This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3072608, IEEE Transactions on Mobile Computing

A Framework for Behavioral Biometric Authentication using Deep Metric Learning on Mobile Devices

Cong Wang, Yanru Xiao, Xing Gao, Li Li and Jun Wang

Abstract—Mobile authentication using behavioral biometrics has been an active area of research. Existing research relies on building machine learning classifiers to recognize an individual's unique patterns. However, these classifiers are not powerful enough to learn the discriminative features. When implemented on the mobile devices, they face new challenges from the behavioral dynamics, data privacy and side-channel leaks. To address these challenges, we present a new framework to incorporate training on battery-powered mobile devices, so private data never leaves the device and training can be flexibly scheduled to adapt the behavioral patterns at runtime. We re-formulate the classification problem into deep metric learning to improve the discriminative power and design an effective countermeasure to thwart side-channel leaks by embedding a noise signature in the sensing signals without sacrificing too much usability. The experiments demonstrate authentication accuracy over 95% on three public datasets, a sheer 15% gain from multi-class classification with less data and robustness against brute-force and side-channel attacks with 99% and 90% success, respectively. We show the feasibility of training with mobile CPUs, where training 100 epochs takes less than 10 mins and can be boosted 3-5 times with feature transfer. Finally, we profile memory, energy and computational overhead. Our results indicate that training consumes lower energy than watching videos and slightly higher energy than playing games.

Index Terms—behavioral authentication, on-device deep learning, privacy preservation, deep metric learning.

1 Introduction

Smartphone has become an indispensable part of our lives. Being a rich mine of personal and corporate data, it is usually sought by cybercriminals as a gateway to key information. The cost of losing one is beyond the replacement of hardware, since data found on the device is usually private and sensitive. As a result, over 65% of users have enabled authentication on their devices [1]. The security-usability conflict has been a persistent design challenge of mobile authentication. From the early methods of passcode, swipe pattern, to the recent advance of fingerprints and FaceID, authentication still requires deliberate attention from the user. On the other hand, behavioral biometrics [10]–[13], [15], [16], [18]–[22] provide implicit channels to identify the user, as they reflect the internal characteristics of a user, and are difficult to replicate. These have made behavioral biometrics a promising (second-factor) authentication to resolve the fundamental usability issues.

Unfortunately, the identification accuracy is still far from satisfaction in the production environments. The behavioral dynamics evolve due to a complex combination of external factors such as sickness, injury and emotion, thus intensifying the intra-class variations and hamper classification. Existing research built around machine learning (ML) leverages statistical properties to recognize users [10]–[13], [15], [16], [18]–[22]. They barely meet practical requirements since hand-crafted statistical features have low discriminative power. Their implementations further ignore important aspects of where and how the model should be trained: these always have profound security and privacy implications. For example, they loosely aggregate user data for training without security precautions. Users may be unwilling to upload private data. On the system level, the read access to motion sensors is not restrictive in the OS [30]–[34], [56], a strong attacker can exploit these side channels to gather motion data. As the same data is also used to train the classifier, the attacker can launch a replay attack by programming an apparatus to generate the identical pattern as the sniffed data [55]. The existing research has yet to consider these issues in the context of mobile authentication.

Meanwhile, propelled by the latest advance in mobile processors, smartphone becomes an ideal platform to execute ML at the data source. The recent efforts mainly improve *inference* from a pre-trained deep neural network model [2]–[5]. Yet, inference from a static model still has a significant gap from being cognizant, since ML relies on the basics that the test samples are independently and identically drawn from the same distribution of training. Deep classifiers are not good at extrapolation when data comes from a different distribution, but it is quite common in mobile applications. This requires the model to adapt to the new distribution, e.g., re-training or finetuning the model to tackle behavioral dynamics. Therefore, being truly intelligent should bring *training* back into the loop.

A natural solution is to securely offload training to the

- *C. Wang and Y. Xiao are with the Department of Computer Science, Old Dominion University, Norfolk, VA, USA, E-mail: {c1wang, yxiao002}@odu.edu*
- *X. Gao is with the Department of Computer Science, University of Delaware, Newark, DE, USA, Email: xgao@udel.edu*
- *L. Li is with Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen, China, Email: li.li@siat.ac.cn*
- *J. Wang is with Futurewei Technologies, Santa Clara, CA, Email: jun.wang2@huawei.com*
- *Correspondence to c1wang@odu.edu.*

cloud. One can host a secure enclave for each user and securely schedule re-training on-demand [24]. However, it also comes with some limitations: 1) *Network connectivity*. Network connection is necessary to upload the new data and receive the updated model. Users may be hesitant when their LTE data plan is metered. 2) *Scalability and Service Cost*. Such centralized approach is not scalable to a large population of users as each user requests dedicated resources from cloud servers. Either the end user or the service provider has to pay for the service, e.g., the Amazon's containerized application at $0.04 per CPU/hr [6] would accumulate to a large amount if constant training is required. Other privacy-preserving techniques are also available [26], [28], [47], but they often come with a nontrivial accuracy loss.

Can we just incorporate training on mobile devices? What are the gaps between mobile and ML that limit the algorithms to perform as they should? Instead of the foreseeable challenges of computation and power in the system architecture, this paper focuses on application-level challenges of *privacy* and *accuracy* to build a comprehensive framework for behavioral authentications. To achieve high accuracy in authentication, we re-formulate the classification problem into deep metric learning [45], [46] to learn a distance metric between the similar and dissimilar samples, so that the features have high discriminative power. Since metric learning learns from pairwise inputs, we develop a new sampling technique that achieves data balance within the memory constraint. After the model is trained, to enhance the reliability of decisions in vibrant mobile environments, we develop a space-time decision fusion algorithm that generates a decision with high confidence level from multiple inference. The results can be fed back to schedule training hence close the loop of learning. Although on-device learning preserves data privacy by default, side-channel leaks from motion sensors in the OS leaves the door open for attackers to spoof authentication via replay attacks [55]. We build a new defense mechanism into the framework to recognize and reject fraudulent samples sniffed from the motion sensors. It is achieved by embedding a noise fingerprint inside the sensing signal and supervising the neural network to distinguish between genuine samples from the ones with noise perturbation. We also explore using feature transfer to speed up training convergence on mobile, while securing all the intermediate activations/model parameters. The main contributions are summarized below.

- We develop a framework for behavioral authentication that the entire loop of training and inference are executed on device. The implementation on Android demonstrates that training is not only feasible but also quite fast with feature transfer (within 5s/epoch on Huawei Mate10 for 400 samples).
- We enhance the discriminative power of the classifier through deep metric learning. Our experiments demonstrate 10-15% improvement of authentication accuracy on different datasets and achieve an accuracy of 0.94 on a large dataset with 153 participants. We also analyze and compare various types of data representations and loss objectives thoroughly.
- We successfully mitigate potential side-channel leaks and reduce attack success ratio below 10% while preserving the usability of benign applications.
- We evaluate the framework extensively on three public datasets and profile learning performance and cost on various smartphone models. To the best of our knowledge, this is the first work that implements both training and inference, and addresses the associated challenges for behavioral authentication on battery-powered mobile devices.

The rest of the paper is organized as follows. Section 2 studies the background and related literatures. Section 3 and Section 4 present the system model and design. Section 6 evaluates the framework with necessary discussions in Section 7 and Section 8 concludes this work.

2 Background and Related Works

2.1 Behavioral Authentication

Commodity mobile devices usually feature an array of sensors that capture the information of acceleration, orientation, angular velocity, magnetic field, etc. These data always implies the behaviorial biometrics of the user, as gait [10]–[14], touch patterns [15], keystroke dynamics [16], micro-movement [17], [18] or their combinations [19], [20], eye movement [21] and even breath [22] are proven to be successful in differentiating human subjects. A system process can run continuously in the background for implicit authentication with no deliberate attention from the user [50], which makes behavioral biometrics an ideal *second factor* for authentication. For example, in [15], four tapping features are analyzed when users type five different PINs. Micro-movements of the phone are considered along with touch-based [18] and signature-based [17] features for authentication. Multi-modalities from hand movement, orientation, and grasp features are considered in [19]. An ensemble of biomedical signals is gathered by medical sensors for continuous authentication in [20]. Audio signatures from sniff, normal and deep breathing are captured via the microphone to identify users [22]. This paper adopts *gait* as a representative among behavioral biometrics. Clinic research has suggested gait as a strong indicator to distinguish between human subjects [10]–[14]. Statistics of correlation, fourier coefficients, histogram are analyzed in [11] and more statistical features are considered in [12]. In [10], an algorithm is proposed to detect gait cycles and compute similarity score with a heuristic algorithm. In [13] and [14], time series are segmented for similarity measurement using dynamic time warping, that relies on the alignment of gait cycles.

The research of behavioral identification has been on the side of exploring different modalities to identify the user. Statistical features such as average max, min, median acceleration, absolute distance, standard deviation, histogram, periodicity, Fourier coefficients or Discrete Cosine Transform coefficients [10]–[13] are typically extracted to train a classifier using a dataset collected in confined environments with minimum interference. In practice, outliers and abrupt changes could easily mislead those statistical classifiers with less discriminative power. To this end, recent studies start to adopt deep neural networks for "automated" feature extraction [51], [52]. Though promising results are shown, they still leave a considerable gap from the high-accuracy requirement in production environments. Furthermore, a

common oversight from the previous works is *where* and *how* the model should be trained, in the shadow of potential threats from privacy leakage on cloud servers to side channel exploits on mobile devices. In this paper, we develop a comprehensive framework to incorporate the entire loop from data generation, computation and decision making on mobile devices, and design a mechanism to defend against side-channel attacks.

2.2 Privacy Preserving and On-Device Learning

Privacy preservation has been extensively studied recently in the machine learning community. The research takes three main directions. The first is the algorithmic direction such as differential privacy [47] and data projection [28]. Differential privacy introduces noise into the training process so adversaries cannot detect the presence or absence of a user [47]. Autoencoder is utilized to transform sensitive features into a latent space for non-sensitive inference [28]. Since these schemes are always subject to the privacy-utility conflict, it is hard to provide rigorous security guarantees and an accuracy loss has to be paid. The second direction is homomorphic encryption that allows curious third parties to perform meaningful computations on encrypted data [25], [26]. CryptoNets use fully homomorphic encryption to encrypt the data from the client and receive an encrypted prediction from the cloud. A square activation function is adopted to bridge the gap between the cryptographical and neural operations, which is only suitable for inference computation due to the accumulation of error in training. The work of [25] designs a two-party protocol to protect both the data and the model using a partially homomorphic encryption. Since homomorphic encryption and decryption are computationally intensive, the computation, energy and communication overhead is prohibitive for mobile devices.

On-device Learning. The third approach is to implement computations directly at the data source, so private information never leaves the device. The existing works either develop new variants of applications such as activity recognition [23] and mobile vision [48], [49], or optimize inference execution on mobile devices using reduced precision and weight pruning [2], [4]. These techniques aim to reduce the run-time redundancy of the neural networks. A direct method is to round the original model parameters in 32-bit floating point to 8-bit integer, so 75% size can be saved [2]. Another approach is to prune the connections with near-zero weights after training [4]. A large and complex teacher network can be also used to train a small student network for comparable results, thus distilling the knowledge to run the small network on mobile devices [5]. These methods are effective for running inference on mobile devices. Nevertheless, they left training out of the loop and are not able to timely engage the dynamics in mobile applications.

Training needs to store all the intermediate results (feature maps, convolution outputs) in the memory for backpropagation. This demands at least twice memory usage compared to inference [7], [8]. For inference, as long as the layers are not referenced any more, its memory can be released and re-used to reduce the peak memory footprint [9]. For training, the intermediate results are persistent in memory until being consumed during the backpropagation, thereby rendering less chance to optimize. The existing architecture and software stack target at mobile workloads that are bursty in nature (user interactions) such as web browsing, texting, etc. The emergence of sustained, highly paralleled workloads have not been fully explored so far. Whether the memory capacity, power management policy and multi-core CPUs are capable to handle neural computation and, meet the application-level requirements are still unknown. This work fills such gap to explore the challenges of training by building the entire framework of behavioral authentication on mobile devices.

