Privacy-Enhancing Fall Detection from Remote Sensor Data Using Multi-Party Computation

Pradip Mainali*
pradip.mainali@onespan.com
OneSpan
Brussels, Belgium

Carlton Shepherd*
carlton.shepherd@onespan.com
OneSpan
Cambridge, United Kingdom

ABSTRACT

Motion-based fall detection systems are concerned with detecting falls from vulnerable users, which is typically performed by classifying measurements from a body-worn inertial measurement unit (IMU) using machine learning. Such systems, however, necessitate the collection of high-resolution measurements that may violate users' privacy, such as revealing their gait, activities of daily living (ADLs), and relative position using dead reckoning. In this paper, we investigate the application of multi-party computation (MPC) to IMU-based fall detection for protecting device measurement confidentiality. Our system is evaluated in a cloud-based setting that precludes parties from learning the underlying data using multiple, disparate cloud instances deployed in three geographical configurations. Using a publicly-available dataset, we demonstrate that MPC-based fall detection from IMU measurements is practical while achieving state-of-the-art error rates. In the best case, our system executes in 365.2 milliseconds, which falls well within the required time window for on-device data acquisition (750ms).

CCS CONCEPTS

• **Security and privacy** → **Security services**; *Distributed systems security*; *Domain-specific security and privacy architectures*; • **Computer systems organization** → Cloud computing.

KEYWORDS

Fall detection, multi-party computation (MPC), Internet of Things (IoT), cloud computing

ACM Reference Format:
Pradip Mainali and Carlton Shepherd. 2019. Privacy-Enhancing Fall Detection from Remote Sensor Data Using Multi-Party Computation. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3339252.3340500

*Both authors contributed equally to this research.

ARES '19, August 26–29, 2019, Canterbury, United Kingdom

ACM ISBN 978-1-4503-7164-3/19/08…$15.00
https://doi.org/10.1145/3339252.3340500

1 INTRODUCTION

Falls are the leading cause of fatal injury and the most common cause of nonfatal, trauma-related hospital admissions among older adults, with 20–30% of mild-to-severe injuries and 40% of injury-related fatalities [34]. The annual medical costs attributable to fatal and nonfatal falls was estimated at $50bn (USD) in 2015 [16]. Fall detection systems are concerned with automatically detecting the occurrence of falls within the homes of elderly and disabled users, with the aim of minimising response times to potential injuries and facilitating independent living [33]. These systems involve the collection of data, such as accelerometer data from a body-worn device [1, 14, 25, 27, 31, 38, 47], video imagery from cameras in the home [15, 24, 30, 48], and proximity sensors in ceilings [43]. This data is typically classified with respect to a dataset of known fall and non-fall data using machine learning.

The deployment challenges surrounding fall detection systems have, to date, remained largely out-of-scope in current work, which implicitly assume a local deployment or focus solely on classification performance using off-line analyses. Recent work has touted the benefits of a cloud-based, software-as-a-service (SaaS) approach for mobility-based sensing systems, where applications and their underlying platform can be updated, upgraded, monitored and secured with greater flexibility [13, 17, 20]. This is important in managing large numbers of remote devices, where patch penetration remains a significant challenge [35]. Indeed, failing to update devices could lead to potentially unsafe misclassification errors due to accuracy discrepancies between implementations.

This principle of *algorithm agility* provides greater flexibility in changing the underlying classification algorithm in a cloud-based scenario without potential penetration issues of patching and updating large numbers of devices in the field. Another benefit lies in the separation of the platform development, e.g. device hardware, firmware and software, from the algorithm implementation itself. In this paradigm, individual device OEMs[1] are relinquished from the responsibility of implementing and maintaining the detection algorithm. Rather, this can be performed by a dedicated third-party with the potential of supporting devices from multiple OEMs in a service-oriented architecture, which we focus upon in this work.

A myriad of security and privacy concerns remain, however, regarding large volumes of data being harvested, used or, indeed, misused by SaaS providers [49]. At worst, sensor measurements collected from users' devices could be exploited to infer their activity patterns and whereabouts, with major privacy implications. This is compounded by the risks of disclosing plaintext measurements, say, after the exploitation of a vulnerable service, such as insecure

[1] OEM: Original Equipment Manufacturer.

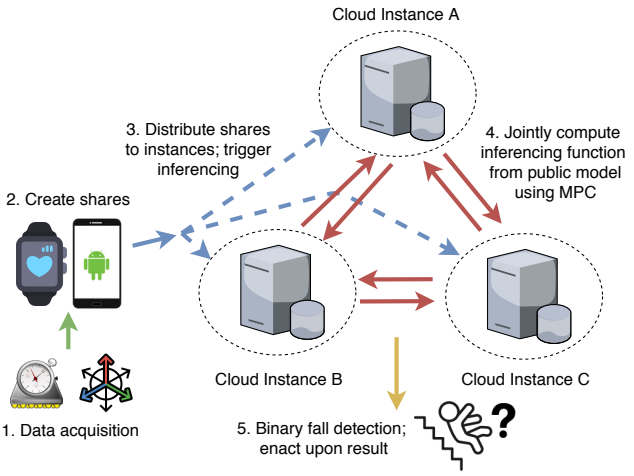

Figure 1: High-level workflow for MPC-based fall detection.

AWS S3 buckets [7]. A large corpus of work has already demonstrated identifying users based on their gait from accelerometer and other inertial measurement unit (IMU) data [18, 19, 22, 29, 44]; their position using dead reckoning [23, 37, 46]; and determining activities of daily living (ADLs), like whether the user is walking, sleeping, and sitting [21, 41]. Existing work on privacy-enhancing fall detection has focussed on non-cryptographic techniques for video-based methods, including image blurring, silhouetting and foreground extraction. However, these still reveal significant information about users, such as their presence. Moreover, they do not generalise to the significant proportion of fall detection proposals using time-series IMU measurements from personal devices, such as smartphones and smartwatches [1, 5, 31, 38, 45].

