Human Activity Recognition in Mobile Edge Computing: A Low-Cost and High-Fidelity Digital Twin Approach with Deep Reinforcement Learning

Chenyu Wang, *Member, IEEE*, Zhipeng Cai, *Fellow, IEEE*, Yingshu Li, *Senior Member, IEEE*

***Abstract*—The presence of digital twins (DTs) has expanded within the consumer electronics area due to its inherent benefits of high-fidelity modeling and predictive insight. However, DT deployment remains costly and constrained in providing the fundamental functionalities required by consumer electronics systems due to massive computation and communication. The emergence of mobile edge computing (MEC) has made DT deployment feasible to achieve data-driven modeling and consumer-centric concurrently with low communication costs. This article proposes an MEC-based DT deployment scheme in the smart home domain. The cloud service can predict subsequent sensor updates leveraging the MEC platform's human activity recognition result of residential environments. In addition, with deep reinforcement learning (DRL), it can track the essential data to maintain sensor update consistency between both physical and virtual sides. We implement experimental evaluation with two real-world residential datasets, and the results demonstrate that our scheme maintains high-level sensor update consistency while being energy-efficient in different residential environments, which illuminates the promising prospects of DT implementation in MEC and consumer electronics.**

***Index Terms*—Mobile edge computing, digital twin, deep reinforcement learning, human activity recognition, sensor inference**

I. Introduction

A hypothesis pursued by a few people revolves around the co-existence of other universes alongside ours. In these alternate universes, an event we recognize might have entirely different outcomes, and these hypothetical alternate universes are called *parallel universes*. Ever since its conception, the idea of parallel universes has inspired us to depict a different and better world and formalize the states and mechanisms of entities within our reach. Doing so encourages one to simulate and access all possible results through many decision checkpoints. Following the principle of parallel universe, something is called *digital twin* (DT) and is referred to as a digital replica of a set of living or non-living physical entities in our world.

A DT represents its physical twin as a minuscule abstraction of the world and stands in a small parallel universe. This is done by including all its physical twin components, which reveals the potential sub-system evolution in many applications of information technology, industry, and manufacturing sectors [1]. Furthermore, leveraging the economical and convenient DT setup allows more flexible approaches to simulate and validate our design and models and extend the settings in the real world. In this digital era, the DT technique has garnered massive attention due to its merits of being intelligent, efficient, economical and sustainable, and has been proposed to be integrated into many complicated and rigorous scenarios, such as the DT human [2], DT city [3], and DT visualization [4], which targets on enhancing the information convergence between the physical and digital spaces of the systems.

Nonetheless, the effectiveness of DTs is significantly constrained by physical limitations such as network conditions, computational capacity, and energy costs associated with the deployed facilities and equipment [5]–[7]. With the prevalence of ever-increasing Internet of Things (IoT) devices, the potential DT designs for these devices are even more restricted as they are impractical and inefficient in constantly formulating real-world conditions and updating the DT status of physical devices. To achieve improved synchronization, it is essential to consider the unique features obtained from the status of existing data while injecting insight into the future status. Furthermore, it is justifiable to develop dedicated models that work in conjunction with different IoT ecosystems, e.g., residential environments with a variety of human behavior and device data generation patterns [8]–[10]. One of the most critical points to maintaining the physical twin and DT consistency is facilitating timely updates of time-sensitive data that helps to prevent unnecessary resource wastage.

Thanks to the *mobile edge computing* (MEC) technique, which brings considerable computing resources closer to the place of data generation, we have gained the capability of addressing matters of change responsively. With the advancement of hardware, mobile edge devices tend to be more computatively designed and capable of dealing with massive local requests in the IoT context [11]–[13]. However, there is still a communication bottleneck between edges and the central server, colloquially called the *cloud*, which is known for being relatively cost-inefficient, insecure, and lagged in dealing with remote data [14]. Before pushing service requests to the cloud, the data preprocessing at the edge side would significantly reduce workloads and enable more concurrent threads in the cloud. The preprocessing strategies can be customized regarding edge servers and their users' physical environments,

This work was supported by the National Science Foundation (NSF), USA, under Grant 2315596, Grant 2244219 and Grant 2146497. *(Corresponding Author: Yingshu Li.)*

C. Wang, Z. Cai, and Y. Li are with the Department of Computer Science, Georgia State University, Atlanta, GA, 30303, USA (e-mail: cwang50@student.gsu.edu, zcai@gsu.edu, yili@gsu.edu).

thus enabling personalized DT constructions on the cloud site. The edge server is responsible for the preprocessing and securing mechanism, which obscures local features, enables essential information flow, and enhances synchronization [15], [16].

This article fully takes the characteristics of edge and cloud computing into consideration. Leveraging *human activity recognition* (HAR) results, we propose an MEC-assisted DT deployment scheme to increase DT fidelity regarding sensor update inference at a low cost. With the emerging *deep reinforcement learning* (DRL) technology, the edge servers deployed in residential environments can investigate the local caches of sensor readings and obtain residents' activity patterns. It is also trained to extract the essential features that facilitate the sensor update inference in the cloud server and decisions for DT updates, leading to highly consistent sensor update sequences in the cloud's virtual space and the residential environment's physical space while reducing the communication cost of cloud-edge synchronization.

The remainder of this article is organized as follows. We introduce the related work in Section II and present the system model and some key concepts, such as the data format and communication mechanism, in Section III. After being aware of an overview of the system, the setup and algorithms are clarified in Section IV. Finally, we evaluate the performance and effectiveness of our DT formation in Section V and conclude this article in Section VI.

II. Related Works

A. Sensor-based Human Activity Recognition

The proposition of DT humans probably enhances the future full lifecycle health management [17] and mobility service in transportation [18]. These functionalities are closely tied to human activity patterns. Developing a precise and realistic DT human requires dense and heterogeneous placement of advanced sensors and the collection of massive amounts of data. Sensor-based HAR is a feasible approach to deducing human behavioral patterns.

Broad literature has considered sensor-based HAR as an economical solution [19]. Pattamaset *et al.* [20] divided the indoor sensors into multiple groups to identify human activities based on the room distribution to reduce the irrelevant data upload. Bouchabou *et al.* [21] directly measured the changes in each sensor deployed in residential environments, converted the changes into vectors of feature, and trained a convolutional neural network model to identify human activities. Gumaei *et al.* [22] introduced a multimodal body sensory approach to realize human activity with a hybrid deep learning model.

Despite the primary domain of activity recognition in the edge server, the sensory-data-based services running in the cloud might target the sensor networks and their management and measurement. The existing methods [23]–[25] addressed the sensor inference by concentrating on correlation-based inference, causal representation domain adaption, or resorting to hidden Markov models; however, none of them interpreted sensor update inference conducted at the cloud server in the DT deployment context and connected it to regular HAR tasks in the edge server.

Our approach extends beyond pure sensor-based HAR solutions, demonstrating the application of low-cost MEC-based DT deployment. In this article, we utilize HAR and sensor update inference as an illustrative example to demonstrate the practicability of DT deployment in an MEC environment.