Side-channel Attacks on Motion Sensors. Android, one of the most popular mobile OS, only asks for permissions to access sensitive data (such as contacts, messages, camera, GPS, etc) [30]. However, the read access from other sensors is not restricted. This has opened the door for attackers to launch various kinds of motion-based side-channel inference attacks. For example, early works have shown that the accelerometer can be exploited to extract text inputs from the touchscreen [31]. The works later exploit other sensing vectors to recognize speech from gyroscope [32], sniff app usage from magnetometer [34] or utilize manufacture nuances to fingerprint users [33]. Such side-channel vulnerability has direct impact on the implementation of authentication, an essential risk that has been overlooked in the behavioral authentication community [10]–[20]. Powerful attackers can launch replay attacks from sniffed sensing data, e.g., programming a robot to generate the same accelerometer reading to bypass gait authentication [55]. Common defense includes injecting random noise or reduce sampling rates to obfuscate the sensor data [33], [56]. Their effectiveness depends on how much the data is "blurred" from the original signal, which adversely impacts the usability of the normal applications. This paper integrates a joint design of authentication with noise injection, by utilizing the high representational power of neural networks, so that the noise level required for protection can be reduced significantly.

2.3 Deep Metric Learning

Loss function is important for achieving high accuracy of deep models. *Softmax* loss is the most popular one in classification tasks, which consists of the output feature vector (the penultimate layer), the softmax function and the cross-entropy loss. The softmax function produces a probability distribution over the class labels. While the correct class labels are represented by one-hot encodings, the cross-entropy loss are trained to make the output feature vector as close as possible to the one-hot vector. Softmax is widely adopted in the state-of-the-art CNN designs [35]–[37] because of its mathematical tractability and clear probabilistic interpretation. Unfortunately, it only encourages the separation of features, but the learned features are not discriminative enough [38], [39]. A quick fix is to increase the amount of data or the depth of the network. However, gathering more data also involves nontrivial efforts to cleanse, augment and label the new data; increasing the network depth could easily violate the peak memory bound on embedded devices. As a result, softmax still falls short in many large-scale identification tasks [40] as its accuracy cannot meet the strict requirements in practice, particularly, for security-critical applications such as authentication.

Instead of mapping feature vectors in a closed set of class labels, deep metric learning learns the relative distances between samples via a pairwise similarity loss function and transforms features into a space that can be measured by, e.g., the Euclidean distance [45], [46]. In softmax function, the gradient is computed for each sample with the correct class label, whereas the gradients in metric learning depend on the pairwise correlations between samples. Given a similarity metric, similar samples are projected into neighborhood locations in the metric space and dissimilar samples are pulled apart. It has achieved higher accuracy than softmax in different tasks of facial recognition [40], object classification [41] and person re-identification [42]. The design of the loss function determines the discriminative power and the corresponding sampling efforts. For instance, the Siamese network [45], [46] adopts the contrastive loss. The Triplet network introduces an additional anchor point and generates a ranking of the closest images to an input [40], [43]. However, triplets also expand the pairwise sampling effort from $\mathcal{O}(n^2)$ to $\mathcal{O}(n^3)$. The computational cost also grows linearly with the number of neural network branches. Further extensions have adopted quadruplets [44] at much higher sampling ($\mathcal{O}(n^4)$) and computational cost. In this paper, we focus on the Siamese Network given its moderate space/computational complexity and explore the design space on resource-constrained embedded devices under limited data, storage and memory space to bring the entire loop of authentication on-device.

3 System Architecture and Model

In this section, we describe the system architecture and threat model for deep behavioral authentication. The proposed framework is depicted in Fig. 1.

Threat Model. The authentication module reads authentic sensor data from the standard API (e.g., `SensorManager` in Android). We assume that sensor data is trustworthy as its integrity can be protected by hardware security extension technologies such as ARM TrustZone [54]. Meanwhile, sensor data is available to all applications, including malware, since modern OS like Android or iOS does not set restrictions to access sensors. By exploiting sensor data (e.g., accelerometer, gyroscope, magnetometer), numerous side-channel attacks [31]–[34] have been reported to sniff PIN/swipe patterns, password, app usage or even speech.

The main performance metric is the authentication accuracy, measured by whether the system can defend against spoofing attacks and recognize its owner. Specifically, we consider two types of attackers: *passive* and *active*. *Passive* attackers [16] use their own data or samples from a large database to spoof the authentication system. This case is common since a random attacker may obtain a device lost by the victim. Without any prior knowledge on the behavioral pattern, the passive attacker can only retrieve data from a large public database and launch brute-force attacks to unlock the device. *Active* attackers can directly and stealthily collect sensory data from the user's smartphone. This could be achieved by tricking the user into installing a third-party app. We further assume the powerful active attacker can generate the exact same gait pattern as the sniffed sensory data by programming an apparatus such as a robot [55].

One effective countermeasure to side-channel attacks is sensor data obfuscation [33], which injects random noise to user-level apps so that malware cannot recognize sensitive operations. The noise level should balance the usability of benign apps while obfuscating malicious ones. Thus, to mitigate side-channel attacks, the obfuscation technique is employed in the target smartphone by wrapping the `SensorManager` API. For instance, Slogger [56] can inject noise into various sensor outputs. The noise is transparent to the authentication module. While applications (including malware) can only obtain obfuscated data through the wrapped interface, they can apply various denoising techniques to restore the original data. Since our design is centered around on-device implementation, the cloud only plays a secondary role to provide samples from the negative classes. We assume cloud providers are honest but curious: they follow protocols but are free to use what they see to learn private information from users.

4 Framework Design

We present the main design of the framework in this section, including the pair sampling, learning technique, decision-making mechanism and opportunities to speed up the computation.

4.1 Memory-efficient Pair Sampling

We first propose a new sampling technique to construct balanced pairs under the memory limit. The Siamese Network takes pairwise samples of spectrograms converted from the tri-axial accelerometer reading (detail of the conversion is discussed in Section 5.1). The positive pairs represent similar samples from the same class and the negative pairs represent dissimilar samples from different classes. The positive samples are collected on the mobile device during the bootstrapping phase and the negative samples are supplied by the cloud as pre-loaded into the storage. Training takes batched input in memory sampled from flash storage. To avoid the latency accessing the storage, the framework maintains a pool of sampled pairs in memory. This makes sampling crucial because of data imbalance and increased memory footprint.

In authentication, the number of negative samples is much larger than the positive ones. For n negative classes with s samples of each class and r positive samples collected at the mobile user, there are r^2 positive pairs and $n \cdot s \cdot r$ negative pairs. With a large number of negative classes, $n \cdot s$ is much larger than r. Hence, the total number of negative pairs is much larger than the positive ones. Since loading all the negative pairs into memory may lead to memory error, the goal is to keep a random subset within memory limits.

We develop a balanced reservoir sampling algorithm based on [57]. A buffer size of $2R$ is found from hardware configuration or test (half for positive and half for negative pairs). The size determines a trade-off between memory usage and variety of negative records. Small R could lead to severe overfitting and large R risks of having memory error. To maximize coverage, we set $R = r^2$ so all positive samples are utilized for training and make sure that the total size of $2R$ is within the memory capacity. The algorithm continuously adds record into the reservoir till the $(T+1)$-th

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3072608, IEEE Transactions on Mobile Computing

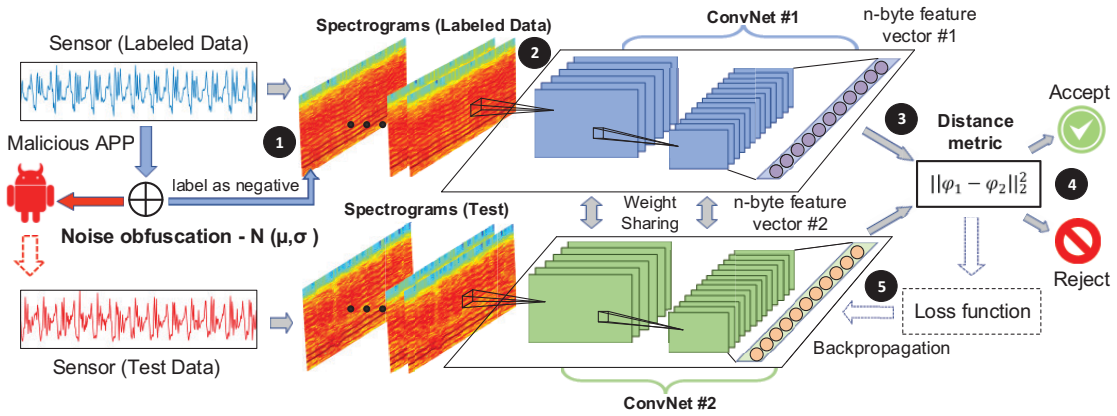


Fig. 1: System architecture on mobile devices: ❶ it takes raw sensor inputs, transforms them into mid-level representations (spectrograms [53]); ❷ processes the representations with the neural network; ❸ computes a distance metric from the feature vectors; ❹ generates a decision; ❺ backpropagates the error if training is scheduled.

Algorithm 1: Memory-efficient Sampling

Input: r^2 positive and $n \cdot r \cdot s$ negative pairs, memory bound $2R$.
Output: a balanced set of samples of size $2R$.
Set of all negative pairs $\mathcal{N}$, $R = r^2$, $|\mathcal{N}| = n \cdot r \cdot s$.
for $T \leftarrow 1, \cdots, R$ **do**
  $\mathcal{R} \leftarrow \mathcal{R} + (i \in \mathcal{N})$.
for $T \leftarrow R+1, \cdots, nrs$ **do**
  if *probability* $p > \frac{R}{T}$ **then**
    $\mathcal{R} \leftarrow \mathcal{R} - (i \in \mathcal{R}) + (i \in \mathcal{N})$.
  else
    $\mathcal{R} \leftarrow \mathcal{R}$.

record, $T = R$. If $T > R$, a random pair in the reservoir is replaced with probability $\frac{R}{T}$ or the reservoir is kept the same with probability $1 - \frac{R}{T}$. After the sequential pass through all the records, the buffer forms a random set from the pool of negative samples.

4.2 Feature Embedding from Siamese Network

The existing work relies on the *softmax* loss with less discriminative power [51]. This section describes a new set of feature embeddings learned from the Siamese Network. As shown in Fig. 1, the Siamese Network incorporates two identical branches of convolutional neural networks with shared model weights. First, the 1D sensing signal is pre-processed (discussed in Section 5.1), converted into a 2D representation and made into positive and negative pairs following Algorithm 1. The Siamese Network takes a pair of inputs to feed into the two identical branches and yields two feature embeddings φ_1, φ_2. A distance metric function $f(\varphi_1, \varphi_2)$ is applied to compute the distance between them, e.g., the l_2 distance $f(\varphi_1, \varphi_2) = \|\varphi_1 - \varphi_2\|_2$. The distance is used to train a *contrastive loss* [45] and map feature vectors to a space in which similar samples have closer distance whereas dissimilar samples are far apart (separated by a margin). For a pair i, j of dataset $\mathcal{D}$, the contrastive loss function is defined as,

$$\mathcal{L}_c = \sum\nolimits_{i,j \in \mathcal{D}} y(\varphi_1^{(i)}, \varphi_2^{(j)}) f(\varphi_1^{(i)}, \varphi_2^{(j)})^2 + (1 - y(\varphi_1^{(i)}, \varphi_2^{(j)})) \max(m - f(\varphi_1^{(i)}, \varphi_2^{(j)}), 0)^2, \quad (1)$$

in which label $y(\varphi_1^{(i)}, \varphi_2^{(j)}) = 0$ for dissimilar pairs and $y(\varphi_1^{(i)}, \varphi_2^{(j)}) = 1$ for similar pairs. m is the margin that separates the dissimilar samples. If the pair is similar (positive), the loss is $f(\varphi_1^{(i)}, \varphi_2^{(j)})^2$; if the pair is dissimilar (negative), the loss is $\max(m - f(\varphi_1, \varphi_2))^2$. When $f(\varphi_1, \varphi_2) > m$, the loss is zero, i.e., dissimilar pair with distance larger than the margin has zero loss. It means that the loss would not encourage further separation when the distance between a dissimilar pair is already larger than the margin.

