This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822

Edge-assisted Real-time Motion Capture System

Chen Qian, Danyang Li, *Student Member, IEEE;* Jingao Xu, *Member, IEEE;* Zheng Yang†, *Fellow, IEEE* Qiang Ma, *Member, IEEE;*

Abstract—Real-time motion capture stands as a pivotal enabler for human-centric applications, such as AR interactions, VR live-streaming, and immersive gaming. Deploying motion capture models on edge/cloud server to provide scalable services is emerging as a promising trend. However, due to the computational burden of model inference and the inevitable network transmission delays, mainstream DNN-based systems struggle to deliver real-time motion capture services. In this paper, we propose edgeMoCap, the first edge-assisted real-time DNN-based motion capture system. At the heart of edgeMoCap is a motion capture-tracking architecture optimized for edge-client heterogeneous computing capabilities. We dig into the spatio-temporal correlation between human motions and body-attached markers to: (i) distill a lightweight tracker for real-time motion tracking on client devices, and (ii) efficiently partition functional modules and schedule motion data. Together, these efforts establish a parallel capture-tracking architecture that enhances edge-client collaboration, ensuring accurate and real-time motion capture. We fully implement edgeMoCap on devices with varying computational resources and network conditions, conducting extensive experiments with diverse users and motions. The results show that edgeMoCap achieves an average latency of 6ms and an accuracy of 3.8cm, outperforming comparative methods by 70%. *Code and data will be made publicly available before publication.*

Index Terms—Edge computing; Motion capture; Motion tracking; Real-time applications

1 Introduction

Recent advancements have positioned real-time motion capture as a critical technology for high-value applications such as virtual interactions and live streaming [1], [2], [3]. Optical-based motion capture, recognized as the industry standard, stands out for its precision and efficiency in recording detailed, coordinated motions [4], [5], [6]. This technology typically employs multi-view infrared cameras to track 3D markers attached to the human body, allowing for accurate reconstruction of skeletal motion [7].

Generally, real-time motion capture systems aim to compute and relay human motion information with imperceptible delay, ideally less than 35 milliseconds [8]. Within those optical-based systems, the most computationally intensive task lies in the reconstruction of skeletal motion from markers. Current reconstruction practices fall into two primary categories: (i) *evidence-driven* method [9], [10], [11], [12], [13], based on empirical rules of human kinetics [14], and (ii) *data-driven* method [7], [15], utilizes deep neural network (DNN) to learn complex priors of human motion from data [16] . Compared to the explicit rules of the former, DNN-based methods offers robustness in handling with challenging motion, and has been proven to be the most promising solution [2], [15].

With the rapid advancement of edge computing resources [17], [18], [19], deploying motion capture systems on edge devices to provide scalable, low-cost motion capture services has become feasible [20], [21], [22]. However, our three-month field study reveals that deploying existing DNN-based systems at the edge to provide motion capture services encounters two major challenges that impact real-time performance, as shown in Fig. 1-(a).

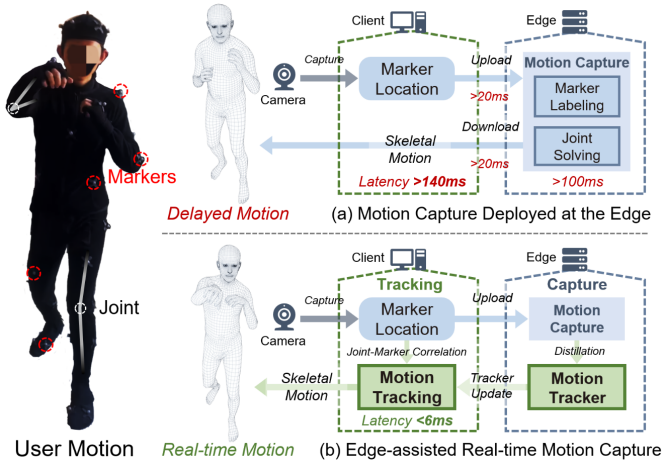


Fig. 1: **Motion capture architecture comparison:** (a) Motion capture system that fully deploy models at the edge suffer from both model inference latency (i.e., $>100ms$) and synchronization latency (i.e., $>40ms$), resulting in delayed motion feedback on the client. (b) The edge-assisted motion capture architecture employed by edgeMoCap utilizes a motion tracker distilled from the motion capture model, allowing the client to track skeletal motions directly from marker locations in real-time (i.e., $6ms$). The upgraded modules of edgeMoCap are highlighted in green.

• **Significant computational overhead.** To accommodate the complex motion priors of the human body, current practices typically involve models with a large number of parameters (e.g., >350MB [15]). Consequently, even on edge devices with advanced computational capabilities (e.g., GeForce RTX 3090 Ti), the latency for a single inference typically exceeds $100ms$.

• **Inevitable transmission latency.** The uploading of marker

- *Chen Qian, Danyang Li, Zheng Yang, Qiang Ma are with the School of Software and BNRist, Tsinghua University, Beijing, China, 100084. E-mail: {chen.cronus.qian, lidanyang1919, hmilyyz}@gmail.com, thu-maq@mail.tsinghua.edu.cn*
- *Jingao Xu is with the Department of Computer Science, Carnegie Mellon University. E-mail: xujingao13@gmail.com*
- †*Zheng Yang is the corresponding author*

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822

data and the downloading of motion results inevitably suffer from network delay and fluctuations (e.g., overall transmission latency $>40ms$), which compromise the real-time motion response at client and may introduce continuous ghosting effects in downstream tasks [23], [24], [25].

As the edge-client collaboration paradigm evolves [26], [27], [28], [29], we identify an opportunity to leverage the accurate motion capture results from edge, coupled with the rapid feedback capabilities of client, to overcome above challenges and achieve real-time motion capture. **Key Insight:** The human skeleton is composed of numerous joints, each strongly correlated with spatially adjacent markers. This correlation enables us to track skeletal motion from specific joint poses with their subsequent updated marker locations.

Our Work: Motivated by this opportunity, we design and implement **edgeMoCap**, an **edge**-assisted real-time **Mo**tion **Cap**ture system. As illustrated in Fig. 1-(b), through leveraging *joint-marker correlation*, edgeMoCap capitalizes on low-frequency skeletal motion captured by DNNs at the edge, allowing lightweight client to accurately track high-frequency motion updates in real time.

Realizing the *capture-tracking* based edge-client collaborative architecture to boost edgeMoCap's accuracy and efficiency, poses several challenges that are addressed in this work: (i) how to design a lightweight motion tracker that models the spatial-temporal relationships between markers and joints; (ii) how to generalize the joint-marker correlation across different types of user body shape; (iii) how to promptly identify and synchronize critical motion capture results to maintain subsequent motion tracking. Overall, edgeMoCap excels across three layers:

- On the Client layer, we design a *Joint-Marker Correlation-driven Motion Tracking* module. By exploring the local spatial correlation between marker locations and joint motion, we employ two sequential trackers to separately track the locations of markers and the poses of joints. This simple yet effective design enables edgeMoCap to track skeletal motion in real time on lightweight client devices.
- On the Edge layer, we introduce *Online Learning-based Tracker Generalization* to enhance the motion tracker's adaptability across users of various body shapes. edgeMoCap is capable of promptly distilling explicit joint-marker correlation from data-driven implicit model during the continuous motion capture process, allowing for the real-time updating of tracker on the client.
- On the Schedule layer, we implement an *Event-responsive Motion Synchronization* strategy. edgeMoCap swiftly identifies two types of marker-related events on the client that affect accurate motion tracking (i.e., the occlusion and de-occlusion of markers). In response, the edge performs timely motion re-captures and tracker updates to ensure consistent tracking performance.

We have fully implemented edgeMoCap on a variety of edge and client devices with different computational and network conditions. Our comprehensive experiments involved 6 users with various body shapes to perform 36 distinct motions, collecting 90 sequences of combined motions with 114,086 frames. We compared the performance of edgeMoCap with two state-of-the-art (SOTA) data-driven motion capture system (MoCap-Solver [15] and Mosh++ [16]). The results demonstrate that edgeMoCap consistently achieves an end-to-end latency of $< 6ms$ across different scenarios, with a real-time accuracy exceeding comparative methods by $> 70\%$.

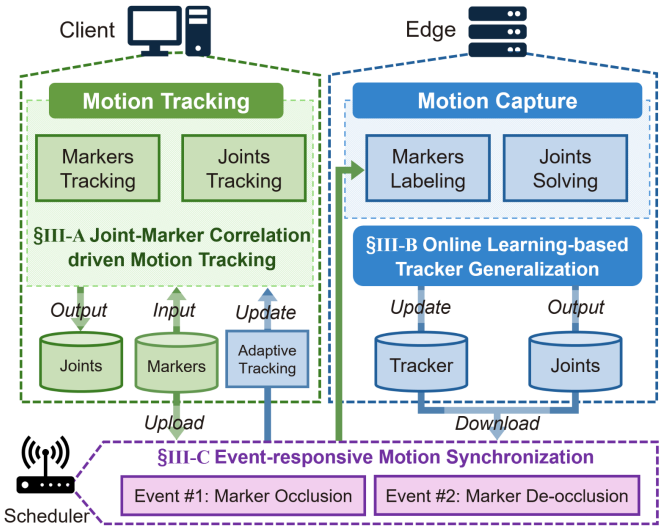


Fig. 2: edgeMoCap Overview

In summary, this paper makes the following contributions.