A promising solution is multi-party computation (MPC), where measurements are split using a linear secret sharing scheme (LSSS) and distributed to mutually distrusting parties. After this, functions can be jointly computed without parties disclosing their plaintext inputs to others. Existing literature work has already shown that machine learning inferencing, upon which many fall detection systems rely, can be performed efficiently using MPC. This includes MPC-based image recognition using support vector machines (SVMs) [28] and remote rehabilitation treatment classification from patient data using decision trees [11].

In this work, we present the implementation and evaluation of the first system for perform MPC-based fall detection from IMU device data in order to protect measurement confidentiality. MPC is used to perform the requisite feature extraction and classification without the parties learning the contents of IMU measurements received from the device. We evaluate the system, illustrated in Figure 1, using an off-the-shelf device and three MPC parties deployed using a public cloud service. In short, this work presents the following contributions:

- The first framework for performing privacy-enhancing fall detection from device IMU measurements using MPC. The proposal preserves measurement confidentiality in a cloud-based model while performing feature extraction and classification using machine learning.
- A two-part evaluation with experiments conducted using a publicly-available fall detection dataset. Our inferencing framework executes faster than the time necessary for on-device measurement acquisition, and achieves error rates commensurate with state-of-the-art schemes.

1.1 Document Structure

This paper proceeds with a general review of fall detection systems in Section 2, which is followed by a discussion of the privacy challenges associated with existing systems. In Section 3, we present the assumptions and threat model in greater detail, before introducing and motivating the use of MPC as a solution to the aforementioned challenges. This section also develops the design of the proposed framework after which, in Section 4, the implementation details are discussed. Section 5 presents an evaluation of the proposed framework, comprising a discussion of the methodology and experimental results in the form of classification and computational performance. Lastly, we conclude our work in Section 6, including a discussion of future research directions.

2 FALL DETECTION

Fall detection systems fall broadly into two categories: 1), those using devices situated in the user's ambient environment; and, 2), those using body-worn devices (BWDs), e.g. smartwatches. We now describe prominent existing schemes and their operation. This section is not an exhaustive analysis—for this, the reader is referred to Mubashir et al. [33] and Delahoz et al. [12]—but, rather, we aim to summarise the salient approaches and results.

2.1 Ambient Device-based Systems

Privacy-enhancing fall detection has focused almost entirely on non-cryptographic techniques from ambient-based devices, such as silhouetting, blurring and foreground masking from camera data. Mastorakis and Makris [30] and Zhang et al. [48] employ RGBD data from a Microsoft Kinect to detect five ADLs, using only the depth values and foreground mask of the RGB data, thus occluding users' facial features. Tao et al. [43] study the use of a ceiling-based sensor network comprising infrared cameras for fall detection in a home environment. The sensors output binary values to indicate the existence of persons underneath, which are used to model a map of the user's home. Edgcomb and Vahid [15] explore four techniques for privacy-enhancing video-based fall detection: *blurring* (of the user), *silhouetting*, and covering the user with a graphical *box* and *oval*. Raw video imagery is first taken from a stationary in-home camera, before applying one of these techniques.

2.2 BWD-based Systems

The use of ambient-based devices has inherent issues associated with detecting falls in occluded areas: fitting multiple cameras to cover all possible locations in the user's living environment—bathroom, bedroom, kitchen, and so on—may impose significant costs relating to unit installation and maintenance. Another paradigm employs devices likely to be already owned and carried by the user, such as a smartwatch or smartphone. Unlike the previous section, we are unaware of proposals that investigate privacy-enhancing fall detection from BWDs to the best of our knowledge.

As such, we provide a general review of highly-cited and state-of-the-art work from the literature.

Mauldin et al. [31] present SmartFall, which uses accelerometer data sampled at 31.25Hz from a user-worn smartwatch to detect user falls. Smartwatch data is streamed to the user's smartphone that extracts features pertaining to the vector magnitude of the accelerometer data and the difference between the maximum and minimum magnitudes over a 750ms sliding window (Equations 1 and 2). Naïve Bayes, Support Vector Machine (SVM) and Recurrent Neural Network (RNN) classifiers are evaluated using labelled data from seven volunteers between 21–55 years old, alongside the Farseeing dataset [32] with data from 2,000 participants. Results of 0.37–0.79 precision, 0.55–1.0 recall, and 0.65–0.99 accuracy are reported, depending on the chosen dataset and classifier.

$$|\vec{v}| = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

$$\Delta s = \max(|\vec{v}_i|) - \min(|\vec{v}_j|), i \neq j \tag{2}$$

Liu and Cheng [25] investigate IMUs placed at users' waists that collects accelerometer measurements at 200Hz. The vector magnitude is computed, along with the fast changed vector, vertical acceleration and posture angle between the vertical acceleration and gravity over a 0.1s sliding window. The features are inputted to an SVM with labelled fall data from 15 volunteers who performed 10 simulated falls and 11 ADLs repeated 10 times. The proposal yields error rates of 98.38% accuracy, 97.40% recall, and 99.27% precision.

Santoyo-Ramón et al. [38] investigate IMUs attached to the user's chest, waist, wrist and thigh—the data from which is aggregated into 15 second segments by a smartphone. A sliding window of 0.5s is used in which the fast changed vector of accelerometer measurements is computed alongside the magnitudes' mean, standard deviation and mean absolute distance, and the mean rotation angle and mean module of the (Y, Z) and (X, Z) components. The features are inputted to SVM, k-Nearest Neighbour, Naïve Bayes and Decision Tree classifiers, and evaluated using data from 19 subjects who performed 746 ADLs and falls. The geometric mean of the precision and recall is used as the performance metric, with results of 0.614–0.999 depending on the algorithm and sensor combination.