Joint Loss. A slightly different loss function $\mathcal{L}_s$ is proposed in [46], that maps dissimilarity into a probability prediction with sigmoid activation so the network can be trained with cross-entropy loss. The advantage is that no margin needs to be pre-determined,

$$\mathcal{L}_s = \sum\nolimits_{i,j \in \mathcal{D}} y(\varphi_1^{(i)}, \varphi_2^{(j)}) \log p(\varphi_1^{(i)}, \varphi_2^{(j)}) + (1 - y(\varphi_1^{(i)}, \varphi_2^{(j)})) \log(1 - p(\varphi_1^{(i)}, \varphi_2^{(j)})). \quad (2)$$

The construction of the loss function plays an important role to capture similarity in different applications. To capitalize from the potential advantages of distance and probabilistic metrics, we combine them into a new joint loss function. The goal is to minimize the total loss $\mathcal{L}_t$ with a balancing parameter γ,

$$\mathcal{L}_t = \mathcal{L}_c + \gamma \mathcal{L}_s, \quad (3)$$

where $\mathcal{L}_s$ is the cross-entropy loss in Eq. (3) and $\mathcal{L}_c$ is the contrastive loss in Eq. (1). γ is a scaling parameter used for balancing the two functions. The feature embeddings can be directly measured in terms of a distance metric (i.e., l_2 distance in this paper), and these different loss formulations are evaluated in Section 6.

4.3 Decision Fusion and Feedback

After the model is trained, the inference module takes input from sensors and outputs a classification decision. The decision based on a single shot of inference is not reliable because interference, outliers, and behavioral instability persist at run-time. The goal is to reach a high confidence within minimum observation time. We build an algorithm on top of the inference module to fuse multiple inferences across the spatial and temporal dimensions. We leverage the positive training samples as the ground truth and assess whether the input data is similar or dissimilar to the training

samples, i.e., pair the input (testing data) with the positive training samples. Since paring with one training sample is not sufficiently representative, for input x_i at time i, x_i is paired with k samples randomly selected from the training set on mobile ($k > 1$). The average distance from x_i to the k samples are computed as $d_i = \sum_{j=1}^{k} d(x_j, x_i)/k$, in which $d(x_j, x_i)$ is the distance between input x_i and training sample x_j.

Not only could the selection of training samples have imperfections, the incoming data may also have disturbances. After the spatial evaluation, we progress along the time dimension to fuse multiple decisions $\{y_1, y_2, \cdots, y_n\}$. After the i-th evaluation, it either decides to *accept* (H_0), *reject* (H_1) or *continue* to observe y_{n+1}. The module defines two kinds of errors: false negative α and false positive β. The objective is to minimize the expected time of evaluation and satisfy the error constraints, which is formulated as Sequential Probability Ratio Test (SPRT) [58]. SPRT progresses by assessing a likelihood ratio λ_n for the n-th observation,

$$\lambda_n = \frac{p(y_1, \cdots, y_n|H_1)}{p(y_1, \cdots, y_n|H_0)} = \prod_{i=1}^{n} \frac{p(y_i|H_1)}{p(y_i|H_0)}. \tag{4}$$

The second equality holds because samples are independently randomly drawn. We extend SPRT for the distance metric (contrastive loss). Pairs with distance less than the margin threshold are considered as similar; otherwise, they are dissimilar. To convert the distance into probabilistic representation, we approximate it with a normal distribution but in an upside-down bell shape, which has been used for risk assessment to impose a loss on off-target products in manufacturing [71]. We adopt the same analogy here since distances that are close to 0 or margin m have a high probability of being similar or dissimilar respectively. The margin threshold is typically set to $\frac{m}{2}$ for a balanced set of positive and negative pairs. The distance around $\frac{m}{2}$ means the classifier is unsure about the pairs so a lower probability is given.

$$p(d_i|\mu, \sigma^2) = 1 - \phi(\frac{d_i - \mu}{\sigma^2}). \tag{5}$$

The mean and variance can be obtained from the learned distance of the training data. Combining (4) and (5), the ratio is,

$$\frac{p(y_i = 0|H_1)}{p(y_i = 0|H_0)} = \phi(\frac{d_i - \mu}{\sigma^2})/(1 - \phi(\frac{d_i - \mu}{\sigma^2})) \tag{6}$$

$$\frac{p(y_i = 1|H_1)}{p(y_i = 1|H_0)} = (1 - \phi(\frac{d_i - \mu}{\sigma^2}))/\phi(\frac{d_i - \mu}{\sigma^2}) \tag{7}$$

According to [58], the decision strategy is,

$$S_n^* = \begin{cases} H_0, \lambda_n \leq B \\ H_1, \lambda_n \geq A \\ continue, B < \lambda_n < A \end{cases} \tag{8}$$

We set the two thresholds A and B suggested by [58], $A = (1 - \beta)/\alpha$, $B = \beta/(1 - \alpha)$. The sequence moves within the open interval (B, A) till a decision is made. Intuitively, if consecutive decisions of acceptance are made, the likelihood ratio shrinks multiplicatively. Any rejection along the way would drive the ratio to an opposite direction towards the upper threshold until a threshold is met. The decision of S_n^* is examined closely to schedule training.

Algorithm 2: Decision Fusion and Feedback

Input: Testing pairs (x_j, x_i), $1 \leq j \leq k$. k pairs randomly drawn from training set. False negative α and false positive β, threshold $A = (1 - \beta)/\alpha$, $B = \beta/(1 - \alpha)$.
Output: Decision S_n^* and training schedules.
Initialize false negative counter $c \leftarrow 0$, and threshold T.
while $c < T$ **do**
  $n \leftarrow 0$
  while $B < \lambda_n < A$ **do**
    $\overline{d_i} \leftarrow \sum_{j=1}^{k} d(x_j, x_i)/k, p(d_i|\mu, \sigma^2) \leftarrow 1 - \phi(\frac{d_i - \mu}{\sigma^2})$.
    $\lambda_n \leftarrow \prod_{i=1}^{n} \frac{p(d_i|H_1)}{p(d_i|H_0)}$.
    if $\lambda_n \geq B$ **then**
      $S_n^* \leftarrow 1$ and **Break**.
    if $\lambda_n \leq A$ **then**
      $S_n^* \leftarrow 0$ and **Break**.
    $n \leftarrow n + 1$
  Output optimal decision S_n^*.
  if *Given true label* H_0, $S_n^* = H_1$. **then**
    $c \leftarrow c + 1$
Schedule training of $\mathcal{M}_t$ with new data $\mathcal{D}_t$.

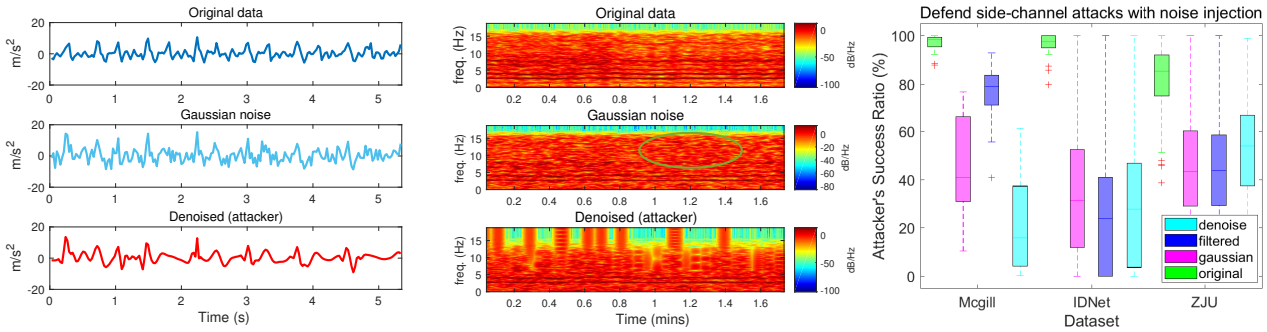
Feedback. We examine the testing accuracy as a feedback to schedule model re-training and adapt variations. In authentication, if the decision outputs a false negative, the screen is mistakenly locked by the (second-factor) behavioral authentication, but the user later logins with her face or fingerprint (that verifies the decision is indeed a false negative). If such situations exceed a certain number, it indicates that the user's behavior may have undergone a substantial change and training is scheduled with a mix from the new data. Incorporating training on mobile could immediately respond to these deviations thereby closing the loop of learning on mobile devices. The scheme is summarized in Algorithm 2 and evaluated in Section 6.8.

4.4 Defend Side-channel Leaks and Active Attacks

Although well-trained neural networks could achieve high accuracy, the sensitive data might be also leaked via side channels. Skilled attackers can trick the user into downloading an app that stealthily captures the motion data, and then replay them to gain the access via programming an apparatus [55]. A typical countermeasure is to obfuscate the sensor output by injecting random noise [33]. However, our experiments show that simple noise injection fails to fully prevent the attack. Specifically, we obfuscate the data with a *zero-mean* gaussian noise, whose standard deviation is set to equal the original signal over a moving window. As circled in the middle picture of Fig. 2(b), the obfuscation introduces a few new energy components at higher frequencies in the spectrograms, which enable the authentication module to recognize attackers to some extent. Fig. 2(c) shows the attacker's success ratio. Without any protection, the attacker can easily accomplish 80-100% success rate (the true positive rate). With noise injected, the success rate drops to an average of 50%. However, it is still not sufficiently secure. In extreme cases, some individuals are still subject to 100% attack success rate even when noise is injected.

One reason that simple noise injection does not work here is that neural network is robust to random noise. It extracts a meaningful combination of features towards a minimization of the loss objective, and serves as an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3072608, IEEE Transactions on Mobile Computing



(a) (b) (c)

Fig. 2: Preliminary assessment of simple noise injection (a) raw sensing data with/without noise; (b) relevant spectrograms; (c) attacker's success ratio.

information bottleneck that finds a compressed mapping of input that preserves maximally possible information of the output [59]. Thus, redundant information including small noise and interference, which does not interfere with the main structure, is thrown away. As a result, the attackers can still succeed. Obviously, a successful obfuscation requires raising the standard deviation of the noise (e.g. surpassing the standard deviation of the signals) to generate larger noise. However, it will inevitably impact the usability of legitimate apps.

Furthermore, attackers can apply various denoise techniques to potentially raise the success rate if the original waveform is not changed by the denoise method. In our preliminary experiment, we measure the attackers' success rate by applying two denoise methods: total variation proximity operators [60] and 1D gaussian filter. As shown in the spectrograms in Fig. 2(b), the new energy components at higher frequencies are removed by denoising. In our experiment, by applying appropriate denoising methods on Mcgill and ZJU datasets, attackers can achieve higher success rates.

Our approach. We propose a new defense mechanism against strong adversaries by making a small extension, that supervises the neural network to learn the injected noise and use it as a hidden fingerprint. The idea is to embed a slightly noised signature in the sensing signal, and instruct the neural network to recognize such signature if the attacker launches a replay attack from the sniffed data. Specifically, our design is backed by the recent discovery that the neural networks can even fit unstructured random noise with random labels [61]. It is also supported by the studies in adversarial learning [62], which supervises the classifier to distinguish random perturbations induced from the adversarial examples [63]. Here, our intention is that if the pair of genuine and noised data (genuine plus noise) is labeled as negative, the Siamese Network is being supervised to map them into different areas in the metric space, similar to how the adversarial examples are learned [62], i.e., where the "noise" are treated as hidden features perceivable by the neural network. Admittedly, this would make the network more difficult to train, in order to learn the patterns from the additive noise and distinguish from the genuine data. With the superb representational capability of the neural network to approximate continuous functions (the universal approximation theorem [64]), it is able to fit the noise part as validated in our evaluation. By labeling them as negative, the system thus forcibly learns the nuances between genuine and noised samples.