B. DT Deployment in MEC Environments

The advancement of MEC has enabled DT deployment to be more realistic and accessible for mobile end-users. Wang *et al.* [18] proposed a data-driven and cloud-edge-device system to facilitate the mobility DT creation by interconnecting various mobility entities. Fan *et al.* [26] proposed a vehicular lane-changing solution for connected autonomous vehicles with real-time safety property and foresight intelligence, where sensing and computing capability are enhanced in mobile edge servers. Experience of the physical MEC network is formulated in a DT format to support real-world decisions. Do-buy *et al.* [27] introduced a DT-aided design for industrial automation, which captures the real-time status of IoT devices and MEC servers and determines the offloading scheme that optimizes the end-to-end latency while subjecting to multiple quality-of-service and resource constraints. Zhou *et al.* [28] proposed a dual-reinforcement learning scheme with DT layers in an edge-cloud structured system to determine client node and global aggregation frequency for federated learning. The edge server has demonstrated notably powerful computation and communication capabilities in these scenarios. Nonetheless, the challenge remains in effectively and seamlessly integrating high-level DT with MEC-based interconnected services.

III. System Model

With real-world data, DTs stack various functional layers to the stunning virtual creation, acting beyond the traditional simulation technology and interacting with reality proactively. In this circumstance, a huge amount of data assimilation is essential to the DT construction, which is also greatly limited by infrastructures, such as network throughput and computational capacity. Specifically, in residential IoT services, the home environment is equipped with various sensors and devices, such as motion detectors, cameras, temperature-humidity sensors, and wearable devices, which serve as fundamental sources of information.

A cloud platform would assimilate the data generated from IoT sensors (e.g., HAR) in a variety of residential environments to create the DTs of specific domains (e.g., sensor update inference) for further usages, e.g., security setting adjustment, sensor management, and elderly care. A more frontier application is the DT incorporation into the metaverse to promote the virtual reality fusion experience with a broad range of sensors [29]. The data generated from many sensors of numerous residential cases can accumulate to a vast volume, posing significant traffic burdens on the cloud server.

Consequently, the formation of high-fidelity DT objects encounters critical difficulties in terms of accessibility and availability. As an intermediary, introducing edge servers can play a significant role in alleviating these concerns. The edge server manages and preprocesses the raw log of deployed sensors within a reachable distance of the home environment. It extracts and uploads only the essential features, guaranteeing that data utilization on the cloud site for DT construction is legitimate and efficient.

A. Problem Formulation

We assume a total of N IoT sensors are deployed in a home environment monitored by a dedicated edge server, and the status of each sensor can be captured in every timestamp indexed by t, denoted by

$$\mathbf{s}_t = (s_{1,t}, s_{2,t}, \cdots, s_{N,t}),\ t \in \mathbb{N}.$$

The edge server manages the status changes of sensors. A status change c is denoted as

$$c = (n, t, s_{n,t}, y_t),\ \forall s_{n,t} \neq s_{n,t-1},\ t > 0,$$

where y_t is the resident activity label of time t in an activity set $\mathcal{Y}$, and each change c is associated with an activity label. When serving the residential environment, the edge server maintains the logs of sensor status change sequence[1] as

$$\mathcal{C} = \left\{c_1, c_2, \cdots, c_T | t^{(c_1)} \leq t^{(c_2)} \cdots \leq t^{(c_T)}\right\}.$$

Taking HAR as an example, the edge server is responsible for tracking the status of sensors and directly responding to human activities, e.g., health and well-being assessments; DTs at the cloud server focus on extensive management purposes and enhanced services. In this article, we consider the inference of sensor update in DTs, i.e., $\mathcal{N} = \{n_{c_1}, n_{c_2}, \cdots, n_{c_T}\}$.

One important concept related to DTs, namely fidelity, is broadly introduced to measure the matching degree between virtual creations and physical entities. The DT fidelity is determined by how accurately the virtual model can represent actual sensor changes in the residential environment. Though the edge server can access the sensor status, uploading raw sensor status to the cloud is inappropriate considering the privacy issue [30] or domain difference between the edge and cloud server. Instead, sensor status can be passed to a feature extractor for further domain adaption before being uploaded to the cloud server. However, this compression in an extractor might hamper the fidelity of constructed DTs.

Another obstacle affecting the DT fidelity is the acquisition of data samples from edge servers. In many instances, continuous data sampling is disadvantageous when adding DTs on the cloud server due to the dense connectivity with diverse MEC environments. Moreover, some information is not critical to DT maintenance. As a result, when constructing DTs, a cloud server should selectively identify the contribution of potentially perceived changes in sensors at a large scale. This raises the requirement regarding intelligent decision-making of connection.

In essence, the demands for secure and selective data sampling underscore the importance of the semantic analysis capability of DTs in envisioning real-time data processing. This also ensures that sensitive information is appropriately

[1]This is referred to as "sequence" throughout the rest of this article.

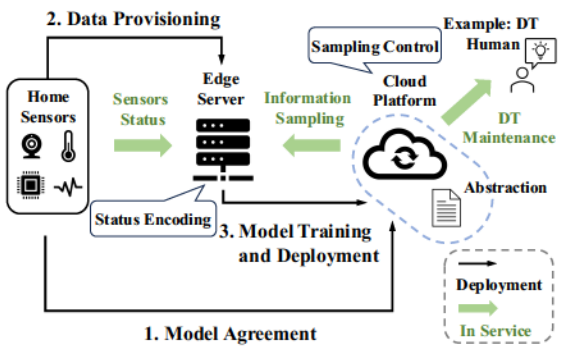

Fig. 1: The overview of the MEC system design.

managed and protected by the edge server. Our effort, with the advance of MEC and DRL, tackles the stated concerns effectively when deploying DTs.

B. System Design

We present an overview of the MEC system design in Fig. 1. As shown in the *model agreement* process of Fig. 1, **(1)** residents are required to consent to the construction of the DT in the cloud and the cloud's access to data in the MEC environment. **(2)** Residents permit the MEC server to collect datasets, including the status changes of sensors and their associated activity labels, and train the models for smart home responses. MEC server proceeds *data provisioning* to aid the DT construction in the cloud, in which sensitive information has been removed. **(3)** In the *model training and deployment* process, the edge server trains a model to transform the status change log into abstraction, i.e., *feature extractor*, and another model that contextualizes the abstraction within the service domain, i.e., *activity classifier*. **(4)** The cloud server is involved in model training with the MEC-granted data for management purposes, e.g., inferring the upcoming sensor change in real-time and determining the timing for data synchronization between the edge and cloud server.

Once the training is finished, these models are encapsulated in the edge and cloud servers to serve the residents. We will next reveal DT construction details.

IV. Deployment Details for DT Construction

This section will first explain the motivation behind our design and deployment. We realize that it is critical to consider the intelligence in DT construction, specifically:

- **Intelligent Compression**: The sensor status changes cannot be directly revealed to the cloud because of privacy concerns; however, the cloud still needs access to the information regarding the point of interest to maintain DT. The edge server can compress the sensory observations, allowing for the upload of a condensed abstraction to represent the current states of all sensors.
- **Intelligent Sampling**: Merely utilizing the sensor update prediction in the cloud is insufficient to construct a high-fidelity DT; it is still necessary to sample some key information from the edge server to maintain consistency