(1) We propose edgeMoCap, as far as we are aware of, the first edge-assisted real-time DNN-based motion capture system. edgeMoCap promotes the practical deployment of cost-effective, scalable motion capture services.

(2) We reveal the correlation between skeletal motion and marker locations, and from this insight, we derive a generalizable explicit motion tracker from data-driven implicit model, to enable light-weight client to infer motion in real time.

(3) We implement edgeMoCap under varying computational and network conditions. The evaluation results highlight the efficacy of our proposed design. We will open-source edgeMoCap, offering useful tools for academia and industry to fast prototype real-time motion capture systems.

The rest of this paper is organized as follows. We first present the overview of edgeMoCap's edge-assisted architecture in §2, followed by detailed descriptions of the function modules and synchronization strategy in §3. The implementation and evaluation are presented in §4. We discuss the related work in §5 and conclude edgeMoCap in §6.

2 System Overview

We first briefly introduce the existing motion capture system. Then, we detail the edge-assisted architecture adopted by edgeMoCap.

2.1 Existing Motion Capture System

Recovering human skeleton from imperfect motion capture data generally involves two steps. The first step is marker data cleaning and classification, known as *Markers Labeling*. The second step is solving for the human skeleton from the labeled marker data, referred to as *Joints Solving*.

Markers Labeling. In motion capture systems, the output from infrared cameras typically consists of incomplete 3D points. The process of associating these 3D points with the markers on the captured object is called marker labeling [30]. During the marker labeling process, outlier points

captured by the cameras can also be cleaned up as they cannot be assigned to any corresponding labels.

Joints Solving. Regressing the translations and rotations of human joints from the labeled markers is called joints solving [15]. One of the common approach is encoding input markers into embedding space and decoding skeletons from embedding space.

2.2 edgeMoCap Overview

edgeMoCap achieves real-time motion tracking on the client side using a lightweight *Motion Tracker* that built upon low-frequency skeletal motions. This edge-capture and client-tracking decoupled architecture , as illustrated in Fig. 1-(b), circumvents (i) the inference delay of DNN models on the edge side, and (ii) the transmission latency of motion information over the network.

The overview of edgeMoCap is outlined in Fig. 2. From a top perspective, edgeMoCap adheres to the system abstraction similar to existing edge-client collaboration paradigm, encompassing the client, edge, and schedule layers. We explore the specific workflow within this edge-assisted motion capture architecture and outline the innovative functional modules designed to enhance the system's capabilities across all three layers.

On the Client layer, corresponding to the existing two steps of motion capture, edgeMoCap employs a *Joint-Marker Correlation-driven Motion Tracking* module (3.1) that includes both a *Marker Tracker* and a *Joint Tracker*. The former tracks preprocessed marker data and assigns labels to the markers. The latter tracks joint motions based on the location changes of markers.

On the Edge layer, a general DNN-based motion capture model is running continuously. This model can batch output marker labels and reconstructed human pose. Moreover, to improve the generalization of edgeMoCap, local tracker can be adaptively updated by *Online Learning-based Tracker Generalization* (3.2). This module fine-tunes a neural network using the currently collected data in an online learning manner, which ensures that the learned results can adapt to the body shape of human in motion.

On the Schedule layer an *Event-responsive Motion Synchronization* (3.3) strategy is specially designed for handling changes in the number of markers caused by marker occlusion. When the number of markers decreases due to occlusion, Scheduler update *Joint Tracker* with the results from the edge side to quickly adapt to the latest marker distributions. When the number of markers increases as they reappear after occlusion, scheduler synchronizes both *Marker Tracker* and *joint tracker* with edge to rapidly utilize all markers information.

3 Modules Design of edgeMoCap

3.1 Joint-Marker Correlation-driven Motion Tracking

A simple and effective tracker design is key to achieving low end-to-end latency in real-time motion capture. The process begins with the input of raw marker locations, and finishes with the reconstruction of human skeleton required by the rendering module. The whole process comprises three main steps.

The first step is a simple denoising process to remove outliers from the input MoCap data. The second step involves identifying the labels of markers, that is, the corresponding positions of human body parts. The third step uses the locations and labels of markers to calculate the locations and rotations of human joints. For the latter two steps, we have respectively designed the Marker Tracker and the Joint Tracker.

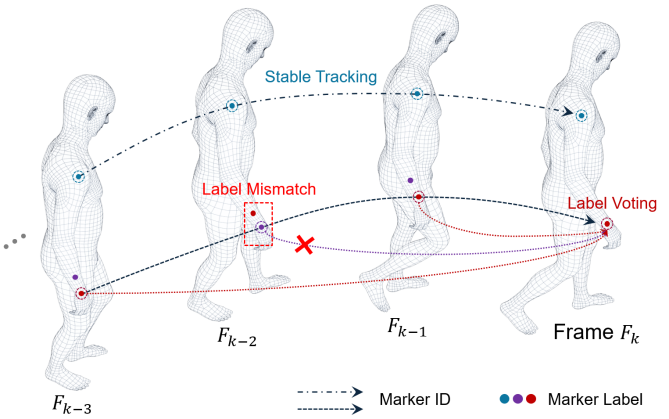


Fig. 3: **The process of marker tracking.** The marker on the shoulder, which maintain relatively stable locations, consistently show correct labeling results. In contrast, the marker on the hand, subject to vigorous movements, encounter a label mismatch at frame F_{k-2}. Despite this, our label voting strategy ensures that subsequent tracking of this marker is not compromised. Markers with the same ID are connected by dashed lines, and different colors are used to indicate each marker's label.

3.1.1 Marker Tracker

In the process of human motion capture, calibrated infrared cameras output the world coordinates of marker locations in three-dimensional space. Based on the knowledge of human kinematics, local devices can predict the potential locations of markers in the next frame and track their movement accordingly. The same marker identified across different frames is assigned the same marker ID. Thus, it can be inferred that the label of a marker with the same ID should be consistent across different frames.

As illustrated in Fig. 3, to obtain the label of the marker with ID I_d in frame F_k, Marker Tracker refers to the labels in historical frames. Considering that the edge model may incorrectly recognize marker labels in complex movements, the labels of markers with ID I_d in historical frames may not all be correct. To achieve relatively accurate recognition results, Marker Tracker initiates a polling in historical frames, where markers with the same ID undergo label voting.

$$x_d = \text{argmax}_{x \in A_d} \left(\sum_{i=1}^{n_d} \mathbf{1}(a_{di} = x) \right). \tag{1}$$

As shown above, $A_d = \{a_{d1}, a_{d2}, \ldots, a_{dn_d}\}$ denotes all labels of markers with ID I_d in historical frames. n_d is the number of historical frames referenced, and $\mathbf{1}$ is the indicator function, which is equal to 1 only if $a_{di} = x$. The equation (1) indicates finding x_d as the label that appears most frequently in A_d, to be used as the label in current frame. Thus, it completes label tracking for the marker with ID I_d, and this rule will be applied to all markers.

3.1.2 Joint Tracker

Under high sampling rates (e.g., 60 fps or 120 fps), the inter-frame variation of human joint poses can be regarded

as a small-step motion that satisfies the local rigid-body assumption. For a single rigid bone segment with a joint as reference, let the neighboring visible markers be $t_{mi}(t) \in \mathbb{R}^3, i = 1, 2...M$. In the local coordinate system of the joint, the vector from the joint to marker i is r_i. For frame F_k, the global pose of the joint is (R_j^k, t_j^k). The ideal rigid model is

$$t_{mi}^k = R_j^k r_i + t_j^k. \tag{2}$$

At high frame rates, the inter-frame motion is a small rotation and a small translation:

$$R_j^k = R_j^{k-1} \exp([\delta\omega]_\times) \approx R_j^{k-1}(I + [\delta\omega]_\times), \tag{3}$$

$$t_j^k = t_j^{k-1} + \delta\tau, \tag{4}$$

where $\delta\omega \in \mathbb{R}^3$ is the infinitesimal rotation, and $[\cdot]_\times$ denotes the cross-product matrix. $\delta\tau$ is the translation increment.

Subtracting two consecutive frames yields the linearized marker displacement:

$$\Delta t_{mi} := t_{mi}^k - t_{mi}^{k-1} \approx \delta\tau + [\delta\omega]_\times r_i. \tag{5}$$

Eq.5 is the standard first-order model of rigid-body motion: each displacement of marker equals a common translation plus a tangential shift induced by the rotation about the joint, which immediately yields two estimators.

