PACAS: A Privacy-Aware Smart Camera System

Keyang Yu
Department of Computer Science
Colorado School of Mines

Dong Chen
Department of Computer Science
Colorado School of Mines

***Abstract*—Smart cameras have been increasingly deployed in smart homes for remote monitoring and enhancing home security. However, extensive recent research has uncovered potential user privacy threats associated with popular commercial camera systems. Some manufacture design of these commercial camera systems usually requires smart camera users to relinquish their control of camera recorded data. For instance, these cameras often upload camera recordings to their cloud servers to enable advanced data analysis for camera app services. To facilitate enhanced camera services, the data may be further shared with on-path vendors, third parties of manufacturers, and cloud providers, potentially allowing them to access video footage or image captures without users' awareness or meaningful consent.**

To address this problem, we design a new smart camera system—PACAS that enables people to regain the control of their data while still retaining access to regular camera services. We evaluate PACAS using multiple camera video footage traces and on multiple real-world camera prototypes. We show that PACAS can achieve the performance of 0.5 second living streaming latency at a frame rate of 30 fps and a resolution of 240x320 on average. PACAS offers compelling evidence that smart camera systems can deliver on-device modern capabilities comparable to those provided by current commercial systems, all while upholding user privacy.

***Index Terms*—Internet of Things, User Privacy, Tiny ML**

I. INTRODUCTION

The global number of smart cameras was forecast to continuously increase between 2023 and 2027 by in total 81.84 million household [6]. In particular, North America smart home security cameras market size was valued at USD 2.99 billion in 2022 and is expected to grow at a compound annual growth rate of 19.8% from 2023 to 2030 [16]. Commercial cameras that are quite successful, since they often offer several essential monitoring benefits, e.g., live feeding, video recording and motion detection.

Unfortunately, extensive recent research [4] has shown that commercially available smart cameras often follow a threat model that mandates undue trust by design. Manufactures or their service providers are granted unlimited access by default to their camera recorded contents of any user who has deployed their smart camera systems [4]. In addition, Amazon's Ring unit has repeatedly handed over its users' doorbell camera footage to law enforcement without their consent, according to new findings from an ongoing investigation [1] led by Massachusetts Senator Ed Markey, D-Mass. Google's current smart home division—Nest Labs has been told to hand over data on 300 separate occasions since 2015 based on a little-documented transparency report from Nest [10]. These commercial smart camera systems could become massive surveillance systems. Smart camera system users are loosing control over their own camera recorded video and image data, and this is compromising their user privacy.

Most recent research [3], [4], [7], [11], [20], [22], [23], [25] focus on designing customized or dedicated hardware or software to enable image or video frame encryption to protect smart camera systems. Unfortunately, these prior approaches either require additional hardware support or do not adequately address the right of sole ownership (replying on other parities), which is crucial for practical deployment. Furthermore, they often lack practical system performance benchmarking and evaluation mechanisms to assess the effectiveness of protection methods across various hardware level camera devices.

To address these issues, we design a fully-local, privacy preserving, embedded smart camera system—PACAS that enables users to significantly prevent privacy leakage through their camera recorded data. Our insight is that the best privacy preserving approach is to keep the data as close as possible to its source, which in our case, on the camera itself. To avoid the uncontrolled data sharing with other parties, we harness AI at the edge and Tiny ML approaches to create a fully-local, user privacy-preserving embedded camera system. This system could deliver the same functionality as commonly found in commercially available smart cameras while safeguarding privacy by retaining data locally. PACAS does not assume any additional hardware support and provides a full stack evaluation towards real world deployments. In essence, we first profile the capabilities of different embedded devices through Raspberry Pi simulation, by limiting the CPU frequency and available memory. We then conduct the camera service (re)training and prediction approaches. Subsequently, we design tiny intelligent camera services that provide comparable features found in commercially available cameras. By leveraging Convolutional Neural Networks (CNNs) model, PACAS can utilize the hardware resource of multiple levels of embedded device, to provide a fully-local security camera service. By doing so, we make the following contributions.

Challenges. We first explore and emphasize the key challenges in designing our fully-local, privacy preserving, embedded smart camera system—PACAS.

PACAS Design. We present the design of PACAS, which enable users to regain the control of their smart camera systems and significantly reduce their privacy leakage through their camera recorded data. To achieve a fully-local security camera service, PACAS first profiles embedded devices in different levels of performance by limiting the CPU frequency and

2024 IEEE Cloud Summit | 979-8-3503-7006-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CLOUD-SUMMIT61220.2024.00035

979-8-3503-7006-5/24/$31.00 ©2024 IEEE
DOI 10.1109/Cloud-Summit61220.2024.00035

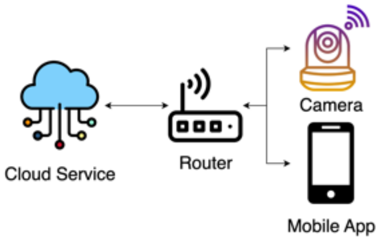


Fig. 1. The data pipeline of a typical smart camera system.

available memory though Raspberry Pi simulation. PACAS then harnesses this profiling information to construct the camera service (re)training and prediction system services. Subsequently, we design tiny intelligent camera services that offer the same features found in commercially available cameras. Lastly, PACAS provides a comprehensive full-stack evaluation.