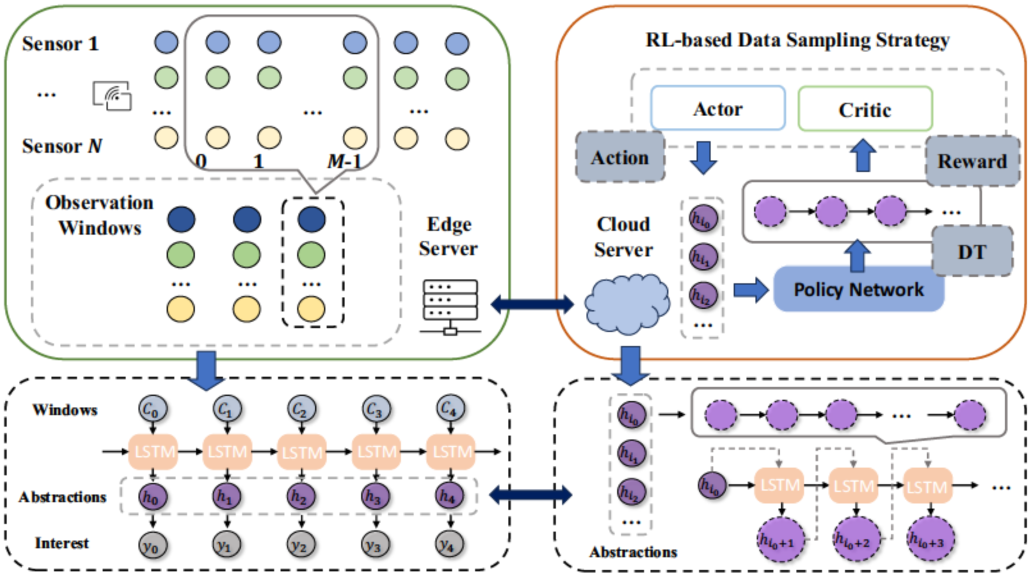

Fig. 2: The detailed framework DRL deployment for optimizing data uploading strategy.

between the physical and digital realms. This necessitates associating the point of interest with the observation and the subsequent decision-making policy.

Fortunately, deep learning technology can be used to facilitate the subject association, and the generated abstraction in terms of HAR can be directly utilized to enhance sensor update inference. Also, according to the observation from edge servers, we can derive the decision-making policy with optimized sampling cost and physical-digital consistency. This gives rise to our scheme, and the general idea includes aligning the observation with the upcoming sensor change using a *long-short-term memory* (LSTM) mechanism that captures the long- and short-term features from historical sensory behaviors [31], [32], and deriving a decision-making policy that maximizes the reward obtained from sampling key changes with DRL technology [33]. A detailed deployment MEC-based DT deployment framework with DRL is illustrated in Fig. 2.

A. Information Abstraction with Embedding

We adopt a dedicated abstraction model $\mathcal{M}_{abs}$ in the edge server that exploits implicit sensor update correlation and outputs state abstraction. For an arbitrary observation window, we employ the LSTM units to extract the sequential information of sensor status changes and associate them to their activity, which has been broadly adopted in HAR context [21], [34], [35]. To learn the features of different activities, we group every M consecutive status changes with the same activity label into an observation window, i.e., $\mathcal{C}_k = \{c_{k_1}, c_{k_2} \cdots, c_{k_M}\} \subset \mathcal{C}$ and $y_{c_{k_1}} = y_{c_{k_2}} \cdots = y_{c_{k_M}}$. For a window $\mathcal{C}_k$, we take each change c_{k*} and combine sensor names and their corresponding readings $\{s_{n_i,t_i}\}_{i=1}^{M}$ to form the vocabularies. For example, a motion sensor [36] named *M004* and with a status of *ON* results in a word *M004ON*. Then we convert all words into embedding vectors $\{\mathbf{x}_i\}_{i=1}^{M}$ with a vector length of d_{embed} units using the embedding layer [37], i.e., $\mathcal{L}_{embed}$, where

$$\mathbf{x}_i = \mathcal{L}_{embed}(n_i, s_{n_i,t_i}), \forall n_i \in \{N\}, s_{n_i,t_i} \in \{s_{n,t}\}.$$

Next, one can feed the embedding vectors into an LSTM layer $\mathcal{L}_{lstm}$ and generate a set of hidden states $\mathbf{h}_{\mathcal{C}_k}$ with a vector length of d_{hidden} units as the sensor status abstraction, i.e.,

$$\mathbf{h}_{\mathcal{C}_k} = \mathcal{L}_{lstm}\left(\{\mathbf{x}_i\}_{i=1}^{M}\right).$$

Hereby, we complete the component of a feature extractor. The abstractions from the feature extractor will be associated with their corresponding activity $y_{\mathcal{C}_k}$, e.g., ***Read***, with an activity classifier. The activity classifier is accomplished by feeding the hidden states to a fully connected (FC) layer that maps the vectors of a length of d_{hidden} units to a new one with $d_{output} = |\mathcal{Y}|$ units, i.e., $\mathcal{L}_{fc}$, followed by the prediction of

$$\hat{y}_{\mathcal{C}_k} = \operatorname{argmax}_j \log \operatorname{Softmax}\left(\mathbf{W}_{abs}^T \mathbf{h}_{\mathcal{C}_k} + \mathbf{b}_{abs}\right),$$

where $\mathbf{W}_{abs}$ and $\mathbf{b}_{abs}$ are the weight and bias parameters of $\mathcal{L}_{fc}$, and j is the position index of a vector that indicates the predicted activity. Hereby, an abstraction module $\mathcal{M}_{abs}$, including a feature extractor and activity extractor, is trained by minimizing the negative log-likelihood loss (NLL) between predictions and the ground truth of activity labels [38].

B. Sampling Timing Determination

The edge server maintains the log of sensor status changes but will not upload them unless it pertains to meaningful updates in DT refreshing, which fundamentally avoids unnecessary communication while envisioning DT freshness. However, one could not foresee if the upcoming sensor status update would lead to a significant DT refresh. This criticality of sensor updates for DT construction has rarely been studied, and it is also challenging to determine due to the context variety.

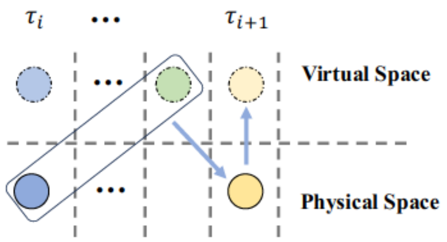

Fig. 3: The mechanism to determine the upload timing.

However, the cloud server can impose historical data replay on the training dataset to derive an optimal policy for determining the update sampling timing to aid DT representation. Formally, denote the *sampling timing set* of a sequence $\mathcal{C}$ as

$$\mathcal{T}_{\mathcal{C}} = \{\tau_i = t_j\}_{t_j \in \{T\}},$$

where τ_i indicates the i-th sampling and t_j is the time of the j-th status change in a sequence.

As a compromise for ensuring DT availability, one should selectively drop some sampling opportunities, thus leading to some unavoidable information loss regarding sensor update changes. Nevertheless, our goal is the creation of a virtual space that mirrors the physical space closely. Maintaining space consistency between physical and virtual space in our discussed domain implies the consistency between the sensors to update inferred from the states in the cloud and in the actual residential setting. With the state of a predicted sequence stored in the cloud, the cloud server could predict the following sensor to update, determine if synchronization with the real world (i.e., edge server) is required, and request for the connection in advance.

This is partly done by a generation model $\mathcal{M}_{gen}$ with LSTM units predicting the next updated sensor in the DT. The training process starts by feeding a window into an LSTM layer to obtain a hidden abstraction containing the information of the next sensor to update, namely $\hat{\mathbf{h}}_{\mathcal{C}_k}$. This hidden abstraction of DT will be concatenated with the feature from $\mathcal{M}_{abs}$ of edge to predict the next sensor to update $\hat{n}_{\mathcal{C}_k}$. Next, the vector of raw predictions on $\hat{n}_{\mathcal{C}_k}$ will be compared with the actually changed sensor $n_{\mathcal{C}_k}$ to calculate the NLL loss. The loss will be used to update the generation model $\mathcal{M}_{gen}$, which gradually gains the capability of predicting the next sensor to update with the input of the currently stored sensor updates in DT. The predicted sensor will be appended to the window and fed back to the LSTM layer in the next timestamp recurrently.