Doukas et al. [14] investigate the use of an accelerometer attached to the user's foot for fall detection. The sensor data is transmitted to a receiving device, such as a laptop, where it is normalised and inputted to an SVM classifier in raw form after applying a Kalman filter. The evaluation reports a classification accuracy of 98.2% using a limited set of labelled data from two volunteers.

Cao et al. [5] present E-FallD, which uses accelerometer data from an Android smartphone. The vector magnitude of the accelerometer components is computed and a global threshold is fixed for identifying falls based on the phone's measured acceleration. The work also investigates multiple thresholds customised to the user's gender, age, and body mass index (BMI). The authors recruit 20 participants of varying gender, age and BMI, who perform a total of 400 falls and 1200 ADLs. The approach yields results of 86.75% precision and 85.5% recall using global thresholds, and 92.75% precision and 86.75% recall with customised thresholds.

Similarly, Vilarinho et al. [45] use accelerometer data from an Android smartphone and smartwatch. Here, a rule-based system is devised for threshold-based detection using the vector magnitude, fall index, and absolute vertical acceleration as features over a 0.8s sliding window. The approach is evaluated using data from three participants, who performed 12 falls and 7 ADLs each, with a reported 0.68 classification accuracy, 0.78 precision, and 0.63 recall.

Yavuz et al. [47] propose another system based on accelerometer values from an Android smartphone. In contrast, the authors use solely frequency domain features, namely the wavelet coefficients after applying a discrete wavelet transform. A threshold-based approach is used to test the coefficients generated from the phone against those of previously observed falls. 100 fall sequences from five volunteers were collected to evaluate the system, with results of 46-95% precision and 88-90% recall.

Albert et al. [1] use accelerometer values from an LG G1 Android smartphone located at the user's pelvic region. 178 features across several categories are subsequently extracted: moments, e.g. kurtosis and skew; moments between successive samples; smoothed root mean squares; extremities, e.g. min and max; histogram values; Fourier components; mean acceleration magnitude; and cross products. The features are inputted to five classifiers—SVM, Logistic Regression, Naïve Bayes, Decision Tree and kNN—and evaluated using data from 18 simulated falls from 15 subjects, resulting in 98% classification accuracy.

Musci et al. [34] evaluate the use of recurrent neural networks (RNNs) and long short-term memory units (LSTMs), and propose a deep learning architecture for ternary fall detection for whether or not a fall has occurred, and whether one is about to occur. The proposal is evaluated using the SisFall dataset from [42], comprising accelerometer and gyroscope measurements from a smartphone and a custom IMU attached to the waists of 38 users; the dataset is labelled with respect to the three aforementioned states. A confusion matrix of the three classes reports 92.86–97.16% classification accuracy following hyperparameter optimisation.

Luštrek and Kaluža [27] explore the use of a body area network (BAN) comprising 12 IMUs at the user's shoulders, wrists, hips, knees and ankles. Features are collected regarding the coordinates of units in a body coordinate system, their velocities, absolute distance, and relative angles. The features are used as input to seven machine learning algorithms—SVM, DT, kNN, NB, Random Forest, Adaboost, and bagging—implemented in the Weka library. Three participants were recruited who performed a total of 45 activities: 15 falls and 10 lying down, sitting down and walking ADLs each. 73.4–96.3% classification accuracy is reported depending on the algorithm used, with SVM yielding the best case.

2.3 Discussion

Generally, fall detection systems rely upon traditional supervised learning algorithms and, more recently, deep neural networks (LSTMs and RNNs), trained over labelled fall data collected *a priori*. The majority of existing work models binary classification, i.e. 'fall' or 'not fall', while other work also models whether a fall *may* be about to occur and classification of general ADLs, such as sitting, walking, and lying down.

Additionally, the generation of *global* models that generalise to all users, rather than individuals, is also a common facet of current work. For BWD-based systems, accelerometer measurements are

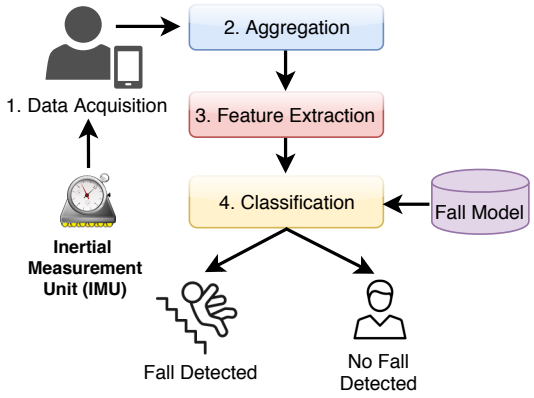

Figure 2: Overview of IMU-based binary fall detection.

used ubiquitously from which a variety of features are extracted prior to classification, which is illustrated in Figure 2 for the binary case. Most work focuses on time-domain features, such as the vector magnitude, arithmetic mean, standard deviation, and maximum and minimum values. The choice of algorithm remains varied, although SVMs tend to exhibit the best performance with 0.9838 [25], 0.982 [14] and 0.73–0.99 [31] accuracy. Recent work [31, 34] has begun to explore the application of deep learning using LSTMs and RNNs with promising results—0.9286–0.9716 [34] and 0.85–0.99 [31] accuracy—but work in the area remains limited.

