CLONE: Collaborative Learning on the Edges

Sidi Lu, Yongtao Yao, and Weisong Shi *Fellow, IEEE*

***Abstract*— The proliferation of edge computing technologies has boosted the development of new applications for a plethora of edge devices. However, many applications face privacy issues and bandwidth limitations. To solve these limitations, we propose a collaborative learning framework on the edges, named CLONE, which is steered by the real-world datasets collected from a large electric vehicle (EV) company and a grocery store of a shopping mall, respectively. We categorize two application scenarios for CLONE, *i.e.,* CLONE in the training stage (CLONE_*training*) and CLONE in the inference stage (CLONE_*inference*). As to CLONE_*training*, we choose the failure prediction of EV battery and associated components as the first use case. While as for CLONE_*inference*, customer tracking in a grocery store is selected as another case study. In this work, the goal of the CLONE is to support real-time training and inference for connected vehicles and marketing intelligence services. Our experimental results on the EV data show that CLONE is able to reduce model training time without sacrificing algorithm performance. Furthermore, the experimental results on the video data from the grocery store reveal that CLONE is a useful approach to solve the multi-target multi-camera tracking problem in a collaborative fashion.**

***Index Terms*—Collaborative learning, Distributed artificial intelligence, Edge computing, Electric vehicle battery failure prediction, Multi-target multi-camera tracking**

I. Introduction

Edge Data Explosion and Challenges: The wide deployment of 4G/5G has enabled connected vehicles and Internet Protocol cameras (IP cameras) as the perfect edge computing platforms for a plethora of new intelligent services. In the meanwhile, with the burgeoning growth of the Internet of Everything (IoE), the amount of data generated by these edge devices has increased dramatically [1]. For instance, a connected and autonomous vehicle (CAV) could produce around one gigabyte of data per second [2] and generate more than 11 TB of privacy-sensitive data on a daily basis [3], [4]. Besides, IP cameras could generate over 2,500 petabytes of data per day [5], [6]. Such huge volumes of data inevitably bring challenges to researchers and domain experts — processing big data often requires a large amount of computation and memory resources, which hinders the application of deep learning algorithms on the resource-constrained edge devices with the stringent latency [7].

Popular Solutions and Limitations: In this context, cloud-only approaches [8], [9], [10] and cloud-edge methodologies [11], [12], [13], [14] have been proposed to process big data generated by edges in the past few years through computation offloading [14], [15], service scheduling [16], [17], virtual machine migration [18], [19], and so on. These methods both incorporate powerful cloud to resolve bandwidth limitations and provide near real-time services, but they both require sending amounts of raw data to the cloud via the wireless network. Consequently, data transferring may become the latency bottleneck and incur a high bandwidth cost [20]. Besides, even if data is compressed in the edge before being sent to the cloud [13], the original sensitive data might be exposed, which may create a potential threat of privacy leakage. Therefore, it can be seen that various intelligent applications on the edges are facing challenges in decreasing latency and protecting privacy [21].

Insights of Edge-Edge Research: In the meanwhile, as the computation and memory resources of edge devices have become more and more powerful [12], questions arise whether always incorporating cloud to process data is desirable moving forward, and whether we are able to implement novel approaches on the edge side to face the challenges of analyzing big data. With these insights, we propose a collaborative learning setting on the edges (CLONE) which is able to mainly demonstrate the effectiveness of latency reduction and privacy-preserving. The basic framework of the CLONE is shown in Fig. 1.

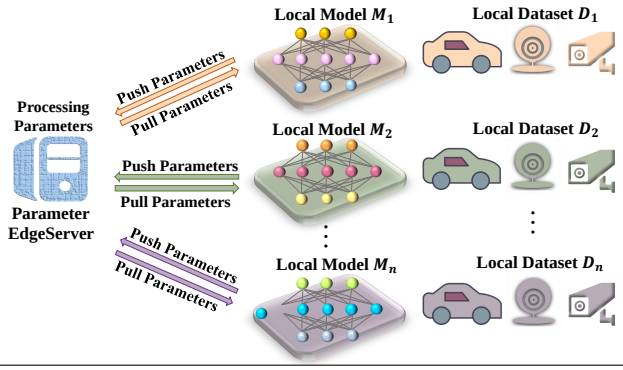


Fig. 1. The framework of CLONE. Each edge node trains/runs the neural network model locally based on its private data and push their own parameters to the Parameter EdgeServer during the training/inference process. The Parameter EdgeServer is responsible for performing aggregations or other necessary operations on the uploaded parameters and sending the updated parameters back to the edge nodes.

Contributions of This Work: Our core contributions are not in the development of machine learning-based models that are built on top of well-understood and mature models such as long short-term memory networks (LSTMs) [22], [23] and gradient boosting decision tree (GBDT) [24]. Instead, the core innovation of our study is in (i) combining state-of-the-art algorithms on top of the Federated Learning concept so that these algorithms can support dynamic distributed learning

This work was supported in part by National Science Foundation (NSF) grant IIS-1724227.

S. Lu and Y. Yao are with the Department of Computer Science, Wayne State University, Detroit, MI 48202, USA (e-mail: lu.sidi@wayne.edu; yongtaoyao@wayne.edu).

W. Shi is with the Department of Computer Science, Wayne State University, Detroit, MI 48202, USA (e-mail: weisong@wayne.edu).

TABLE I
SUMMARY OF OBSERVATIONS AND IMPLICATIONS. RF, GBDT, AND LSTMS REPRESENT THE RANDOM FOREST, GRADIENT BOOSTING DECISION TREE, AND LONG SHORT-TERM MEMORY NETWORKS RESPECTIVELY.

Observations	Implications	Sections
Adding driver behavior metrics will improve the prediction accuracy for the failure of EV battery and associated accessories.	Researchers could build EV failure prediction models based on driver behavior metrics to build personalized models in real-time.	Section III-C.
LSTMs outperform RF and GBDT in our two experimental groups.	LSTMs are potentially more accurate to predict EV failures.	Section III-C.
Compared with stand-alone learning (ALONE) on the edges, CLONE is able to reduce model training time without sacrificing algorithm performance. With more edge nodes involved, the advantages of CLONE in training time reduction will be more obvious.	CLONE is capable to provide efficient edge computing solutions for various edge services which have stringent requirements on the real-time performance and privacy protection.	Section III-F.
CLONE is a useful solution to solve the multi-target multi-camera tracking problem, and it enables applications such as visual surveillance and suspicious activity detection.	CLONE is also capable to track multiple players for team sports and track pedestrian for CAVs.	Section IV.
Compared with the cloud-edge approach, the main advantages of CLONE is to speed up the analysis tasks and protect user privacy better as it does not need to transfer any amounts of sensitive dataset via the network.	Researchers may leverage the CLONE framework to deal with real-time video processing task whose performance is significantly influenced by video transmission.	Section V.

(*e.g.,* collaborative reliability analysis and computer vision computation), and (ii) providing experimental evidence to establish that CLONE could be employed in both model training and inference phases, which provides actionable insights on using the CLONE framework to support other real-world intelligent services that require the collaboration of diverse edge devices.

To be concrete, this paper presents a collaborative learning framework for edges, which could be used in the training phases (CLONE_*training*) and inference phases (CLONE_*inference*) with the effectiveness of privacy-preserving and latency reduction. Regarding CLONE_*training*, we choose the failure prediction of electric vehicle (EV) battery and related components as the first use case. As to CLONE_*inference*, we choose customer tracking in a grocery store as the second use case, showing that CLONE is a powerful solution to the multi-target multi-camera tracking problem. Specific contributions are listed as follows.

- ❖ We have demonstrated the applicability of CLONE in two typical edge computing scenarios, covering both the training and inference at the edges in a collaborative fashion.
- ❖ To the best of our knowledge, this is the first work to predict an imminent failure of EV battery and associated accessories based on the real-world EV dataset which involves driver behavior metrics.
- ❖ Our analysis reveals that adding driver behavior metrics is able to improve the prediction accuracy for the failures of EV battery and associated accessories.
- ❖ CLONE_*training* has the capability to reduce the training time significantly without sacrificing prediction accuracy.
- ❖ CLONE_*inference* is a powerful solution to the multi-target multi-camera tracking problem.

Table I summarizes our observations and implications, and it serves as a roadmap for the rest of the paper. Section II provides the framework descriptions of CLONE and the main differences between CLONE_*training* and CLONE_*inference*, as well as the configuration information of the total edge devices employed in our two use cases. Section III and Section IV present our basic steps and results of the experiments for the two case studies. Section V reviews and compares with some previous related works. Finally, Section VI concludes our contributions and discusses the possible improvements of CLONE and other potential use cases.

II. CLONE DESIGN

Based on the progress of developing a neural network model, we divide two categories of application scenarios for CLONE, *i.e.,* CLONE in the training stage (CLONE_*training*) and CLONE in the inference stage (CLONE_*inference*). In this section, we introduce the core ideas of CLONE and illustrate the differences between CLONE_*training* and CLONE_*inference*. Then we describe the hardware configuration information for the experiment setup of the two case studies.

A. Framework Description

As has been mentioned, Fig. 1 presents the basic framework of CLONE. These two types of CLONE share the same core ideas — the training/inference tasks are both solved by a group of distributed participating edge nodes which are coordinated by a Parameter EdgeServer. Each edge node has its local training dataset that is never uploaded to the Parameter EdgeServer or transferred to the cloud. In CLONE, each edge node trains/runs the neural network model locally based on its private data and push their own parameters to the Parameter EdgeServer during the training/inference process. The Parameter EdgeServer is responsible for performing aggregations or other necessary operations on the uploaded parameters and sending the updated parameters back to the edge nodes.

B. Differences of Two Application Scenarios

However, there are three main differences between CLONE_*training* and CLONE_*inference*, *i.e.,* the types of transmitted parameters, the tasks of the Parameter EdgeServer, and the transmission manners are different. Table II summarizes the main differences between CLONE_*training* and CLONE_*inference*, and it serves as a roadmap for the next sections which describe two use cases related to these two application scenarios.