The extension can be realized by creating a simple wrapper around `SensorManager`, attackers using the noised data for replay attacks would be recognized by the system. Meanwhile, our mechanism significantly reduces the level of noise without sacrificing much usability from the benign applications, while the naive noise injection requires a large noise level for successful obfuscation. To mitigate the impact of possible denoising from the attacker, the system can further predict the potential classical denoise algorithms that might be used by attackers. The authentication module can generate the denoised pairs beforehand and similarly label them as negative for training. The detailed evaluation of our proposed defending system is presented in Section 6.7.

4.5 Speed up Convergence via Privacy-Preserved Feature Transfer

Training learns hundreds of thousands of parameters through backpropagation, which could take more than hundreds of epochs till convergence. The previous work proposed to partition the neural network between the cloud and mobile device in a layer-wise manner [29]. We build on this approach to speed up convergence via feature transfer. With domain similarities, knowledge learned from the source can be efficiently repurposed for the target domains. For neural networks, the high-level features learned from the first few layers are more generic, while the low-level features are more specific to the classification tasks [65] (e.g., the early layers learn general features like edge detectors to identify the concentration of frequency energy from the sensing signals).

To initiate, the cloud (*source*) and the mobile (*target*) agree on a partial network structure of the first k layers, e.g., the first two convolutional layers. The agreement includes the model weights and hyper-parameters. For the target model, a few adaptation layers are introduced. Thus, the high-level features could be efficiently reused on user's mobile device. We can utilize a source model $\mathcal{M}_s$ trained on the public dataset with n_s classes in the cloud, and transfer the learned features for the n_t classes in the target model $\mathcal{M}_t$, when the source and target domains do not overlap. Specifically, samples x from the source domain are passed through $\mathcal{M}_s$ until the k-th cutoff layer, where x is represented as an n-byte feature vector. Next, the model parameters $\mathcal{M}_s$ are transferred to the target model $\mathcal{M}_t$ for the first k layers, along

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3072608, IEEE Transactions on Mobile Computing

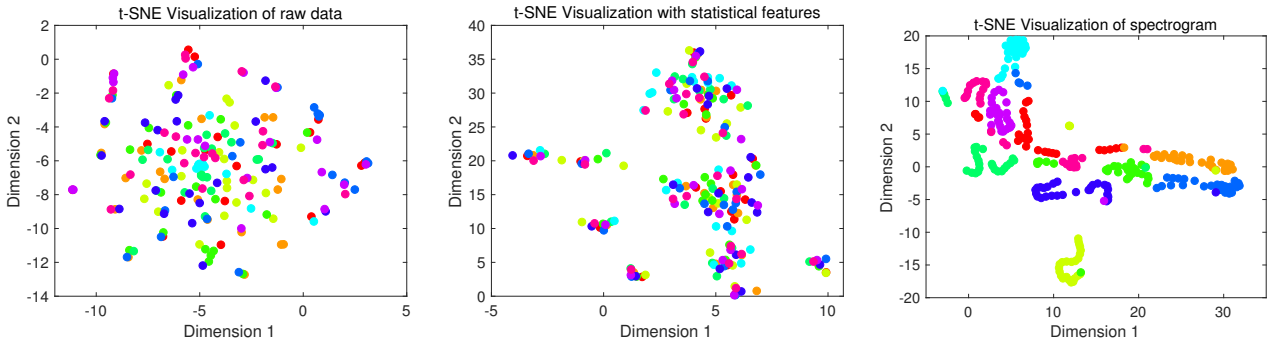


Fig. 3: t-SNE visualization (best view in color; each color represents samples from an individual) (a) raw walking data (b) nine statistical features (c) spectrogram.

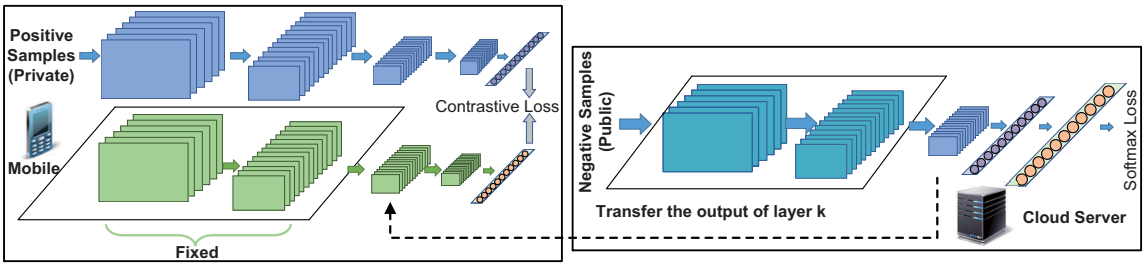


Fig. 4: An example of transferring features of the negative samples from the first two convolutional layers to the mobile device.

with all the feature vectors. To initiate training on mobile, the target model freezes weights of the first k layers. The error is backpropagated from the last layer to the $(k+1)$-th layer. The weights of the adaptation layers are adjusted according to stochastic gradient descent. Fig. 4 shows an example that transfers the output of the first two convolutional layers to the mobile device for the negative samples, where the positive samples from the user are computed on-device.

Note that the cloud is not aware of the layer structure, loss function or model weights beyond the k-th layer on the mobile devices. We show by experiments that this approach has great potentials of adaptation, such as allowing the source and target to have different loss functions (softmax for $\mathcal{M}_s$ and contrastive loss for $\mathcal{M}_t$) and heterogenous sensor hardware with different sampling frequencies. The target model can still learn effectively and enjoy massive speed-up of convergence with little accuracy loss.

Remarks on Privacy and Communication. Other than the active attackers, privacy exploits attempt to reconstruct the original data from feature activations [66] or model parameters [67]. Our design is robust against these exploits since: 1) activations generated from the target model are kept on mobile thus curious cloud providers cannot invert the private data; 2) though data can be recovered from the shared weights of k layers on mobile by curious users, the data is public and carries little business value; 3) the model weights beyond the k-th layers on mobile are not disclosed to anyone else, hence a third party cannot recover private data from the model parameters.

The cost from network communication is associated with the computational speed-up. Fortunately, the neural architectures need to remain compact to fit into the device's memory, so the increased network overhead to transmit the intermediate features is manageable for on-device implementation. For T-mobile 4G LTE at 60 Mbps or the campus WiFi network at 90 Mbps downlink speed, the communication takes about 1-2 mins for the architectures in Table 1. Features can be downloaded in an offline fashion when the mobile device is connected to the WiFi. To further reduce communication overhead, compression can be applied to take advantage of the inherent sparsity in the feature activations, e.g., the zero-value compression [69] and ZLIB compression [68] can reduce the network bandwidth by $3-5\times$ [70].

5 Implementation

5.1 Data Pre-processing

Data processing lays the foundation to achieve high accuracy. Here, we formally discuss the intuition behind using *spectrograms* to represent sensing signals gathered by accelerometer sensors. Unlike images, the accelerometer signal is one-dimensional time series. Existing research mainly works in the time-domain and requires cycle extraction [51] or segmentation [34]. Cycle extraction looks for cyclic patterns between local minima or maxima algorithmically, but is prone to error in the presence of noise. Segmentation divides the signal into many overlapped pieces that expands the dataset by many folds. Here, we adopt a new approach to model walking data as speech and demonstrate its advantages in the following.

Walking consists of a set of motions from the body parts (torso and limb), which shares similarities with speech from their generation mechanisms. While speaking, the pulse from vocal cords is modulated in frequency through the throat cavity and reshaped by the articulators (tongue, mouth, lips) to produce sound. Gait signals generate a similar pattern from the body parts. Based on these observations, it is reasonable to model gait as speech, which is typically analyzed in spectrogram [53].

A spectrogram uses three dimensions to represent signal energy as a function of time (x-axis) and frequency (y-

axis). It breaks data into segments of short intervals, takes short-time Fourier transform in each segment and assigns frequency spectrums into different bins of magnitude. Each bin stands for the frequency scale perceived. Spectrogram concatenates multiple quasi-stationary cycles to generate a 2D output. This way, learning can be performed effectively using convolutional neural networks.

The compelling advantage of spectrogram is suggested by Fig. 3 (visualized by the t-SNE tool to reduce the high-dimensional data into 2D [72]). Fig. 3(a) shows the raw sensing signal and Fig. 3(b) visualizes the data with nine statistical features. Though these statistical features form a distinguishable trend, they are still not powerful enough; in sharp contrast, data points are clustered in a more organized manner in Fig.3(c), so it is much easier to build a classifier and recognize different individuals.

5.2 Model Development

Model architecture determines the representational power, memory requirement, and computational intensity. In this paper, we evaluate three convolutional neural network architectures extended the families of *LeNet* [73], *VGG* [36] and *MobileNetv2* [3]. Though an alternative is to use the recurrent neural networks [52], the computation intensity is much higher. We customize these classic models to add or prune layers in order to yield similar input dimension at the dense layer as their original implementation with the ImageNet. The spectrograms of (x, y, z) axis are stacked vertically to form 33×42 images.

In particular, LeNet repeats two blocks of 5×5 convolutional and max pooling layers followed by densely connected layers. We add one more 3×3 convolutional layer and prune one dense layer to get 4 weight layers (therefore the name LeNet4). VGG repeats two 3×3 convolutional layers to achieve similar receptive field with the 5×5 convolution, but much less computation/parameters. Multiple such blocks are stacked to learn complex relations among the features. We repeat the blocks three times and introduce one more convolutional layer before the last max pooling, thus making VGG8 a heavy-weight network with 8 weight layers. We also implement the latest MobileNetv2. The model stacks inverted residual blocks (inv_res_bl) to take low-dimensional representation, and then expands to the high-dimension for efficient feature extraction by the depthwise convolution. The blocks are connected with bypass links to make deeper structures trainable with less degradations. The max pooling layer is replaced by a convolution stride of 2, e.g. $(1, 1, 2)$ represents two blocks with a stride of 1 followed by a block with a stride of 2. Table 1 summarizes the model architectures and layer-wise parameters.

5.3 Mobile Development

The choice of the software framework is crucial since training requires backpropagation. Despite a handful of available frameworks, most of them (e.g. *Tensorflow Lite* [74]–[76]) have tailored backpropagation and left only the inference part to compute from pre-trained models. This way, no intermediate gradient values need to be stored and the memory/code can be optimized. In this paper, to enable training on the mobile device, we develop the system on a Java-based framework called *DL4J* [77]. Since the two Siamese branches are identical, only one copy of the model is stored in memory. During testing, we notice that deeper structures could cause `OutOfMemoryError` due to a large number of parameters and batched data processing. To mitigate, we set `largeHeap` to give the application a 512 MB heap capacity.

6 Experimental Evaluation

This section conducts a thorough evaluation of the framework for deep behavioral authentication. The main goals of the evaluations are: 1) investigate the accuracy and computational cost of different models and approaches; 2) examine cost savings and performance impact from feature transfer; 3) validate system robustness against both random and active attacks; 4) profile performance and overhead on various smartphone models.

6.1 Dataset and Experimental Settings

To make the benchmarks comparable, the experiments are based on public datasets: Mcgill [78], IDNet [79], ZJU [80] and Osaka [81] gait datasets. Note that this paper focuses on algorithm design and system integration rather than collecting, analyzing or deriving data from human subjects. Thus, an IRB approval is not required. With a total coverage of around 1,000 individuals, we believe the four datasets are sufficient to validate the system in various scenarios.