Rather than using machine learning for the discovery of optimal decision boundaries, another widely-used approach has been the use of manually set threshold values, which are discovered experimentally. Proponents of this approach have touted faster classification as an advantage over machine learning-based systems, and thus more suitable for constrained devices; however, such approaches generally yield higher error rates and are evaluated over limited datasets, e.g. with five [47] and three [45] participants. We present a summary of BWD-based systems in Table 1.

3 PRIVACY-ENHANCING BWD-BASED FALL DETECTION

To the best of our knowledge, privacy-enhancing BWD-based fall detection is yet to be investigated. Accelerometers, used in the bulk of current proposals, measure fine-grained movements at high frequency rates (typically >30Hz) on commodity devices, such as smartphones and smartwatches. A wealth of literature exists surrounding the identification of users based on their gait using accelerometer measurements alone from such devices [18, 19, 22, 29, 44]. Furthermore, we are also aware of IMU measurements being used for recognising non-fall ADLs [21, 41] and performing user position tracking using dead reckoning [23, 37, 46], which could be exploited to violate user privacy.

3.1 Applying Multi-Party Computation (MPC)

We observe that MPC can be applied to acquire data, extract features, and perform fall classification in a way that preserves IMU measurement confidentiality. In this work, we tackle the base case of MPC using arithmetic secret sharing from Shamir's secret sharing scheme (SSSS) [40], which is briefly described as follows.

Table 1: Comparison of BWD-based fall detection systems.

Proposals	Algorithms	Error Rates
SmartFall [31]	NB, SVM, RNN	0.37–0.79 (P), 0.55–1.0 (R), 0.65–0.99 (A)
Liu & Cheng [25]	SVM	0.9927 (P), 0.9740 (R), 0.9838 (A)
S-R et al. [38]	SVM, kNN, NB, DT	0.614–0.999 (GMPR)
Doukas et al. [14]	SVM	0.982 (A)
E-FallD [5]	Thresholds	*GT*: 0.8675 (P), 0.855 (R) *CT*: 0.9275 (P), 0.8675 (R)
Vilarinho et al. [45]	Thresholds	0.78 (P), 0.63 (R), 0.68 (A)
Yavuz et al. [47]	Thresholds	0.88–0.90 (P), 0.46–0.95 (R)
Albert et al. [1]	SVM, LR, NB, DT, kNN	0.982 (A)
Musci et al. [34]	RNN, LSTM	0.9286–0.9716 (A)
Luštrek & Kaluža [27]	SVM, DT, kNN, NB, RF, Adaboost, Bagging	0.732–0.963 (A)

P: Precision, R: Recall, A: Classification accuracy, GMPR: Geometric mean of precision and recall.

3.1.1 Shamir's Secret Sharing Scheme. SSSS divides a secret $\mathcal{S}$ into n shares, $\mathbf{s} = \{s_1, s_2, \ldots, s_n\}$, such that knowing t shares from $\mathbf{s}$ makes $\mathcal{S}$ easy to compute, but knowing fewer shares than t renders $\mathcal{S}$ undetermined. The scheme relies on the fact that t points are required to construct a polynomial of degree $t-1$, which is defined in the field $\mathbb{F}_P$. Next, $t-1$ positive integers, $a_1, \ldots, a_{t-1}$ are chosen with $a_i < P$, with the secret assigned as $a_0 = \mathcal{S}$. The polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{t-1} x^{t-1}$ is built with which n points are constructed from $i = 1, \ldots, n$, to retrieve pairs $(i, f(i))$. Any subset with t distinct pairs can be used to recover the constant term, a_0, i.e. the secret, which is found using Lagrange interpolation.

3.1.2 MPC from Arithmetic Secret Sharing. An important observation is that SSSS allows each party to locally compute linear combinations of secrets and public values. Firstly, let us define $[x]_i$ as the i^{th} share of data item, x. SSSS allows parties to locally compute the following arithmetic operations over their shares [6]:

- *Addition* ($[c] \leftarrow [a] + [b]$): Each party p_i can compute a new share, $[c]_i$, that is the addition of two other shares, i.e. $[c]_i = [a]_i + [b]_i$.
- *Addition of a secret and public value* ($[c] \leftarrow [a] + \alpha, \alpha \in \mathbb{F}_P$): Party p_i can compute a new share from the addition of a held share and a public value $\alpha \in \mathbb{F}_P$; that is, $[c]_i = [a]_i + \alpha$.
- *Multiplication of a secret and public value* ($[c] \leftarrow [a] \cdot \alpha, \alpha \in \mathbb{F}_P$): p_i can compute a new share of the multiplication of a held share with a public value, $[c]_i = [a]_i \cdot \alpha$.

Table 2: Communication cost for MPC functions using arithmetic secret sharing [6].

Function	Communication Cost (in rounds)
Multiplication	1
Inner Product	1
Inverse	1
Random element	1
Random bit	3
Equality (=)	2
$\geq$	$2m + 4$
$\leq$	$2m + 4$
AND, OR	$2\log(k)$

Where k is the number of operands and m is the bound $x, y \in [0, 2^m - 1]$ wherein operands x and y lie.

Computing a new share that is the multiplication of two others, $[c] = [a] \cdot [b]$, is less straightforward: given that SSSS uses polynomials of degree t, then $[a \cdot b] \neq [a]_i \cdot [b]_i$, as the resulting polynomial has degree $2t$. A method for computing t-degree polynomials representing $[c] \leftarrow [a] \cdot [b]$ was presented in [4]. As opposed to entirely local computation for addition of shares and the addition and multiplication of a share with a constant value, multiplying shares necessitates one round of communication. While many functions can be computed using only these operations, they preclude integer comparison, e.g. =, ≠, <, and >, and Boolean operations, e.g. AND and OR. The reader is referred to [6] for a comprehensive description and analysis of these protocols; the differences in asymptotic communication complexity are reproduced from [6] in Table 2.