TABLE II
TWO TYPES OF APPLICATION SCENARIOS FOR CLONE.

	CLONE_*training*	**CLONE_*inference***
Use Cases	Predict the failures of EV battery and related accessories.	Customer tracking in a grocery store.
Parameters	The value of network parameters.	Appearance descriptors.
Tasks of EdgeServers	Parameter aggregation.	Store, update and delete trackers, which are defined in Section IV-D.
Transmission	Asynchronous.	Synchronous.
Sections	Section III.	Section IV.

C. Hardware Selection

Applying heterogeneous edge devices may impact the communication latency and therefore the experimental results. In the light of the foregoing consideration, we adopt three types of hardware in total shown in Fig. 2 — Intel Fog Reference Design (FRD), NVIDIA GPU Workstation, and Jetson TX2, to figure out the potential effects of heterogeneous hardware.

Fig. 2. Three categories of hardware. Subfigure 1-3 shows the Intel Fog Reference Design (FRD), Jetson TX2, and NVIDIA GPU Workstation, respectively.

The above three categories of edge devices are equipped with diverse processors, operating systems (OS), and so on. Intel FRD leverages the field-programmable gate array (FPGA) technology [25] to cast the proprietary hardware in a closed chassis allowing users to configure and program it for diverse edge use cases [26]. Unlike other edge devices that are equipped with GPUs, Intel FRD host two Intel Arria 10 GX1150 FPGAs that are capable to provide a more consistent throughput for various workload sizes [27]. As powerful hardware with the high-quality components (4 × GeForce RTX 2080 Ti graphics cards), NVIDIA GPU Workstation is capable to deliver the cluster-level performance for even the demanding applications [28], [29]. Jetson TX2 is one of the prominent power-efficient embedded AI platforms that enables server-class computing performance on the edge devices [30]. We describe the detailed configuration information of these edge devices in Table III.

TABLE III
HARDWARE SETUP FOR CLONE.

	Intel RFD	**NVIDIA GPU Workstation**	**Jetson TX2**
CPU	Intel Xeon E3-1275 v5	Intel Xeon E5-2690 v4	ARMv8
GPU	NONE	GeForce RTX 2080 Ti	NVIDIA Pascal
Frequency	3.6GHz	2.6GHz	2GHz
Core	4	14	6
Memory	32GB	64GB	8GB
OS	Linux 4.13.0	Windows 10	Linux 4.4.38

III. USE CASE I: FAILURE PREDICTION OF EV BATTERY AND RELATED ACCESSORIES

In this section, we present our experimental results of CLONE_*training* setting on the first use case — failure prediction of EV battery and associated accessories. The main idea is to predict upcoming failures in a collaborative fashion by capturing individuals' driving characteristics. We also compare the performance of stand-along learning (ALONE) with CLONE_*training* in two aspects: (i) training time, and (ii) evaluation scores including precision, recall, accuracy, and F-measure. After that, we discuss critical observations based on the comparison results.

A. Background

Recently, EVs have received significant attention as an essential part of the efficient and sustainable transportation system. As the key component of EVs, the battery system largely determines the safety and durability of EVs [31], [32]. Due to the aging process or abuse maneuvers during the real operation, various faults may occur at each constituent cell or the associated accessories. Therefore, it is essential to develop early failure detection techniques to ensure the availability and safety of EVs through anticipated replacements. Besides, we believe driver behavior metrics such as speed and acceleration reflect the usage of an EV, which could be the main root causes of EV's failure. Hence, how to build a personalized model to predict the failure of EV battery and related components, and what's the correlation between driver behavior metrics with the failure prediction are both open problems. In this work, we choose the failure of EV battery and related accessories as our case study to show how the CLONE solution trains the failure prediction models to ensure the sustainable and reliable driving in a collaborative fashion.

B. Data Description

The study of the first use case presents an analysis of EV health characteristics based on the data measured at and collected from a large EV company. We analyze three different models of EVs, and the corresponding data is reported and collected every 10 milliseconds during the whole 6-hour duration of the data collection period. The battery cells of these three EVs are made by the same battery vendor (BAK company) with different battery types — Lithium Cobalt Oxide (LCO) and Nickel Cobalt Manganese (NCM). The battery pack of each EV consists of 96 battery cells with 30 temperature sensors to detect and report cell temperatures in real-time.

In general, our data set is collected from the core control systems of EVs, which includes the vehicle control unit (VCU) [33], motor control unit (MCU) [34], and battery management system (BMS) [35]. BMS [35] is responsible for the battery maintenance and state estimation. The VCU [33], as a key component of the whole EV, sends orders to

other modules based on the driver manipulation (such as gear signal, accelerator pedal signal, and vehicle mode) via CAN communication network [36]. MCU [34] controls the wheel motor locally according to the command from VCU. Failures in the MCU may cause abnormal motor torque outputs, which in turn affect the vehicle safe driving. Therefore, the functional safety of these systems is particularly important. Fig. 3 shows the structure diagram of the core control systems.

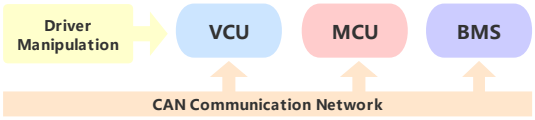


Fig. 3. Three Core Control Systems of EVs.

More specifically, we analyze EV data in the two aspects: (1) EIC attributes, and (2) driver behavior metrics. Here, EIC refers to electric, instrumentation, and computer control systems [37]. It includes battery features collected from BMS (most commonly used for EV battery durability analysis by other studies [31], [38], [39], [40], [41]) and the data reported from other control systems. The number of available features is more than 250, but not all features have useful value (except for NONE and constant value). For our study, 42 features listed in Table IV and Table V were selected as three EVs reported these attributes and the value of these attributes varies over time.

TABLE IV
SELECTED EIC FEATURES.

Voltage	Temperature	Power & Energy
BMS_BattVolt	InCar_Temp	BMS_BattSOC
BMS_CellVoltMax	Environment_Temp	BMS_MaxChgPwrCont
BMS_CellVoltMax_Num	BMS_BattTempAvg	BMS_MaxChgPwrPeak
BMS_CellVoltMin	BMS_Inlet_WaterTemp	BMS_MaxDchgPwrCont
BMS_CellVoltMin_Num	BMS_Outlet_WaterTemp	BMS_MaxDchgPwrPeak
MCUF_Volt	BMS_MaxTemp	VCU_Batt_Comp_Pwr
MCUR_Volt	BMS_MinTemp	VCU_Batt_PTC_Pwr
Current	BMS_TempMaxNum	**Error Info**
BMS_BattCurr	BMS_TempMinNum	BMS_BatterySysFaultLevel
MCUF_Curr	MCUF_Temp	BMS_Low_SOC
MCUR_Curr	MCUR_Temp	VCU_PTC_ErrSta

TABLE V
SELECTED DRIVER BEHAVIOR METRICS.

Driver Behavior	VehicleSpeed	Acceleration	Steering
	YawRate	WheelSpeedFL	WheelSpeedFR
	WheelSpeedRL	WheelSpeedRR	EmergencyStop
	AccPedalPosition	BrakePedalPosition	

Driver behavior metrics are collected from VCU and sensors.

Most of the selected features could be understood intuitively from Table IV and Table V; hence, we choose some vague features to give our explanations. Taking account into the CAN communication demand of different modules, the communication network has been set several bus nodes, including MCU for the front wheels (MCUF) and rear wheels (MCUR). "MCUF_Volt" and "MCUR_Volt" represent the voltage value of MCUF and MCUR. Positive Temperature Coefficient heater (PTC) is the heating unit in EV. "VCU_PTC_ErrSta" shows the failure status of battery PTC, which preheat the battery at low temperature to make sure the battery is able to work properly. "BMS_BatterySysFaultLevel" indicates the healthy states of BMS. State of Charge (SOC) is the indicator of left capacity, and "BMS_Low _SOC" shows the SOC states of EVs. "YawRate" reflects the overall tilt state of EVs. "AccPedalPosition" and "BrakePedalPossition" show the real-time percent of the gas pedal and brake pedal that the driver pressed on. As to the three error info attributes, they both show the degradation process from healthy to failed. Besides, "Comp" is an acronym of the compressor.

C. Experiment Design of ALONE

*1) **Experiment Goals**:* Before employing CLONE, we first combine the whole real-world dataset of three EVs to train different machine learning models on an Intel FRD. We term this stand-alone learning as ALONE. The goal is to figure out a suitable algorithm to predict failures, explore the influence of the driver behavior metrics on EV failure prediction, and provide baseline experimental results to compare it with the algorithm performance of CLONE_*training*.

*2) **Experiment Groups**:* To show the impact of driver behavior metrics on the battery failure prediction, we conduct experiments on two experimental groups. Our first step is to combine all selected EIC attributes and driver behavior metrics to train models using different methods, and we label this group as ED Group. Then, we exclude all driver behavior metrics but keep EIC attributes, and we denote it as E Group.

*3) **Experiment Setup**:* We tackle the failure prediction problem using random forest (RF) [42], gradient boosted decision tree (GBDT) [43], [44], and long short-term memory networks (LSTMs) [45], [46] since they have become highly successful learning models for both classification and regression problems. The models learned in this use case are implemented in Python, using tensorflow 1.5.0 [47], keras 2.1.5 [48], and scikit-learn libraries [49] for model building. To evaluate the proposed prediction approach, we use 5-fold cross-validation [50], which is a validation technique to assess the predictive performance of the machine learning models and to judge how models perform to an unseen dataset [51].