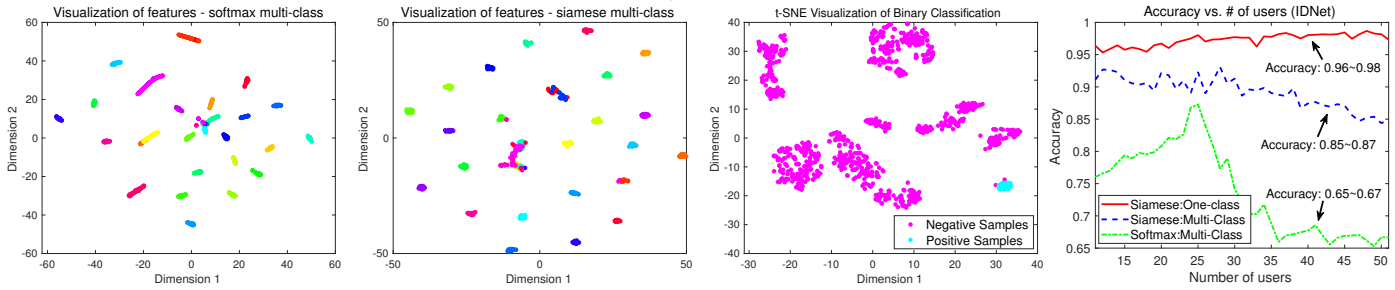
In particular, Mcgill includes 15-min walk of 20 people on two different days. IDNet is collected in a more vibrant environment with 50 participants, where there is no restriction of phone types and clothes (different clothes would lead to slightly different gait patterns from the same individual). ZJU collects gait data from 153 individuals in 3 different sessions using 5 body sensors of low sampling rates. Osaka records 1-minute walk of 744 subjects. Due to short recordings (only 1-2 spectrograms), we cannot perform meaningful training so it is utilized as a large database from which attackers may launch random attacks. Since some individuals have much less or missing data in IDNet and ZJU, we remove those individuals for data balance. This ultimately brings them to 30 and 136 individuals respectively.

The datasets are first split into 80% for training and 20% for testing. Then Algorithm 1 is adopted to form balanced positive and negative pairs from the training and testing set respectively. For SPRT test, the pairs are generated by randomly pairing training samples with testing samples as described in Section 4.3, where the training samples act as the ground truth. This simulates the run-time when new motion data is evaluated against training samples as the ground truths. To assess the performance of authentication, we mainly focus on the mean Average Precision (mAP), which is the average percentage of true authentication over the total number of testing. We also evaluate the trade-offs between false rejection (the genuine user is falsely rejected) and false acceptance (an imposter is falsely accepted) using different margin threshold.

We set the margin $m = 1.5$ in the contrastive loss (Eq. (1)) based on the best Equal Error Rate from the experiment discussed in Section 6.8. $\gamma = 0.1$ is set empirically for the joint loss (Eq. (3)) so the training is led by the contrastive loss. For fast prototyping, we first develop the model and

LeNet4		VGG8		MobileNetv2	
#layer blocks	# param.	# layer blocks	# param.	# layer blocks	# param.
$32\times$conv2d$(5,5)$+pool	0.96K	$2\times(64\times$conv2d$(3,3)$+pool$)$	39.2K	conv2d$(3,3,2)$	1.9K
$64\times$conv2d$(5,5)$+pool	51.5K	$2\times(128\times$conv2d$(3,3)$+pool$)$	222.5K	$(16,32)\times$inv_res_bl$(1,1,2)$	47.3K
$32\times$conv2d$(3,3)$	51.4K	$3\times(128\times$conv2d$(3,3)$+maxpool$)$	444.3K	conv2d$(1,1)$	1.1K
dense(128)	82.5K	dense(128)	327.8K	dense(64)	61.8K
contrastive/x-entropy loss	186.36 K	contrastive/x-entropy loss	1033.8K	contrastive/x-entropy loss	112.1 K

TABLE 1: Summary of model architectures



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

Fig. 5: Multi-class and binary classification via t-SNE visualizations (a) softmax (multi-class); (b) siamese (multi-class); (c) siamese (binary); (d) accuracy vs. user number. (Best view in color)

		baseline			siamese multi-class			siamese binary-class (20% data)		
		softmax(sw)	softmax(spgm)	osvm	contrastive	x-entropy	joint	contrastive	x-entropy	joint
Mcgill	LeNet4	0.774	0.881	0.542	0.918	0.940	0.925	0.966	0.934	**0.975**
	VGG8	0.752	0.902	0.672	0.925	0.952	0.931	0.962	0.906	**0.973**
	Mobilenet	0.682	0.811	0.581	0.865	0.926	0.923	0.847	0.901	**0.957**
IDNet	LeNet4	0.726	0.842	0.552	0.884	0.903	0.910	0.937	0.899	**0.943**
	VGG8	0.764	0.875	0.561	0.916	0.934	0.915	0.908	0.901	**0.941**
	Mobilenetv2	0.770	0.776	0.591	0.876	0.912	0.867	0.910	0.921	**0.945**
ZJU	LeNet4	0.442	0.646	0.511	0.681	0.804	0.779	0.941	0.926	**0.972**
	VGG8	0.463	0.743	0.523	0.769	0.841	0.800	0.936	0.851	**0.981**
	Mobilenetv2	0.591	0.471	0.510	0.706	0.778	0.743	0.895	0.835	**0.921**

TABLE 2: Model accuracy of different loss functions for the siamese network

evaluate authentication accuracy, security and performance in *Tensorflow* [74] with Nvidia Tesla P100 GPU, and then develop the learning module on Nexus 6/6P, Huawei Mate 10 and Google Pixel2 using DL4J [77]. A large batch size of 128 is used while training on GPU for 100 epoches. The learning rate is set to 0.03 with the RMSprop optimizer. Accuracy is averaged over 10 different runs and each run draws a random subset of samples to construct balanced paring between positive and negative pairs following Algorithm 1. During our testing, we find that the maximum batch size for Nexus 6 (oldest phone in our test) is 56 pairs. To test various models and avoid memory errors, we set the batch size to 20 on mobile.

6.2 Authentication Accuracy

We first evaluate the authentication accuracy by comparing models, data representation and learning mechanisms on different datasets in Table 2. We validate the choice of spectrogram by comparing with the pre-processing technique of sliding window (SW) [34] on the temporal data both using softmax (cols. 1, 2). As envisioned by the t-SNE visualization of Fig.5, spectrogram achieves a significant accuracy gain of over 10% (col. 4). A one-class SVM (osvm) is used in [51] to detect outliers from imposters. It takes features from the last convolutional layer learned from the softmax function to train an osvm using positive samples only. Unfortunately, though osvm can handle 80-90% outliers, it fails to generalize to the positive samples, which results in high rate of false rejections. Thus, the total accuracy is just slightly better than random guesses (col. 3).

Softmax vs. Contrastive Loss. Our motivation to use the Siamese Network is because of the higher discriminative power on small data. To validate, we first visualize the features learned by softmax and siamese (contrastive loss) in Figs. 5(a) and (b), where the colors represent the feature vectors of different subjects in 2D. Features learned by softmax are not sufficiently discriminative where the distance along the feature vectors from the same individual could be similar to a different individual. We further notice that some features belong to different individuals are mapped to the same vector space in 2D. These findings are in line with [82] (softmax tends to underperform). Contrastive loss from the siamese network offers improvements by mapping feature activations into a condensed, compact set of spaces. This validates the higher discriminative power of deep metric learning than softmax especially with less training data. Not only via feature visualization, the authentication accuracy also indicates 8-15% improvements between the two methods (col. 2 and 4-6 of Table 2).

Binary vs. Multi-class Classification. We discuss the impact of formulating the problem into either the binary or multi-class classification problem. Multi-class classification requires all the pairs between different classes to be labeled whereas binary only labels one vs. the rest. The former is more suitable for recognition tasks where a centralized model is trained to identify different users. The recognition

model can be also migrated to the mobile devices for authentication [51]. However, it is subject to potential security risks when a malicious end user attempts to invert training data of other individuals [67]. In addition, we investigate the performance gap between the two formulations in terms of model accuracy.

To simulate limited mobile storage, only 20% data from the training set is used for binary classification but evaluated on the entire test set. This is challenging for recognition since the neural network can only "see" from a small subset of training data. A model is trained for each individual and the results are averaged. Fig. 5(c) visualizes binary classification. It only distinguishes the positive samples from the rest and the negative samples can be mapped to similar locations in space without causing an error. Nevertheless, multi-class classification still has to separate all the individuals by a margin. This makes it difficult to differentiate the hard examples, which are those in the training set that are difficult to be classified correctly (Figs. 5(a)(b)). To test the scalability of multi-class classification, we show the results in Fig. 5(d) by increasing the number of classes in IDNet. The accuracy declines with a growing number of classes in the system. Hence, model capacity should keep growing as new users subscribe to the service. This would require extensive maintenance efforts in distributed mobile environments. As projected in Fig. 5(d), accuracy is independent from the system scale using binary classification with a fixed network architecture.

Table 2 summarizes the overall accuracy comparison. With multi-class classification using the Siamese Network, accuracy still declines a little with an increasing number of classes (e.g. from 0.952 of Mcgill with 20 people down to 0.841 of ZJU with 136 people). By reducing the problem into binary classification, the accuracy stays above 90%. Among them, the new joint loss accomplishes the best accuracy with over 95% correctness. This is because the joint loss balances the two loss functions and combines the model outputs for higher fidelity.

We also notice some interesting phenomenon that the cross-entropy loss is better than the contrastive loss for multi-class classification, but the opposite for binary classification. The difference between them is that the cross-entropy generates a probabilistic decision, rather than a deterministic distance metric from the contrastive loss. In our experiment, we discover that contrastive loss is more prone to error during multi-class classification in the presence of hard examples. Due to space limit, we would further investigate this issue in our future work. Finally, we alter the model into VGG8 and MobileNetv2. VGG8 achieves the best accuracy in most cases. With 40% less parameters, MobileNetv2 suffers 8-26% accuracy loss compared to LeNet4. This indicates that networks particularly optimized on model parameters and computer vision tasks may perform poorly on mobile sensing tasks, compared to simple solutions of stacking convolutional layers such as VGG8.

6.3 Resource Requirement

To quantify the performance and resource requirements of the mobile sensing task, we conduct more experiments to illustrate the relations between model parameters, floating point operations (FLOPS), and accuracy in Fig. 6. We alter the structures by shrinking/expanding filter size, numbers, and adding/removing convolutional or pooling layers. For the same model, in general, more parameters bring higher representational power at the risk of overfitting and cost of computation. From Fig. 6(a), VGG8 is more stable than others in terms of accuracy. Once the number of parameters exceeds a million, the models tend to overfit. Mobilenetv2 can be tailored to only weigh half of LeNet4, but the performance is not stable. Fig. 6(b) also indicates that it incurs nontrivial GPU time if the FLOPS increase. Fig. 6(c) shows that LeNet4/VGG8 are more competitive than Mobilenetv2 for the datasets in terms of computation time and accuracy.

To facilitate mobile development, we conduct the following experiments using LeNet4 and keep the consistency through the rest of the experiments. Fig. 6(d) shows the training time per epoch on mobile devices. We plot in 3D for better visualization of the impact from the convolutional and dense layer. Training on mobile devices is not only feasible, but actually much faster than expected. For a deep model with 650K parameters and 400 samples, it only takes the latest Pixel2 or Mate10 less than 5 seconds to complete one training epoch. Thus, training 100 epochs takes less than 10 mins. Even the old Nexus 6 finishes around 10 seconds per epoch. During the experiment, we notice that the speed bottleneck of convolutional layers is magnified on mobile devices due to less processing power from the mobile CPUs and memory. As observed in Fig. 6(d), with more convolutional layers, training time surges sharply. However, increasing computations of the dense layer has less impact on performance. Interestingly, we are even able to train some networks with over a million parameters, as long as most of the parameters reside in the dense layer. Equipped with the capability to learn, model updates can be scheduled efficiently without external efforts from service providers.