3.2 Assumptions and Threat Model

An outsourcing model is assumed whereby a body-worn device streams IMU measurements to a cloud-based service that performs the feature extraction and classification. The focus is on binary classification for inferring the occurrence of falls using a globally-trained model in line with the bulk of existing work from Section 2. We divide the entities involved into two categories.

3.2.1 Source device. We assume the existence of a body-worn device—a smartphone in the user's pocket, smartwatch, a sensor placed on the user's belt, or otherwise—with an IMU. The device is also assumed to possess the ability to split measurements into shares using SSSS and stream them to an Internet-facing service over a secure channel, such as TLS using a pre-installed certificate. In this work, the IMU measurements are considered to be trusted; the focus is on the unauthorised disclosure of measurements after departing the device over a network to a cloud-based service.

3.2.2 Cloud service. Comprises the software components—database instance, REST APIs, and so on—for receiving, storing and classifying falls from IMU sensor data on a public cloud platform. The primary threat is the exfiltration and disclosure of raw IMU measurements from a remote attacker, such as from an unsecured, public-facing database, e.g. AWS S3 bucket, which could be used to model and infer users' general ADLs, position, or gait for user identification. This work tackles the base case for MPC where parties operate under an honest-but-curious model, whereby the protocol specification is executed as intended. We also assume that the output, i.e. whether or not a fall occurred, is learned by the cloud services in order to contact an emergency number or provide alternative remediation. We note that *some* information may be revealed by the communication between the device and the cloud service, such as the absence of receiving protected IMU data at certain times of day, e.g. at night where the user deactivates their smartphone or smartwatch before sleeping. However, this is orthogonal to the ultimate goal of this work of preventing either the service or an external adversary from learning additional information from IMU measurements beyond that intended for detecting falls.

3.3 Workflow

The principle stages of the proposed system were illustrated previously in Figure 1 and described as follows:

(1) *Data acquisition.* The collection of tri-axial IMU data on the device, e.g. smartphone, at an appropriate sampling frequency. This may be unimodal input, i.e. *only* accelerometer [5, 14, 25, 31, 47]; multi-modal, e.g. accelerometer and gyroscope together [38]; or aggregating measurements from multiple sensors in a body-area network [27].

(2) *Secret sharing.* Given a tri-axial IMU sample, $[A_x, A_y, A_z]$, the master device splits each component into n shares using SSSS. In this work, we set n equal to the number of authorised communicating parties, while the reconstruction threshold is fixed at $t = 3$. In short, IMU samples are split as follows:

$$\mathbf{D} = \begin{pmatrix} [A_x]_0 & [A_x]_1 & [A_x]_2 & \cdots & [A_x]_n \\ [A_y]_0 & [A_y]_1 & [A_y]_2 & \cdots & [A_y]_n \\ [A_z]_0 & [A_z]_1 & [A_z]_2 & \cdots & [A_z]_n \end{pmatrix}$$

Where the rows comprise the individual shares of a particular component, while each column represents the i^{th} share of all components.

(3) *Share distribution.* The device individually transmits the column elements from $\mathbf{D}$ to each cloud service instance; that is, $[A_x]_0$, $[A_y]_0$ and $[A_z]_0$, is sent to party P_0, $[A_x]_1$, $[A_y]_1$ and $[A_z]_1$ to P_1, and so on. By sharing each sample independently, no single cloud-based party may recover the underlying IMU samples assuming the reconstruction threshold above.

(4) *MPC-based inferencing.* The parties use arithmetic operations upon the received shares to perform fall detection feature extraction, such as computing the vector magnitude of IMU samples, and subsequent classification using a desired machine learning classifiers, e.g. support vector machine (SVM), logistic regression (LR), and naïve Bayes (NB).

(5) *Classification decision.* The resulting label, $\mathbf{L} \in \{0, 1\}$, is used to denote the occurrence of a fall in a binary fashion. This value may be used to raise an alarm or telephone an emergency contact number.

4 IMPLEMENTATION

This section describes the implementation of the device- and cloud-side systems introduced in the previous section.

Table 3: MPC server location configurations using AWS.

Config	Party 1	Party 2	Party 3
A	US_Oregon	Canada_Central	US_Ohio
B	US_Oregon	Canada_Central	Europe_London
C	US_Oregon	Europe_London	Asia_Singapore

4.1 Device Application

A Python application was developed that reads CSV IMU data line-by-line from file and creates shares of each sensor modality component using SSSS. The resulting shares are base 64-encoded and JSON-formatted, and transmitted over TLS to their respective cloud parties using the method described in Section 3.3. The application was deployed on a Raspberry Pi 3 Model B, with a Broadcom BCM2837 system-on-chip with a quad-core ARM Cortex-A53 at 1.2GHz, 1GB RAM, and Broadcom BCM43438 WiFi module (802.11N, 150 Mbit/s at 2.4GHz). The board was connected to a corporate WiFi network over which all communications occurred. Benchmarking procedures were also implemented using the Python `time` module, which wraps functions from the GNU C standard library, for measuring the upload time for transmitting the shares and receiving an acknowledgement from the server.

4.2 Cloud Framework

We developed a cloud-based framework for performing MPC using arithmetic secret sharing comprising three applications.

Web server. A Python web server application that implements a public-facing service for retrieving JSON-formatted shares from the device application; performs database initialisation, storage, and retrieval; and bootstraps the launching of MPC-based operations with other parties and returns the result to the device.

MPyC instance. We make use of MPyC—an open-source Python library developed by Schoenmakers [39] intended for the rapid prototyping of MPC-based systems. MPyC is based on the VIFF framework by Damgård et al. [9], which implements numerous MPC operation protocols using arithmetic secret sharing, and abstracts the complexity of network management, e.g. socket connections, message formatting and transmission, and reconstructing and computing upon shares between the parties.