During the training phase, we first discover the best value of parameters for RF, GBDT, and LSTMs using the hold-out method [52]. Then, we conduct a grid search on these values of parameters to find the best combination that achieves the highest performance. During the testing phase, the first step is to determine how long the prediction horizon should be. After conducting a series of sensitivity studies showing the changes in the value of loss function which captures the error of the model on the training dataset, we choose 15 seconds as our prediction horizon so that it is able to predict failures for the next 1,500 data points. The second step is to measure the wellness of our prediction approaches. We take F-measure as the priority criterion, which is the harmonic average of precision and recall. We also consider other measure criteria simultaneously, such as precision, recall, and accuracy.

*4) **ALONE Experiment Results**:* For each method, we conduct the testing phase five times and get the precision, recall, accuracy, and F-measure for each time. We calculate the average values of these evaluation scores of five times, and the average values are shown in Fig. 4.

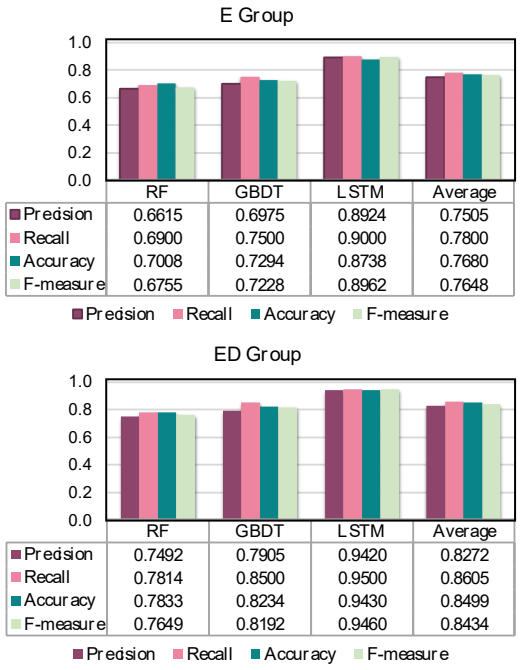


Fig. 4. ALONE Experiment Results.

*5) **Observations of ALONE experiments**:* Based on our experimental results, we have the following observations:

- ✠ Excluding driver behavior metrics results in around 8% reduction in the average F-measure (shown in Fig. 4).
- ✠ LSTMs outperforms RF and GBDT in both two groups.

The first observation shows that driver behavior metrics such as speed, acceleration, and steering are potentially good indicators of the failures of EV battery and associated accessories. However, driving is significantly influenced by the current driver, weather, location, and traffic conditions, which requires a personalized model to capture the real-time driving pattern.

As to the second observation, although the degradation process of EV failures is complicated, battery and associated accessories do have their hidden failure patterns [32], [53]. As the more sophisticated neural networks including historical information, LSTMs are good at capturing hidden patterns of battery failures based on historical data and generating a sequence of predictive data points to predict the incoming patterns.

D. *CLONE*_training *Framework*

As shown in Fig. 5, each edge node (vehicle) is responsible for continuously performing training locally based on its private data. When a single edge node (vehicle) finishes one epoch [54], which refers to the number of iterations related to the total input dataset, it will push the value of current parameters to the Parameter EdgeServer, where the parameter values are aggregated to compute the weighted average value. Then, each edge node (vehicle) will immediately pull the updated parameter values from the Parameter EdgeServer, and set the updated parameters as their current parameters to start the next epoch. The above steps will be repeated as necessary.

Note that when a new edge node (vehicle) joins in, it will pull the current aggregated parameters from the Parameter

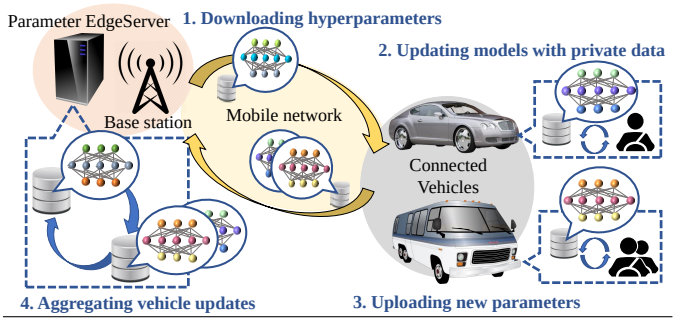


Fig. 5. Each vehicle trains the neural network model locally based on its private data. Then, the value of current parameters from each vehicle is uploaded to the Parameter EdgeServer, where those parameters are aggregated and sent back to vehicles.

EdgeServer first, and set them as the initial parameters for the first round of training, which is able to speed up the training process for unseen edge nodes (vehicles). Besides, since it is asynchronous communication, for each edge node (vehicle), there is no need to stop and wait for other edge nodes to complete an epoch, which greatly reduces the latency. To illustrate the aggregation protocol of CLONE_*training*, we need to introduce the loss function first, which is defined as follows:

$$Loss = \sum_{i} \left[\hat{y}_{(i)} * \lg(y_{(i)}) + (1 - \hat{y}_{(i)}) * \lg(1 - y_{(i)})\right] \quad (1)$$

Here, $\hat{y}_i$ is the predicted output of a machine learning model, and the scalar y_i is the desired output of the model for each data sample i. We then define the formula to aggregate and update parameters as follows:

$$\begin{cases} P(p) \leftarrow \frac{Loss(v)}{Loss(p)+Loss(v)} P(p) + \frac{Loss(p)}{Loss(p)+Loss(v)} P(v) \\ Loss(p) \leftarrow Loss(v) \end{cases} \quad (2)$$

Where P represents the value of a parameter, and $Loss$ stands for the value of the loss function. Besides, p refers to the Parameter EdgeServer, and v represents a specific edge node (vehicle). According to the above formula, if the model on an edge node achieves more accurate results (lower value of loss function), it will assign a higher weight to the parameters uploaded by this edge node, so that the required training time could be minimized efficiently to reach a certain accuracy level.

E. *Implementation of CLONE*_training

*1) **Experimental Goals**:* We can see in Section III-C5 that driver behavior metrics have non-negligible impacts on the prediction of EV failures, and employing LSTMs could achieve better results compared with RF and GBDT. Therefore, in this section, we aim to deploy LSTMs-based collaborative learning approaches on edges based on EIC attributes and driver behavior metrics. The main goal is to construct personalized models by continuously tuning parameters in a collaborative fashion while protecting driver privacy and predicting failures in real-time.

*2) **Hardware Selection**:* To build a heterogeneous hardware cluster to represent different models of EVs, we adopt two different types of hardware — Intel FRD and Jetson TX2, with

different CPUs, memory, and so on (shown in Table III). More specifically, we choose one Intel FRD as the Parameter EdgeServer, and we treat other two Intel FRDs and one Jetson TX2 as the edge nodes for vehicles to continuously "learn" latent patterns.

*3) **Experiment Setup**:* In Section III-C, we trained an accurate LSTMs model with 4 layers on the front and followed by a fully connected layer (dense layer). Now, we aim to deploy a distributed LSTMs in a collaborative fashion with the same number of layers (same hyperparameters) on edge nodes. We first distribute our whole dataset into three edge nodes, so that each edge node (vehicle) has its locally private dataset. Note that the data collection periods of three EVs are not the same, which means that the dataset is not evenly distributed.

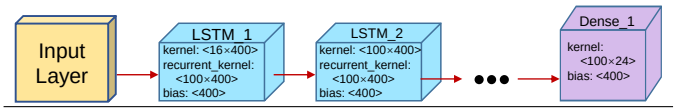


Fig. 6. Model Parameters.

Fig. 6 shows the parameter distribution of the LSTMs model on the first two LSTMs layers (marked as LSTM_1 and LSTM_2) and the last fully connected layer (labeled as Dense_1). The "kernel" and "recurrent_kernel" are the parameter vectors, and the number inside angle brackets represents the shape (size) of parameters for each vector. We can see that parameter vectors are usually high-dimensional, *i.e.,* there are a huge amount of parameters. For example, $\langle 16 \times 400 \rangle$ indicates that there are 16×400 of parameters. Our whole network contains up to 297,700 parameters, including the weights and the biases. Weight is able to reflect the strength of the connection between input and output. Bias shows how far off the predictions are from the real values. During each epoch, edge nodes (vehicles) push all parameters to the Parameter EdgeServer to get the updated value of parameters. Same as Section III-C, we use a 5-fold cross-validation method to evaluate experiment results of CLONE, which will be present in Section III-F to have a direct comparison with ALONE.

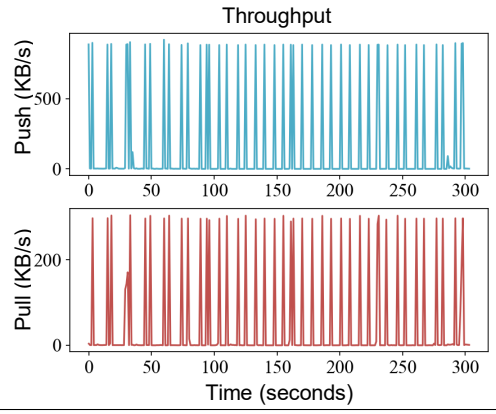


Fig. 7. Throughput at the Parameter EdgeServer.

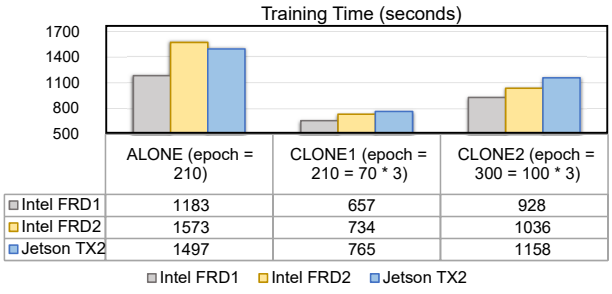
*4) **Throughput**:* Fig. 7 shows the throughput at the Parameter EdgeServer when three work nodes are working together. The Parameter EdgeServer receives parameters during the push process and sends parameters during the pull process. We can see that the data throughput peak is relatively stable and the peak appears intermittently. Besides, the maximum throughput for the push and pull process is around 750 KB/s and 250 KB/s, indicating that there is no big pressure on the network throughput. Fig. 7 also proves that the push process is usually much slower than the pull process for CLONE_*training*, which was concluded by the work of [55]. This observation shows the importance to investigate methods that are able to reduce the communication latency of the push process in the future work.