The rotation term varies with r_i, whereas the translation $\delta\tau$ is identical for all adjacent markers. Therefore, taking a weighted average of Δt_{mi} yields the common translation component.

$$\hat{\delta\tau} = \frac{\sum_i w_i \Delta t_{mi}}{\sum_i w_i} = W_t^k * \Delta t_m. \tag{6}$$

By fitting a bias term t_b to compensate for the deviation introduced after removing each rotational component, the offset of the current joint can be expressed as:

$$t_j^k = W_t^k * \Delta t_m + t_j^{k-1} + t_b. \tag{7}$$

Given direction $u_i^k = R_j^k r_i / \|r_i\|$, we can derive the following expression based on the above formulations:

$$(u_i^k - u_i^{k-1}) + \frac{t_b}{\|r_i\|} = -[u_i^k]_\times \, \delta\omega_k. \tag{8}$$

Denoting the left-hand side of the above equation as ΔR_m^k, it can be obtained through the rotation of the joint-marker vector. By performing linear regression over all vectors, the small rotation increment can be estimated as:

$$\hat{\delta\omega}_j^k = W_R^k \, \Delta R_m^k. \tag{9}$$

The rotation of joint can thus be expressed as:

$$R_j^k = \exp\left([\hat{\delta\omega}_j^k]_\times\right) R_j^{k-1}. \tag{10}$$

To capture the relationship between joints and markers in the above formulation, we design two neural networks to estimate the weights and biases (3.2). These networks are trained to learn the motion patterns between neighboring joints and markers from large-scale data.

To intuitively illustrate the motion transformation, we visualize two representative frames in Fig. 4 and interpret their correlation based on the above formulations. The elbow joint state at frame F^k is computed based on the transformation of its previous state at frame F^{k-1}. Between these two frames, the joint motion T^k consists of both translation and rotation. The observable information originates from the positional variations of the markers adjacent to the elbow joint. By learning two types of joint-marker association weights, W_t^k corresponding to translational movement and W_R^k corresponding to rotational movement, the system can rapidly infer the joint state (t_j^k and R_j^k) from the observed marker dynamics.

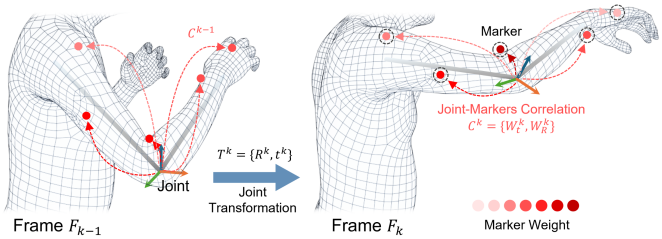


Fig. 4: **Joint-marker correlation.** The case illustrates the pose changes of the elbow joint with its associated markers during a lifting motion. The joint-marker correlation is represented by a dashed line. Darker markers represent higher weights in the correlation.

Additionally, to address the occlusion scenarios, Joint Tracker rebalances the weights of unobscured markers for each joint. This is achieved by reallocating the weights of the occluded markers as negative values, proportional to the weight ratios of the remaining visible markers. This weight adjustment is capable of controlling the error within a reasonable range in a short period of time.

3.2 Online Learning-based Tracker Generalization

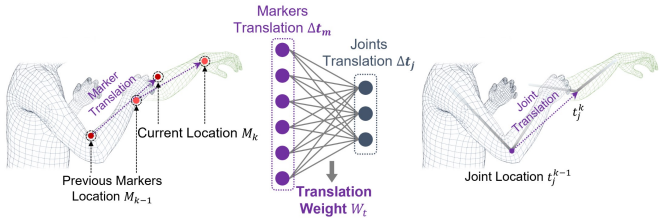
To fully leverage the advantages of data-driven methods learning for prior knowledge of human motions, we design a weight learner and complete its pre-training. The weight parameters of this learner, W_t and W_R, are preloaded in Joint Tracker. Following the concept of knowledge distillation from model compression, these two sets of weight parameters will undergo online learning and would be adaptively transmitted to Joint Tracker during system operation.

To accommodate local devices with varying computing capabilities and to allow for rapid iterative fine-tuning, we use a single fully connected layer as the primary model. As shown in Fig. 5, the model used for translation calculations takes the translation of markers between two adjacent frames as input, and it outputs the translation of joints. The model used for rotation calculations receives the rotational changes of vectors, formed by markers and joints between two adjacent frames, and it outputs the rotational changes of joints.

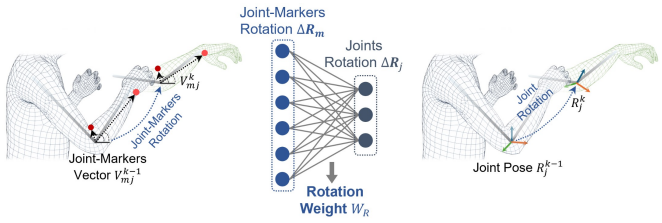
3.2.1 *Translation Network*

M_{k-1} and M_k represent the three-dimensional locations of markers in frames k-1 and k, respectively. Also, the displacement of markers' positions between the two frames is calculated and denoted as Δt_m. The output of translation network is the location displacement of the joints from frame k-1 to frame k. By inputting the known three-dimensional location of the joints at frame k-1, denoted as t_j^{k-1}, the joint location in frame k, t_j^k, can be obtained.

To enhance the robustness of network and to improve the smoothness of the generated results, we input a sequence of ten consecutive frames (N_f) to the network, and t_j^{k-1} for the



(a) Translation Learner



(b) Rotation Learner

Fig. 5: **Online learning-based tracker generalization.** We utilize two fully connected neural networks to model the linear relationships between changes in marker locations and joint pose transformations. (a) The translation learner receives marker translations as inputs and predicts skeletal translations as outputs. (b) The rotation learner processes the rotation of vectors between joints and markers, delivering predicted skeletal rotations. Both networks are trained using supervision signals sourced from the motion capture model's outputs. The translation and rotation weights, distilled in real-time, are employed to update the motion tracker.

last nine frames are the generated results. The designed loss is as follows:

$$L_t = \sum_{i}^{N_f} \sum_{j}^{n_j} \boldsymbol{\varphi}(\hat{t}_j^i, t_j^i), \tag{11}$$

where n_j denotes the number of joints, and $\boldsymbol{\varphi}(\hat{t}_j^i, t_j^i)$ represents the Euclidean distance between the joint locations generated by the network and the actual joint locations. By optimizing network parameters through continuous training iteration, the translation network can learn the best way to express joints translation using marker movements.

3.2.2 Rotation Network

The rotation changes of a single joint are actually related to some nearby markers. We have established a list of nearby relevant markers for each joint, and we design a rotation network to explore how the location changes of these nearby markers reflect the rotation changes of a specific joint.

As the Fig. 5 shown, using the three-dimensional locations of markers m in the nearby list of joint j, and the locations of joint j in frame k-1 and frame k, vectors V_{mj}^{k-1} and V_{mj}^{k} can be calculated from joint j pointing to markers m respectively. Quaternion ΔR_{mj} which represents the rotation of vectors formed by the joint-markers from frame k-1 to frame k, then can be estimated. The specific approach is: (i) Calculating the angle θ between the two paired vectors in adjacent frames and determining the rotation axis $\mathbf{n}$. (ii) Obtaining the rotation between the two vectors and expressing the rotation as a quaternion. The formula is:

$$\Delta R_{mj} = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(n_x i + n_y j + n_z k), \tag{12}$$

where n_x, n_y, n_z are the coordinate components of the vector $\mathbf{n}$, and i, j, k are the imaginary unit components of the quaternion ΔR_m.

The rotation quaternions constructed from different paired vectors serve as inputs to the network, and its output is the quaternion representation ΔR_j, which is the joints' rotation between two adjacent frames. Combining ΔR_j with the known joint Pose R_j^{k-1} from frame k-1, the relative rotation of joints R_j^k comparing to their initial state can be calculated for frame k.

Similar to the design of translation network, the input to rotation network includes ten consecutive frames (N_f), and R_j^{k-1} for the last nine frames are computed using generated results. To align with the edge model's output format for rotations, the quaternion is reconstructed as a rotation matrix.

The loss is designed as the angular error between two rotation matrices, expressed by the following formula:

$$L_R = \sum_{i}^{N_f} \sum_{j}^{n_j} \cos^{-1}\left(\frac{\mathrm{tr}(\hat{R}_j^i R_j^{iT}) - 1}{2}\right), \tag{13}$$

where n_j denotes the number of joints. $\hat{R}_j^i$ denotes the rotation matrix output by the network for the j-th joint in the i-th frame, and R_j^{iT} represents the transpose of the actual rotation matrix for the j-th joint in the i-th frame.

3.2.3 Adaptive Update of Weights

The single fully connected layer has limited capacity. It is only able to learn linear relationships between inputs and outputs, which may result in insufficient generalization capability. Knowledge Distillation is a model compression technique primarily used to transfer the knowledge of a large, complex model (called the "teacher model") to a smaller, more efficient model (called the "student model"). This method helps the smaller model learn the performance capabilities of the larger model while maintaining a compact size and high efficiency.