Database. Stores time-series IMU shares received from the device application; our implementation uses MongoDB[2], a JSON document-based NoSQL database.

4.2.1 Deployment. We deploy our architecture using three MPC parties deployed in geographically disparate locations available via AWS: US West (Oregon), US East (Ohio), Central Canada, Europe (London), and South East Asia (Singapore). These configurations are listed in Table 3, representing a strictly North American deployment (Config A), North America and Europe (Config B), and North America, Europe and Asia (Config C). The three server-side applications, listed above, were deployed collectively on separate AWS EC2 instances in their respective region; each instance was also pre-loaded with the other instances' TLS certificates and static IP addresses in configuration files. We use three `t2.micro` AWS instances, featuring a single-core Intel Xeon Scalable Processor at 3.3 GHz, with 1GB RAM and up to 0.72Gbit/s network throughput. At the time of publication, this incurs an operating cost of $0.0116 (USD) per hour, or $8.47 per month per instance [2].

[2]MongoDB: https://www.mongodb.com

4.2.2 Feature Extraction. In this work, we implement MPC-based feature extraction for replicating the SmartFall proposal by Mauldin et al. [31], which uses two features: the samples' vector magnitudes, and the difference between the maximum and minimum magnitudes, Δs (Equations 1 and 2 respectively), over a sliding window of 750ms. This is managed in our framework by aggregating shares in an array of size 24, assuming a sampling frequency of 31.25Hz as in [31]. The vector magnitude is found for each sample via the Newton-Raphson method for computing the square root, while Δs relies upon MPyC's maximum and minimum functions of a list of secret-shared elements. These are internally implemented by recursively finding the maximum (or minimum) of each list half, computing their difference, and performing a $\geq$ or $\leq$ operation with zero—the costs of which were listed in Table 2.

An important observation is that the choice of features dramatically affects the asymptotic cost of MPC-based operations. Features reliant upon large numbers of multiplication, comparison or Boolean operations will necessitate many rounds of inter-party communication versus features that can be computed locally, such as additions or multiplications with public values, as noted in Section 3.1.2. We note that the SmartFall features, i.e. the vector magnitude and computing Δs in a sliding window, imposes significant cost for the square root and minimum and maximum functions respectively.

To address this, we also explore derivative-based features, which are used extensively for feature extraction in computer vision tasks [3, 8, 26]. In particular, inspired by the SURF features used in Bay et al. [3], we use the sum of the derivatives and the sum of the squared derivatives for each data channel within the window, resulting in a six-dimensional feature vector, f, listed in Equation 3.

$$f = \sum \frac{dx}{dt}, \sum \frac{dy}{dt}, \sum \frac{dz}{dt}, \sum \left(\frac{dx}{dt}\right)^2, \sum \left(\frac{dy}{dt}\right)^2, \sum \left(\frac{dz}{dt}\right)^2 \quad (3)$$

Where $\frac{dx}{dt}$, $\frac{dy}{dt}$, and $\frac{dz}{dt}$ represent the differentials of the x, y an z measurement components respectively. The derivative is computed by convolution as follows:

$$\frac{du}{dt} = A_u * h \quad (4)$$

Where A_u is the u-th component of an accelerometer measurement and h is a filter kernel with impulse response $[-0.5, 0, 0.5]$. This feature extraction scheme is substantially more efficient, requiring only a single round of inter-party communication for each derivative squaring operation; the summing operations can be performed locally without communication, as can the convolution operation using a filter kernel that performs multiplications with constant values.

4.2.3 Privacy-Enhancing Inferencing. We developed a range of MPC-based functions for privacy-preserving inferencing from measurement shares, which were implemented using the fundamental operations from Section 3.1.2. At present, we have developed MPC-based implementations of the following classifiers.

Algorithm 1: Inner product with LSSS [6].

Data: $[\mathbf{a}] = ([a_1], [a_2], \ldots, [a_m]), [\mathbf{b}] = ([b_1], [b_2], \ldots, [b_m])$
Result: $[c]$, where $c = \sum_{j=1}^{m} a_j \cdot b_j$
foreach $i \in \{1, 2, \ldots, 2t+1\}$ **in parallel do**
  Party P_i computes $d_i \leftarrow \sum_{j=1}^{m} [a_j]_i \cdot [b_j]_i$
  P_i shares d_i into $[d_i] \leftarrow SSSS(d_i)$
end
$[c] \leftarrow \sum_{i=1}^{2t+1} ([d_i] \cdot \prod_{\ell=1, \ell \neq i}^{2t+1} \frac{-\ell}{i-\ell})$
return $[c]$

Logistic Regression (LR) is a linear model of the probability of a binary response given a feature vector, $\vec{x} = [x_1, x_2, \ldots, x_n]$. LR applies the logit function to $\vec{x}$ and parameters, $\vec{\theta}$, that minimise some cost function, e.g. ordinary least squares. The decision boundary, h, is represented by a linear combination of the parameters and features; that is, $h(\vec{\theta}, \vec{x}) = \sum_{i=0}^{n} \theta_i x_i$.