*F. Comparison between CLONE_*training *and ALONE*

In this subsection, we present the experimental results of CLONE_*training*, and compare it with the algorithm performance of ALONE in two aspects: (i) training time, and (ii) evaluation scores including precision, recall, accuracy, and F-measure.

To have a clear comparison, we conduct experiments on three experimental groups. The first group is ALONE, and we set the epoch of ALONE equal to 210. The second group is CLONE with the epoch of 70 for each edge node, and we label it as CLONE1. Since there are three edge nodes in CLONE1, the equivalent number of iterations in total is also 210 (70×3). As to the third group (CLONE2), the epoch is 100 for a single edge node, which results in 300 (100×3) of total iterations.

*1) **Training Time Comparison**:* We first profile and compare how the training time is spent on the three experimental groups, which is shown in Fig. 8.



	ALONE (epoch = 210)	CLONE1 (epoch = 210 = 70 * 3)	CLONE2 (epoch = 300 = 100 * 3)
Intel FRD1	1183	657	928
Intel FRD2	1573	734	1036
Jetson TX2	1497	765	1158

Fig. 8. Training Time Comparison.

For ALONE, the used training time varies with different edge devices — it takes 1183s and 1573s on two Intel FRDs respectively, while taking 1497s to execute the training task on Jetson TX2. As to CLONE1, the training time of each edge node is much lower than the training time of the single edge node of ALONE. Since there are three edge nodes in CLONE1, the training time of CLONE1 should be one-third of ALONE theoretically. However, due to the inevitable delay of the parameter transmissions, the training time of CLONE1 is greater than one-third of ALONE. We then increase the epoch value from 70×3 to 100×3 (CLONE2), and it can be seen that the required training time is longer than CLONE1 as it has a larger number of iteration related to the input dataset during the training phase, but it still less than ALONE which has a lower epoch value. *Note that with the participation of more*

*edge nodes and larger size of the input dataset, the advantages of CLONE_*training *in training time reduction will be more obvious.* Note that the training time in Intel FRD1 and FRD2 are different, which might be due to the difference in the time length of usage, system environment, and the distance to the router.

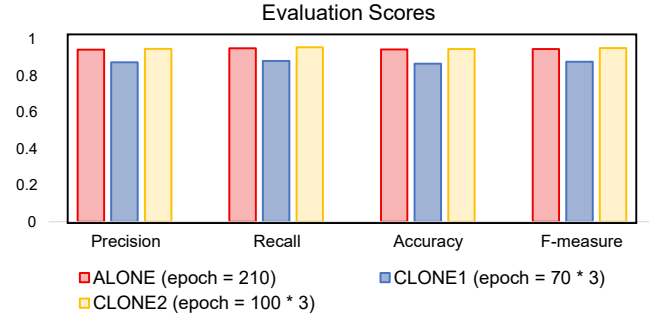


Fig. 9. Algorithm performance.

2) ***Evaluation Score Comparison****:* We then calculate the average evaluation scores for each group, which is shown in Fig. 9. Comparing ALONE and CLONE1, we can see that the overall evaluation scores of CLONE1 are lower than ALONE. This could be caused by the fact that the prediction accuracy will be improved with the increasing number of iterations passing the full dataset through the current model, and ALONE has a higher epoch value (210) than CLONE1 (70). However, the evaluation score of CLONE could be further improved by increasing the epoch value, *e.g.,* the evaluation scores of ALONE and CLONE2 are relatively equal. Besides, in the CLONE setting, due to the hardware difference, powerful edge nodes that have a higher FLOPS may train the model faster than other edge nodes. Here, FLOPS is an abbreviation for floating-point operations per second ("S" stands for the time unit, *i.e.,* second), which is often used to estimate the computation power of a device (greater GFLOPS usually indicates a more powerful computation resource and faster computation speed). Therefore, during the same training time period, powerful edge nodes can achieve a higher value of epoch, *i.e.,* finish more training passes than other edge nodes; therefore, the output of a powerful edge node will make a greater contribution to the training of the global model, which leads to better performance when the trained model performs inference on the data generated by the powerful node. When the parameters of the poor training results are uploaded to the Parameter EdgeServer, the global accuracy of CLONE1 will be influenced. This may explain the performance gap between ALONE and CLONE1 whose total epoch values are the same.

However, when we further increase the value of epoch (CLONE2), it is capable to achieve high evaluation scores as ALONE. Note that, by observing Fig. 8, the training time of CLONE2 is much lower than ALONE, even though CLONE2 has a higher epoch.

G. Discussion

Compared with ALONE, CLONE_*training* is able to reduce model training time without sacrificing algorithm performance. With more edge nodes involved, the advantages of CLONE in training time reduction will be more obvious. Besides, CLONE_*training* provides personalized models to predict the failures of EV battery and associated accessories considering the current driver behaviors, and CLONE_*training* is capable to speed up the analysis tasks while protecting user privacy better as it does not need to transfer any portion of the sensitive dataset via the network.

IV. Use Case II: Customer Tracking in a Grocery Store

In this section, we describe our experiment steps and results of CLONE_*inference* setting on the second use case, *i.e.,* customer tracking in a grocery store, showing how CLONE_*inference* could be adopted to solve the multi-target multi-camera tracking (MTMCT) problem. Different from the first case study that compares ALONE with CLONE_*training* in terms of the training time and evaluation scores, we focus on three aspects: (i) inference time, (ii) frames per second (FPS), and (iii) throughput per second.

A. Background

As another perfect edge computing platform, surveillance embedded IP cameras are widely adopted in the crowded indoor region and public outdoor area [56]. For example, at grocery stores, IP cameras are used to record video segments with the initial purpose of theft deterrence and employee protection. With the prevalence of computer vision and machine learning technologies, these IP cameras have been equipped with novel real-time marketing intelligence functions which were impossible before, such as customer facial recognition, personalized shopping assistance, and purchase habit modeling [57], [58], [59], [60]. Although these new functions of IP cameras are capable to help retailers better market their product, the video data generated from the IP cameras is privacy-sensitive as it contains a huge amount of personal information [61], [62]. Therefore, how to prevent machine learning-driven surveillance from violating privacy and confidentiality is one of the most essential challenges [63].

We choose customer tracking in a grocery store as one of our case studies to show how CLONE could be used in edge video analysis, solving the multi-target tracking problem under the multi-camera system. Although a multitude of previous works achieve excellent performance in multi-target tracking [64], [65], [66], [67], [68], multi-camera system tracking problem still remains as a very new topic comparing to the classical single-camera tracking [69], [70].

B. Data Description

The study of the second use case is steered by the video dataset collected from 8 retail surveillance cameras at a middle-sized grocery store during the same time period. More specifically, we collect the 3-hour video data from 8 cameras of 3 manufacturers and 6 models — Arecont Vision (AV20175DN-NL), Axis (Q6000-E, Q6045-E), and Canon (VB-H41, VB-H45, VB-M50B). These cameras capture the top-down views monitoring for both the incoming and outgoing customer flow at the entry gate, exit gate, check-out

gate, condiment section, snack foods section, and cosmetic section, providing both disjoint and overlapped fields of view. We connected these cameras via Milestone ONVIF Bridge [71], which is an open ONVIF interface of video-sharing from XProtect video management software (VMS) systems to other IP-based systems, to collect and store over 22 gigabytes of full HD videos (1920 $\times$ 1080 pixels) with a frame rate of 30 FPS (frames per second). Fig. 10 present eight different views from eight IP cameras. In order to protect privacy, we only illustrated the black-white version of the video image in this paper.

Fig. 10. Examples of the eight views from eight IP cameras.

C. Experiment Design of ALONE

We first conduct multi-target tracking experiments on an Intel FRD and the NVIDIA GPU Workstation respectively based on the three hours of video dataset of a single camera, before employing CLONE_*inference*. Note that the edge nodes (single camera) of CLONE_*inference* just have two more operations than ALONE, *i.e.,* pushing appearance descriptors and pulling appearance descriptors, which will be further described in Section IV-D; hence, we conduct the ALONE experiment first to serve as a baseline for the evaluation of the subsequent CLONE_*inference* experiment.

*1) **Algorithm Description of Multi-target Tracking**:* The solution of multi-target tracking could be broadly divided into two phases: (i) detecting phase, and (ii) tracking phase. During the detecting phase, we implement YOLOv3 (You-Only-Look-Once-v3) [72], a fast and accurate detector to detect targets and compute bounding boxes, so that each target is enclosed in a corresponding bounding box for each frame. We then conduct target tracking through the Deep SORT algorithm [67], [68], an extension to SORT [73] that incorporates deep appearance descriptors to reduce the number of identity switches caused by the long periods of occlusions. When a new detection is associated with a tracked target, the newly detected bounding box is used to update the current state of the target.

Appearance Descriptors: For each bounding box, Deep SORT first generates two types of association descriptors — appearance descriptor and motion descriptor, capturing the corresponding appearance and motion information respectively. The appearance information of the bounding box is a 128-dimensional feature obtained through the Wide Residual Network [74] with two convolutional layers and six residual blocks, which makes appearance descriptors irreversible to the targets' original images. Deep SORT calculates the smallest cosine distance between the appearance descriptors of detected bounding boxes and the tracked bounding boxes to quantify the appearance association.

Motion Descriptors: The motion information is defined on the eight-dimensional state space $(x, y, r, h, v_1, v_2, v_3, v_4)$, where x and y represent the horizontal and vertical pixel of the center position of a bounding box, r and h indicate the aspect ratio and the height of the bounding box respectively, while v_1, v_2, v_3, v_4 represent the velocities at the four vertices of the bounding box. Kalman filter framework [75] is responsible for predicting targets' next motion states. Deep SORT then calculates the Mahalanobis distance between the motion descriptor of the detected bounding boxes and predicted motion states to characterize the degree of motion matching.