In practice, the whole $\mathcal{T}_{\mathcal{C}}$ can be broken down into multiple $\mathcal{T}_{\mathcal{C}_k}$ for better traceability and feature capturing. To showcase the derivation of each $\mathcal{T}_{\mathcal{C}_k}$, we plot the mechanism of determining each synchronization sampling as illustrated in Fig. 3. Initially, the cloud server possesses the updated sensor information from the cloud in time τ_i. The cloud server will not set up communication with the edge server until the time τ_{i+1}. During this period, the DT will be refreshed by $\mathcal{M}_{gen}$. In time τ_{i+1}, the edge server uploads the $(i+1)$-th actual updated sensor to the cloud server to refresh the DT. The refreshed DT will be further used for future sampling timing decisions.

The determination of the sampling timing set will influence the quality of DT, specifically reflected in the fidelity, i.e., $F_{\mathcal{C}}^{\mathcal{T}_{\mathcal{C}}}$. Conclusively, the optimization goals consist of maximizing the consistency fidelity of all decision-making slots with the sampling timing set $\mathcal{T}_{\mathcal{C}}$ for a given sequence $\mathcal{C}$, denoted as

$$F_{\mathcal{C}}^{\mathcal{T}_{\mathcal{C}}} = \frac{1}{T} \sum_{t \in \{T\}} \mathbf{1}_{\{n(\mathbf{h}_{cloud,t}) = n_t\}},$$

where $n(\mathbf{h}_{cloud,t})$ and n_t are the inferred sensor and actual updated sensor in time t, respectively and $\mathbf{h}_{cloud,t}$ is the abstraction of cloud's DT in time t. It is worth noting that we will denote the abstraction as $\mathbf{h}_{\mathcal{C}_k}$, $\mathbf{h}_{\tau_i}$, $\mathbf{h}_t$, $\mathbf{h}_{cloud}$ or $\mathbf{h}_{edge}$ interchangeably based on the reference of windows, sampling decisions, timesteps, or the locations of generation.

To optimize consistency, an ideal sampling timing sequence would entail the synchronization of each updated sensor, ensuring that all sensor update patterns are instantaneously incorporated into the DT. Nonetheless, as previously detailed from the resource perspective, this approach can be both unnecessary and difficult to implement. Formally, the average communication efficiency $Q_{\mathcal{C}}^{\mathcal{T}_{\mathcal{C}}}$ with the sampling timing set $\mathcal{T}_{\mathcal{C}}$ for a given sequence $\mathcal{C}$, can be represented by

$$Q_{\mathcal{C}}^{\mathcal{T}_{\mathcal{C}}} = \frac{|\mathcal{T}_{\mathcal{C}}|}{|\mathcal{C}|}.$$

The overall performance of a DT deployment for $\mathcal{C}$ is improved by maximizing

$$\mathcal{R}_{\mathcal{C}}(\mathcal{T}_{\mathcal{C}}) = F_{\mathcal{C}}^{\mathcal{T}_{\mathcal{C}}} - Q_{\mathcal{C}}^{\mathcal{T}_{\mathcal{C}}}.$$

However, $\mathcal{C}$ cannot be predetermined and undergoes dynamic changes in practical applications, and an optimal $\mathcal{T}_{\mathcal{C}}$ is not directly solved. We propose a DRL framework that learns from the training data and can be applied to potential activity transitions in the real world. A DRL-based sampling model $\mathcal{M}_{sap}$ can indicate if the system should schedule another real-world synchronization sampling in the next physical sensor update, which will be detailed in Section IV-C. Intuitively, we can directly replay the historical sequence with $\mathcal{M}_{gen}$ and derive the targeting sampling model $\mathcal{M}_{sap}$ with the synchronization supported by the MEC environment. Now, we start a problem formalization of the *Markov Decision Process* (MDP) to derive our DRL method [33], [39].

C. DRL Deployment Details

Aiming to improve the overall consistency fidelity of the generated DT while reducing the communication cost in the cloud server, the deployment is realized by the sampling model $\mathcal{M}_{sap}$, which takes cloud DT abstraction $\mathbf{h}_{cloud}$ as input. The state of cloud DT abstraction can indicate the choice of whether the next actual updated sensor recorded in the edge server should be synchronized or not. This sampling timing determination problem can be formulated as an MDP, represented by $\Gamma = (\mathcal{O}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{O}, \mathcal{A}, \mathcal{R}, \mathcal{P}$ and γ are the sets of observations and actions, reward function, transition function, discounted factor, respectively, and the details are as follows:

*1) **Observation space** $\mathcal{O}$:* For observation windows, we discretize their sensor readings and tokenize the combination of each sensor name and its reading. With $\mathcal{M}_{gen}$ and *Tanh* activation [40], the observation of DT abstraction can be converted into a hidden state vector $\mathbf{h}_{cloud}$ with a fixed length d_{hidden} and each position in $\mathbf{h}_{cloud}$ is in the range of $(-1, 1)$. $\mathcal{M}_{sap}$ takes $\mathbf{h}_{cloud}$ as input, which leads to the observation $o \in \mathcal{O} = \{(-1,1)\}^{d_{hidden}}$.

*2) **Action space** $\mathcal{A}$:* An action $a \in \mathcal{A}$ dictates the sampling strategy of the cloud server based on the perceived information in both physical and virtual space, which can take a value from the set $\{0, 1\}$, with 1 indicating sampling from the edge server and 0 otherwise.

*3) **Reward function** $\mathcal{R}$:* After deciding on an action, one can receive an instant reward $r \in \mathcal{R} = \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ to imply the contribution of this action to DT fidelity and sampling cost. The system is devoted to constantly optimizing the expected discounted return as $\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right]$, where γ is the discount factor that accounts for the long-term effect on aggregated rewards. Specifically, the reward is formulated as:

$$r_t = \begin{cases} 1 - a_t & \text{if } n(\hat{\mathbf{h}}_t) = n_t, \\ 1 - a_t - \kappa & \text{elsewise,} \end{cases}$$

where κ is the penalty for the sensor inference accuracy compromise. With this reward design, the system guarantees the consistency reward while considering sampling cost in synchronization.

*4) **Transition function** $\mathcal{P}$:* According to the decision of action a_t, the probability of state transition is given as $\mathcal{P}$, which follows $p(o_{t+1}|o_t, a_t) = \mathcal{O} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$. With $\mathcal{P}$, the long-term reward can be estimated, thus enabling the refinement of $\mathcal{M}_{sap}$ that considers the future impact. In this context, the strategy is to determine if the next actual updated sensor in physical space should be sampled.

We adopt an Advantage Actor-Critic architecture [41], [42] to establish $\mathcal{M}_{sap}$, which learns an actor and critic from the training data to determine the sampling timing and estimates the long-term reward based on the current decision, respectively. As shown in Fig. 2, the actor of $\mathcal{M}_{sap}$ takes the abstraction of DT, i.e., $\mathbf{h}_{cloud}$ as input to derive the action. By responding based on the actions of the actor, instant rewards are calculated to shape the critic that evaluates the long-term reward resulting from adopting the current action.