A *Support Vector Machine* (SVM) attempts to find a hyperplane that maximises the margin (distance) between linearly separable data points of classes in the training set. We implement a traditional SVM with linear kernel for binary classification, whereby samples are classified using a given set of weights, $\vec{w}$, feature vector, $\vec{x}$, bias, b, and evaluating Equation 5. Note that, for inferencing, both SVM and LR reduce to a single inner product. With MPC, a naïve approach requires m rounds of communication to compute each $\theta_i x_i$; however, [10] demonstrated how to compute the inner product of two vectors using LSSS requiring only a single round of communication with (non-interactive) local summations, which is reproduced in Algorithm 1 for two shared vectors, $[\mathbf{a}]$ and $[\mathbf{b}]$.

$$C = \begin{cases} 1, & \text{if } \vec{\theta} \cdot \vec{x} + b \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Naïve Bayes (NB) is a probabilistic classifier that uses Bayes' Theorem with a strong independence assumption between features. A feature vector is subsequently classified by maximising the posterior probability for each class label using the prior and conditional probabilities computed in the training phase. In our case, we use Gaussian NB to compute the log-likelihoods of real-valued inputs, parameterised by mean μ with variance σ^2 and evaluated as follows:

$$g_i(\vec{x}) = \ln(P(C_i)) - \sum_{j=1}^{n} \frac{||x_j - \mu_{ij}||^2}{2\sigma_{ij}^2} \tag{6}$$

Where n is the number of features and $P(C_i)$ is the probability of the occurrence of class C_i.

5 EVALUATION

This section presents a two-part evaluation comprising: 1), *classification error rates* achieved by our framework using MPC-based implementations of SmartFall and derivative-based features with the three inferencing algorithms; and 2), *computational performance*, for evaluating the time complexity of the framework.

5.1 Classification Error Rates

We now present error rates achieved by the models used in our MPC-based implementations of SmartFall [31] using the standard and derivative-based features described in Section 4.2.2. The authors of [31] provide a publicly-available dataset comprising labelled IMU accelerometer measurements. These correspond to binary fall events from seven participants between 21–55 years old wearing a Microsoft Band 2 smartwatch.

Table 4: Classification error with SmartFall features.

Algorithm	Accuracy	Precision	Recall	F1-Score
LR	0.933	0.780	0.688	0.731
NB	0.926	0.704	0.762	0.732
SVM	0.936	0.819	0.663	0.733

LR: Logistic Regression, NB: Naïve Bayes, SVM: Support Vector Machine.

Table 5: Classification error with derivative-based features.

Algorithm	Accuracy	Precision	Recall	F1-Score
LR	0.936	0.771	0.812	0.791
NB	0.934	0.725	0.903	0.804
SVM	0.937	0.762	0.835	0.797

An off-line Python application was used to generate the model parameters. Firstly, a sliding window of 750ms was used with data sampled at a frequency of 31.25Hz, over which the vector magnitude and Δs were computed. This was repeated for the derivative-based feature extraction. Next, six classifiers—Gaussian NB, SVM (with linear kernel), and logistic regression models for both feature types—were trained independently using the Scikit-Learn library [36]. The inclusion of logistic regression goes beyond the classifiers evaluated in [31]. The dataset was split using a 80:20 training-test set ratio, and five-fold cross-validation was used to select the best performing model, whose final error rate was measured using the test set. The parameters of the best performing models were subsequently exported and implemented within the inferencing procedures detailed in Section 4.2.3.

Tables 4 and 5 presents the error rates for each model with respect to classification accuracy, precision, recall, and F1-scores averaged across all participants for the SmartFall and derivative-based features respectively. The achieved results, i.e. 93.6% accuracy and 0.733 F1-score in the best case for SVM, are commensurate with the original SmartFall proposal and other fall detection work listed previously in Table 1. We note that the derivative features outperform the original features in terms of precision, recall and F1-score, e.g. 0.804 F1-score for NB versus 0.732, and 0.835 recall for SVM versus 0.663; however, the classification accuracy is broadly similar for both feature types.

5.2 Computational Performance

This suite of experiments evaluates the latencies incurred by the device and cloud entities.

5.2.1 Device-side Latency. Here, we measure the average time to both construct the shares using SSSS and their transmission time to $n = 3$ parties, with a reconstruction threshold of $t = 3$. This is

Table 6: Mean feature extraction and inferencing times (milliseconds) using SmartFall features for each configuration; S.D. shown in brackets.

	Config A		Config B		Config C	
Alg.	**FE**	**IN**	**FE**	**IN**	**FE**	**IN**
LR	|	308 (91)	|	383 (104)	|	507 (150)
SVM	**779 (119)**†	—	**1005 (125)**	—	**1204 (194)**	—
NB	|	509 (103)	|	557 (226)	|	692 (114)

†: Common feature extraction for all algorithms, | : Repeat all column-wise entries, —: Repeat previous entry (column-wise), Alg.: Algorithm, FE: Feature extraction, IN: Inferencing.

Table 7: Mean feature extraction and inferencing times (milliseconds) using derivative-based features.

	Config A		Config B		Config C	
Alg.	**FE**	**IN**	**FE**	**IN**	**FE**	**IN**
LR	|	297 (106)	|	365 (95)	|	490 (104)
SVM	**68 (15)**†	—	**102 (19)**	—	**111 (24)**	—
NB	|	552 (103)	|	602 (106)	|	742 (113)

†: Common feature extraction for all algorithms, | : Repeat all column-wise entries, —: Repeat previous entry (column-wise), Alg.: Algorithm, FE: Feature extraction, IN: Inferencing.

measured for the sharing and transmission of samples in a sliding window of 750ms (21 samples at 31.25Hz). The mean share construction time, averaged across all windows in the dataset, took 6.1ms (σ = 0.9), with the upload time taking 398ms (σ = 23.9ms), 404ms (σ = 25.9ms) and 493.2ms (σ = 33.9ms) for configurations A, B and C respectively. These timings are measured by the device-side Python application using the `time` module, which executed upon a Raspberry Pi 3 located in Belgium, Europe. The device specifications were listed previously in Section 4.1.