Short-term and Long-term Tracking: The Mahalanobis distance gives the information of targets' possible locations from the motion aspect, which is particularly effective for short-term prediction and tracking. Besides, the cosine distance considers appearance information, which is particularly useful to reduce identities switches caused by the long-term occlusions, especially when motion is less discriminative. Consequently, Deep SORT combines the Mahalanobis distances and cosine distance as an indicator to figure out whether a frame-by-frame association is admissible to keep tracking multiple targets.

*2) **Experiment Setup**:* We employed YOLOv3 [72] and Deep SORT algorithms [76] to detect person and then achieve multi-target tracking. The models learned in this use case are also implemented in Python, using tensorflow 1.5.0 [47], keras 2.1.5 [48], and scikit-learn libraries [49] for model building. We conduct experiments on an Intel FRD and the NVIDIA GPU Workstation respectively. Note that NVIDIA GPU Workstation is a more powerful device than Intel FRD, and it is capable to provide a cluster-level performance. We compare the experimental results in terms of inference time and frames per second (FPS), to explore the impact of different hardware on the speed of video analysis and inference for multi-target tracking problem, and to provide a baseline performance results for the experiments of CLONE_*inference*.

*3) **ALONE Experiment Results**:* Our experimental results show that the average FPS on the Intel FRD is around 3.25, and the average FPS on the NVIDIA GPU is 14.88. To better explore and evaluate ALONE results in terms of the

inference time, we also further divide the solution of multi-target tracking into four phases: (i) target detection, (ii) feature extraction, (iii) data association, and (iv) tracker output. "Target detection" refers to the process of leveraging YOLOv3 to detect customers and compute the corresponding bounding boxes. "Feature extraction" indicates the phase of generating appearance and motion descriptors for each customer frame by frame. "Data association" represents the process of combining the Mahalanobis distances and cosine distance to figure out if a specific association is acceptable. Finally, "tracker output" refers to the phase of getting the output of the trackers, *i.e.,* showing the qualitative results based on the original video. Fig. 11 shows the average inference time per frame on the aforementioned four phases of multi-target tracking.

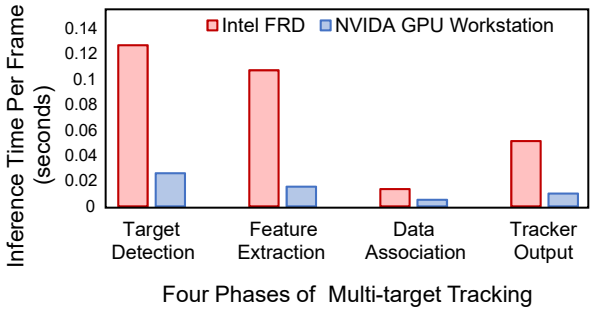


Fig. 11. Inference time per frame on the four phases of multi-target tracking.

*4) **ALONE Experiment Observations**:* Based on our experimental results, we have the following observations:

- NVIDIA GPU workstation takes significantly less time in four stages of inference than the Intel FRD.
- The average FPS of the complete inference process on the Intel FRD is not high — around 3.25.
- No matter on the Intel FRD or the NVIDIA GPU, "target detection" takes the longest time during the inference process, while "data association" takes the shortest inference time on average.

The first observation is in line with expectations since the NVIDIA GPU workstation is more powerful than the Intel FRD. As to the second and the third observation, we give our explanation as follows. The not high value of FPS on the Intel FRD is not only due to the limitations of hardware processing capability but also because of the full HD video's high resolution, which penalizes the frame-by-frame video analysis speed. Most importantly, the algorithms used to detect targets also significantly influence the performance of the multi-target tracking approach (Deep SORT). In this use case, we implement YOLOv3 to detect targets and compute the corresponding bounding boxes. However, the model size of YOLOv3 is around 237 MB, and itself is responsible for detecting up to 80 categories of objects for each frame. Although in this use case, we only pay attention to person detection (customer detection), the detection of the person actually masks all non-human detection results after YOLOv3 detects all types of objects in each frame. In other words, this algorithm is not only for human detection, which is one of the reasons for the slowness. For the use case that only focuses on person detection or single object detection, exploring a more suitable approach is necessary.

Besides, we can see in Fig. 11 that leveraging NVIDIA GPU Workstation may help to shorten inference time. However, its high price has determined that it is unrealistic to widely adopt NVIDIA GPU Workstation currently. Our CLONE experiments below are all performed on the Intel FRDs (isomorphic), and we focus on the performance changes of CLONE_*inference* as the number of edge nodes increases.

*D. CLONE_*inference *Framework*

CLONE_*inference* aims to give a simple yet powerful solution to capture the multi-camera trajectories. For a specific target, we term a set of the historical appearance descriptors and the corresponding ID as a tracker. Since the Parameter EdgeServer is responsible for storing, updating, and deleting trackers, we describe CLONE_*inference* from the three aspects as follows.

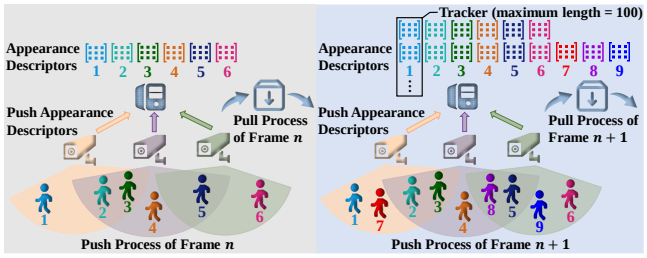


Fig. 12. CLONE in the inference stage (CLONE_*inference*). We take two consecutive frames as an example to illustrate how CLONE is able to solve the multi-target multi-camera tracking problem.

*1) **Store Trackers**:* Fig. 12 shows the detailed framework of CLONE_*inference* for the use case of multi-target multi-camera tracking (MTMCT). Each edge node (camera) first detects targets, assigns unique IDs, and computes the associated appearance descriptors within each frame via the local algorithms. The appearance descriptor is a 128-dimensional feature obtained through a state-of-the-art neural network which is illustrated in Section IV-C. Due to the internal complexity and nonlinear structure of the neural network, the appearance descriptor is irreversible to the target's original images that could potentially be mishandled. When an edge node (camera) finishes the computation of all targets' appearance descriptors for a frame, it will push the current descriptors and associated IDs to the Parameter EdgeServer immediately. The Parameter EdgeServer stores the appearance descriptors from all edge nodes (cameras). After all edge nodes finish the pushing process for a frame at the same timestamp, they will pull trackers from the Parameter EdgeServer (this step is based on the requirement of processing videos frame-by-frame, so all edge nodes are strictly synchronized, and we discussed the limitations and improvements of this step in Section IV-G). Next, the local algorithm of edge nodes quantifies the appearance association between the newly detected targets with trackers. The above steps will be repeated as necessary to keep tracking. Section IV-C will provide a detailed description of the algorithm used to quantify the appearance association.

*2) **Update Trackers**:* In this framework, the Parameter EdgeServer only stores up to a hundred latest appearance descriptors for a unique ID, *i.e.,* the maximum number of the historical appearance descriptors contained in a tracker is 100. This is because we only need to focus on the latest appearance descriptors to keep tracking a target. Besides, storing the latest descriptors is able to control the storage and computation at a suitable intensity. An example of updating trackers is shown in Fig. 13. As to a newly detected target on an edge node, assume the associated ID has been already stored on the Parameter EdgeServer, if the length of the corresponding tracker is less than 100 (such as target 1,2,5,6 in Fig. 13), then this new appearance descriptor will be added to the related tracker. Otherwise, the latest 100 appearance descriptors will replace the previously stored descriptors. When there is no corresponding ID stored on the Parameter EdgeServer, this appearance descriptor will be assigned a new ID to create a new tracker (such as target 7 in Fig. 13).

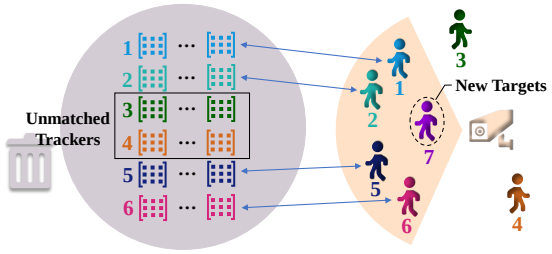


Fig. 13. An example of updating and deleting trackers. The right circle represents a set of the trackers on the Parameter EdgeServer, and the left sector represents a set of the detected targets and their corresponding IDs on an edge node.

*3) **Delete Trackers**:* Tracked targets do not always stay in the surveillance area. Therefore, in this work, the Parameter EdgeServer is also responsible to free up storage space by deleting the unmatched trackers that have not been associated with detected targets in the last consecutive 30 frames. An example of deleting trackers is also shown in Fig. 13.

*E. Implementation of CLONE_*inference

*1) **Experiment Goals**:* The goal of this subsection is to figure out the impact of different numbers of edge nodes on the performance of the CLONE_*inference*. In this use case, we do not consider the influence of heterogeneous edge devices. The main reason is that we want to process the video stream in real-time, which requires all edge nodes to process the video roughly synchronously, so the time that each edge node waits for each other could be reasonably controlled. For the heterogeneous edge devices, we discuss the possible solutions to solve the MTMCT problem in Section IV-G.

*2) **Experiment Setup**:* Based on the description of the multi-target tracking algorithm (presented in Section IV-C1) and the framework description of CLONE_*inference* (illustrated in Section IV-D), we assemble a processing pipeline of CLONE_*inference* to solve the MTMCT problem shown in Fig. 14. In Fig. 14, the yellow area indicates the CLONE_*inference* setting on the case study of MTMCT, and the rest area of Fig. 14 presents how to track multiple targets.