Formally, the actor and critic algorithms are concretized as parameterized functions that perform vectorization operations with the provided input:

- **Critic**: The critic is a parameterized function $V^{\theta_v}(\cdot)$ with parameters θ_v. With the critic, an advantage can be estimated that yields the difference between the returns and the baseline $V(o_t)$, as

$$A(o_t, a_t) \approx r_t + \gamma V^{\theta_v}(o_{t+1}|o_t, a_t) - V^{\theta_v}(o_t) = \delta(o_t).$$

  We can estimate the critic parameters with the replay of the training data.
- **Actor**: The actor is a parameterized function $\pi^{\theta_u}(a_t|o_t)$ with parameters θ_u, which quantifies the probability of choosing action a_t based on o_t. We expect to use the advantage function to guide the optimization of the policy. Thus, the policy gradient of the actor with the current critic can be calculated by differentiating the loss

$$L(\theta_u) = \log \pi^{\theta_u}(a_t|o_t)\delta(o_t).$$

  Meanwhile, we also consider the critic, which is updated by minimizing the loss function of

$$L(\theta_v) = \frac{1}{2}\delta^2(o_t).$$

Finally, we jointly optimize the θ_v and θ_u by considering the overall loss

$$J(\theta_u, \theta_v) = L(\theta_u) + L(\theta_v).$$

We present learning details of our proposed DRL-based sampling control in Algorithm 1. The replay of derived rewards on the training data will be utilized to update both actor and critic parameters, finally leading to the sampling model $\mathcal{M}_{sap}$.

Algorithm 1: DRL-based Sampling Control.

```
Input: Timesteps for per actor update $M$, mini-batch
       size $\mathbf{B}$, training dataset in format of
       observation windows.
Output: Sampling model $\mathcal{M}_{sap}$.
1  Initialization:
2  Initialize the parameters of critic and actor for $\mathcal{M}_{sap}$,
     namely $\theta_v$ and $\theta_u$, ;
3  for $m = 1$ to max-iterations do
4      Stage 1: Generating Trajectories
5      Reset the environment, get initial observation $\boldsymbol{o}_{(0)}$
         with $\mathbf{B}$ sequences;
6      for $k = 0$ to $M - 1$ do
7          Sample actions $\boldsymbol{a}_{(k)}$ using $\theta_u$ given $\boldsymbol{o}_{(k)}$;
8          Execute $\boldsymbol{a}_{(k)}$ in DT system;
9          Observe reward $\boldsymbol{r}_{(k+1)}$ and next state $\boldsymbol{o}_{(k+1)}$;
10         Store the transition from policy
             $< (\boldsymbol{o}_{(k)}, \boldsymbol{a}_{(k+1)}, \boldsymbol{r}_{(k+1)}, \boldsymbol{o}_{(k+1)} >$;
11     Stage 2: Parameters Training
12     Compute advantage estimates $\delta$ of the current
         policy function using critic parameters $\theta_v$ for the
         stored transitions;
13     Update the overall loss $J$ wrt. $\theta_u$ and $\theta_v$ within the
         batch and backward the gradient from loss;
14 Return the final sampling model $\mathcal{M}_{sap}$.
```

V. Experimental Evaluation

In this section, we adopt two real-world human activity datasets, Aruba and Milan [36], [43], to demonstrate the practicability of our proposed framework. Each dataset is used to represent one residential environment and a specific MEC environment is set up to serve individual needs. Both datasets include a table of sensor change events with attributes of *date*, *time*, *sensor value*. Specifically, each event may include a *begin* or *end* tag along with the activity category to indicate the action transition. The Aruba identifies 12 activity categories, namely *Meal_Preparation*, *Relax*, *Eating*, *Work*, *Sleeping*, *Wash_Dishes*, *Bed_to_Toilet*, *Enter_Home*, *Leave_Home*, *Housekeeping*, *Respirate*, and *Other* while Milan identifies 15 activity categories as *Bed-to-Toilet*, *Chores*, *Desk_Activity*, *Dining_Rm_Activity*, *Eve_Meds*, *Guest_Bathroom*, *Kitchen_Activity*, *Leave_Home*, *Master_Bathroom*, *Meditate*, *Watch_TV*, *Sleep*, *Read*, *Morning_Meds*, *Master_Bedroom_Activity*. The capability of dealing with a wide spectrum of human activities will greatly demonstrate the practicability of our proposed DT deployment scheme. The sensor and activity columns in the dataset are extracted as described in Section IV-A.

Firstly, we pre-segment the whole dataset according to the tags of *begin* and *end* of all activity labels. For each segment, we construct multiple observation windows with a length of $M = 100$ and step size 1, in which all sensor status changes are of the same activity classes. For segments shorter than 100 units in length, we pad the windows with zeros at the beginning. We remove all windows regarding the label *Other* to focus on meaningful DT construction in targeting activities. We downsample the Aruba dataset to ensure comparability with the Milan dataset in terms of data point volume. To enable our proposed training scheme, all observation windows are split into a training set and a testing set with a ratio of 7:3, and the data in the training set are further partitioned into training and validation sets with the same ratio. In the experiments, we train $\mathcal{M}_{abs}$, $\mathcal{M}_{gen}$ and $\mathcal{M}_{sap}$ with Pytorch 2.1.0 and CUDA version 12.0 on a device with Intel Xeon Gold 6242@2.60 GHz CPU and NVIDIA Tesla V100 GPU. All the following training was conducted over three random seeds.

A. Performance in Model $\mathcal{M}_{abs}$

Firstly, we train the abstraction model of edge server $\mathcal{M}_{abs}$ processing the data in HAR domain. It includes an embedding layer $\mathcal{L}_{embed}$ that converts each tokenized sensor reading in the given window into vectors of length $d_{embed} = 64$. Next, the embedding vectors are passed to $\mathcal{L}_{lstm}$, generating a hidden state vector of length d_{hidden}. The hidden state will pass the *Tanh* activation function along with $\mathcal{L}_{fc}$ and LogSoftmax function to determine the activity label among d_{output} options. The model is optimized by Adam [40] with a learning rate of 1e-3. The batch size is set as 2048. We adopt the early stopping mechanism, which terminates the training when the classification loss stops to decrease in the validation dataset for 100 episodes. The model with the best performance in the validation dataset is used for testing.

The weighted F1-scores [44] and NLL loss for the testing dataset in Aruba and Milan are presented in Table I. In both residential environments, the capability and robustness of the model are generally improved as d_{hidden} increases. This result indicates the importance of the feature-capturing capability of the abstraction, which impacts the future DT construction. However, there is minimal room for performance enhancement when considering a sufficiently large d_{hidden}. In practice, d_{hidden} should be delicately set to trade off the model performance and the communication cost. The service provider can consider an acceptable length of hidden features in model deployment.

We illustrate confusion matrices of Aruba and Milan residential environments from $\mathcal{M}_{abs}$ with $d_{hidden} = 256$ in Fig. 4. The figure shows that most activity classes can be correctly recognized by $\mathcal{M}_{abs}$, demonstrating that domain knowledge regarding HAR is embedded in the abstraction. Meanwhile, we can also observe some fundamental defects in the abstraction model as some types of observation windows cannot be correctly identified in both datasets. For example, some of *Leave_Home* windows labeled by 4 are mistreated as *Enter_Home* denoted by 2 in Aruba; most of *Eve_Med* windows labeled by 4 in Milan are misidentified as *Morning_Med* with a label 11. The cause could stem from the ambiguity in the chosen features. Given the low weight of the number of samples relative to the total, its impact on the weighted F1 score is subtle. To address precision challenges related to feature representativeness in the physical domain, additional effort should be directed toward enhancing data quality and refining model architecture which is not the scope of this article.