Learning from the insight above, we use the edge motion capture model, which utilizes a large number of parameters to learn human motion knowledge, as the teacher model, and leverage its outputs as ground truth to fine-tune the weight learner. Through knowledge transfer, the weight learner can adjust the fitting function in real time and update the weight parameters. The weight learner continuously learns the representation of newly input human motions from the teacher model, which has the excellent generalization ability. And it transmits the newly acquired online knowledge to the Joint Tracker through weights updates.

It is worth noting that, due to insufficient data for initial fine-tuning, the parameters in weight learner are not stable enough, and the weights in Joint Tracker would not be updated immediately. When the results from the fine-tuned network show less error compared to the results from original network, the transmission and update of weights to the local tracker will be triggered.

3.3 Event-responsive Motion Synchronization

To quickly obtain accurate joint motion calculated by the edge server, the local device continuously sends the locations of markers from newly arrived frames to the edge

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822

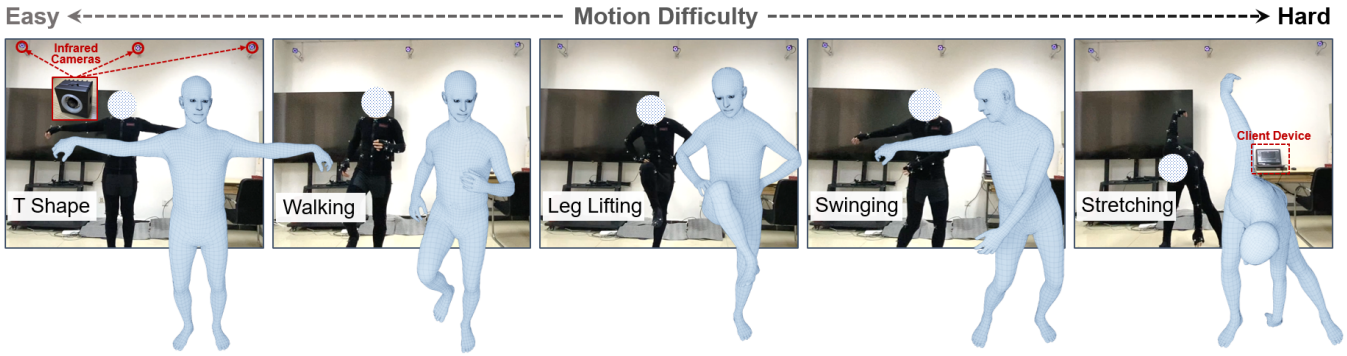


Fig. 6: **Experimental Setup and Qualitative Results:** Eight infrared cameras encircling the laboratory are utilized to capture markers attached to the motion capture suit. The markers locations was transmitted to client devices and collaborated with remote edge devices for real-time motion capturing. The human body mesh illustrates edgeMoCap's real-time motion estimation performed by the same user, with the difficulty of the motions (i.e., the extent of markers occlusion) progressing from left to right.

in a data stream. Due to limited computing capabilities at the edge side, newly arrived frames are stored in a buffer, awaiting the completion of the current processing round in the motion capture model. However, in marker occlusion scenarios, it is difficult for local tracker to maintain satisfied performance due to marker information loss. To better address the local tracker's timely needs for generated results from edge in occlusion scenarios, we designed a scheduler that responds to occlusion events in human motion.

Event #1: Marker occlusion. Due to factors such as the insufficient number or improper deployment of infrared cameras, and the complexity of human movement, markers may be occluded during motion, which can result in incomplete input marker data. However, Joint Tracker relies on the locations of all markers, according to the initial weights, to track the translation and rotation of joints.

To quickly adapt to tracking scenarios with missing markers, the scheduler issues an instruction to the edge to update the weights. This instruction guides the edge server to rapidly fine-tune a new set of adaptive weights in weights learner. The newly learned weights will be transmitted to the local Joint Tracker to complete the weight update.

Event #2: Reappearance of occluded markers. When an occluded marker reappears with a new marker ID, Marker Tracker cannot identify and track the marker due to the absence of label information corresponding to that marker ID in the historical frames. Also, Joint Tracker cannot use data from the unidentified marker.

To quickly identify label of the marker, the scheduler issues a rapid solving instruction to the edge server. Upon receiving this instruction, the server terminates current inference process and directly perform inference for the batch containing the newly arrived frame. Simultaneously, the server readjust the weight learner for a rapid fine-tuning. The outputs of the edge model and the updated weights can be quickly transmitted and applied to the local Tracker.

In conjunction with Sec.3.2.3, we decide when the local tracker should refresh its regression weights with an event-driven, threshold-based scheduler. Upon the arrival of an edge result, the client computes the discrepancy between (i) the pose previously produced by the local tracker and (ii) the newly returned pose from the edge. A weight refresh is triggered when this discrepancy exceeds a preset error threshold.

This design has two key benefits. First, human motion and marker visibility changes are highly bursty and non-periodic. Updating only when the measured error spikes allows the system to react immediately at those moments, instead of refreshing on a fixed schedule that may come either too late or when nothing has changed. This also avoids unnecessary bandwidth consumption and avoids interrupting the edge pipeline with needless parameter switches. Second, the trigger is not based on the self-estimated reliability signal from the edge model, but on an externally validated discrepancy: we directly compare the client prediction against the latest corrected result from the teacher model in joint space. This gives an objective signal of actual tracking drift.

TABLE 1: Details of Data Collection in Different Motion Difficulties

Motion Difficulty	No.of User	No.of Motion	No.of Frames	Markers Occlusion (Avg.)
Easy	6	15	47,141	0
Medium	4	12	36,415	2
Hard	4	9	30,530	5

4 Evaluation

In this section, we first present the experimental methodology (4.1), followed by the overall performance of edgeMoCap (4.2). We demonstrate the robustness of edgeMoCap (4.3) by testing edgeMoCap in different application scenarios and conduct ablation studies (4.4) to illustrate the effectiveness of different modules.

4.1 Experimental Methodology

Evaluation Datasets. We conduct our experiments using one synthetic dataset and two real datasets, each featuring characters with slightly varied marker configurations due to inaccuracies in marker placement. The synthetic dataset is generated from CMU MoCap dataset [31] using SMPL body model [32], and one of the real dataset is from SOMA [30].

Besides that, we also collected a new dataset in our experiment setting. The evaluation spanned three months, with about 2 hours of parallel client operation per day during data collection. As shown in Table 1, 6 users with various body shapes attended the experiments, and they were asked to perform 36 distinct motions. Based on the complexity of human movements, we classified motions

into three categories by difficulty: easy (e.g., walking, clapping), medium (e.g., sitting down, throwing), and hard (e.g., dancing, jumping).

We follow the occlusion strategy adopted in MoCap-Solver for the CMU test data. In the experiments, a total of 56 markers are used. To evaluate system performance under occlusion, 10% of the markers are randomly masked in separate trials. To better approximate real-world long-term occlusion, each occluded marker remains invisible for 100 consecutive frames. In real-world datasets, marker occlusion mainly depends on the self-occlusion of actors during motion capture.

Experiment Setting. Fig. 6 shows the experimental scenarios in a laboratory. We set up 8 infrared cameras with 60 fps to cover different perspectives of the scene. The actor wears a motion capture suit that fits closely to the markers and completes several movements with different motion difficulties in the space. The local device we use in the experiments is equipped with Apple M3 Pro chip, and the edge server is equipped with GeForce RTX 2080 Ti with 11GB memory.

Evaluation Metrics. (i) End-to-end latency. In real-time applications, it is crucial to focus on the errors stemming from computational delays, which can result in rendering outdated poses. Therefore, we define end-to-end latency as the production time interval between the current received frame from cameras and the current rendering frame from system to assess the computational delays of systems. (ii) Real-time accuracy. We use both translation vector and rotation matrix to represent a 6-DoF human joint. The translation error (in mm) is calculated by joints location distance, and the rotation error (in $^\circ$) is calculated by the smaller angle between the two rotation matrices. Also, the error is computed between the current received frame and the current rendering frame to assess the overall performance of systems.

Baselines. Considering both the performance of models and the capability to process quickly in markers occluded scenes, we chose SOMA [30] as the method for markers labeling and MoCap-Solver [15] as the method for joints solving on the server side. Within the proposed framework, any robust DNN-based optical motion capture model can serve as the teacher model for knowledge distillation. MoCap-Solver follows an autoencoder (AE) architecture. The encoder learns latent representations of skeletal structure, marker configuration, and temporal motion dynamics, while the decoder reconstructs clean marker trajectories and joint sequences from these latent variables. Each module is implemented using linear layers, and the model contains approximately 374.4 MB of trainable parameters.

Furthermore, based on the same network conditions setup, we compare the system performance with MoCap-Solver, Mosh++ [16] and OpenMoCap [33] in real-time manner.