Although the Deep SORT algorithm extracts not only appearance descriptors but also motion descriptors, the participating edge nodes just need to transmit appearance descriptors to the Parameter EdgeServer. The reasons as follows. The motion information of the same target captured by different cameras is inevitably different. For a particular camera, the motion descriptors obtained locally only assist its camera to track the target. Therefore, transmitting motion information does not help the collaborative working of the various cameras, which is why we do not consider the transmission of motion descriptors in CLONE_*inference*.

*3) **Experimental Results**:*

Tracker Output: The qualitative result of CLONE_*inference* is shown in Fig. 15. Fig. 15 presents two examples of multi-camera results from our trackers on the grocery store video dataset with the same time point. The two customers with id 117 and id 118 in Fig. 15 are simultaneously tracked in two different camera videos. The same customer is assigned with the same id on both edge nodes (cameras), which is exactly the purpose of CLONE_*inference*.

Throughput: To show the impact of the different numbers of participating edge nodes on the performance of CLONE_*inference*, we conduct experiments on four experi-

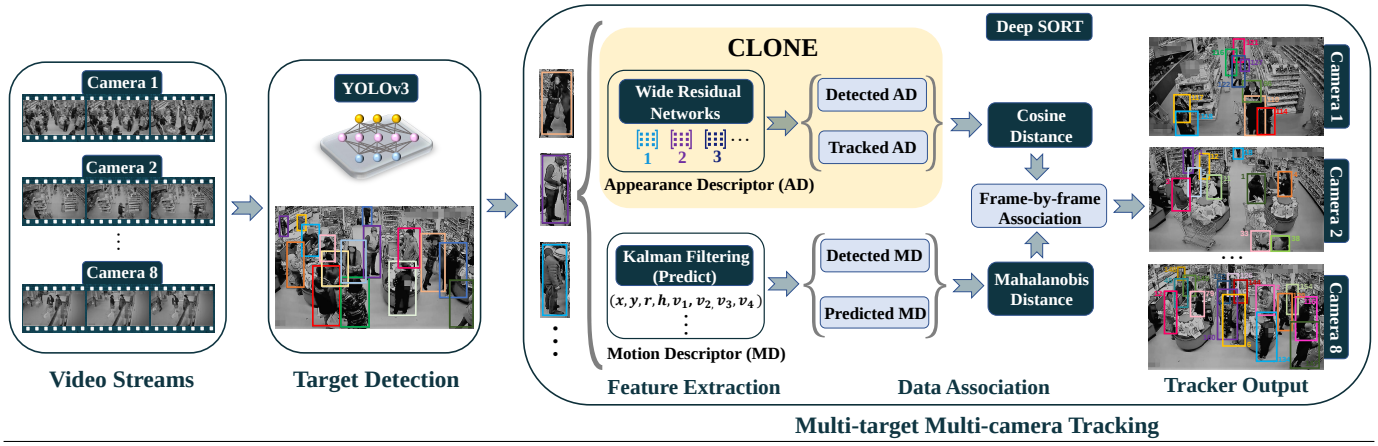


Fig. 14. An illustration of our pipeline for the multi-target multi-camera tracking in CLONE.

Fig. 15. An exemplary output of CLONE's trackers.

mental groups, which consist of 2, 3, 5, and 8 Intel FRDs respectively. Fig. 16 shows the throughput at the Parameter EdgeServer for these four groups when 2, 3, 5, and 8 edge nodes are working at the same time. The Parameter EdgeServer receives parameters during the push process and sends parameters during the pull process.

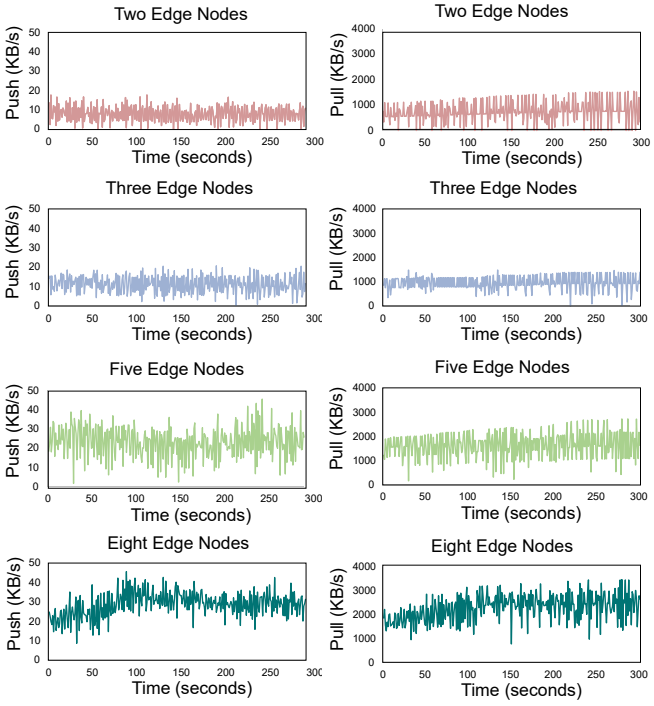


Fig. 16. Throughput of four experimental groups at the Parameter EdgeServer.

Unlike the throughput result of Section III-E4, we can see in Fig. 16 that the pull process has overall much higher throughput than the push process. This is because, in the push process, only the appearance descriptors related to the currently detected targets are pushed. However, in the pull process, edge nodes need to pull all the stored trackers, *i.e.,* stored ids and the corresponding historical appearance descriptors, from the Parameter EdgeServer. The heavier workload of the pull process leads to a higher throughput.

*F. Comparison between CLONE_*inference *and ALONE*

In this subsection, we present the experimental results of CLONE_*inference* (four experimental groups) with ALONE in terms of accuracy and FPS.

As to ALONE, the accuracy is around 63.7%, and for the CLONE, its accuracy achieves 56.4% on average. We also observed that with the increasing number of edge nodes, the accuracy of CLONE_*training* will decrease.

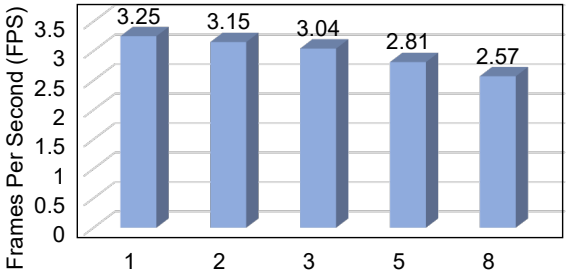


Fig. 17. The Average FPS on a single edge node for ALONE and the four CLONE_*inference* groups.

Fig. 17 shows the average value of FPS on a single edge node for ALONE (the number of edge nodes is one) and the four CLONE_*inference* groups, which consists of 2, 3, 5, and 8 participating edge nodes respectively. It can be seen that when there are more edge nodes working together, the average FPS on an edge node will decrease, which is in line with our expectation. The increasing number of the participating edge nodes leads to the increasing number of frames that need to be processed at the same time, which in turn leads to an increasing amount of appearance descriptors' transmission. Therefore, FPS will decrease when there are more edge nodes participate in the CLONE_*inference*. But we can also see in Fig. 17 that FPS is not decreasing dramatically. We may adopt more efficient neural networks and figure out more suitable parameters for CLONE_*inference* to control the reasonable value of FPS when there are more edge nodes working together.

G. Discussion

In this case study, we focus on the isomorphic edge devices for CLONE_*inference* to make sure the video streams of different cameras could be processed roughly synchronously. Employing isomorphic edge nodes is able to guarantee to some extent that the speed at which they process video streams is not much different, which is capable of reasonably controlling the time that each edge node waits for each other. Consequently, the latency could be controlled. However, in the real application scenarios, always adopting isomorphic edge devices is not easy or even unrealistic. The first solution is to make the edge device that processes video data faster "wait" for slower devices, but this solution inevitably influences the efficiency of CLONE_*inference*. The second solution is to let

the edge device that processes the video data slower to "chase" faster one, *i.e.,* skipping unprocessed frames to catch up with the faster devices. The disadvantage of the second solution is that the accuracy of the algorithm could be negatively affected.

Besides, in this work, we deal with the offline video data, and we claimed "after all edge nodes finish the pushing process for a frame at the same timestamp, they will pull trackers from the Parameter EdgeServer" (described in Section IV-D), to process video data frame-by-frame, making all edge nodes are strictly synchronized. Otherwise, for two edge nodes with different analyzing speed, the time gap between the timestamps of the current frames that they are processing will become larger and larger. However, in the real application scenarios, CLONE_*inference* could be employed to analyze live video. In this case, each edge node extracts the frame corresponding to its current time, and the edge node with slower processing speed will automatically skip some frames so that the video frames processed by each edge node are roughly synchronized.

V. Related Work

Recently, machine learning algorithms have been widely deployed across a variety of domains, but it is difficult to deploy neural networks on edge devices with limited computation and storage resources. In this context, model compression technologies (such as parameter pruning [77], [78], low-rank approximation [79], [80], and knowledge transfer [81], [82]) as well as lightweight machine learning algorithms (such as MobileNets [83], Xception [84], and Squeezenet [85]) have been proposed to tackle this challenge, but they can not guarantee to solve the problem completely when the training data and machine learning models are particularly large. Another popular choice to address this limitations is employing distributed data flow systems such as MapReduce [86], Spark [87], Naiad [88], and XGBoost [89]. They are able to robustly scale with the increasing dataset size, but when training complex neural network tasks, the data flow systems fail to scale as they are inefficient at executing iterative workloads [90], [91].

This restriction sparked the development of distributed machine learning (DML) algorithms [92], [93], [94], [95], [96], [97] to aid the implementation of complex iterative neural networks. Later on, federated learning (FL), a novel DML, was initially proposed by Google researchers [55], which is perfect to train models in a collaborative manner while preserving the privacy of sensitive data. Google describes the approach of FL in four simple steps — nodes download the current model from the server and start training based on local data. Then, each node computes an updated model and sends the updates to the server. Next, the server aggregates these updates. After that, the server sends a federated model to nodes (repeat as necessary) [98], [99], [100], [55], [101].