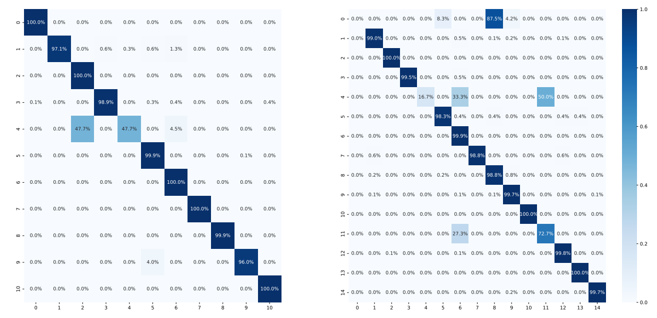

Fig. 4: A sample of the confusion matrices of Aruba (left) and Milan (right) datasets obtained from $\mathcal{M}_{abs}$ with $d_{hidden} = 256$. The darker cells along the diagonal signify the higher number of samples, illustrating a more capable model.

B. Performance in Model $\mathcal{M}_{gen}$

We consider the temporal correlation in the sequence using LSTM units and conduct real-time pattern recognition for the next updated sensor inference using the generation model, i.e., $\mathcal{M}_{gen}$. In DT deployment, the cloud server can sample the actual updated sensors and HAR features from the edge server. The updated sensor sequence encoded to a state $\mathbf{h}_{cloud}$ while incorporating stored $\mathbf{h}_{edge}$, is fed into FC layers to infer the next sensor to update. In our experiment, we directly set the hidden state of the LSTM encoder as $d_{hidden} = 256$ to encode the sensor sequence and obtain enough domain information. Related model training parameters are the same as $\mathcal{M}_{abs}$.

Our proposed $\mathcal{M}_{gen}$ takes both the DT status and the edge domain information of HAR into consideration in the way of concatenating both abstractions of edge and cloud, namely *Edge-Cloud-Fusion*. To evaluate its performance, we compare it with two baselines: 1) *Edge-FC* utilizes only feature obtained from the edge server to generate sensor word; and 2) *Cloud-LSTM* recurrently append the next sensor from training data while predicting the next sensor to update. We evaluate the performance of these methods in Table II.

	Weighted F1-Score			
	$d_{hidden} = 32$	64	128	256
Aruba	99.36 ± 0.02	99.57 ± 0.09	99.73 ± 0.07	$\mathbf{99.79 \pm 0.03}$
Milan	99.49 ± 0.05	99.67 ± 0.07	$\mathbf{99.71 \pm 0.05}$	$\mathbf{99.71 \pm 0.05}$
	Average NLL Loss			
	$d_{hidden} = 32$	64	128	256
Aruba	0.026 ± 0.005	0.014 ± 0.003	0.010 ± 0.001	$\mathbf{0.009 \pm 0.003}$
Milan	0.023 ± 0.001	0.016 ± 0.003	$\mathbf{0.015 \pm 0.005}$	0.016 ± 0.004

TABLE I: The model performance of $\mathcal{M}_{abs}$ in terms of *Weighted F1-Score* and *Average NLL Loss* with different hidden state length d_{hidden}.

	Aruba		**Milan**	
	Average Precision (%)	**Average NLL Loss**	**Average Precision (%)**	**Average NLL Loss**
Edge-FCC	61.85 ± 0.27	1.102 ± 0.007	62.33 ± 0.33	1.050 ± 0.009
Cloud-LSTM	78.77 ± 0.66	0.691 ± 0.011	92.38 ± 0.11	0.279 ± 0.003
Edge-Cloud-Fusion	$\mathbf{85.28 \pm 0.47}$	$\mathbf{0.527 \pm 0.004}$	$\mathbf{93.96 \pm 0.08}$	$\mathbf{0.212 \pm 0.001}$

TABLE II: The model performance of $\mathcal{M}_{gen}$ in different schemes on next sensor inference task with hidden state length $d_{hidden} = 256$.

We can easily observe the superiority of *Edge-Cloud-Fusion* method over the other methods, which reaches an accuracy of 85.28% and 93.96% on Aruba and Milan, respectively. These results surpass those of the *Cloud-LSTM* method by 6.51% and 1.58%, respectively. As the *Edge-FC* only relies on the HAR domain knowledge, it neglects the mechanism of sensor update, thus failing to yield sensor change patterns correctly. Although *Cloud-LSTM* captures the pattern of sensor updates, it lacks the context information, thus leading to fidelity loss. As shown in Table II, it has been noted that domain knowledge effectively transitions from HAR to sensor inference on the cloud server by leveraging features extracted at the edge server, as demonstrated by the performance of *Edge-Cloud-Fusion*. This implies that the cloud server can infer the sensor changed in a timestamp by utilizing the edge abstraction and $\mathcal{M}_{gen}$. The inference performance can be influenced by many factors, such as activity complexity, the number of sensors involved, and underfitting. We will prove that a robust and capable generation model is crucial to the efficiency of DT construction and the efficacy of $\mathcal{M}_{sap}$.

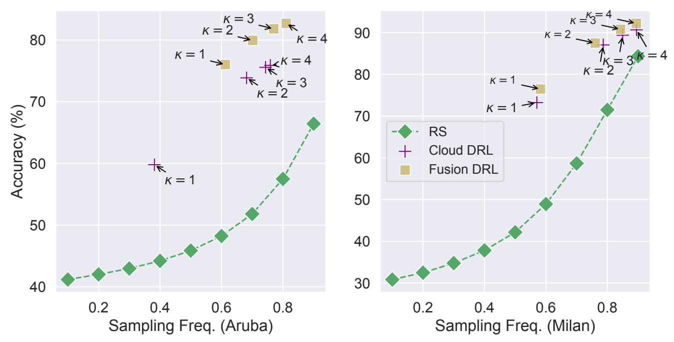

Fig. 5: The accuracy-sampling efficiency performance of $\mathcal{M}_{sap}$ with DRL-based Sampling and Random Sampling (RS) strategies in Aruba (left) and Milan (right) datasets and different penalty settings with $d_{hidden} = 256$.

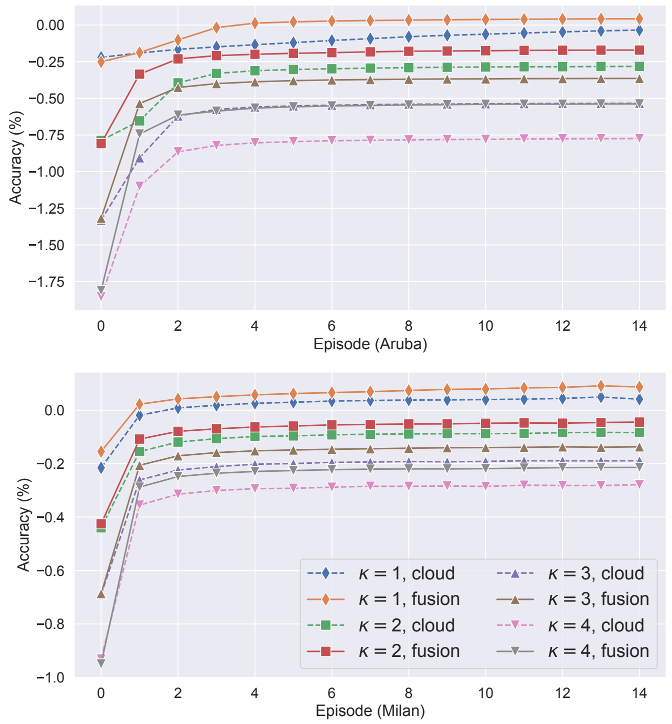

Fig. 6: The reward performance of $\mathcal{M}_{sap}$ with *Cloud DRL* and *Fusion DRL* in Aruba (upper) and Milan (lower) datasets and different penalty settings with $d_{hidden} = 256$.