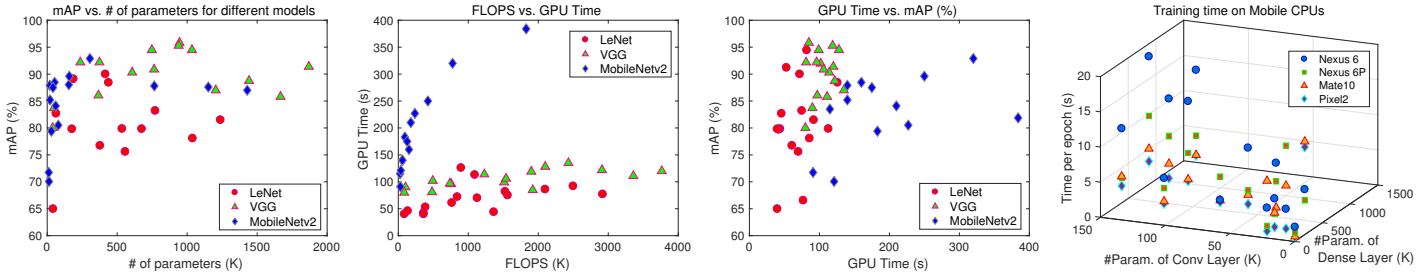
6.4 Speed up on Mobile by Feature Transfer

Since convolutional layers learn common features, these features can be efficiently transferred from the cloud for computation efficiency. To see such potential, the following cases are evaluated: 1) freeze all convolutional layer weights (*fconv1-3*); 2) freeze the first two convolutional layer weights (*fconv1-2*); 3) freeze the first convolutional layer weights (*fconv1*), and train the rest of the layers. The source model conducts multi-class classification on the dataset (public) without the presence of the target user (private). At the target user, it performs the binary classification based on the

feature transfer		Mcgill	IDNet	ZJU	gain/loss
Mcgill	*fconv1-3*	0.933	0.903	0.907	-5.2%
	fconv1-2	0.948	0.927	0.918	-3.5%
	fconv1	0.953	0.941	0.948	-1.9%
	gain/loss	-2.1%	-4.2%	-4.2%	–
IDNet	*fconv1-3*	0.876	0.941	0.896	-3.3%
	fconv1-2	0.922	0.951	0.911	-0.9%
	fconv1	0.933	0.957	0.936	+0.5%
	gain/loss	-2.7%	+1.3%	-2.3%	–
ZJU	*fconv1-3*	0.808	0.810	0.829	-12.5%
	fconv1-2	0.836	0.818	0.833	-11.3%
	fconv1	0.832	0.804	0.847	-11.3%
	gain/loss	-11.6%	-13.0%	-10.5%	–

TABLE 3: Accuracy with feature transfer

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3072608, IEEE Transactions on Mobile Computing

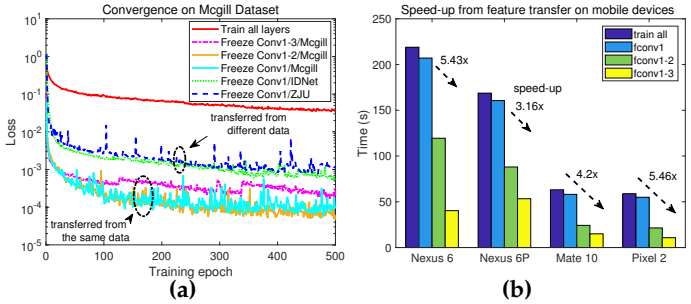


(a) (b) (c) (d)

Fig. 6: Evaluation of resource requirement vs. accuracy on GPU and mobile platforms using IDNet (a) mAP vs. parameters; (b) FLOPS vs. GPU time; (c) GPU time vs. mAP; (d) Parameters (Conv and Dense Layers) vs. Mobile CPU Time.

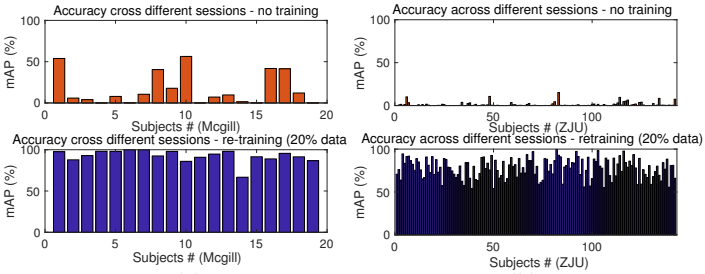
weights transferred from the source model. Note that this implementation is robust against privacy exploits since the private activations are kept on mobile and the transferred features are public. We also evaluate scenarios when different public data are available, by alternating the source data between the other two datasets. This allows us to examine the generality of features and their impact on accuracy and convergence. If the source and target models permits easy domain adaptations, the cloud no longer needs to tightly match the hardware configuration with the user device.

Fig. 7(a) shows the convergence of a random individual from the Mcgill dataset. We can see that feature transfer offers at least two orders of magnitude speed-up in terms of convergence. Features learned from data gathered with different settings offer significant boost as well. For instance, for the loss value to converge to 0.05, the original training takes 325 epochs. With feature transfer, it only takes 2 epochs from the same dataset, 5 and 4 epochs for different IDNet and ZJU datasets, respectively. We then evaluate the speed-up on mobile devices and measure the total computation time to finish 50 epochs of training, as shown in Fig. 7(b). Freezing all the convolutional layers offers 3-5 times of speed-up. If one additional convolutional layer is released, the gain is still over 2 times. The speed-up comes with a little accuracy loss due to the discrepancy among domain features (illustrated in Table 3). Training the dense layers only has 3-5% accuracy loss on Mcgill, IDNet, and 12% on ZJU dataset. The accuracy can be improved by fine-tuning more layers (e.g. to 0.9% and 3.5% for Mcgill and IDNet). Transferring from a different dataset only incurs minor accuracy loss (1-3% on average). This indicates that the proposed architecture is robust to re-use features for the new target domain, though device settings such as sampling frequency (sensors) can be different.



(a) (b)

Fig. 7: Boost from feature transfer (a) speed of convergence; (b) speed-up on mobile devices.



(a) (b)

Fig. 8: Acceptance rate across different sessions (a) Mcgill; (b) ZJU.

6.5 Robustness against Intra-class Variations

We show that incorporation of training on mobile devices offers fast response to intra-class variation when behavioral biometrics evolve. We utilize the Mcgill and ZJU datasets since they record more than two sessions of a subject on different days (Mcgill) and months (ZJU). To see whether the system can still recognize its owner, we examine the acceptance rate. If the acceptance rate is low, the model is likely to reject the genuine user and degrade usability significantly. In the upper figures (*no training*) of Fig. 8, each user trains a model in session 1 and directly tests on the data from session 2. As we observe, the acceptance rate is quite low if the model is not updated. Mcgill dataset across several days only yields 16.3% average acceptance, and the rate drops to 1.1% for ZJU over a longer period. It certainly indicates that pre-trained models cannot adapt to new data distributions.

With continuous model updates, we fine-tune the model from the previous weights with a lower learning rate, and only use 20% of the new data. The bottom figures in Fig. 8 shows the mean acceptance percentage over all fine-tuning epochs, which quickly brings it back to 92.4% and 77.6% for Mcgill and ZJU, respectively. The best acceptance percentage of some users can hit 100% indicating that the fine-tuned model can almost perfectly adapt to the new data.

6.6 Robustness against Random Attacks

A *random attacker* tries to gain system access using his own walking data (gait) or data retrieved from a large database. Since behavioral patterns are extremely difficult to mimic by observation, we use Osaka as the database to launch attacks. These samples are entirely new to the model from unknown data distributions. We train users in the three datasets and enumerate through all the attacking samples (1684 spectrograms) for each user. As shown in Table 4, the

Dataset	All	Batch 4	Batch 8	Batch 16	Batch 32
Mcgill	0.05%	0.003%	0.003%	0.000%	0.000%
IDNet	2.36%	2.18%	2.014%	1.682%	1.024%
ZJU	0.346%	0.028%	0.010%	0.004%	0.001%

TABLE 4: Success ratio of passive attacks using Osaka dataset

success ratio is below 3%. Once the results are fused with 32 samples randomly selected from the training data, the ratio further declines to 1% in the worst case. This rate could be easily reduced to zero by incorporating high-level security mechanisms such as limiting the number of trials.

6.7 Robustness against Active Attacks

Next, we evaluate the system robustness against *active* attacks. Sec. 4.4 has shown that simple noise injection does not work well for obfuscation. In addition to *Gaussian* noise, we further evaluate *Laplacian* and *Uniform* noise with the standard deviation set to the original signal over a finite moving window. Laplacian noise is also used in differential privacy for mathematical tractability. We adopt the three types of noise to evaluate their properties regarding obfuscation and impact on usability. We choose a typical application of pedometer step counter to assess usability in the presence of noise. Fig. 9 shows the attacker's success ratio versus the pedometer error for different noise distributions. We alter the input in three ways. 1) *noise/train*: noise samples are paired with genuine ones in the training set and labeled as negative. 2) *denoise/no train*: attacker applies a state-of-the-art denoise technique called total variation proximity operators [60] on (1). The classifier takes no countermeasure. 3) *denoise/train*: the classifier makes a successful prediction about the denoise scheme and labels the denoise pairs as negative for training.

Fig. 9(a-b) indicate that the proposed mechanism is capable of defending against active attacks when the Siamese Network is supervised to learn the difference from the attack samples (noised or denoised). Learning the noised signals can drop the success rate from 50% to less than 10%. However, without considering possible denoise from the attacker in the classifier, there is still around 20% success rate even when the neural network has learned the noised signals. Once denoise is considered in training, the attacker can no longer succeed. For usability, the noise distributions incur 7-13% error for the step counter. IDNet gathered from a more vibrant environment has higher intra-class deviations. When the standard deviation of noise is set to as large as the original signal, the step counter is subject to a higher error rate.

An anomaly is ZJU in Fig. 9(c), in which body sensors of low sampling rate are used. The additive noise has much higher frequency thus is bound to be filtered out by the neural networks. The error of step counter is almost doubled due to the local peaks of the noisy spikes being mistakenly recognized as gait cycles. Using random noise, we do not see much security improvement but a sharp usability degradation. Instead of noise distributions with high frequency, we further test a sinusoidal wave with a low frequency (identical to the gait signal with a much smaller amplitude). The sine wave is merged into the sensing signal and difficult to extract since the oscillating frequency is kept as a secret. On the other hand, the new frequency components are evident enough to be recognized by the neural network through training. As shown in Fig. 9(c), the attacker's success ratio quickly drops to nearly zero for most of the 136 individuals in ZJU. Our new finding suggests that random noise is not always a good solution to balance security and usability. The actual obfuscation should be considered with respect to the types of data source. For a better balance of security and usability, obfuscation with hidden regularity can be considered as a signature.

6.8 Impacts from Accuracy Requirements and Margin

This subsection evaluates the choice of two important parameters: the accuracy requirements in terms of false rejection/acceptance rate α, β and margin m in the Siamese Network. α, β are inputs from the users. We set them to 0.01 for demonstration. It means that we want the authentication to reach 99% confidence about its decisions based on multiple observations. This is equivalent to $A = B = 99$ in Eq. (8). For Eq. (5), the mean of distance is set to $m/2$ for balanced data and the variance are obtained on the training samples.

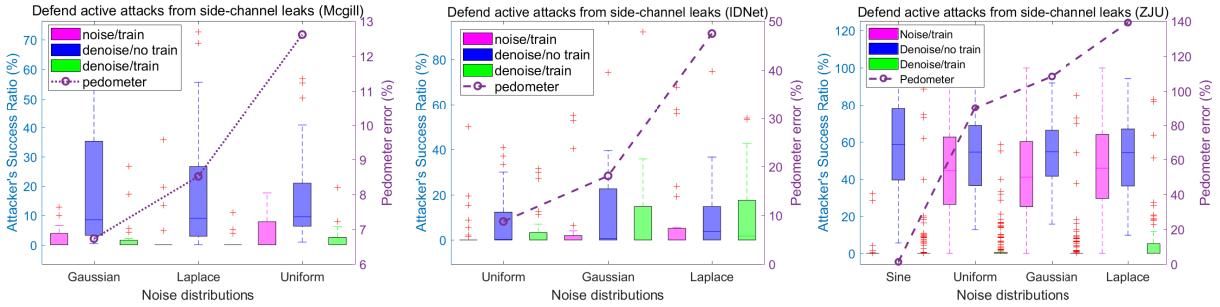
Fig. 10(a) demonstrates the decision-making process. When the likelihood ratio hits the upper shaded area, the decision is to reject; otherwise, the decision is to accept. Normally, 5-6 batch iterations are needed to reach a confident decision. To see more of how it evolves, we select some hard samples and mix them with random samples. Then the classifier is less confident based on the single batch, thus it progresses to the next iteration until a shaded region is hit. The process can be thought as a competition between the decisions to either accept or reject. If a majority of the new data indicates positive, the decision is inclined to accept though a few false ones may drag the curve towards the opposite direction en-route. As we can see, SPRT reduces authentication instability at a little cost of extended response time. Fig. 10(b) shows time durations of making batched inference on mobile devices (from batch sizes of $4-56$). Since less parallel resources are available on the mobile platform (the CPU cores are fully utilized), the inference time increases almost linearly with the input batch size. The computation takes less than 1.5s for all the devices to process a single batch. In normal situations, reaching a confident decision of 5-6 iterations takes about 6s and 1.5s on Nexus 6/6P and Pixel2/Mate10 respectively, which is quite acceptable to run as background processes.

