remains practical for real world 5G or 6G edge deployments, even under cross-operator coordination scenarios.

6. Conclusions and Future Work

We propose MSQ, a three-dimensional decision model that jointly considers mobility, sociality, and QoS awareness to optimize task offloading of MEC, where dynamic and resource-constrained conditions are critical. Our framework integrates lightweight KF for mobility prediction, Rényi entropy for social scoring, and QoS function. The experiments for small and medium scale deployments confirm that MSQ achieved up to 78% reduction in average latency, 66% lower energy consumption, and 64% improvement in load balancing compared with a random offloading strategy while maintaining below one ms per decision and consistent stability across scenarios. Beyond averages, its 95th and 99th percentile task completion times also reflect fewer service interruptions and higher user perceived reliability under real-time workloads. Finally, MSQ's per-decision latency remains sub-millisecond, demonstrating low-latency performance and practical deployability.

Future work will focus on enhancing the adaptability and generalization capability of MSQ under more dynamic and large-scale MEC conditions. MSQ should support heterogeneous task types and multi-service applications, where offloaded workloads vary in computation intensity, data volume, and latency sensitivity. Such heterogeneity may require dynamic parameter tuning and priority-aware scheduling, which could be explored using hybrid optimization and runtime feedback mechanisms. Another concern is about the integration of learning-augmented prediction modules. This component should continuously refine mobility and trust estimations based on online user behavior. In addition, we plan to extend MSQ toward federated and cooperative MEC architectures, where multiple edge domains collaboratively share information to optimize offloading while preserving data privacy. Such an approach could enable scalable cross-operator task management and reduce control overhead in large networks. Finally, with the evolution of 6G, IoT, and vehicular paradigms, we aim to evaluate MSQ in ultra-dense, high-mobility environments involving interactions between autonomous vehicles and infrastructures. This extension will require refining the QoS utility model for ultra-reliable low-latency communications and context-driven service continuity. Doing so could bridge the gap between theoretical optimization and real-world deployment in next-generation edge ecosystems.

Author Contributions Statement

Van Long Nguyen Huu – Conceptualization, Methodology, and Supervision: Proposed research ideas, Constructed the overall framework, and supervised project execution. Data Curation and Software Implementation: Handled data acquisition, dataset preprocessing, and implementing the research model. Writing – Drafted the initial manuscript, contributed to the literature survey, and documented the technical background of the study.

An Cong Tran – Model Training, Validation, and Performance Evaluation: Led the model training process, validated results using standard metrics, and benchmarked performance against existing methods. Formal Analysis, Visualization, and Statistical Analysis: Performed in-depth analysis of experimental results, prepared performance charts, and ensured the statistical robustness of the evaluation. Writing – Review and Editing, and Project Management: Reviewed and edited the manuscript, ensured clarity and coherence, and helped coordinate project milestones and deadlines.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement

The authors declare no conflicts of interest.

Funding Declaration

The authors received no financial support for the research, authorship, and publication of this article.

Data Availability Statement

This study analyzed publicly available datasets. The dataset can be found here: "https://ieee-dataport.org/open-access/crawdad-copelabsusense", accessed on "20 March 2026". The source code that support the findings of this study is available from the corresponding author upon reasonable request.

Ethical Declarations

This research is purely theoretical and simulation-based with publicly available dataset. It thus does not involve human subjects, animals, or any experiments requiring ethical approval.

Acknowledgments
We sincerely thank the experts for their professional evaluation and valuable recommendations, which have contributed to improving the quality of the experiment and the reliability of its results.

Declaration of Generative AI in Scholarly Writing
During the preparation of this work the authors used Grammarly solely to check grammar, spelling errors, and improve language clarity. The authors reviewed and edited the content and take full responsibility for the content of the publication.

Abbreviations
The following abbreviations are used in this manuscript:

MEC - Mobile Edge Computing
ES - Edge Server
QoS - Quality of Service
Mobile User - MU
Affinity Propagation - AP
Kalman Filtering - KF
Deep Learning - DL
Long Short Term Memory - LSTM
Federated Deep Reinforcement Learning - FDRL
Mobility, Sociality and QoS framework - MSQ

APPENDIX A\B\C…, with appendix tile
Not applicable.