C. Performance in Model $\mathcal{M}_{sap}$

The DRL-based $\mathcal{M}_{sap}$ encodes the inferred sensor into a cloud feature as the current state, represented by a vector length of $d_{hidden} = 256$. In training, γ is set as 0.99, learning rate is set as 3e-4, and batch size $\mathbf{B}$ is 256. We simulate a baseline strategy in which the cloud server randomly samples the actual updated sensors from the edge server with a fixed probability. In comparison, our proposed DRL-based method penalizes those sampling slots that generate a fault prediction in the reward function and selects the slots to sample from the edge according to the guidance of $\mathcal{M}_{sap}$. We set the penalty κ in reward to multiple values. With a large penalty, the policy tends to prevent fidelity loss, thus encouraging more sampling. Also, we explore the impact of the edge feature in the sampling process. The method that merely uses the features of the cloud server is denoted as *Cloud DRL* while the method that takes the fusion abstractions from $\mathcal{M}_{gen}$ as inputs is denoted by *Fusion DRL*. Fig. 5 plots the performance of $\mathcal{M}_{sap}$ with *Cloud DRL* and *Fusion DRL* Sampling and Random Sampling (RS) strategies in Aruba (left) and Milan (right) datasets obtained from $\mathcal{M}_{abs}$ with $d_{hidden} = 256$. We can easily observe that the DRL methods outperform the RS strategy as they gain insight into DT and associate each state with their importance. This significantly reduces the unnecessary data synchronization with the edge server. With the same number of times in sampling data from the edge, the accuracy of sensor inference with the DRL-based methods is higher than the RS method. When comparing *Fusion DRL* with *Cloud DRL*, it is evident that for different penalty values, the scatter plots of *Fusion DRL* are positioned in the upper left relative to those of *Cloud DRL*, which exhibits better accuracy-sampling efficiency. This observation underscores the significance of integrating physical and virtual elements. This conclusion is further illustrated in Fig. 6, which plots the obtained rewards from the testing dataset in different training episodes based on *Cloud DRL* and *Fusion DRL*. As shown in Fig. 6, we depict the training trajectories of *Cloud DRL* and *Fusion DRL* under various penalty values using dashed and solid lines, respectively. It is clear that all rewards initially increase at the start of training and then converge after a certain number of episodes, illustrating the effectiveness of the DRL method. Furthermore, the rewards achieved with *Fusion DRL* are higher than those from *Cloud DRL* under the same dataset and penalty setting, demonstrating the effectiveness of edge-cloud fusion in practice.

VI. Conclusion

In this article, we propose a low-cost and high-fidelity digital twin (DT) implementation scheme with the mobile edge computing (MEC) environment to aid the cloud service. Taking the residential human activity recognition task as an example, we first employ an abstraction model with the long short-term memory (LSTM) to extract an abstract representation aligned with the point of interest of service. Next, we exploit the feature extracted from the abstraction model and train a data upload timing model with deep reinforcement learning (DRL) to improve the DT fidelity while reducing communication costs. With the aid of mobile edge servers, the deployment can be customized for different consumer scenarios. Extensive empirical studies with different residential environments demonstrate the applicability of our proposed deployment scheme. In the future, we will continue to optimize the deployment of low-cost and high-fidelity DTs in extensive emerging consumer electronics paradigms, such as blockchain and metaverse.

References

[1] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, vol. 94, no. 9-12, pp. 3563–3576, 2018.

[2] R. Martinez-Velazquez, R. Gamez, and A. El Saddik, "Cardio twin: A digital twin of the human heart running on the edge," in *2019 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. Istanbul, Turkey: IEEE, 2019, pp. 1–6.

[3] M. Farsi, A. Daneshkhah, A. Hosseinian-Far, and H. Jahankhani, *Digital Twin Technologies and Smart Cities*. Berlin/Heidelberg, Germany: Springer, 2020.

[4] G. Schroeder, C. Steinmetz, C. E. Pereira, I. Muller, N. Garcia, D. Espindola, and R. Rodrigues, "Visualising the digital twin using web services and augmented reality," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. Poitiers, France: IEEE, 2016, pp. 522–527.

[5] C. Wang, Z. Cai, and Y. Li, "Sustainable blockchain-based digital twin management architecture for iot devices," *IEEE Internet of Things Journal*, vol. 10, no. 8, pp. 6535–6548, 2022.

[6] Z. Cai and T. Shi, "Distributed query processing in the edge-assisted iot data monitoring system," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 679–12 693, 2020.

[7] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4854–4866, 2018.

[8] L. Corneo, C. Rohner, and P. Gunningberg, "Age of information-aware scheduling for timely and scalable internet of things applications," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. Paris, France: IEEE, 2019, pp. 2476–2484.

[9] X. Zhou, X. Ye, I. Kevin, K. Wang, W. Liang, N. K. C. Nair, S. Shimizu, Z. Yan, and Q. Jin, "Hierarchical federated learning with social context clustering-based participant selection for internet of medical things applications," *IEEE Transactions on Computational Social Systems*, 2023.

[10] O. Akbarzadeh, S. Hamzehei, H. Attar, A. Amer, N. Fasihihour, M. R. Khosravi, and A. A. Solyman, "Heating-cooling monitoring and power consumption forecasting using lstm for energy-efficient smart management of buildings: A computational intelligence solution for smart homes," *Tsinghua Science and Technology*, vol. 29, no. 1, pp. 143–157, 2023.

[11] X. Zhou, W. Liang, K. Yan, W. Li, I. Kevin, K. Wang, J. Ma, and Q. Jin, "Edge-enabled two-stage scheduling based on deep reinforcement learning for internet of everything," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3295–3304, 2022.

[12] K. Sultana, K. Ahmed, B. Gu, and H. Wang, "Elastic optimization for stragglers in edge federated learning," *Big Data Mining and Analytics*, vol. 6, no. 4, pp. 404–420, 2023.

[13] K. Lone and S. A. Sofi, "A review on offloading in fog-based internet of things: Architecture, machine learning approaches, and open issues," *High-Confidence Computing*, p. 100124, 2023.

[14] X. Zhou, W. Liang, I. Kevin, K. Wang, Z. Yan, L. T. Yang, W. Wei, J. Ma, and Q. Jin, "Decentralized p2p federated learning for privacy-preserving and resilient mobile robotic systems," *IEEE Wireless Communications*, vol. 30, no. 2, pp. 82–89, 2023.

[15] X. Zhou, W. Liang, I. Kevin, K. Wang, and L. T. Yang, "Deep correlation mining based on hierarchical hybrid networks for heterogeneous big data recommendations," *IEEE Transactions on Computational Social Systems*, vol. 8, no. 1, pp. 171–178, 2020.

[16] D. Yu, Z. Xie, Y. Yuan, S. Chen, J. Qiao, Y. Wang, Y. Yu, Y. Zou, and X. Zhang, "Trustworthy decentralized collaborative learning for edge intelligence: A survey," *High-Confidence Computing*, p. 100150, 2023.