The Siamese Network separates the positive and negative samples apart by *margin* m. In our experiment, where the positive pair distance stays close to 0, $m = 0.5$ maps the negative pair distance to around 2 and $m = 3$ maps it to 4.5. Intuitively, a small margin leads to higher error rate because dissimilar pairs are closer and possibly misclassified as similar pairs, or vice versa. A large margin makes it difficult to train the classifier in terms of slower convergence as shown in Fig. 10(c). For balancing the rates between false acceptance and false rejection, the desirable margin is around $m = 1.5$. We also demonstrate the margin threshold from 0.1 to 3 in Fig. 10(d). If the distance is below the margin threshold, the test pair is similar or dissimilar otherwise. For Mcgill, our framework achieves an equal error rate (EER) around 96.3% when the margin threshold is set to $\frac{1}{2}m$.

6.9 Profile System Overhead

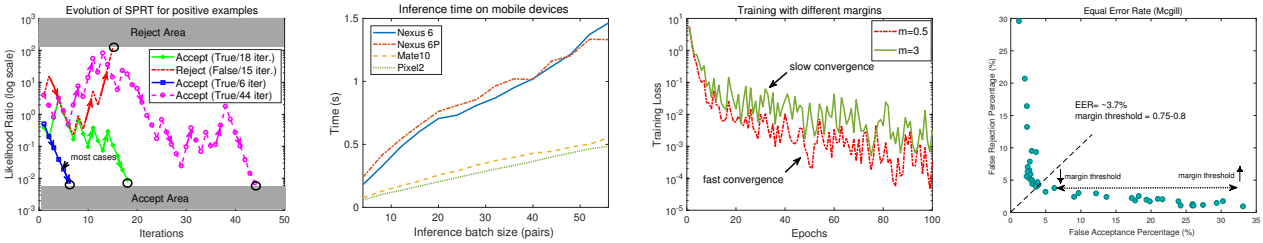
Memory. We use the Android Profiler to measure the memory consumption of the app during training in Fig. 11.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3072608, IEEE Transactions on Mobile Computing



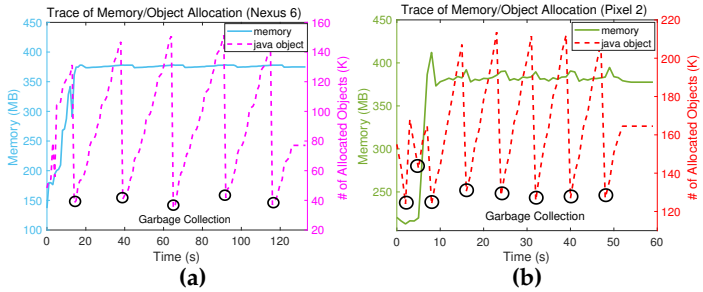
(a) (b) (c)

Fig. 9: Defend active attacks through side-channel leaks (a) Mcgill; (b) IDNet; (c) ZJU.



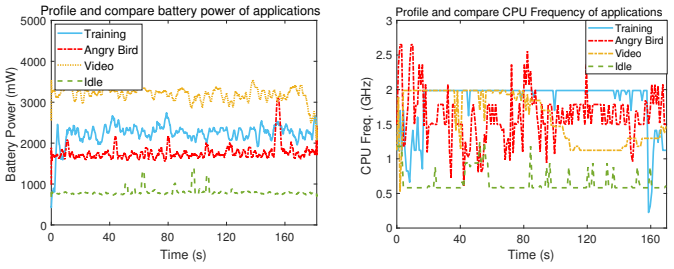
(a) (b) (c) (d)

Fig. 10: System metrics (a) SPRT for positive samples (b) batched inference time on mobile (c) speed of training convergence with different margins (d) equal error rate (Mcgill).



(a) **(b)**

Fig. 11: Trace of memory/object allocation during mobile training (a) Nexus 6; (b) Pixel 2.



(a) (b)

Fig. 12: Profiling battery power and CPU frequency of different applications

To save space, we show the traces of Nexus 6 and Pixel 2 (the oldest and newest of our collection). Nexus 6 has a quad-core of 4×2.7 GHz. Pixel 2 features an octa-core with 4×2.35 GHz plus 4×1.9 GHz CPUs. Once the app starts, it loads the native code, training samples and network model into the mobile memory. Sample paring is conducted on the device at the beginning. Since DL4J is not optimized for the mobile environment, the native/code occupies about 130 MB. When training is initiated, new objects are allocated and once the app approaches the assigned memory limit, a garbage collection is triggered to release the objects, which could pause the app for a minimum amount of time (several ms). When multi-threads are enabled in DL4J with OpenBLAS, the training process enjoys much better performance with an octa-core processor on Pixel 2. Hence, we see a steeper line of object allocation on Pixel2, which completes the training by only half of the time with Nexus 6.

Battery Power and CPU Frequency. We profile the battery power using the Monsoon power monitor [83] and CPU frequency by the Trepn Profiler [84]. We measure the battery power and average CPU frequency of the 4 cores on Nexus 6 while (1) training, (2) playing angry bird, (3) watching an MP4 video in MX player, and (4) idling, in Fig. 12. Training runs at 2.0 GHz set by the default governor and its battery power consumes at the level of 2000 mW, which consumes about 1% total battery during 2.5 mins. Training introduces an additional 28% energy overhead compared to angry bird, but consumes 25% less energy compared to watching a video. The results suggest that training consumes more energy than mobile games but less intensive than watching videos. Since model update is less time-sensitive compared to interactive apps, it can be delegated as a background service and scheduled on-demand while the phone is charging or idling. The default CPU governor can be also adjusted adaptively to optimize performance and power consumption.

7 Discussion

The primary goal of this paper is to meet the application requirements from authentication and explore whether the Android OS with the default settings can accommodate persistent workloads like training on consumer mobile devices (e.g., with the default `interactive` power management policy), in contrast to the conventional mobile workloads that are bursty in nature. Our implementation not only shows

that it is feasible to execute training, but also reasonably fast using the multi-core CPUs (partially because we cannot run ultra deep models under the memory capacity). We keep the power management policy unchanged because the vendor-supplied drivers have heterogenous configurations regarding the CPU frequencies, proprietary task migration between the Big.LITTLE CPU clusters as well as the complex thermal behaviors. In the experiment, we notice some thermal throttling among the older generations (Nexus 6/6P), that the governor actively reduces frequency on the course of training or deactivates CPU cores at the big cluster. It leads to noticeable performance slowdown, but is an active measure to protect the CPU subsystem and battery from overheating.

To conserve energy, one direction on the OS level is to exploit co-running opportunities by scheduling learning with an appropriate foreground process [85], but at non-trivial performance trade-offs to slowdown the background learning process. Optimizations at the architecture level are more effective, such as improving the intra-layer [86], [87], cross-layer data locality [88], [89] and reducing data transfer via zero-value compression [90]. These works are complementary to our research and we expect the designs to be integrated into specialized processors in the mobile architectures to carry out training tasks in the near future.

Other than authentication, the framework also provides the basis to launch federated learning tasks [91], [92], a new computing paradigm for privacy-preserved collaboration among the mobile users, where on-device training is an essential element. To this end, we also expect that the proposed framework can provide a guideline to explore the design space for many federated applications looking forward.

8 Conclusion

This paper incorporates training on mobile devices and tackles the privacy and performance challenges for behavioral authentication. Empowered by deep metric learning, a comprehensive framework is designed to improve discriminative power and tackle side-channel leaks. Our extensive experiments demonstrate the security and robustness of the proposed design against intra-class variations and imposters that are out-of-distributions. We anticipate the presented system would offer insights and opportunities to enhance deep learning on mobile devices.

Acknowledgments

This work was supported in part by the US NSF grant numbers CCF-1850045, CNS-2044841, CNS-2054657 and the State of Virginia Commonwealth Cyber Initiative (cyberinitiative.org).

References

[1] V. Bruggen, D. Liu, S. Kajzer, M. Striegel, A. Crowell and D. D'Arcy, "Modifying smartphone user locking behavior", *ACM SOUPS*, 2013.

[2] Low precision GEMM library, https://github.com/google/gemmlowp

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", *IEEE CVPR*, 2018.

[4] S. Han, H. Mao, W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding", *ICLR*, 2016.

[5] J. Ba and R. Caruana, "Do Deep Nets Really Need to be Deep?", *NIPS*, 2014.

[6] AWS Fargate Pricing, https://aws.amazon.com/fargate/pricing/

[7] W. Jiang, Y. Ma, B. Liu, H. Liu, B. Zhou, J. Zhu, S. Wu and H. Jin, "Layup: Layer-adaptive and Multi-type Intermediate-oriented Memory Optimization for GPU-based CNNs", *ACM Trans. on Architecture and Code Optimization*, vol. 16, no. 4, 2019.

[8] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li S. L Song, Z. Xu and T. Kraska, "SuperNeurons: Dynamic GPU Memory Management for Training Deep Neural Networks," *ACM PPoPP*, 2018.

[9] A. Sarma, H. Jiang, A. Pattnaik, J. Kotra, M. T. Kandemir and C. Das, "Cash: Compiler Assisted Hardware Design for Improving DRAM Energy Efficiency in CNN Inference," *ACM Memsys*, 2019.

[10] Y. Ren, Y. Chen, M. C. Chuah, and J. Yang, "User verification leveraging gait recognition for smartphone enabled mobile healthcare systems," *IEEE TMC*, vol. 14, no. 9, pp. 1961-1974, 2015.

[11] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S. M. Makela and H. A. Ailisto, "Identifying users of portable devices from gait pattern with accelerometers," *IEEE ICASSP*, 2005.

[12] T. Hoang and D. Choi, "Secure and privacy enhanced gait authentication on smart phone", *The Scientific World Journal*, 2014.

[13] W. Xu, G. Lan, Q. Lin, S. Khalifa, N. Bergmann, M. Hassan and W. Hu, "KEH-gait: towards a mobile healthcare user authentication system by kinetic energy harvesting", *NDSS*, 2017.

[14] Y. Li, M. Xie and J. Bian, "SEGAUTH: A segment-based approach to behavioral biometric authentication", *IEEE CNS*, 2016.

[15] N. Zheng, K. Bai, H. Huang and H. Wang, "You are how you touch: user verification on smartphones via tapping behaviors", *IEEE ICNP*, 2014.

[16] P. Negi, P. Sharma, V. S. Jain, B. Bahmani, "K-means++ vs. behavioral biometrics: one loop to rule them all", *NDSS*, 2018.

[17] A. Buriro, B. Crispo, F. DelFrari and K. Wrona, "Hold & sign: a novel behavioral biometrics for smartphone user authentication", *IEEE Security and Privacy Workshops*, 2016.

[18] C. Bo, L. Zhang, T. Jung, J. Han, X. Li and Y. Wang, "Continuous user identification via touch and movement behavioral biometrics", *IEEE IPCCC*, 2014.

[19] Z. Sitova, J. Sedenka, Q. Yang, G. Peng, G. Zhou, P. Gasti and K. Balagani, "HMOG: New behavioral biometric features for continuous authentication of smartphone users", *IEEE Tranactions of Information Forensics and Security*, vol. 11, no. 5, pp. 877-892, May 2016.