References

[1] Liang J, Ma B, Feng Z, Huang J. Reliability-Aware Task Processing and Offloading for Data-Intensive Applications in Edge computing. IEEE Transactions on Network and Service Management 2023. Doi:10.1109/TNSM.2023.3258191.
[2] Haibeh LA, Yagoub MCE, Jarray A. A Survey on Mobile Edge Computing Infrastructure: Design, Resource Management, and Optimization Approaches. IEEE Access 2022;10:27591–610. Doi:10.1109/ACCESS.2022.3152787.
[3] Loutfi SI, Shayea I, Tureli U, El-Saleh AA, Tashan W. An overview of mobility awareness with mobile edge computing over 6G network: Challenges and future research directions. Results in Engineering 2024;23:102601. Doi:10.1016/j.rineng.2024.102601.
[4] Dong S, Tang J, Abbas K, Hou R, Kamruzzaman J, Rutkowski L, et al. Task offloading strategies for mobile edge computing: A survey. Computer Networks 2024;254:110791. Doi:10.1016/j.comnet.2024.110791.
[5] Xu C, Xu C, Li B, Li S, Li T. Joint Social-Aware and Mobility-Aware Computation Offloading in Heterogeneous Mobile Edge Computing. IEEE Access 2022;10:28600–13. Doi:10.1109/ACCESS.2022.3158319.
[6] Zaman SK uz, Jehangiri AI, Maqsood T, Ahmad Z, Umar AI, Shuja J, et al. Mobility-aware computational offloading in mobile edge networks: a survey. Cluster Computing 2021;24:2735–56. Doi:10.1007/s10586-021-03268-6.
[7] Gong Y, Hao F, Wang L, Zhao L, Min G. A Socially-Aware Dependent Tasks Offloading Strategy in Mobile Edge Computing. IEEE Transactions on Sustainable Computing 2023;8:328–42. Doi:10.1109/TSUSC.2023.3240457.
[8] Zhou L, Zhang L, Zhao C, Laili Y, and LX. Diverse task scheduling for individualized requirements in cloud manufacturing. Enterprise Information Systems 2018;12:300–18. Doi:10.1080/17517575.2017.1364428.
[9] Ojima T, Fujii T. Resource management for mobile edge computing using user mobility prediction. 2018 International Conference on Information Networking (ICOIN), 2018, p. 718–20. Doi:10.1109/ICOIN.2018.8343212.
[10] Rui L, Wang S, Wang Z, Xiong A, Liu H. A dynamic service migration strategy based on mobility prediction in edge computing. International Journal of Distributed Sensor Networks 2021;17:1550147721993403. Doi:10.1177/1550147721993403.
[11] Deng M, Li S, Jiang X, Li X. Vehicle Trajectory Prediction Method Based on “Current” Statistical Model and Cubature Kalman Filter. Electronics 2023;12. Doi:10.3390/electronics12112464.
[12] Mehmood A, Muhammad A, Mehmood F, Song W-C. Enhancing Vehicle Location Prediction Accuracy with Road-Aware Rectification for Multi-Access Edge Computing Applications. Mathematics 2024;12. Doi:10.3390/math12243980.
[13] Zaki AM, Elsayed SA, Elgazzar K, Hassanein HS. Quality-Aware Task Offloading for Cooperative Perception in Vehicular Edge Computing. IEEE Transactions on Vehicular Technology 2024;73:18320–32. Doi:10.1109/TVT.2024.3444591.
[14] Jahandar S, Kouhalvandi L, Shayea I, Ergen M, Azmi MH, Mohamad H. Mobility-Aware Offloading Decision for Multi-Access Edge Computing in 5G Networks. Sensors 2022;22. Doi:10.3390/s22072692.
[15] Xia C, Jin Z, Su J, Li B. Mobility-Aware Offloading and Resource Allocation Strategies in MEC Network Based on Game Theory. Wireless Communications and Mobile Computing 2023;2023:5216943. Doi:10.1155/2023/5216943.
[16] Chen L, Du J, Zhu X. Mobility-Aware Task Offloading and Resource Allocation in UAV-Assisted Vehicular Edge Computing Networks. Drones 2024;8. Doi:10.3390/drones8110696.
[17] Hossein Shokouhi M, Hadi M, Pakravan MR. Mobility-Aware Computation Offloading for Hierarchical Mobile Edge Computing. IEEE Trans on Netw and Serv Manag 2024;21:3372–84. Doi:10.1109/TNSM.2024.3386845.