Implementation and Evaluation. We implement PACAS using the most widely used programming language—*Python* for tiny machine learning models, micro controllers, and firmware. To assess and benchmark PACAS, we conduct evaluations with two real cameras, including the Arducam 5MP Camera and the OpenMV Cam H7 Smart Camera. Additionally, we use multiple camera prototypes created using Raspberry Pi Zero 2 W, Pi 2 Model B, Pi 3 Model B+, and Pi 4 Model B to simulate various hardware levels of camera systems. We find that PACAS excels in effectively and collaboratively managing local devices within the monitored environment. It successfully prevents the leakage of sensitive user information with nearly negligible system overhead increase.

Releasing Data and Source Code. Our new approaches to prevent user privacy leakage from smart camera systems is quite general, and could be applied to address similar user privacy issues in other domains. We will release the source code and datasets of PACAS to broad user privacy and IoT research communities on our lab GitHub website.

II. Background and Related Work

A. Privacy Threat Model

Figure 1 shows the typical data streaming pipeline of a commercial camera system. The camera monitors and records local video and image data once the camera monitoring mode is enabled (a.k.a. armed) by users. The data is transmitted from the camera to its remote cloud servers where the manufactures host their machine learning-based and computer vision-based applications (e.g., motion detection, package delivery notification). Once the cloud servers process the uploaded data, service notifications and other results will be sent back to users via smart phone camera apps. Our research problem here is to design a new secure camera system that could not only provide the regular features and functionalities but also preserve user privacy. Our new camera system will help people regain the control over their private information which could be inferred from their camera data.

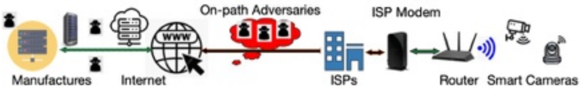


Fig. 2. Overview of our privacy threat model.

As shown in Figure 2, we are broadly concerned with the ability of external adversaries (e.g., Internet service provider, on-path network observers, manufactures, cloud storage providers, and their third parties) to infer user private information from camera recorded data. The camera recorded data, including video frames and image frames. And these potential adversaries may be incentives to infer user private information from the camera reported environment where users do not want to share this privacy-sensitive information with them. We only trust the devices owned by the users to run our secure camera system.

We assume external adversaries can use any sophisticated user information inference techniques, such as machine learning, deep learning, or other statistical methods to infer certain types of the observed user private information for the recorded video and image data. Thus, inferring or discovering user activities through their camera recorded data is considered as an opposition to users' privacy preferences.

In particular, we are concerned with four types of user privacy inference attacks: i) *Learning user occupancy from the data.* This includes whether a home or space is occupied and when; ii) *Counting occupants from the data.* This includes how many occupants are staying in the monitored space and also how many visitors in a day, week or month; iii) *Learning occupant daily routine from the data.* These user activities are inferred from image or video frames and may include when users come and go, when they perform their daily activities, such as going to work, going out for groceries, walking with their pets, playing in backyard; iv) *Re-identifying occupants from the data.* Given the significant improvements on computer vision models, it is often trivial for attackers to learn and apply face recognition models to identify an occupant's face from large amount of uploaded image or video frames. The attackers could potentially link this information to users' public social network accounts, leading to even more severe privacy breaches. To exacerbate the situation further, much of the recorded camera data already contains geotags. Considering the wealth of personal information involved, this privacy breach can lead to significant identity theft risks. In addition to the above-mentioned short-term user activities, adversaries may also infer more comprehensive and longer-term user activities, such as whether a household has a baby, and whether they go on vacation on weekends.

B. Related Work

While there is relatively less focus on *privacy-preserving* camera systems, there is a substantial body of work dedicated to enhancing the *security* of smart camera systems. We next examine the gaps in prior smart camera security work [3], [7], [11], [20], [22], [23], [25] and privacy enhancing camera system [4]. The security enhancement approaches often leverage encryption, sign, and customized hardware to encrypt or blurring security sensitive areas of image or video frames.

TABLE I
COMPARISON OF RECENT SMART CAMERA SYSTEMS

	Additional Hardware	Right of Ownership	Single Host	Practical Evaluation
PrivacyCam [7]	✓	×	✓	✓
TrustCAM [23]	✓	×	✓	×
SoC [11]	✓	×	✓	✓
Signcryption [20]	×	×	✓	×
Pinto [25]	×	×	✓	✓
CaCTUs [4]	✓	✓	✓	×
TrustEYE.M4 [22]	✓	✓	✓	✓
SecureCam [3]	✓	×	✓	×

We classify them into different categories, including hardware, software, and hybrid approaches.