The main difference between the conventional DML and FL is that FL using decentralized data residing on the end devices (edge computing environment), while conventional DML is employed in the data center environment [102]. In data centers, worker nodes fetch the data from the shared storage at the beginning of the training process. Hence, the data samples obtained by different nodes are usually independent and identically distributed (i.i.d.). However, in FL, the data is collected at edges directly and stored persistently; thus, the data distribution at different edge nodes is usually non-i.i.d.

Our work is inspired by FL, but there are four main differences between the conventional FL and CLONE. Simply speaking, conventional FL synchronously aggregates the parameters during the training process to learn a communal model. While CLONE could be employed in the training phase and inference phase, so the tasks of the Parameter EdgeServer include but not limited to aggregate parameters. Table VI summarizes the main differences. Traditional FL only uses the boosting technique to optimize the distributed isomorphic model, while CLONE generates different architecture models for different scenarios.

TABLE VI
Differences between conventional FL and CLONE.

	Conventional FL	**CLONE**
Application Scenarios	Training.	Training and inference.
Parameter Transmission	Synchronous.	Asynchronous / Synchronous.
Personalization	No personalization.	Personalized model.
Tasks of Parameter EdgeServers	Aggregating parameters.	Including but not limited to aggregate parameters.

TABLE VII
Advantages of CLONE.

Advantages	**Description**	**Outperform**
User Personalization	Each edge node trains a model locally based on its private data; local model will be updated according to the dynamic changes of the local dataset.	DML, FL, Cloud-only method.
Privacy Preserving	The training data could be always kept in its original source.	DML, Cloud-only method, and cloud-edge method.
Latency Reduction	Analyze data on board; edge nodes just need to push parameters to the Parameter EdgeServer instead of amounts of data.	DML, Cloud-only method, Cloud-edge method.
Security Protection	Reduce security risks by limiting the attack surface to only the edges, instead of the edges and the cloud.	Cloud-only method, Cloud-edge method.
Offline Working	Since the models are present on the edges, the local model work even when there is no internet connection available.	Cloud-only method, Cloud-edge method.

At the same time, with the prevalence of DML, cloud-only approaches [8], [9], [10] and cloud-edge methodologies [103], [12], [13], [102] have been proposed to tackle the challenges of analyzing big data in the real-time fashion, which need to upload a portion of raw data to cloud. Different from these approaches, we just focus on collaborative learning on only the edge side, and CLONE has three main strengths: (i) reduce power consumption by eliminating the use of central data centers, (ii) speed up the analysis tasks as it always analyzes real-time data on-board and just need to communicate with the parameter edge about the current parameters [104], [22], and (iii) CLONE is capable to reduce security risk by limiting the attack surface to only the edges. As a summary,

the advantages of CLONE compared with the conventional DML, FL, cloud-only approaches, and cloud-edge methods are shown in Table VII.

VI. Conclusion and Future Work

Conclusion Remarks: In this paper, we proposed a collaborative learning framework on the edges (CLONE), including CLONE_*training* and CLONE_*inference*. It demonstrated the effectiveness of privacy serving and latency reduction. For CLONE_*training*, we chose the failure prediction of EV battery and associated components as our first case study, and the corresponding experimental results showed that CLONE_*training* could reduce model training time without sacrificing algorithm performance. Besides, we found that adding driver behavior metrics could improve the prediction accuracy for the EV failure prediction. As to CLONE_*inference*, customer tracking in a grocery store was selected as the second case study. We presented a detailed description of how the CLONE_*inference* solution could be employed to solve the multi-target multi-camera tracking problem.

Possible Improvements: There are some possible improvements for CLONE. We list three of them for the discussion. (i) *Bandwidth demand*: as the increasing number of edge nodes or the participation of larger neural networks, the communication of CLONE may be limited by bandwidth. In this context, we could leverage the Parameter EdgeServer group. In the group, Parameter EdgeServers communicate with each other. Each Parameter EdgeServer is only responsible for a portion of parameters, and they work together to maintain globally shared parameters and their updates. (ii) *Aggregation protocol:* it is essential to find a suitable aggregation rule for the Parameter EdgeServer to aggregate parameters, which requires excessive experiments based on the specific experimental conditions. (iii) *Push/pull latency:* as to CLONE_*training*, pushing parameters to the Parameter EdgeServer is usually much slower than pulling parameters. As to CLONE_*inference*, the case is different. It is essential to investigate methods to reduce push/pull latency (possible solutions include structured updates and sketched updates [55]).

Potential Use Cases: In this work, we choose two case studies, *i.e.,* failure prediction of EV battery and related components as well as the customer tracking in a grocery store to show how CLONE solution could be employed in the training stage and the inference stage. There are a variety of other meaningful use cases that CLONE could help, particularly for two types of scenarios: (i) real-time applications that require developing suitable machine learning algorithms on the edges, and (ii) due to the privacy or/and the large network bandwidth constraints, the training dataset cannot be moved away from its source.

References

[1] C. Savaglio, P. Gerace, G. Di Fatta, and G. Fortino, "Data mining at the IoT edge," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–6.

[2] B. L. Mearian. (2013) Self-driving cars could create 1GB of data a second. [Online]. Available: https://www.computerworld.com/article/2484219/emerging-technology/self-driving-cars-could-create-1gb-of-data-a-second.html

[3] T. Rossi, "Autonomous and ADAS test cars produce over 11 TB of data per day," October 10, 2018. [Online]. Available: https://www.tuxera.com/blog/autonomous-and-adas-test-cars-produce-over-11-tb-of-data-per-day/

[4] S. Lu, X. Yuan, and W. Shi, "EdgeCompression: An integrated framework for compressive imaging processing on CAVs," in *Proceedings of the 5th ACM/IEEE Symposium on Edge Computing (SEC)*, 2020.

[5] SecurityInfoWatch, "Data generated by new surveillance cameras to increase exponentially in the coming years," January 20, 2016. [Online]. Available: https://www.securityinfowatch.com/video-surveillance/news/12160483

[6] G. Forecast, "Cisco visual networking index: Global mobile data traffic forecast update 2017–2022," *Update*, vol. 2017, p. 2022, 2019.

[7] G. Ulm, E. Gustavsson, and M. Jirstrand, "OODIDA: On-board/off-board distributed data analytics for connected vehicles," *arXiv preprint arXiv:1902.00319*, 2019.

[8] J. Bort. (2016) The 'Google Brain' is a real thing but very few people have seen it. [Online]. Available: http://www.businessinsider.com/what-isgoogle-brain-2016-9.

[9] N. Jouppi. (2016) Google supercharges machine learning tasks with TPU custom chip. [Online]. Available: https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip

[10] B. Lovejoy. (2015) Apple moves to third-generation Siri backend, built on open-source Mesos platform. [Online]. Available: https://9to5mac.com/2015/04/27/siri-backend-mesos/

[11] S. Claudio and F. Giancarlo, "A simulation-driven methodology for IoT data mining based on edge computing," *ACM Transactions on Internet Technology (TOIT)*, 2020.

[12] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *Acm Sigplan Notices*, vol. 52, no. 4, pp. 615–629, 2017.

[13] P. M. Grulich and F. Nawab, "Collaborative edge and cloud neural networks for real-time video processing," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2046–2049, 2018.

[14] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, and G. Fortino, "Autonomic computation offloading in mobile edge for IoT applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[15] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware video analytics on edge computing platform," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[16] K. Lee, J. Flinn, and B. D. Noble, "Gremlin: scheduling interactions in vehicular computing," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[17] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "VideoEdge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[18] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[19] S. Y. Jang, Y. Lee, B. Shin, and D. Lee, "Application-aware IoT camera virtualization for video analytics edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 132–144.

[20] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, K. Karanasos, J. Padhye, and G. Varghese, "Wanalytics: Geo-distributed analytics for a data intensive world," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. ACM, 2015, pp. 1087–1092.

[21] P. Zhang, M. Zhou, and G. Fortino, "Security and trust issues in fog computing: A survey," *Future Generation Computer Systems*, vol. 88, pp. 16–27, 2018.

[22] D. Park, S. Kim, Y. An, and J.-Y. Jung, "LiReD: A light-weight real-time fault detection system for edge computing using LSTM recurrent neural networks," *Sensors*, vol. 18, no. 7, p. 2110, 2018.

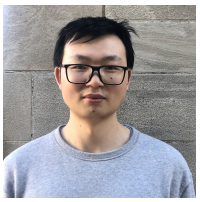
[23] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi, "Making disk failure predictions SMARTer!" in *18th USENIX Conference on File and Storage Technologies (FAST)*, 2020, pp. 151–167.