[20] A. Mosenia, S. Sur-Kolay, A. Raghunathan and N. Jha, "CABA: Continous authentication based on BioAura", *IEEE Transactions on Computers*, vol. 66, no. 55, pp. 759-772, May 2017.

[21] D. Liu, B. Dong, X. Gao, and H. Wang, "Exploiting eye tracking for smartphone authentication", *ACNS*, 2015.

[22] J. Chauhan, Y. Hu, S. Seneviratne, A. Misra, A. Seneiratne and Y. Lee, "BreathPrint: Breathing Acoustics-based User Authentication", *ACM Mobisys*, 2017.

[23] X. Li, Y. Zhang, I. Marsic, A. Sarcevic and R. Burd, "Deep learning for rfid-based activity recognition", *ACM Sensys*, 2016.

[24] S. Tople, K. Grover, S. Shinde, R. Bhagwan and R. Ramjee, "Privado: Practical and Secure DNN Inference", *arXiv preprint arXiv:1810.00602*, 2018.

[25] Q. Zhang, C. Wang, H. Wu, C. Xin and T. V. Phuong,"GELU-Net: a globally encrypted, locally unencrypted deep neural network for privacy-preserved learning", *IJCAI*, 2018.

[26] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," *ICML*, 2016.

[27] X. Jiang, M. Kim, K. Lauter and Y. Song,"Secure outsourced matrix computation and application to neural networks", *ACM CCS*, 2018.

[28] M. Malekzadeh, R. G. Clegg and H. Haddadi, "Replacement autoencoder: a privacy-preserving algorithm for sensory data analysis," *IoTDI*, 2018.

[29] 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 ASPLOS*, 2017.

[30] Android Permission Overview, https://bit.ly/3atJ8LU, Accessed 8-12-2020.

[31] E. Owusu, J. Han, S. Das, A. Perrig and J. Zhang, "ACCessory: password inference using accelerometers on smartphones", *ACM HotMobile*, 2012.

[32] Y. Michalevsky, D. Boneh and G. Nakibly, "Gyrophone: recognizing speech from gyroscope signals", *USENIX Security*, 2014.

[33] A. Das, N. Borisov, M. Caesar, "Tracking mobile web users through motion sensors: attack and defenses", *NDSS*, 2016.

[34] R. Ning, C. Wang, C. Xin, J. Li and H. Wu, "DeepMag : sniffing mobile apps in magnetic field through deep convolutional neural networks", *IEEE Percom*, 2018.

[35] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *NIPS*, 2012.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *ICLR*, 2014.

[37] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE CVPR*, 2016.

[38] W. Liu, Y. Wen, Z. Yu and M. Yang, "Large-Margin Softmax Loss for Convolutional Neural Networks," *ICML*, 2016.

[39] Y. Wen, K. Zhang, Z. Li and Y. Qiao, "A Discriminative Feature Learning Approach for Deep Face Recognition", *ECCV*, 2016.

[40] F Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering", *IEEE CVPR*, 2015.

[41] P. Wohlhart, and V. Lepetit, "Learning descriptors for object recognition and 3d pose estimation", *IEEE CVPR*, 2015.

[42] H. Shi, Y. Yang, X. Zhu, S. Liao, Z. Lei, W. Zheng and S. Li, "Embedding deep metric for person re-identification: A study against large variations," *ECCV*, 2016.

[43] E. Hoffer and N. Ailon,"Deep Metric Learning using Triplet Network", *ICLR Workshop*, 2015.

[44] W. Chen, X. Chen, J. Zhang and K. Huang, "Beyond Triplet Loss: A Deep Quadruplet Network for Person Re-identification," *IEEE CVPR*, 2017.

[45] S. Chopra, R. Hadsell and Y. LeCun, "Learning a similarity metric discriminatively with application to face verification", *IEEE CVPR*, 2005.

[46] G. Koch, R. Zemel, R. Salakhutdinov, "Siamese neural networks for one-shot image recognition", *ICML Deep Learning Workshop*, 2015.

[47] R. Shokri, V. Shmatikov, "Privacy-preserved deep learning," *ACM CCS*, 2015.

[48] X. Zeng X, K. Cao, M. Zhang,"MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images", *ACM Mobisys*, 2017.

[49] A. Mathur, N. Lane, D. Bhattacharya, S. Boran, A. Forlivesi, C. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware", *ACM Mobisys*, 2017.

[50] H. Khan, A. Atwater and U. Hengartner, "Itus: an implicit authentication framework for Android", *ACM Mobicom*, 2014.

[51] M. Gadaleta and M. Rossi, "IDNet: Smartphone-based gait recognition with convolutional neural networks", *Elsevier Pattern Recognition*, vol. 74, pp. 25-37, 2018.

[52] N. Neverova, C. Wolf, G, Lacey, L. Fridman, D. Chandra, B. Barbello and G. Taylor, "Learning Human Identity From Motion Patterns", *IEEE Access*, vol. 4, pp. 1810-1820, 2016.

[53] G. Hinton et. al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups", *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012.

[54] H. Liu, S. Saroiu, A. Wolman, H. Raj, "Software abstractions for trusted sensors", *ACM Mobisys*, 2012.

[55] A. Serwadda and V. V. Phoha, "When kids' toys breach mobile phone security", *ACM CCS*, 2013.

[56] P. Shrestha, M. Mohamed and N. Saxena, "Slogger: smashing motion-based touchstroke logging with transparent system noise", *ACM WiSec*, 2016.

[57] J. Vitter, "Random sampling with a reservoir", *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37-57, 1985.

[58] A. Wald, "Sequential tests of statistical hypotheses," *Annals of Mathematical Statistics*, vol. 16, no. 2, pp. 117-186.

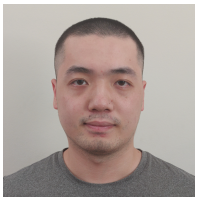
[59] R. Shwartz-Ziv, N. Tishby, "Opening the black box of deep neural networks via information", *arXiv preprint arXiv*, 1703.00810, 2017.

[60] Total Variation proximity operator, https://github.com/albarji/proxTV

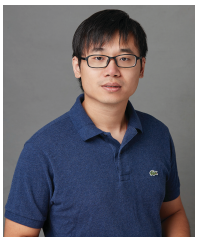
[61] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, "Understanding deep learning requires re-thinking generalization", *ICLR*, 2018.

[62] A. Madry, A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," *ICLR*, 2018.
[63] A. Kurakin, I. Goodfellow and S. Bengio, "Adversarial Examples in the Physical World", *ICLR Workshop*, 2016.
[64] K. Hornik, "Neural Networks", vol. 4, no. 2, pp. 251–257, 1991.
[65] J. Yosinski and J. Clune and Y. Bengio and H. Lipson, "How transferable are features in deep neural networks?", *NIPS*, 2014.
[66] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them", *IEEE CVPR*, 2015.
[67] B. Hitaj, G. Ateniese and F. Perez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning", *ACM CCS*, 2017.
[68] Zlib Compressed Data Format, "https://tools.ietf.org/html/rfc1950," 1996.
[69] L. Villa, M. Zhang and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction," *ACM/IEEE MICRO*, 2000.
[70] G. Georgiadis, "Accelerating Convolutional Neural Networks via Activation Map Compression," *IEEE CVPR*, 2019.
[71] D. Drain and A. M. Gough, "Applications of the upside-down normal loss function", *IEEE Transactions on Semiconductor Manufacturing*, vol. 9, no. 1, pp. 143-145, Feb. 1996.
[72] L. Maaten and G. Hinton, "Visualizing data using t-SNE", *Journal of machine learning research*, pp. 2579-2605, 2008.
[73] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", *in Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.
[74] Tensorflow Lite, https://www.tensorflow.org/lite/
[75] Caffe2 for iOS/Android, https://caffe2.ai/docs/mobile-integration.html
[76] MXNet for smart devices, https://mxnet.incubator.apache.org/faq/smart_device.html
[77] Deep Learning for Java, https://deeplearning4j.org
[78] McGill Dataset, http://jwf.github.io/Humansense-Android-App
[79] IDNet Dataset, http://signet.dei.unipd.it/human-sensing
[80] Y. Zhang, G. Pan, K. Jia, M. Lu, Y. Wang, Z. Wu, "Accelerometer-based gait recognition by sparse representation of signature points with clusters," *IEEE Transactions on Cybernetics*, vol. 45, no. 9, pp. 1864-1875, Sept. 2015.
[81] T. Ngo, Y. Makihara, H. Nagahara, Y. Mukaigawa and Y. Yagi, "The largest inertial sensor-based gait database and performance evaluation of gait-based personal authentication," *Pattern Recognition*, vol.47, no. 1, pp. 222-231, 2014.
[82] Y. Wen, K. Zhang, Z. Li and Y. Qiao, "A discriminative feature learning approach for deep face recognition", *European Conference on Computer Vision*, Springer, 2016.
[83] Monsoon Power Monitor, https://www.msoon.com/online-store
[84] Trepn Power Profiler, https://developer.qualcomm.com/software/trepn-power-profiler
[85] M. Zhu and K. Shen, "Energy Discounted Computing on Multicore Smartphones," *USENIX ATC*, 2016.
[86] T. Chen, Z. Du, N. Sun. J. Wang, C. Wu, Y. Chen, O. Temam, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning", *ASPLOS*, 2014.
[87] Y. Chen, J. Emer and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016.
[88] M. Alwani, H. Chen, M. Ferdman and P. Milder, "Fused-Laer CNN Accelerators", *IEEE/ACM MICRO*, 2016.
[89] B. Akin, Z. Chishti and A. Alameldeen, "ZComp: Reducing DNN Cross-Layer Memory Footprint Using Vector Extensions", *ACM/IEEE MICRO*, 2019.
[90] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, 2018.
[91] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. Arcas, "Communication-efficient learning of deep networks from decentralized data", *AISTATS*, 2017.
[92] C. Wang, X. Wei and P. Zhou, "Optimize Scheduling of Federated Learning on Battery-powered Mobile Devices," *IEEE IPDPS*, 2020.

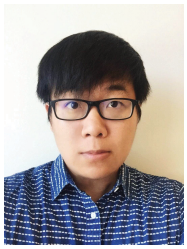
Cong Wang received the B. Eng degree in Information Engineering from the Chinese University of Hong Kong in 2008, M.S. degree in Electrical Engineering from Columbia University in 2009, and Ph.D. in Computer and Electrical Engineering from at Stony Brook University, NY, in 2016. He is currently an Assistant Professor at the Computer Science department, Old Dominion University, Norfolk, VA. His research focuses on addressing security and privacy challenges in Mobile, Cloud Computing, IoT, Machine Learning and System. He is the recipient of IEEE PERCOM Mark Weiser Best Paper Award in 2018, Commonwealth Cyber Initiative Research and Innovation Award, ODU Richard Cheng Innovative Research Award in 2020 and NSF CAREER award in 2021.



Yanru Xiao received the B.S. degree in Computer Science from Central South University, China, 2017. He is currently pursuing PhD of Computer Science at Old Dominion University, VA. His research interests include artificial intelligence and security.



Xing Gao received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2018. He is an Assistant Professor in the Department of Computer and Information Sciences at the University of Delaware, Newark, DE, USA. His research interests include security, cloud computing, and mobile computing.



Li Li is currently an Assistant Professor in ShenZhen Institutues of Advanced Technology, Chinese Academy of Sciences. He received his Ph.D. in Electrical and Computer Engineering from Ohio State University, Columbus, OH, USA in 2018. He received the M.S. degree in electrical and computer engineering from Ohio State University, Columbus, OH, USA, in 2014, and the B.S. degree in electrical engineering from Tianjin University, Tianjin, in 2011. His research interests include mobile computing and cloud computing.

Jun Wang received the PhD degree in computer science from McGill University. He is a senior researcher at Futurewei Technologies, in Santa Clara, CA, where he has been leading research projects on mobile computing, machine learning, and so on. His research interests include machine learning, mobile computing, compiler, computer architecture, among others. He is a member of the IEEE Computer Society.