Hardware-based Approaches. Ihtesham and Bernhard in [11] presented SoC-based camera platform that integrates data and node security into the hardware. It requires a trusted authority for device fingerprinting in key generation and trusts the manufacturer's physically unclonable functions for encryption keys. Changing ownership might necessitate buying a new device. Thomas and Bernhard in TrustCAM [23], features a hardware security module with an Atmel TPM chip for enhanced security, though it does not encrypt video frames and assumes users have the necessary hardware.

Software-based Approaches. Hyunwoo and Jaemin proposed Pinto [25], which is a software solution for producing privacy-protected, forgery-proof videos on low-end IoT cameras. It allows privacy filtering in post-processing while maintaining real-time video authenticity. However, it might leave other identifiable information unobscured. EYE.MP4 [22] by Winkler and Rinner, and signencryption [20] by Ullah et al. provide secrecy and integrity for video frames, but issues like video deletion and delegation are not fully explored.

Hybrid Approaches. Ankur and T.E.'s PrivacyCam [7] implemented PICO to secure real-time video feeds on Blackfin DSP, enabling surveillance while maintaining privacy. Yet, it doesn't fully safeguard privacy in smart cameras and leaves some vulnerabilities. CaCTUs [4] offers commercial-grade smart camera features, utilizing direct pairing and cryptographic algorithms for data management, but relies on smartphones for encryption settings and doesn't fully consider commercial hardware limitations. Ifeoluwapo et al.'s approach [3] focuses on selective encryption for surveillance videos, but also depends on a host machine for encryption tasks.

Observation. As shown in Table I, hardware-based or hybrid solutions, such as PrivacyCam [7], TrustCAM [23], SoC [11], Signcryption [20], CaCTUs [4], TrustEYE.M4 [22], and SecureCam [3], often could support stronger encryption or blurring algorithms. Hardware-based approaches (i.e., PrivacyCam [7], SoC [11]) typically were validated in real applications due to their necessary hardware design and overhead benchmarking. Hybrid approaches (e.g., TrustEYE.M4 [22], CaCTUs [4]) can support right of the ownership on smart cameras, which assumes only the owner is trusted. Software solutions (e.g., Signcryption [20], Pinto [25]) typically does not support right of ownership. Almost all the related work assume their new solutions are deployed or hosted by a single hardware, either a camera itself or a hosting machine.

Prior research has advanced the security of smart cameras but often falls short in privacy preservation for real-world use due to several limitations: reliance on single-host setups that may struggle with resource constraints, inadequate support for ownership rights, dependence on centralized systems or authorities for protection, additional hardware needs for software-based methods, misplaced trust in manufacturers, and outdated privacy threat models. Moreover, there is a lack of performance evaluations for these methods across varying hardware, an oversight our PACAS project aims to address.

III. CHALLENGES

In this section, we outlined the key challenges we met when designing, implementing, and evaluating our PACAS.

Limited Computing Resources. The smart cameras and other devices in the monitored environment may present limited computing resources, including CPU, RAM, ROM, network bandwidth and I/O speed, etc. These resource limited cameras and other IoT devices may not be able to (re)train and predict camera services using standard machine learning or deep learning models. In particular, these devices themselves may not have sufficient memory and storage space to store the camera recorded video and image data for modeling retraining which is the foundations for many camera features. To address this issue, we leverages tiny machine learning models to build adaptive distributed machine learning enabled smart camera services. In addition, we also develop a new camera data relocation approach to mitigate the limited storage issue.

No Advanced Development Library Support. Unlike traditional system development support, such as ready-to-use machine learning or deep learning frameworks or libraries, smart camera systems often do not have any off-shelf toolkit to run their firmware immediately. To address this issue, we develop or tailor a set of tiny machine learning and deep learning models from scratch. We use these lightweight models to develop intelligent camera services that offer the same features found in commercially available cameras.

No Special Hardware Support. Many prior work assumed specially designed or tailored camera hardware to support their functionaries to protect user sensitive data leakage. Unfortunately, this is not the case in real practice. In a real monitoring environment, people often may not be willing to purchase or install additional dedicated device to help mask their privacy in camera data streams. To address this issue, we design a new adaptive distributed system to provide user privacy preserving services. Our system will automatically profile the local IoT environment and determine procedures for distributed model (re)training and data relocation using all the available devices.

Non Practical Trust Models. As we discussed in Section II-B, many recent works assumed the trust relationship among different parities who might be potential on-path attackers for smart camera systems. Several recent approaches still assumed they can trust on camera manufacturers to mask their privacy. Regrettably, this trust model seems unviable, especially in

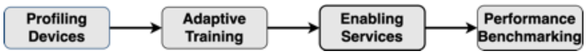


Fig. 3. The operational pipeline of PACAS.

light of recent concerns regarding data resharing without user consent. To address this issue, we only assume the trust on users' local devices to build our new system—PACAS. Our insight for PACAS is that the most effective privacy-preserving approach should operate where the data is generated.