4.2 Overall Performance

4.2.1 Real-time accuracy

Fig. 7a depicts the translation error of the proposed edgeMoCap as well as two other comparative systems. As shown, the average translation error of edgeMoCap is 38.8 mm, which outperforms MoCap-Solver by 77.6%, and exceeds Mosh++ by more than 93.7%. Furthermore, the 90th percentile accuracy outperforms these systems by 76.4%, and 92.7%, correspondingly. These results highlight the substantial enhancement in translation accuracy achieved by our system. Compared with OpenMoCap, edgeMoCap further reduces the average translation error by 70.4%. Although OpenMoCap claims to achieve real-time MoCap reconstruction, its deployment is limited by the computational capacity on the client side. Considering both the network transmission delay and the edge inference latency (approximately 40 ms per frame), such architectures cannot sustain high-frame-rate motion capture in practice.

TABLE 2: Modules Latency Analysis

Modules	Latency (ms)	Fluctuation (ms)
Marker Tracker	0.1	0.07
Joint Tracker	5.25	1.2
Scheduler	2.89	26.1
Transmission	56	574
Motion Capture	251	43
Weights Learner	24	28

Fig. 7b demonstrates the rotation error of the proposed edgeMoCap, again in comparison with MoCap-Solver and Mosh++. The average rotation error of edgeMoCap is 8.8°, whereas MoCap-Solver records an error of 14.1°and Mosh++ shows an error of 25.8°. Compared with OpenMoCap, which reports an average rotation error of 11.89°, edgeMoCap achieves an additional 25.9% improvement in rotation accuracy.

Fig. 6 shows the qualitative results of edgeMoCap. The difficulty of motion from easy to hard, while edgeMoCap tracks the user's motion can output a accurate pose in real-time. Above theoretical and empirical results demonstrate edgeMoCap achieves remarkable performance gains based on the proposed fast and effective tracker and the edge-assisted architecture.

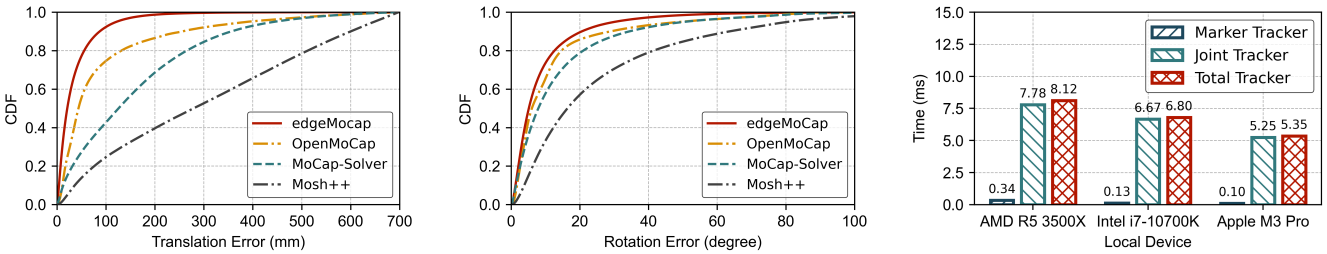
4.2.2 End-to-end Latency

We collected time consumption for each module on SOMA dataset, and calculated the average cost for each frame. The results are shown in Table 2. edgeMoCap adopts a two path architecture. The client side path, consisting of the Marker Tracker and the Joint Tracker, produces real-time pose estimates with end to end latency below 6 ms, while each frame is concurrently transmitted to the edge for model-based motion solving. The returned solving results correct historical states on the client side, indirectly improving the accuracy of the current frame without blocking or delaying the client loop.

Upon data arrival, the scheduler evaluates the error between historical client estimates and newly returned results, handles occlusion events, and decides whether to trigger a weight update. The average latency is 2.89 ms. Dispatching the update to the edge can cause minor timing fluctuations, yet it does not affect the critical path, and the tracker maintains real-time operation.

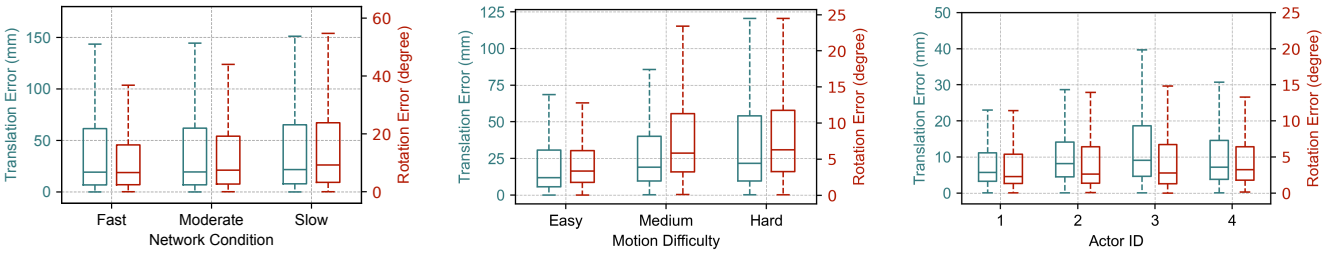
For weights learner, it takes the generated results from motion capture model as ground truth, and the fine-tuning process lasts for 3 epoches in each round. Due to its simple network architecture and the relatively small amount of training data, weights learner costs about 24 ms for an translation and rotation weights update, which demonstrates that well-designed weights learner in edgeMoCap runs in an efficient way.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822



(a) Translation Accuracy Comparison (b) Rotation Accuracy Comparison (c) End-to-End Latency

Fig. 7: Overall Performance



(a) Impact of Network Conditions (b) Impact of Different Motions (c) Impact of Different Actors

Fig. 8: System Robustness Evaluation

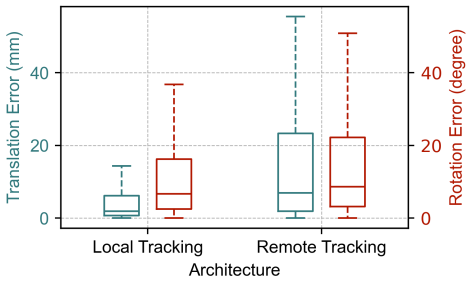


Fig. 9: Effectiveness of Architecture

4.2.3 Resource Utilization and Energy Profiling

During real-time tracking on a MacBook Pro (M3, 18 GB), the client-side process shows an average Energy Impact of 8.52, measured by macOS Activity Monitor. This value corresponds to low-to-moderate energy consumption compared to typical multimedia workloads (Energy Impact $>$ 30). The client-side tracker maintains an average CPU utilization of 97.6% on a single logical core, corresponding to approximately 9% of the total 11-core capacity of the M3 Pro. This confirms that the module occupies roughly one core under full load, leaving substantial computational headroom for concurrent rendering or communication tasks. We further performed a client side deployment on the mobile device (iPhone 15 Pro). The system maintained stable operation, reporting 0.84% memory usage and 17.2% CPU utilization.

4.3 System Robustness Evaluation

We conduct several experiments to test the performance and the generalization of edgeMoCap in different experimental conditions.

4.3.1 Impact of Network Conditions

To verify the tracker's tracking ability under different transmission delays, we test edgeMoCap in fast, moderate, and slow network conditions respectively and evaluate the real-time motion capture accuracy. As shown in Fig. 8a, the mean translation error of edgeMoCap on the synthetic dataset was 7.0 mm, 7.14 mm, and 7.59 mm, respectively; the mean rotation error is 11.5°, 15.3°, and 18.27°, respectively. The average network latency is 34.2 ms in the fast environment and 59.3 ms in the moderate environment. It should be noted that under slow network conditions, the transmission delay can occasionally approach 1 s.The results indicate that as the transmission delay increases, the local tracker can still effectively track joint locations. For the relatively complex rotation tracking, the local tracker can also keep the error within 20°.

4.3.2 Impact of Different Motions

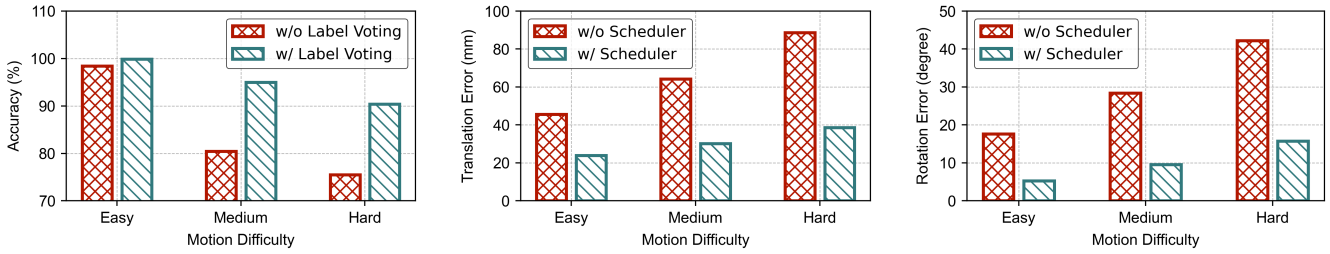
Using edgeMoCap to track human movements of varying difficulties, the error results are shown in Fig. 8b. As the difficulty of the actions increased, the corresponding average translation errors are 27.96 mm, 29.67 mm, and 36.79 mm, and the average rotation errors are 7.39°, 8.57°, and 8.66°. From these results, it can be seen that the translation error difference between simple and difficult actions is less than 1cm, and the rotation error difference is about 1°. This experiment demonstrates the robustness of the system in capturing movements of different difficulty levels.