[17] W. Shengli, "Is human digital twin possible?" *Computer Methods and Programs in Biomedicine Update*, vol. 1, p. 100014, 2021.

[18] Z. Wang, R. Gupta, K. Han, H. Wang, A. Ganlath, N. Ammar, and P. Tiwari, "Mobility digital twin: Concept, architecture, case study, and future challenges," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 452–17 467, 2022.

[19] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-based activity recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 790–808, 2012.

[20] S. Pattamaset and J. S. Choi, "Irrelevant data elimination based on a k-means clustering algorithm for efficient data aggregation and human activity classification in smart home sensor networks," *Inter-*

national Journal of Distributed Sensor Networks, vol. 16, no. 6, p. 1550147720929828, 2020.

[21] D. Bouchabou, S. M. Nguyen, C. Lohr, B. Leduc, and I. Kanellos, "Fully convolutional network bootstrapped by word encoding and embedding for activity recognition in smart homes," in *Deep Learning for Human Activity Recognition: Second International Workshop, DL-HAR 2020, Held in Conjunction with IJCAI-PRICAI 2020, Kyoto, Japan, January 8, 2021, Proceedings 2*. Springer, 2021, pp. 111–125.

[22] A. Gumaei, M. M. Hassan, A. Alelaiwi, and H. Alsalman, "A hybrid deep learning model for human activity recognition using multimodal body sensing data," *IEEE Access*, vol. 7, pp. 99 152–99 160, 2019.

[23] S. Silvestri, R. Urgaonkar, M. Zafer, and B. J. Ko, "A framework for the inference of sensing measurements based on correlation," *ACM Transactions on Sensor Networks (TOSN)*, vol. 15, no. 1, pp. 1–28, 2018.

[24] X. Zhou, X. Zheng, T. Shu, W. Liang, I. Kevin, K. Wang, L. Qi, S. Shimizu, and Q. Jin, "Information theoretic learning-enhanced dual-generative adversarial networks with causal representation for robust ood generalization," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[25] P. Liu, S.-K. Nguang, and A. Partridge, "Occupancy inference using pyroelectric infrared sensors through hidden markov models," *IEEE Sensors Journal*, vol. 16, no. 4, pp. 1062–1068, 2015.

[26] B. Fan, Y. Wu, Z. He, Y. Chen, T. Q. Quek, and C.-Z. Xu, "Digital twin empowered mobile edge computing for intelligent vehicular lane-changing," *IEEE Network*, vol. 35, no. 6, pp. 194–201, 2021.

[27] T. Do-Duy, D. Van Huynh, O. A. Dobre, B. Canberk, and T. Q. Duong, "Digital twin-aided intelligent offloading with edge selection in mobile edge computing," *IEEE Wireless Communications Letters*, vol. 11, no. 4, pp. 806–810, 2022.

[28] X. Zhou, X. Zheng, X. Cui, J. Shi, W. Liang, Z. Yan, L. T. Yang, S. Shimizu, I. Kevin, and K. Wang, "Digital twin enhanced federated reinforcement learning with lightweight knowledge distillation in mobile networks," *IEEE Journal on Selected Areas in Communications*, 2023.

[29] D. Van Huynh, S. R. Khosravirad, A. Masaracchia, O. A. Dobre, and T. Q. Duong, "Edge intelligence-based ultra-reliable and low-latency communications for digital twin-enabled metaverse," *IEEE Wireless Communications Letters*, vol. 11, no. 8, pp. 1733–1737, 2022.

[30] Y. Qu, L. Ma, W. Ye, X. Zhai, S. Yu, Y. Li, and D. Smith, "Towards privacy-aware and trustworthy data sharing using blockchain for edge intelligence," *Big Data Mining and Analytics*, vol. 6, no. 4, pp. 443–464, 2023.

[31] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[32] R. Yuan, S. Meng, R. Dou, and X. Wang, "Modeling long-and short-term service recommendations with a deep multi-interest network for edge computing," *Tsinghua Science and Technology*, vol. 29, no. 1, pp. 86–98, 2023.

[33] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[34] D. Liciotti, M. Bernardini, L. Romeo, and E. Frontoni, "A sequential deep learning application for recognising human activities in smart homes," *Neurocomputing*, vol. 396, pp. 501–513, 2020.

[35] R. Mutegeki and D. S. Han, "A cnn-lstm approach to human activity recognition," in *2020 international conference on artificial intelligence in information and communication (ICAIIC)*. IEEE, 2020, pp. 362–366.

[36] D. J. Cook, N. C. Krishnan, and P. Rashidi, "Activity discovery and activity recognition: A new partnership," *IEEE transactions on cybernetics*, vol. 43, no. 3, pp. 820–828, 2013.

[37] F. Hill, K. Cho, A. Korhonen, and Y. Bengio, "Learning to understand phrases by embedding the dictionary," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 17–30, 2016.

[38] D. Oh and B. Shin, "Improving evidential deep learning via multi-task learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 7, 2022, pp. 7895–7903.

[39] M. Van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement learning: State-of-the-art*. Springer, 2012, pp. 3–42.

[40] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[41] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[42] M. Han, L. Zhang, J. Wang, and W. Pan, "Actor-critic reinforcement learning for control with stability guarantee," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6217–6224, 2020.

[43] S. Yan, K.-J. Lin, X. Zheng, and W. Zhang, "Using latent knowledge to improve real-time activity recognition for smart IoT," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 3, pp. 574–587, 2019.

[44] N. Rashid, B. U. Demirel, and M. A. Al Faruque, "Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13 041–13 051, 2022.

Chenyu Wang received his B.S. degree from Xiangtan University, Hunan, China, and his M.S. degree from Beijing Normal University, Beijing, China in 2012 and 2016, respectively. He is currently pursuing a Ph.D. degree at the Department of Computer Science in Georgia State University. His research interests include the Internet of Things, network optimization, and machine learning.

Zhipeng Cai received his Ph.D. and M.S. degree from University of Alberta, and B.S. degree from Beijing Institute of Technology, Beijing China in 2008, 2004, and 2001, respectively. Dr. Cai is currently a Professor in the Department of Computer Science at Georgia State University. Dr. Cai's research areas focus on Networking and Big data. He is the steering committee chair of the International Conference on Wireless Algorithms, Systems, and Applications (WASA). Dr. Cai served/is serving on the editorial boards of several technical journals (e.g., IEEE Internet of Things Journal, IEEE Transactions on Knowledge and Data Engineering (TKDE), and IEEE Transactions on Vehicular Technology (TVT)). Dr. Cai is the recipient of an NSF CAREER Award.

Yingshu Li received her Ph.D. and M.S. degrees from University of Minnesota-Twin Cities, Minneapolis, MN, USA in 2005 and 2003, respectively. She received her B.S. degree from Beijing Institute of Technology, Beijing, China in 2001. Dr. Li is currently a Professor in the Department of Computer Science at Georgia State University. Her research interests include Privacy-aware Computing, Management of Big Sensory Data, Internet of Things, Social Networks, and Wireless Networking. Dr. Li is the recipient of the NSF CAREER Award. She has served as an Associate Editor or Guest Editor for some prestigious journals such as ACM Transactions on Sensor Networks, IEEE Transactions on Computers, IEEE Transactions on Network Science and Engineering, and IEEE Internet of Things Journal. She has also served as a Steering Committee Chair, General Chair, Program Chair, and Technical Program Committee member for many international conferences.