IV. Design

In this section, we will explain how we design our security camera system—PACAS.

A. System Design

We design a new fully-local, privacy-aware, embedded security camera system—PACAS that enables users to significantly prevent privacy leakage through their camera recorded data. Our insight is that instead of uploading data to the cloud, we can harness AI@Edge and Tiny ML techniques to create a local camera system that offers the same functionalities as those commonly found in commercially available cameras. This approach ensures privacy by keeping data locally secured.

System Operational Pipeline. Figure 3 shows the system pipeline of PACAS. In essence, PACAS first profile the embedded device capability by simulating with different CPU frequency and available memory on a Raspberry Pi, to determine the amount of training data allocated to the camera with acceptable training latency and model accuracy. PACAS then constructs new camera service (re)training and prediction system services. Subsequently, we design tiny intelligent camera services that offer the same features found in commercially available cameras. Lastly, PACAS incorporates a comprehensive full-stack evaluation.

B. Profiling Heterogeneous Hardware Devices

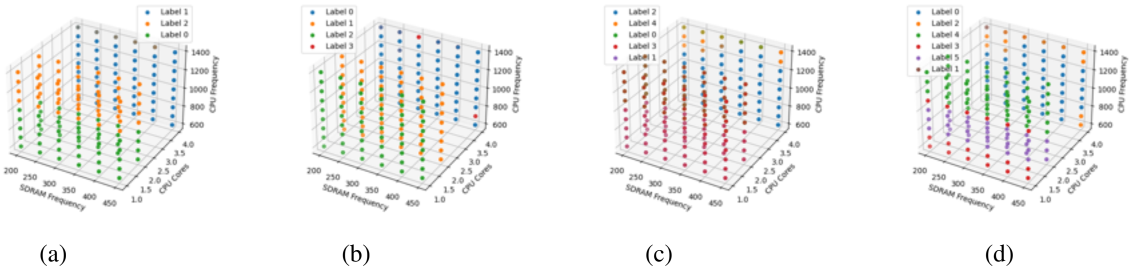
PACAS relies on a heterogeneous hardware device profiling information for workload dispatching and data relocation, which are crucial to enable smart camera services. To build this model, we first leverage multiple hardware devices, including Raspberry Pi Zero 2 W, Pi 2 Model B, Pi 3 Model B+, and Pi 4 Model B, to build camera prototypes to simulate various levels (∼1,597) of camera systems. We then empirically review and identified six principal features, including SDRAM frequency, the number of CPU cores, CPU frequency, and memory allocation and I/O speed, which could comprehensively describe each hardware's computing capabilities. By doing so, we create a set of different hardware profiles, which are representing different level cameras and other in-field devices. Figure 5 (a) illustrated the relationship between training time and CPU frequency of PACAS. We observe that the model training time exhibits a non-linear increase as the CPU frequency is reduced. We find that all of the 5 models on a standard 1,800 MHz CPU camera require less than ∼100 seconds for training. This suggests acceptability in the real-life scenario of training a single camera.

Data Cleaning and Preprocessing. The principal features identified in prior step may be recorded using different measurement units. We could not simply compare and compute the correlations among them to build our model directly. The goal of this data preprocessing is to change raw feature vectors into a representation that is more suitable for the downstream estimators. Standardization of datasets is a common requirement for camera service related machine learning estimators. Given a learning objective, people assumes all features are centered around zero or have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected. To address this issue, we leverage standard scaling, min-max scaling, and Box-Cox transformation to preprocess our training dataset to train our models.

Unsupervised Device Capability Learning. After collecting and preprocessing data with the features from IoT devices, we leverage KMeans [12], DBSCAN [19], OPTICS [2], among others to categorize the hardware configurations. To goal of this unsupervised clustering is to identify how many categories we should have to efficiently characterize the hardware devices, in particular for the cameras. The K-Means algorithm [12] clusters data by trying to separate camera data samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. Affinity Propagation [9] creates clusters by sending messages between pairs of samples until convergence. A dataset is then described using a small number of exemplars, which are identified as those most representative of other samples. We also study the comparison results when we applying different clustering methods over our hardware specification dataset. Initially, as shown in Figure 4 (a)∼(d), we employ a Gaussian mixture model to determine the appropriate cluster size for our camera prototype dataset. Our benchmarking results using commercially available camera specs indicate a cluster size of three is achieving the best performance. Then, we run Gaussian mixture model again across four different hardware enabled camera prototypes. The goal is to ensure our identified cluster size performs well on different camera prototypes. The last step in this unsupervised learning process is to identify the best performing clustering approach. We find that K-Means is the best performing unsupervised approach for our prototype dataset.