It should be acknowledged that the learning of edgeMoCap's tracker is based on the smooth changes between adjacent frames. If the motion is extremely difficult, which disrupts the learned temporal correlation, the experimental results can be affected in a short time. However, because of the designed online-learning strategy, the errors can then be rectified and the overall performance can be kept in a satisfied level.

4.3.3 Impact of Different Actors

To verify the system's generalization ability to human movements of different body types, we carefully selected four participants with varying body types to perform easy motions for this experiment. As shown in Fig. 8c, the results indicate that the average translation error is below 25 mm, and the average rotation error is below 10°. The edgeMoCap's

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822



(a) Label Voting (b) Scheduler (Translation Error) (c) Scheduler (Rotation Error)

Fig. 10: Ablation Study

robustness and adaptability ensure consistent performance across a diverse range of users, further validating its effectiveness in real-world applications. The edgeMoCap's generalization performance across actors with different body types benefits from the weight updates, which we discuss later in ablation study.

4.3.4 *Impact of Different Client Devices*

Due to the simplicity and efficiency of the tracking process, edgeMoCap does not have specialized high demands on the client device. As shown in Fig. 7c, we conducted tracking experiments under different local device conditions (AMD R5 3500X, Intel i7-10700K and Apple M3 Pro), and the tracking process is completed in approximately $10ms$ in each case. Moreover, the tracking process can operate in less than $6ms$ on the local device with advance computing capability. In fact, with the experimental setting of motion capture at 60fps, the tracking process completing within approximately $16ms$ is enough for real-time applications. These experiments demonstrate that the system can operate effectively on local devices with varying computational capabilities.

4.4 Ablation Study

We conduct several experiments to evaluate the effectiveness of edgeMoCap's architecture, implementation, and algorithms.

4.4.1 *Effectiveness of Architecture*

To evaluate the effectiveness of the collaborative architecture that runs the tracker locally and optimizes with the results from edge, we design an experiment based on a synthetic dataset. The comparative architecture, called "Remote Tracking", is designed to reduce the time required to transmit the results of edge model back to the tracker. It operates by running the tracker directly on the edge server, and the calculated results are then transmitted back to the local device for rendering.

The comparison results are shown in Fig. 9. The Local Tracking approach achieved an average translation error of 7.0 mm and an average rotation error of 11.53°, whereas the Remote Tracking approach has an average translation error of 20.23 mm and an average rotation error of 17.51°. In other words, the results of Local Tracking surpass the results of Remote Tracking for more than 65% and 51% respectively. These results emphasize the importance of client-edge collaboration, which leads us to fully utilize the computational resource in local devices. It also underscore the pivotal role of the designed edge-assisted architecture in edgeMoCap.

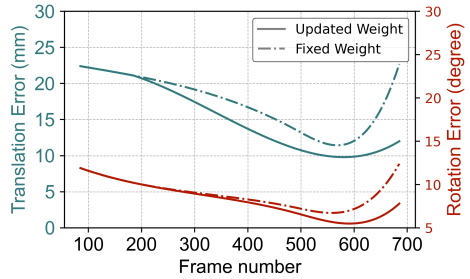


Fig. 11: **Effectiveness of Weights Update.** After about 580 frames, human motion transitions from simple actions such as standing and walking to the challenging action of squatting.

4.4.2 *Effectiveness of Label Voting*

As shown in Fig. 10a, "w/o Label Voting" indicates that Marker Tracker directly uses the labels of the markers from the last frame to label markers with the same ID in the current frame. Since the results of label recognition accuracy are closely related to the motion difficulty, we design the experiments in three comparing groups based on the actions. The figure illustrates that with the Label Voting module, the accuracy of marker labeling by Marker Tracker increased by approximately 1.5%, 18% and 19.7% respectively, with an average accuracy of 95.08%.

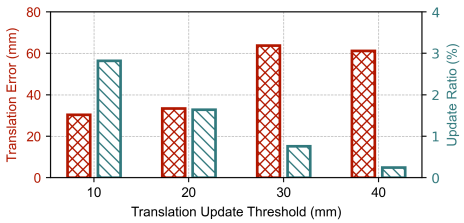
The results demonstrate that label voting improves the effectiveness of recognizing marker labels by referencing all possible previous frames rather than just a single frame. And it also indicates that the model has a better performance on recognizing easy motions. Therefore, for application usage, we strongly recommend users to start from the easy pose and collect the precise historical frame results before motions.

4.4.3 *Effectiveness of Scheduler Synchronization*

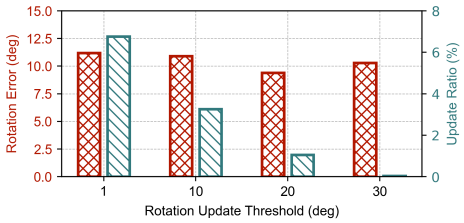
The design of the scheduler primarily addresses the issues of adaptability to information loss in occlusion scenarios and the integration and utilization of recovered information. Therefore, based on the difficulty of the motions, i.e., the number and frequency of occluded markers, we design experiments to verify the effectiveness of the scheduler. The results are shown in the Fig. 10b and Fig. 10c.

As the difficulty of motion increases, both the average translation error and the average rotation error increase. However, the system using the scheduler can keep the average translation error within $40mm$ and the average rotation error within 20°. In contrast, the system without the scheduler experiences a significant increase in errors.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822



(a) Translation



(b) Rotation

Fig. 12: Parameters Study

The translation error reaches more than 80mm, and the rotation error reaches more than 40°. By promptly update the Tracker when markers are occluded, and timely utilize the information when the markers reappear, scheduler helps edgeMoCap better cope with occlusions.

4.4.4 *Effectiveness of Weights Update*

Considering that using motion capture might introduce errors due to different body types and imprecise placement of markers, we applied the weights pre-trained on a synthetic dataset as the initial weights for Joint Tracker, and test the system operation on SOMA dataset. We compared the error change curves without weight updates and with continuous weight updates.

As shown in Fig. 11, in terms of translation error, during the initial phase, due to insufficient testing data, the fine-tuned weights are not applied to the local Joint Tracker. After approximately 200 frames, the weights used in the local Joint Tracker were continuously updated with the parameters fine-tuned at the edge server.

The fine-tuned weights adaptively adjusted the weight distribution according to the placement of markers. Therefore, the system with weight updates shows a significant decrease in both types of errors comparing with fixed weights, and the error increase slower when handling difficult actions.

4.5 Parameters Study

To assess the effect of local tracker update frequency, we evaluated translation and rotation errors under different thresholds. We report the update ratio (number of updates / total frames) and the mean errors in Fig.12. As the translation threshold increases, the update ratio drops markedly while the translation error grows, confirming that timely weight updates are beneficial. Notably, the translation error rises sharply between 2 cm and 3 cm, so we adopt 2 cm as the translation threshold. Interestingly, when increasing from 3 cm to 4 cm, the translation error slightly decreases, likely due to bandwidth consumptions, where the minor gain in update frequency cannot offset the accumulated correction latency introduced by multiple small update operations.

In contrast to translation, the rotation error is less sensitive to the update ratio. To probe why, we logged the gradient norms of the edge-side weight learner under different angular thresholds. At a 1°threshold, the median gradient norm is on the order of 1e-2, suggesting that tiny angular perturbations provide limited learning signal. At a 20°threshold it increases to 1e-1, indicating sufficiently informative updates. Beyond 20°, overall performance degrades, consistent with an insufficient update frequency. We therefore set 20°as the rotation update threshold.

4.6 Evaluation under Challenging Scenarios

4.6.1 *Network Disconnection*

To further evaluate the performance of edgeMoCap under network disconnection, we test various motion types and present the results as shown in Fig.13a and Fig.13b. Once the local device receives the first motion-solving result returned from the edge, the network connection is immediately terminated. The subsequent frame-by-frame inference is then completed locally based on this initial result.

Across the entire CMU test set, the method achieves an average translation error of 11.26 cm and an average rotation error of 29.50 °. Two representative motion types are selected from the test set: simple walking and complex ballet dancing. In the first few frames, since the local device needs to wait for the edge results, the translation error is initially larger for these two motion types involving rapid movements. After receiving the first edge-returned frame, the subsequent fast local solving quickly adapts to the rapid motion, allowing the position error to stabilize within 2 cm.