[18] Huang L, Yu Q. Mobility-aware and energy-efficient offloading for mobile edge computing in cellular networks. Ad Hoc Netw 2024;158. Doi:10.1016/j.adhoc.2024.103472.
[19] Wang N, Li Y, Li Y, Nie H. Fault-tolerant and mobility-aware loading via Markov chain in mobile cloud computing. Sci Rep 2025;15:18844. Doi:10.1038/s41598-025-02529-3.
[20] Cui C, Zhao M, Wong K. An LSTM-Method-Based Availability Prediction for Optimized Offloading in Mobile Edges. Sensors 2019;19. Doi:10.3390/s19204467.
[21] Zaman SK uz, Jehangiri AI, Maqsood T, Haq N ul, Umar AI, Shuja J, et al. LiMPO: lightweight mobility prediction and offloading framework using machine learning for mobile edge computing. Cluster Computing 2022;26:99–117. Doi:10.1007/s10586-021-03518-7.
[22] Zaman SK uz, Jehangiri AI, Maqsood T, Umar AI, Khan MA, Jhanjhi NZ, et al. COME-UP: Computation Offloading in Mobile Edge Computing with LSTM Based User Direction Prediction. Applied Sciences 2022;12. Doi:10.3390/app12073312.
[23] Nishad D, Verma V, Rajput P, Gupta S, Dwivedi A, Shah D. Adaptive AI-enhanced computation offloading with machine learning for QoE optimization and energy-efficient mobile edge systems. Scientific Reports 2025;15. Doi:10.1038/s41598-025-00409-4.
[24] Huang H, Zhan W, Min G, Duan Z, Peng K. Mobility-Aware Computation Offloading With Load Balancing in Smart City Networks Using MEC Federation. IEEE Transactions on Mobile Computing 2024;23:10411–28. Doi:10.1109/TMC.2024.3376377.
[25] Yang P, Li Q, Yan Y, Li X-Y, Xiong Y, Wang B, et al. "Friend is Treasure": Exploring and Exploiting Mobile Social Contacts for Efficient Task Offloading. IEEE Transactions on Vehicular Technology 2016;65:5485–96. Doi:10.1109/TVT.2015.2465392.
[26] Wu D, Shen G, Huang Z, Cao Y, Du T. A Trust-Aware Task Offloading Framework in Mobile Edge Computing. IEEE Access 2019;7:150105–19. Doi:10.1109/ACCESS.2019.2947306.
[27] Teng Y, Liu Z. Edge computing task scheduling method based on user's social relations: a construction and solution for Smart City Library. PeerJ Computer Science 2024;10:e2457. Doi:10.7717/peerj-cs.2457.
[28] Li K, Wang X, Ni Q, Huang M. Entropy-based Reinforcement Learning for computation offloading service in software-defined multi-access edge computing. Future Generation Computer Systems 2022;136:241–51. Doi:10.1016/j.future.2022.06.002.
[29] Belli D, Chessa S, Foschini L, Girolami M. A Social-Based Approach to Mobile Edge Computing. 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, p. 00292–7. Doi:10.1109/TNSM.2023.3258191.
[30] Raju MR, Mothku SK, Somesula MK. DMITS: Dependency and Mobility-Aware Intelligent Task Scheduling in Socially-Enabled VFC Based on Federated DRL Approach. IEEE Transactions on Intelligent Transportation Systems 2024;25:17007–22. Doi:10.1109/TITS.2024.3432177.
[31] Chen M, Wang T, Zhang S, Liu A. Deep reinforcement learning for computation offloading in mobile edge computing environment. Computer Communications 2021;175:1–12. Doi:10.1016/j.comcom.2021.04.028.
[32] Brecko A, Kajati E, Koziorek J, Zolotova I. Federated Learning for Edge Computing: A Survey. Applied Sciences 2022;12. Doi:10.3390/app12189124.
[33] Firdose S, Lopes L, Moreira W, Sofia R, Mendes P. CRAWDAD dataset copelabs/usense (v. 2017-01-27) 2017. Doi:10.15783/C7D885
[34] Zhang Y, He X, Xing J, Li W, Seah WKG. Load-balanced offloading of multiple task types for mobile edge computing in IoT. Internet of Things 2024;28:101385. Doi:10.1016/j.iot.2024.101385.
[35] Sindjoung MLF, Velempini M, Bomgni AB. A MEC architecture for a better quality of service in an Autonomous Vehicular Network. Computer Networks 2022;219:109454. Doi:10.1016/j.comnet.2022.109454.

Authors' Profiles



Van Long Nguyen Huu, received his Master of Embedded Systems from Limoges University, France, in 2014, and his Ph.D of Information Technology from Rennes I University, France, in 2022. He is senior lecturer in College of Information and Communication Technology, Can Tho University, Vietnam from 2010. His area of interest includes cloud and edge computing, caching techniques, embedded systems and cybersecurity.



An. Cong Tran is a senior lecturer at the College of Information and Communication Technology, Can Tho University, Vietnam. He holds a Master's Degree (Hons) in Computer Science from the Asian Institute of Technology (AIT), Thailand, and a Ph.D. in Computer Science from Massey University, New Zealand. His current research interests encompass description logic learning, ontology learning, applications of blockchain in the public sector, and deep learning methods.

How to cite this paper: Van Long Nguyen Huu, An. Cong Tran, "Three-dimensional Decision Task Offloading Model in Mobile Edge Computing", International Journal of Computer Network and Information Security(IJCNIS), Vol.18, No.2, pp.19-37, 2026. DOI:10.5815/ijcnis.2026.02.02