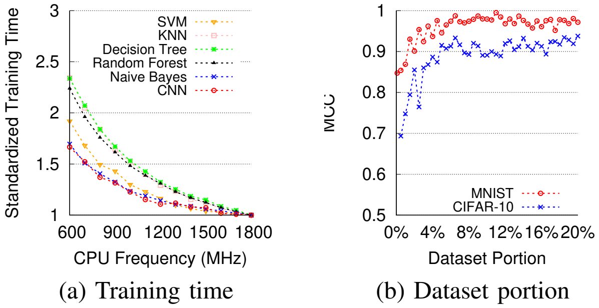
C. Adaptive Camera Service Training

To further reduce the time consumption on training a image identification model, we consider a smart dispatcher which dispatches partial training dataset to the embedded device, according to their performance level, to achieve an adaptive training process. Figure 5 (a) illustrated the relationship between training time and CPU frequency of PACAS in standalone mode. Not surprisingly, we observe that while maintaining MCC within the range of 0.9-1.0, the model training time exhibits a non-linear increase as the CPU frequency is reduced. We find that all of the 5 models on a standard 1,800 MHz



(a) (b) (c) (d)

Fig. 4. The comparison results when using different machine learning models.



(a) Training time (b) Dataset portion

Fig. 5. The adaptive camera model training.

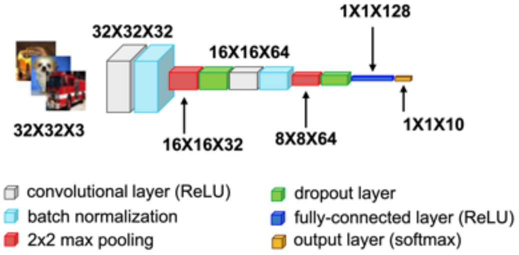


Fig. 6. The overview of our CNNs architecture design.

CPU camera require less than ∼100 seconds for training. This suggests acceptability in the real-life scenario of training a single camera.

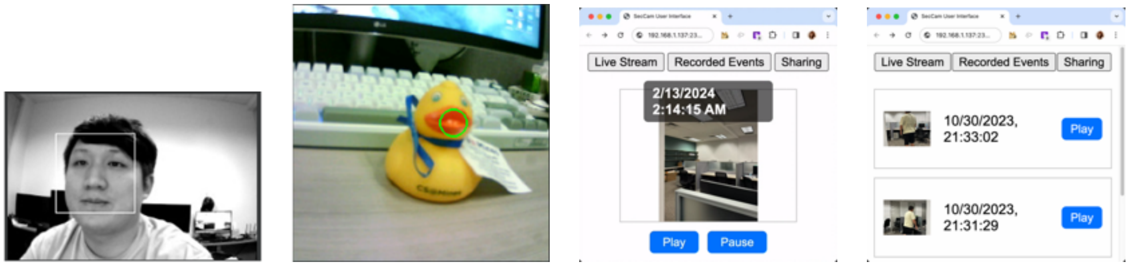
Minimum Amount of Data for Training and Predicting. The question towards real world deployment for a smart camera system is minimum amount of data that is required to train a reasonably accurate model (i.e., when MCC $\geq$ 0.9). To fundamentally understand this problem, we benchmark the distributed learning process by varying the size of data relocated to each participating device. As shown in Figure 5 (b), our results show that after each device receive at least 13% of the dataset, the model accuracy become stable on both big datasets—MNIST [15] and CIFAR-10 [8]. That being said, 13% of the dataset is ideally the minimum amount of data to ensure our training or other service prediction. The time consumption for training a machine learning or deep learning model is positively correlated to the size of the given dataset. This learned cutoff will also help us find out the optimal trade-off point between training speed and model accuracy for most widely used smart camera applications. We will apply this constrain when we schedule workload and data relocation.

D. Enabling Tiny Smart Camera Services

Next, we develop a set of smart camera services including objection detection and motion detection, which are the key foundations to enable a wide set of smart camera features. Unfortunately, as we discussed in Section III, the standard machine learning or deep learning library or framework can not be used directly to address our problem. Instead, we tailor the "large" models into tiny models so that they can run on tiny devices, e.g., cameras, IoT hubs, etc. Next, we tailored a new CNN-based deep learning classifier that can accurately detect objects and motions in camera recorded videos. Below, we describe the design of our CNNs architecture. As shown in Figure 6, our CNNs architecture is comprised of input, convolutional layers (ReLU), batch normalization layer, max pooling, dropout layer, fully-connected layers(ReLU) and output layer (softmax). The inputs are 32x32x32 video frames, and the first two layers are convolutional layers which have 32x32x32 neurons with a rectified linear unit (ReLU). Then, we have another two convolutional layers that have 16x16x65 neurons with ReLUs. After these layers, we employ another convolutional layer with ReLUs, and all these layers have 8x8x64 neurons. Finally, we leverage 2 convolutional layers with ReLUs, and these layers all have 1x1x128 neurons. Among the different groups of convolutional layers, we have 2x2 max pooling which is used to down sample input video frames and reduce their dimensionality. Note that, the way we build this model is quite general, so our model potentially can also be adapted into similar application domains, such as embedded navigation and medical IoT devices.