[24] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision

tree," in *Advances in neural information processing systems*, 2017, pp. 3146–3154.
[25] O. Consortium *et al.*, "OpenFog reference architecture for fog computing," *Architecture Working Group*, 2017.
[26] J. Li, J. Jin, D. Yuan, and H. Zhang, "Virtual fog: A virtualization enabled fog computing framework for internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 121–131, 2018.
[27] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are FPGAs suitable for edge computing?" in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.
[28] F. Spiga and I. Girotto, "phiGEMM: a CPU-GPU library for porting quantum espresso on hybrid systems," in *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2012, pp. 368–375.
[29] I. V. Morozov, A. Kazennov, R. Bystryi, G. E. Norman, V. Pisarev, and V. V. Stegailov, "Molecular dynamics simulations of the relaxation processes in the condensed matter on GPUs," *Computer Physics Communications*, vol. 182, no. 9, pp. 1974–1978, 2011.
[30] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "GPU scheduling on the NVIDIA TX2: Hidden details revealed," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 104–115.
[31] L. Bao, L. Fan, and Z. Miao, "Real-time simulation of electric vehicle battery charging systems," in *2018 North American Power Symposium (NAPS)*. IEEE, 2018, pp. 1–6.
[32] Y. Xing, E. W. Ma, K. L. Tsui, and M. Pecht, "Battery management systems in electric and hybrid vehicles," *Energies*, vol. 4, no. 11, pp. 1840–1857, 2011.
[33] Y. Ma, K. Zhang, J. Gu, J. Li, and D. Lu, "Design of the control system for a four-wheel driven micro electric vehicle," in *2009 IEEE Vehicle Power and Propulsion Conference*. IEEE, 2009, pp. 1813–1816.
[34] H.-P. Li and Y.-w. Li, "The research of electric vehicle's MCU system based on iso26262," in *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. IEEE, 2017, pp. 336–340.
[35] K. W. E. Cheng, B. Divakar, H. Wu, K. Ding, and H. F. Ho, "Battery-management system (BMS) and SOC development for electrical vehicles," *IEEE transactions on vehicular technology*, vol. 60, no. 1, pp. 76–88, 2011.
[36] L. Ran, W. Junfeng, W. Haiying, and L. Gechen, "Design method of CAN BUS network communication structure for electric vehicle," in *International Forum on Strategic Technology 2010*. IEEE, 2010, pp. 326–329.
[37] B. Li, W. Wang, L. Jia, D. Wang, and A. Kong, "Study on HIL system of electric vehicle controller based on NI," in *IOP Conference Series: Materials Science and Engineering*, vol. 382, no. 5. IOP Publishing, 2018, p. 052033.
[38] K. Sarrafan, K. M. Muttaqi, and D. Sutanto, "Real-time state-of-charge tracking system using mixed estimation algorithm for electric vehicle battery system," in *2018 IEEE Industry Applications Society Annual Meeting (IAS)*. IEEE, 2018, pp. 1–8.
[39] X.-W. Yan, Y.-W. Guo, Y. Cui, Y.-W. Wang, and H.-R. Deng, "Electric vehicle battery soc estimation based on gnl model adaptive kalman filter," in *Journal of Physics: Conference Series*, vol. 1087, no. 5. IOP Publishing, 2018, p. 052027.
[40] A. Fotouhi, D. J. Auger, K. Propp, S. Longo, and M. Wild, "A review on electric vehicle battery modelling: From Lithium-ion toward Lithium–Sulphur," *Renewable and Sustainable Energy Reviews*, vol. 56, pp. 1008–1021, 2016.
[41] E. Peled, D. Golodnitsky, H. Mazor, M. Goor, and S. Avshalomov, "Parameter analysis of a practical lithium-and sodium-air electric vehicle battery," *Journal of Power Sources*, vol. 196, no. 16, pp. 6835–6840, 2011.
[42] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomForest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.
[43] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng, "Stochastic gradient boosted distributed decision trees," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 2061–2064.
[44] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.
[45] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[46] F. D. dos Santos Lima, G. M. R. Amaral, L. G. de Moura Leite, J. P. P. Gomes, and J. de Castro Machado, "Predicting failures in hard drives with LSTM networks," in *Proceedings of the 2017 Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, 2017, pp. 222–227.
[47] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 16, 2016, pp. 265–283.
[48] A. Gulli and S. Pal, *Deep Learning with Keras*. Packt Publishing Ltd, 2017.
[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
[50] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 14, no. 2, 1995, pp. 1137–1145.
[51] J. D. Rodriguez, A. Perez, and J. A. Lozano, "Sensitivity analysis of k-fold cross validation in prediction error estimation," *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, vol. 32, no. 3, pp. 569–575, 2010.
[52] J.-H. Kim, "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap," *Computational statistics & data analysis*, vol. 53, no. 11, pp. 3735–3745, 2009.
[53] L. He, E. Kim, K. G. Shin, G. Meng, and T. He, "Battery state-of-health estimation for mobile devices," in *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2017, pp. 51–60.
[54] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.
[55] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.
[56] W. S. Yuwono, D. W. Sudiharto, and C. W. Wijiutomo, "Design and implementation of human detection feature on surveillance embedded ip camera," in *2018 International Conference on Sustainable Information Engineering and Technology (SIET)*. IEEE, 2019, pp. 42–47.
[57] W. Zhu, C. B. Owen, H. Li, and J.-H. Lee, "Personalized in-store e-commerce with the PromoPad: an augmented reality shopping assistant," *Electronic Journal for E-commerce Tools and Applications*, vol. 1, no. 3, pp. 1–19, 2004.
[58] J. Eden, T. Kawchak, and V. Narayanan, "Indoor navigation using text extraction," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2018, pp. 112–117.
[59] A. Cheng. (2019) Why amazon go may soon change the way we shop. [Online]. Available: https://www.forbes.com/sites/andriacheng/2019/01/13/why-amazon-go-may-soon-change-the-way-we-want-to-shop/#435349e56709
[60] M. Tillman. (2019) What is Amazon Go, where is it, and how does it work? [Online]. Available: https://www.pocket-lint.com/phones/news/amazon/139650-what-is-amazon-go-where-is-it-and-how-does-it-work
[61] Q. Burrows, "Scowl because you're on candid camera: Privacy an dvideo surveillance," *Val. UL Rev.*, vol. 31, p. 1079, 1996.
[62] C. Slobogin, "Public privacy: camera surveillance of public places and the right to anonymity," *Miss. lJ*, vol. 72, p. 213, 2002.
[63] F. of Humanity Institute, "Artificial intelligence: American attitudes and trends," January 9th, 2019. [Online]. Available: https://www.fhi.ox.ac.uk/aipublic2019/
[64] D. Musicki, B. F. La Scala, and R. J. Evans, "Integrated track splitting filter-efficient multi-scan single target tracking in clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, no. 4, pp. 1409–1425, 2007.
[65] A. Ess, B. Leibe, K. Schindler, and L. Van Gool, "A mobile vision system for robust multi-person tracking," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.
[66] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 5–18, 2004.
[67] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 3645–3649.
[68] N. Wojke and A. Bewley, "Deep cosine metric learning for person re-identification," in *2018 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2018, pp. 748–756.
[69] W. Liu, O. Camps, and M. Sznaier, "Multi-camera multi-object tracking," *arXiv preprint arXiv:1709.07065*, 2017.

[70] E. Ristani and C. Tomasi, "Features for multi-target multi-camera tracking and re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6036–6046.
[71] ONVIF. (2018) Meet our members: Henrik sydbo hansen of milestone. [Online]. Available: https://www.onvif.org/blog/blog/2018/03/20/meet-members-henrik-sydbo-hansen-milestone/
[72] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
[73] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 3464–3468.
[74] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.
[75] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.
[76] X. Hou, Y. Wang, and L.-P. Chau, "Vehicle tracking using Deep SORT with low confidence track filtering," in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2019, pp. 1–6.
[77] M. Courbariaux, Y. Bengio, and J.-P. B. David, "Training deep neural networks with binary weights during propagations. arxiv preprint," *arXiv preprint arXiv:1511.00363*, 2015.
[78] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.
[79] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277.
[80] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.
[81] B. B. Sau and V. N. Balasubramanian, "Deep model compression: Distilling knowledge from noisy teachers," *arXiv preprint arXiv:1610.09650*, 2016.
[82] P. Luo, Z. Zhu, Z. Liu, X. Wang, X. Tang *et al.*, "Face model compression by distilling knowledge from neurons." in *AAAI*, 2016, pp. 3560–3566.
[83] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[84] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint*, pp. 1610–02 357, 2017.
[85] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.
[86] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[87] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.
[88] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: a timely dataflow system," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 439–455.
[89] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.
[90] K. Zhang, S. Alqahtani, and M. Demirbas, "A comparison of distributed machine learning platforms," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.
[91] C. Boden, T. Rabl, and V. Markl, "Distributed machine learning-but at what COST," in *Machine Learning Systems Workshop at the 2017 Conference on Neural Information Processing Systems*, 2017.
[92] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.
[93] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative MapReduce," in *Proceedings of the 19th ACM international symposium on high performance distributed computing*. ACM, 2010, pp. 810–818.
[94] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.
[95] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, "Spinning fast iterative data flows," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1268–1279, 2012.
[96] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "GraphLab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.
[97] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, and M. Viroli, "Modelling and simulation of opportunistic IoT services with aggregate computing," *Future Generation Computer Systems*, vol. 91, pp. 252–262, 2019.
[98] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.
[99] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.
[100] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," *arXiv preprint arXiv:1812.00535*, 2018.
[101] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.
[102] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *learning*, vol. 8, p. 9, 2018.
[103] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1586–1597, 2017.
[104] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

Sidi Lu received the B.E. degree from Xidian University, Xi'an, China, in 2016. She is currently working toward a Ph.D. degree in Computer Science with Wayne State University, Detroit, USA. Her academic advisor at Wayne State University is Prof. Weisong Shi. Her research interests include reliability analysis, collaborative machine learning, and video processing.



Yongtao Yao is a Ph.D. student under the supervision of Prof. Weisong Shi at Wayne State University, Detroit, USA. He received his B.E. degree from Xidian University in 2016. Shi. His research direction is Edge Computing, and his specific research interests include model scheduling and video processing.

Weisong Shi is a Charles H. Gershenson Distinguished Faculty Fellow and a Professor of Computer Science with Wayne State University, USA, where he leads the Wayne Mobility Initiative (WMI) and directs the Connected and Autonomous dRiving Laboratory (CAR), investigating performance, reliability, power- and energy-efficiency, trust and privacy issues of networked computer systems, and applications. He is one of the leaders in the edge computing research community and has been very actively working on edge computing and its applications in public safety, CAVs, and connected health. His paper entitled "Edge Computing: Vision and Challenges" has been cited more than 2500 times. In 2016, he co-chaired the NSF Workshop on Grand Challenges in Edge Computing. In 2018, Dr. Shi led the development of IEEE Course on Edge Computing. In 2019, Dr. Shi served as the lead guest editor for the edge computing special issue on the prestigious Proceedings of the IEEE journal. He is the Founding Steering Committee Chair of the ACM/IEEE Symposium on Edge Computing (SEC). He is an IEEE Fellow and an ACM Distinguished Scientist.