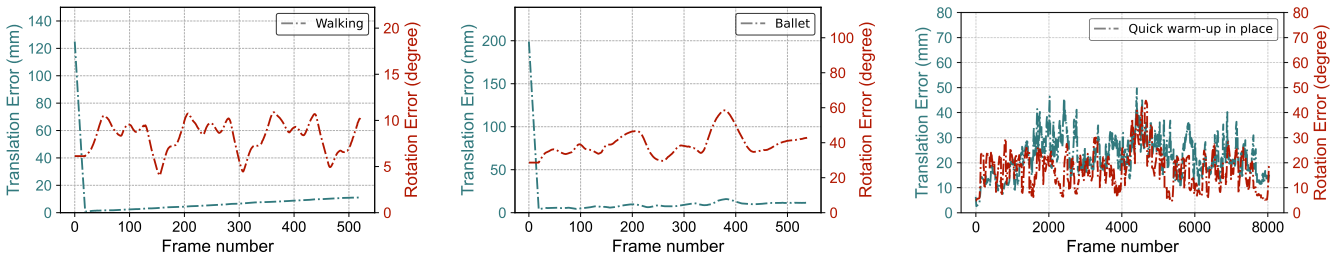
During the initial second of most MoCap sequences, the human body rarely undergoes drastic joint rotations, resulting in a relatively stable error at the beginning of the sequence. For simple motion patterns such as walking, the overall rotational error remains moderate, typically within 10°. When the motion involves non-standard or biomechanically extreme postures, such as single-leg standing with backward leg extension in ballet, the rotational error can surge to as high as 60°.

For relatively simple motions, the client-side solver is sufficient to maintain accurate rotation estimation. However, for motions involving large joint rotations and complex postures, rotation estimation becomes a more challenging task and depends more critically on temporally adjacent, accurate feedback from the edge-side solver. Overall, as a fast solver operating on the client side, the tracker maintains acceptable accuracy within a short time window.

4.6.2 *Long-duration Test*

To assess robustness under prolonged operation, we conducted a real-world MoCap session lasting approximately two hours and report a time slice from the final several minutes in Fig. 13c. The subject began from a static T-pose and then performed a series of large-amplitude warm-up movements in the capture space (e.g., rapid arm swings, torso twists, and badminton-style swings). As the motion complexity increased, the error exhibited modest temporal fluctuations, with a pronounced spike around frame 4000 during a rapid lateral bend with waist flexion. Overall, the error remained at a low level. Benefiting from the edge results that are continuously returned to refine local estimates, the system sustained stable, long-duration operation.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822



(a) Network Disconnection (Easy Motions) (b) Network Disconnection (Extreme Motions) (c) Long-duration Test (Quick Motions)

Fig. 13: Challenging Scenarios

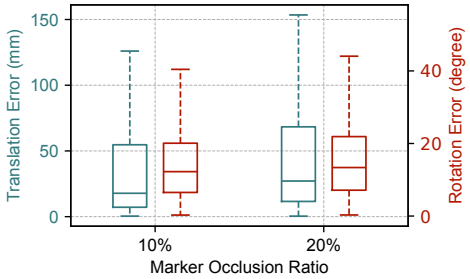


Fig. 14: Extreme Marker Occlusion

4.6.3 *Extreme Marker Occlusion*

To further evaluate system robustness in an extreme scenario, experiments are also conducted with a 20% marker occlusion ratio on CMU test data. The results are summarized and illustrated in Fig.14. With increasing marker occlusion, the rotational error in joint estimation remains relatively stable, while the positional error exhibits a more significant increase. Even under the extreme case of 20% marker occlusion, the estimated positional deviation for most motions remains within 10 cm, indicating that the system can still achieve stable performance under challenging conditions.

5 Related Work

In this section, we review the work in related areas.

Optical-based Motion Capture. Marker-based optical motion capture is currently the most accurate among common modalities such as image-based (e.g., LiveCap [34], MotionBERT [35], HybrIK [36], LitePose [37]) and IMU-based (e.g., Mocap-Everyone-Everywhere [38]) approaches. Based on the type of prior knowledge, Optical-based motion capture methods can be categorized into evidence-driven approaches and data-driven approaches.

Evidence-driven methods are often based on heuristic rules of human motion, combining these rules with physical constraints to enhance motion capture techniques [39], [40]. Aristidou et al. [14] utilizes the high self-similarity of human motion data. Some methods are based on low-rank matrix completion [9], [10], [11]. Other methods based on Kalman filters [12], [13] observe and predict motions, assuming that the motions can be approximated by a Gaussian model. Although these methods can operate in real-time, they rely on certain assumptions to be effective, or they are mainly targeted at solving specific types of noise, which makes them unable to handle complex scenarios.

Data-driven methods can learn implicit prior knowledge from large amounts of training data. Ghorbani et al. [30] constructs an attention network to learn the relationships between markers and label markers accordingly. Mahmood et al. [16] continuously estimates the position of the human body in the next frame through parameter optimization. Other methods [7], [15], [33] use pre-labeled markers as input to solve for the skeleton, achieving high accuracy even in occluded scenarios. However, typical optical pipelines (e.g., OpenMoCap [33]) prioritize accuracy and do not guarantee strict real-time behavior under sustained high frame rate operation (typically $\geq$ 60 fps).

We target real-time optical motion capture on lightweight client devices. To the best of our knowledge, this work propose the first edge-assisted architecture tailored to optical MoCap. Whereas pipelines like OpenMoCap commonly rely on powerful GPUs or batch style processing, this design enables stable real-time execution on resource constrained clients, and the generated results can be efficiently rendered (e.g., VR-Pipe [41]).

Edge-assisted Paradigm for Real-time System. The edge-assisted paradigm creates opportunities for applications that demand real-time performance with limited computing capabilities and require scalable, one-to-many services [42], [43], [44], [45], [46], [47]. Chen et al. [48] assess the latency performance of different edge computing applications. Some works [21], [49], [50] address computational challenges through efficient offloading strategies, which accomplishes a Simultaneous Localization and Mapping (SLAM) service for mobile devices. [51] designed an edge-assisted system to achieve high-precision real-time object detection on existing AR/MR systems. And there are some other works explore applying edge computing architecture in healthcare [52], [53], [54]. While edge-assisted frameworks have established effective paradigms for combining tracking and correction across local devices, these approaches cannot be directly applied to optical motion capture. Optical MoCap systems are inherently complex and large-scale, lacking a lightweight tracker capable of maintaining stable accuracy as in visual-inertial tracking scenarios.

Generalizable Robust Motion Capture. Motion capture applications should be able to accommodate different body types. One approach for generalization is using a standardized skeletal model and adjusting the length and proportions of bones parametrically. Skeleton retargeting [55] is a technique for mapping motion from one skeleton to another. By establishing correspondences between the source and target skeletons, motion data can be retargeted from one body to another. In deep learning, different body parameters are commonly used to represent human models [30], [32]. To improve the generalizability of edgeMoCap, we reference the knowledge distillation method from model

compression to adjust the parameters used in Tracker in an online-learning manner, which enables the system to provide robust services for different body types of users.

6 Conclusion

We have presented and implemented edgeMoCap, the first framework to enable existing motion capture models to provide real-time services at the edge. edgeMoCap (i) distills implicit human motion knowledge from DNN models into explicit joint-marker correlations, supporting lightweight clients to rapidly track skeletal motion locally; (ii) through meticulous tasks segmentation and data synchronization of motion capture and tracking, the proposed edge-client collaboration architecture ensures both real-time and accurate performance. Comprehensive evaluations over three months underscore its superior performance. edgeMoCap marks a pioneering effort to offload motion capture models to the edge, facilitating scalable and real-time services.

References

[1] K. Yamane and J. Hodgins, "Simultaneous tracking and balancing of humanoid robots for imitating human motion capture data," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009.

[2] M. Menolotto, D.-S. Komaris, S. Tedesco, B. O'Flynn, and M. Walsh, "Motion capture technology in industrial applications: A systematic review," *Sensors*, 2020.

[3] A. Elhayek, O. Kovalenko, P. Murthy, J. Malik, and D. Stricker, "Fully automatic multi-person human motion capture for vr applications," in *Virtual Reality and Augmented Reality: 15th EuroVR International Conference, EuroVR 2018, London, UK, October 22–23, 2018, Proceedings 15*, 2018.

[4] X. Yi, Y. Zhou, M. Habermann, V. Golyanik, S. Pan, C. Theobalt, and F. Xu, "Egolocate: Real-time motion capture, localization, and mapping with sparse body-mounted sensors," *ACM Transactions on Graphics (TOG)*, 2023.

[5] A. Chatzitofis, D. Zarpalas, P. Daras, and S. Kollias, "Democap: Low-cost marker-based motion capture," *International Journal of Computer Vision*, 2021.

[6] M. Naeemabadi, B. Dinesen, O. K. Andersen, and J. Hansen, "Influence of a marker-based motion capture system on the performance of microsoft kinect v2 skeleton algorithm," *IEEE Sensors Journal*, 2018.

[7] X. Pan, B. Zheng, X. Jiang, G. Xu, X. Gu, J. Li, Q. Kou, H. Wang, T. Shao, K. Zhou *et al.*, "A locality-based neural solver for optical motion capture," in *SIGGRAPH Asia 2023 Conference Papers*, 2023.

[8] T. Kämäräinen, M. Siekkinen, A. Ylä-Jääski, W. Zhang, and P. Hui, "A measurement study on achieving imperceptible latency in mobile cloud gaming," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, 2017, pp. 88–99.