E. Full Stack Benchmarking and Optimization

Different than prior work, PACAS also provides a full stack benchmarking on system performance using Matthews Correlation Coefficient (MCC) [14] and Adversary Confidence (AC) [26], and privacy preserving guarantee evaluation. PACAS could examine smart camera systems in terms of their CPU utilization, Memory, ROM, Network Bandwidth and I/O once they are in armed mode. PACAS also could preload the common hardware configurations that we found in the



(a) Face Detection (b) Object Detection (c) Live Stream (d) Event Recording

Fig. 7. PACAS's live demonstration of features and functionalities.

commercially available cameras and IoT devices. That being said, PACAS can provide practical evaluations towards real world deployments. In addition, we developed optimization approaches to further enhance PACAS's system performance.

Runtime Data Relocation Adjustment. For instance, PACAS supports runtime improvements on the data relocation method that we use in our second step—adaptive distributed modeling process. PACAS monitors the local model performance on each device and alerts the coordinator to optimize data distribution across different devices.

Dataset Shuffling. Unlike image identification models trained on image datasets, object detection models, which usually implemented on detecting moving human, pets, or vehicles, will be trained based on video footage, i.e, image sets with high time-correlation. Regular dataset shuffling may break such time-correlationship, and may bring significant performance loss especially for some machine learning models with high time-dependency, like Long Short-Term Memory (LSTM).

V. Implementation

We implement PACAS on real cameras—Arducam 5MP Camera and OpenMV Cam H7 Smart Camera. Additionally, we use multiple camera prototypes created using Raspberry Pi Zero 2 W, Pi 2 Model B, Pi 3 Model B+, and Pi 4 Model B, using `Python` and `C` to simulate different level hardware specifications found in commercially available camera systems. We implement the system components using widely available open-source frameworks, including OpenCV [5], Scikit-learn [17], TensorFlow Lite [21], and YOLO [24].

To implement heterogeneous hardware characterizing model, we develop a TensorFlow Lite-based pose estimation application using four datasets. We also use `Crontab` jobs and `shell` scripts to automate this data collection process across different hardware prototypes. We leverage standard scaling, min-max scaling, and Box-Cox transformation to preprocess our datasets. We then use Scikit-learn library to implement unsupervised learning, including K-Means, DBSCAN, OPTICS, among others. We implement Linear Regression, SVR, Random Forest, KNN, Ridge, Lasso, and neural network based models to understand and classify different hardware levels in terms of computing resources to provide smart camera services. To build optimal ensemble model, we implement several model reconstruction and ensemble methods, including Voting, Stacking, Bagging, to collect and reconstruct a "global" model. For extended evaluation, we also use Amazon EC2 [13] t1.micro instance with a cost of $0.0035 per hour to perform computing offload to cloud experiments.

VI. Experimental Evaluation

A. Datasets

Dataset 1: MNIST Dataset. We downloaded the MNIST dataset [15], which has a training set of 60,000 image samples, and a test set of 10,000 image samples. MNIST is commonly used for training various image processing systems. The dataset is also widely used for training and testing in the field of machine learning-based applications.

Dataset 2: CIFAR-10 Dataset. We downloaded the CIFAR-10 dataset [8] that is an established computer-vision dataset used for object recognition. The CIFAR-10 dataset consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains 1,000 randomly-selected images from each class.

B. Experimental Setup

PACAS on Camera #1. We implemented and deployed PACAS on the real camera—Open MV H7 Smart Camera. This PACAS implementation covers all system components, including profiling heterogeneous hardware devices, adaptive camera distributed training and prediction, enabling tiny smart camera services, full stack benchmarking and optimization.

PACAS on Camera #2. We implemented and deployed PACAS on the real camera—Arducam 5MP Camera. This PACAS implementation covers all system components, including profiling heterogeneous hardware devices, adaptive camera distributed training and prediction, enabling tiny smart camera services, full stack benchmarking and optimization.

C. Evaluation Metrics

Matthews Correlation Coefficient (MCC). We note that standard evaluating metrics (e.g, accuracy, F1) would not work well on highly imbalanced training data [18]. We use the MCC [14], a standard measure of a classifier's performance, where values are in the range -1.0 to 1.0, with 1.0 being perfect object detection, 0.0 being random object inference, and -1.0 indicating object inference is always wrong. The expression for computing MCC is below, where TP is the

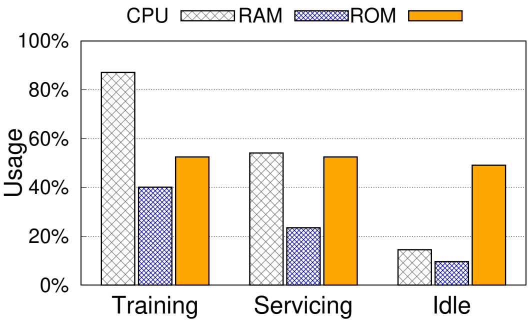


Fig. 8. System overhead comparison across five IoT devices and one camera under collaborative and standalone modes for training and providing services.

fraction of true positives, FP is the fraction of false positives, TN is the fraction of true negatives, and FN is the fraction of false negatives.

$$\frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (1)$$

D. Experimental System Performance Evaluation Results

1) PACAS's Features: As a fully-local, privacy preserving embedded camera system, PACAS is designed to offer comparable capabilities provided by current commercial camera systems, that includes live streaming, event recording triggered by human detection, and object detection for monitoring parcel deliveries. Figure 7 shows the capabilities of PACAS.

2) Quantifying PACAS's Privacy-preserving Performance in Standalone Mode: Figure 5 (a) illustrated the relationship between training time and CPU frequency of PACAS in standalone mode. Not surprisingly, we observe that while maintaining MCC within the range of 0.9∼1.0, the model training time exhibits a non-linear increase as the CPU frequency is reduced. We find that all of the 5 models on a standard 1,800 MHz CPU camera require less than ∼100 seconds for training. This suggests acceptability in the real-life scenario of training a single camera. Note that the result was collected when using 100% of MNIST dataset, and it's far beyond sufficient ∼13% according to the result in Figure 5 (b).

3) Quantifying PACAS's System Overhead: Figure 8 demonstrates the performance of PACAS deployed on camera and other five different IoT devices. For each cluster group, we are reporting CPU utilization, RAM, and ROM usage on the hosted hardware platforms. In Figure 8, we demonstrated three clustered groups, including (re)training, camera servicing, and system idling. Our results show that PACAS on three, four, and five devices in collaborative mode and standalone mode has only a minimal or marginal increase on CPU and RAM utilization, when the smart camera system is armed. That says, PACAS has demonstrated the performance consistency in the ability of protecting user private information across five different devices and one real home camera deployment. Note that, although all the devices are having higher system overhead in the training process than idling mode, it is significant to note that this (re)training process is optional for real-world deployment and up to user personal preference. Many commercially available cameras are only working in pre-trained modeling mode. In addition, we find that ROM utilization remains the same across different devices, which indicates near-zero ROM overhead increasing.

Results: *PACAS almost yields the same ROM overhead across five different devices. PACAS shows the performance consistency in protecting user privacy on five IoT devices and one real camera deployment in both collaborative and standalone modes, with only a minimal or marginal increase on system overhead in terms of CPU and RAM utilization.*

TABLE II
THE PERFORMANCE COMPARISON OF OUR SMART CAMERA SYSTEM

MCC		0.9343
Latency (s)	Modeling	692
	Servicing	1.4

4) Quantifying PACAS's System Latency: We next focus on examining PACAS's system latency on both modeling and servicing. Table II presents the performance comparison results using model accuracy and training and predicting time, with 30% of the shuffled CIFAR-10 dataset for training. Our results indicate that when dispatching partial training dataset to the smart camera, the overall MCC is still close to 1, with a servicing latency lower than 2 seconds. Though the modeling latency, i.e. time consumption for re-training the model reaches 11.5 minutes, the whole re-training process is optional and the user can select to use our pre-trained model directly.

Results: *PACAS consistently performs well, even when migrated from a camera (standalone mode) to multiple local devices (collaborative mode). Furthermore, during these migrations, the service latency of PACAS increases, while the overall training time of PACAS's models decreases.*

5) Examining the quality of PACAS's System Services: Next, we quantify the service quality of PACAS when varying the data relocation size for each participating device. Figure 9 illustrates the performance comparison results. We find that after ∼20% of data relocation, the local models on most participating devices has yielded the MCC as of ∼0.95, which is much closer to 1. After 50% relocation, almost all the devices are achieving their top accuracy. That being said, ∼20% of data relocation is the minimum cut to build a reasonably accurate sub-network model. And this pattern can be consistently observed across four different devices.

Results: *PACAS consistently achieves an MCC accuracy of 0.95 even after relocating approximately ∼20% of the data across four different participating devices. This demonstrates that PACAS can attain high accuracy quickly with only a small portion of the dataset being relocated to the device.*

VII. Conclusion and Future Work

We present a new on device, privacy-preserving smart camera system—PACAS, which enable users to re-gain the control

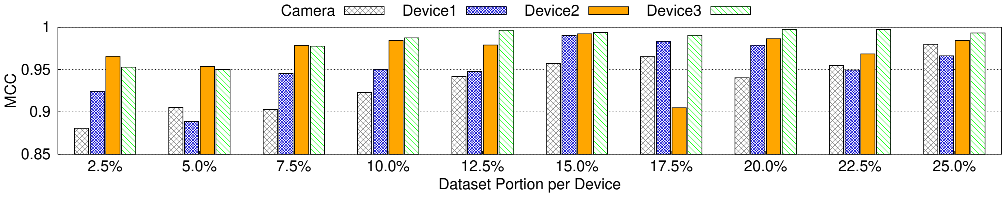


Fig. 9. The comparison of PACAS system when applying different training dataset sizes.