[9] Y. Feng, M. Ji, J. Xiao, X. Yang, J. J. Zhang, Y. Zhuang, and X. Li, "Mining spatial-temporal patterns and structural sparsity for human motion data denoising," *IEEE transactions on cybernetics*, 2014.

[10] Y. Feng, J. Xiao, Y. Zhuang, X. Yang, J. J. Zhang, and R. Song, "Exploiting temporal stability and low-rank structure for motion capture data refinement," *Information Sciences*, 2014.

[11] X. Liu, Y.-m. Cheung, S.-J. Peng, Z. Cui, B. Zhong, and J.-X. Du, "Automatic motion capture data denoising via filtered subspace clustering and low rank matrix approximation," *Signal processing*, 2014.

[12] L. Li, J. McCann, N. Pollard, and C. Faloutsos, "Bolero: a principled technique for including bone length constraints in motion capture occlusion filling," 2010.

[13] A. Aristidou and J. Lasenby, "Real-time marker prediction and cor estimation in optical motion capture," *The Visual Computer*, 2013.

[14] A. Aristidou, D. Cohen-Or, J. K. Hodgins, and A. Shamir, "Self-similarity analysis for motion capture cleaning," in *Computer graphics forum*, 2018.

[15] K. Chen, Y. Wang, S.-H. Zhang, S.-Z. Xu, W. Zhang, and S.-M. Hu, "Mocap-solver: A neural solver for optical motion capture data," *ACM Transactions on Graphics (TOG)*, 2021.

[16] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, "Amass: Archive of motion capture as surface shapes," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019.

[17] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, 2016.

[18] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE access*, 2020.

[19] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, 2017.

[20] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual slam," in *IEEE INFOCOM 2020-IEEE Conference on computer communications*, 2020.

[21] A. J. Ben Ali, M. Kouroshli, S. Semenova, Z. S. Hashemifar, S. Y. Ko, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," *ACM Transactions on Embedded Computing Systems*, 2022.

[22] Y. Chen, H. Inaltekin, and M. Gorlatova, "Adaptslam: Edge-assisted adaptive slam with resource constraints via uncertainty minimization," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, 2023.

[23] B. P. Ortega and J. M. J. Olmedo, "Application of motion capture technology for sport performance analysis," *Retos: nuevas tendencias en educación física, deporte y recreación*, 2017.

[24] E. Van der Kruk and M. M. Reijne, "Accuracy of human motion capture systems for sport applications; state-of-the-art review," *European journal of sport science*, 2018.

[25] B. Jung, H. B. Amor, G. Heumer, and M. Weber, "From motion capture to action capture: A review of imitation learning techniques and their application to vr-based character animation," in *Proceedings of the ACM symposium on Virtual reality software and technology*, 2006.

[26] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[27] Y. Zhao, Z. Yang, X. He, X. Cai, X. Miao, and Q. Ma, "Trine: Cloud-edge-device cooperated real-time video analysis for household applications," *IEEE Transactions on Mobile Computing*, 2022.

[28] A. J. Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: edge-assisted visual simultaneous localization and mapping," in *MobiSys' 20: Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020.

[29] Z. Yang, X. Wang, J. Wu, Y. Zhao, Q. Ma, X. Miao, L. Zhang, and Z. Zhou, "Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision," *IEEE/ACM Transactions on Networking*, 2022.

[30] N. Ghorbani and M. J. Black, "Soma: Solving optical marker-based mocap automatically," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[31] CMU, "CMU Graphics Lab Motion Capture Database." http://mocap.cs.cmu.edu/, 2000.

[32] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "Smpl: A skinned multi-person linear model," in *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. Association for Computing Machinery, 2023.

[33] C. Qian, D. Li, X. Yu, Z. Yang, and Q. Ma, "Openmocap: Rethinking optical motion capture under real-world occlusion," *arXiv preprint arXiv:2508.12610*, 2025.

[34] M. Habermann, W. Xu, M. Zollhfer, G. Pons-Moll, and C. Theobalt, "Livecap: Real-time human performance capture from monocular video," *ACM Transactions on Graphics*, vol. 38, no. 2, pp. 1–17, 2019.

[35] W. Zhu, X. Ma, Z. Liu, L. Liu, W. Wu, and Y. Wang, "Motionbert: A unified perspective on learning human motion representations," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 15 085–15 099.

[36] J. Li, C. Xu, Z. Chen, S. Bian, L. Yang, and C. Lu, "Hybrik: A hybrid analytical-neural inverse kinematics solution for 3d human pose and shape estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 3383–3393.

[37] Y. Wang, M. Li, H. Cai, W. M. Chen, and S. Han, "Lite pose: Efficient architecture design for 2d human pose estimation," *arXiv e-prints*, 2022.

[38] J. Lee and H. Joo, "Mocap everyone everywhere: Lightweight motion capture with smartwatches and a head-mounted camera," *IEEE*, 2024.

[39] VICON, "Vicon Motion Systems," https://www.vicon.com/, 2024.

[40] L. Herda, P. Fua, R. Plankers, R. Boulic, and D. Thalmann, "Skeleton-based motion capture for robust reconstruction of human motion," in *Proceedings Computer Animation 2000*, 2000.

[41] J. Lee, J. Kim, J. Park, and J. Sim, "Vr-pipe: Streamlining hardware graphics pipeline for volume rendering," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 217–230.

[42] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, 2017.

[43] J. Xu, H. Cao, Z. Yang, L. Shangguan, J. Zhang, X. He, and Y. Liu, "{SwarmMap}: Scaling up real-time collaborative visual {SLAM} at the edge," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[44] F. Ahmad, H. Qiu, R. Eells, F. Bai, and R. Govindan, "{CarMap}: Fast 3d feature map updates for automobiles," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[45] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE conference on computer communications*, 2018.

[46] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 conference of the ACM special interest group on data communication*, 2018.

[47] L. Cheng, J. Wang, and Y. Li, "Vitrack: Efficient tracking on the edge for commodity video surveillance systems," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[48] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017.

[49] H. Cao, J. Xu, D. Li, L. Shangguan, Y. Liu, and Z. Yang, "Edge assisted mobile semantic visual slam," *IEEE Transactions on Mobile Computing*, 2022.

[50] D. Li, Y. Zhao, J. Xu, S. Zhang, L. Shangguan, Q. Ma, X. Ding, and Z. Yang, "Reshaping edge-assisted visual slam by embracing on-chip intelligence," *IEEE Transactions on Mobile Computing*, 2024.

[51] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th annual international conference on mobile computing and networking*, 2019.

[52] S. U. Amin and M. S. Hossain, "Edge intelligence and internet of things in healthcare: A survey," *Ieee Access*, 2020.

[53] M. Hartmann, U. S. Hashmi, and A. Imran, "Edge computing in smart health care systems: Review, challenges, and research directions," *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 3, p. e3710, 2022.

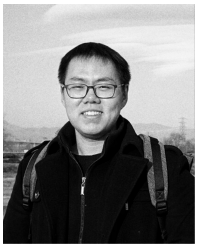
[54] L. Greco, G. Percannella, P. Ritrovato, F. Tortorella, and M. Vento, "Trends in iot based solutions for health care: Moving ai to the edge," *Pattern recognition letters*, vol. 135, pp. 346–353, 2020.

[55] A. Menache, *Understanding Motion Capture for Computer Animation and Video Games*. Understanding Motion Capture for Computer Animation and Video Games, 1999.

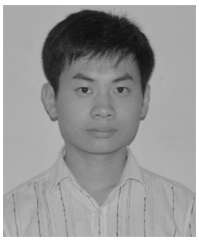
This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3656822

Chen Qian received the B.E. degree in School of Software from Dalian University of Technology in 2020 and the M.E. degree in School of Software from Tsinghua University in 2023. Now she is pursuing a Ph.D. degree in School of Software, Tsinghua University under the supervision of Prof. Zheng Yang.

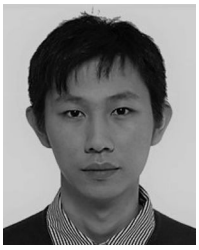
Danyang Li received the B.E. degree in School of Software from Yanshan University in 2019 and the M.E. degree in School of Software from Tsinghua University in 2022. He is currently pursuing his Ph.D. degree in School of Software, Tsinghua University. His research interests include Internet of Things and mobile computing.



Jingao Xu received his B.E. and Ph.D. degree in School of Software from Tsinghua University in 2017 and 2022, respectively. He is now a Postdoc research fellow in School of Software, Tsinghua University. His research interests include Internet of Things and mobile computing.



Zheng Yang is an associate professor at Tsinghua University. He received a B.E. degree in computer science from Tsinghua University in 2006 and a Ph.D. degree in computer science from Hong Kong University of Science and Technology in 2010. His main research interests include Internet of Things and mobile computing. He is the PI of National Natural Science Fund for Excellent Young Scientist and has been awarded the State Natural Science Award (second class).



Qiang Ma received the B.S. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, in 2013. He is currently an Assistant Researcher with Tsinghua University. His research interests include sensor networks, mobile computing, and data privacy.