of their smart camera systems and significantly reduce their privacy leakage through their camera recorded data. PACAS first profiles the performance level of multiple embedded device, to determine the adaptive model training distribution. PACAS then harness this profiling information to construct new adaptive, distributed camera service (re)training and prediction system services. Then, we develop tiny intelligent camera services that offer the same features found in commercially available cameras. Eventually, PACAS incorporates a comprehensive full-stack evaluation. We evaluate PACAS using real cameras, and also multiple camera prototypes. We find that PACAS successfully prevents the leakage of sensitive user information with nearly negligible system overhead increase. PACAS offers an evidence that smart camera systems can deliver advanced capabilities, similar to those provided by current commercial systems, all while upholding user privacy.

We plan to implement PACAS in a distributed manner, as well as adding more features for PACAS. We will also collect more data to further evaluate the performance of PACAS. In particular, we will continue to optimize our data relocation approach in runtime. We will also design new data encryption approaches to defend local adversaries.

Acknowledgements. This research is supported by NSF Grant No. 2238701 and DOE NEXUS Seed Grant. We would like to thank the anonymous reviewers for providing us their insightful comments and valuable feedback, which will significantly improve the quality of this paper.

References

[1] Amazon's ring dinged for handing over footage to law enforcement. https://nypost.com/2022/07/16/amazons-ring-dinged-for-handing-over-footage-to-law-enforcement/.

[2] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[3] Ifeoluwapo Aribilola, Mamoona Naveed Asghar, Nadia Kanwal, Martin Fleury, and Brian Lee. Securecam: Selective detection and encryption enabled application for dynamic camera surveillance videos. *IEEE Transactions on Consumer Electronics*, 69(2):156–169, 2023.

[4] Yohan Beugin, Quinn Burke, Blaine Hoak, Ryan Sheatsley, Eric Pauley, Gang Tan, Syed Rafiul Hussain, and Patrick Mcdaniel. Building a privacy-preserving smart camera system. *arXiv preprint arXiv:2201.09338*, 2022.

[5] Gary Bradski. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.

[6] Number of households with smart security cameras worldwide from 2016 to 2027. https://www.statista.com/forecasts/1301193/worldwide-smart-security-camera-homes/.

[7] Ankur Chattopadhyay and T.E. Boult. Privacycam: a privacy preserving camera using uclinux on the blackfin dsp. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[8] CIFAR, howpublished = https://www.kaggle.com/c/cifar-10/.

[9] Delbert Dueck. *Affinity propagation: clustering data by passing messages*. University of Toronto Toronto, ON, Canada, 2009.

[10] Smart home surveillance: Governments tell google's nest to hand over data 300 times. https://www.forbes.com/sites/thomasbrewster/2018/10/13/smart-home-surveillance-governments-tell-googles-nest-to-hand\ \-over-data-300-times/?sh=10f23fff2cfa.

[11] Ihtesham Haider and Bernhard Rinner. Private space monitoring with soc-based smart cameras. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 19–27, 2017.

[12] Greg Hamerly and Charles Elkan. Learning the k in k-means. *Advances in neural information processing systems*, 16, 2003.

[13] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. Scientific workflow applications on amazon ec2. In *2009 5th IEEE international conference on e-science workshops*, pages 59–66. IEEE, 2009.

[14] Matthews Correlation Coefficient.

[15] MNIST. https://www.kaggle.com/datasets/hojjatk/mnist-dataset/.

[16] North america smart home security cameras market size, share and trends analysis report. https://www.grandviewresearch.com/industry-analysis/north-america-smart-home-security-cameras-market-report.

[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[18] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 209–237, 2019.

[19] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[20] Subhan Ullah, Bernhard Rinner, and Lucio Marcenaro. Smart cameras with onboard signcryption for securing iot applications. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2017.

[21] Pete Warden and Daniel Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[22] Thomas Winkler, Ádám Erdélyi, and Bernhard Rinner. Trusteye.m4: Protecting the sensor — not the camera. In *2014 11th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 159–164, 2014.

[23] Thomas Winkler and Bernhard Rinner. Trustcam: Security and privacy-protection for an embedded smart camera based on trusted computing. In *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 593–600, 2010.

[24] Real-Time Object Detection, howpublished = https://pjreddie.com/darknet/yolo/.

[25] Hyunwoo Yu, Jaemin Lim, Kiyeon Kim, and Suk-Bok Lee. Pinto: Enabling video privacy for commodity iot cameras. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1089–1101, New York, NY, USA, 2018. Association for Computing Machinery.

[26] Keyang Yu, Qi Li, Dong Chen, Mohammad Rahman, and Shiqiang Wang. Privacyguard: Enhancing smart home user privacy. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*, pages 62–76, 2021.