5.2.2 Cloud-side Latency. We measure the average wall-clock times of the MPC-based feature extraction and inferencing procedures, which were described in Sections 4.2.2 and 4.2.3. The total time for classifying each window can be considered as the sum of these two values. These were measured within the server-side Python application also via the `time` module. We report the times for each cloud configuration using the standard SmartFall features in Table 6 and the derivative-based features in Table 7. The results are also illustrated in Figures 3 and 4.

5.3 Discussion

One concern with MPC, raised previously in Section 3.1.2, is the inter-party communication cost for performing multiplication- and comparison-heavy functions, which were used in the feature extraction and inferencing procedures. With the original SmartFall features, inferencing and feature extraction takes approximately 1900ms in the worst case with Naive Bayes under Config C, to ~1100ms in the best case using LR with Config A. We also observe that the feature extraction stage dominates the execution times using this feature type, accounting for approximately two-thirds of each tested algorithm's total time. In contrast, this inverts with

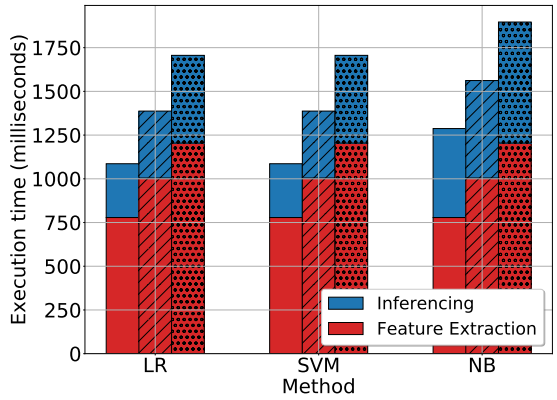

Figure 3: Mean window execution times using SmartFall features for Config. A (no hatch), B (hatched), and C (dotted).

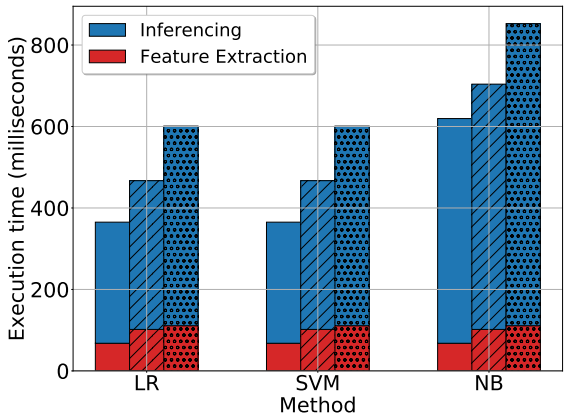

Figure 4: Mean window execution times using derivative features for Config. A (no hatch), B (hatched), and C (dotted).

the use of derivative features, which require fewer rounds of inter-party communication; the *inferencing* stage becomes the dominant factor for all algorithms, rather than the feature extraction. The execution time is also substantially reduced for the derivative-based features; reducing from 1087ms to 365ms using LR with Config A, for example, between the SmartFall and derivative-based features respectively. In general, the derivative-based features are extracted ~11x faster than the originally proposed SmartFall features in [31]; the *total* execution time, including inferencing, exhibits a ~2.8x increase between derivative- and SmartFall-based experiments.

We also see that the SVM and LR algorithms execute in similar times due to the mathematical similarities of inferencing, requiring an inner product operation that necessitates only a single round of communication. NB, however, is relatively expensive from requiring two multiplications for each feature in the window: one for finding the square of the feature subtracted from the mean, and the inverse multiplication (division) of this value by the variance. The choice of server configuration, i.e. the parties' geographical distribution, also has a significant effect on the total execution time; increasing, for example, from approximately 1087ms to 1711ms using SmartFall

features with SVM-based inferencing, corresponding to a ~57% increase from Config A to C. Similar results are exhibited for all SmartFall- and derivative-based experiments.

Lastly, we observe that, in some instances, the total execution time falls within that required for acquiring the measurements within the window (750ms). This applies for all derivative-based prediction using configurations A and B, but only the SVM and LR inferencing algorithms using Config C. This leads us to believe that these would be best suited for real-time prediction.

6 CONCLUSION

A plethora of fall detection systems rely upon IMU data from user-owned commodity devices. However, as we discussed in Section 2, these proposals necessitate the collection of data that can be used to identify users' gait, their position using dead reckoning, and determining their activities of daily living (ADLs). In light of this, we presented the first investigation of the application of multi-party computation for IMU-based fall detection with the aim of preserving the confidentiality of sensor measurements.

We presented the design and implementation of our system in Sections 3 and 4, which was subsequently deployed using three AWS EC2 instances in geographically disparate locations under three configurations. Each instance acted as an MPC participant for jointly computing the feature extraction and inferencing procedures using weights from pre-trained global models. The system was evaluated with the original SmartFall features alongside an exploration of derivative-based features for reducing the communication complexity imposed by MPC operations.

In Section 5, we presented the results from a two-part evaluation of the system's classification error and computational performance. This was conducted using a publicly-available dataset comprising data from seven participants wearing an off-the-shelf device. The performance results indicate that state-of-the-art error rates can be achieved for MPC-based fall detection—achieving 0.926–0.937 classification accuracy and 0.731–0.804 F1-score depending on the chosen algorithm and feature set.

We also showed experimentally that the choices of inferencing algorithm, feature extraction, and server location have substantial effects on computational performance. The worst cases increase the total execution total by almost three-fold versus the best performers, which is dominated principally by the choice of features. In the best cases—for example, using an SVM classifier with derivative-based features in a cloud configuration in the same continent—the total prediction time is less than the on-device data acquisition time. This provides an indication that real-time prediction is feasible.

To conclude, we aim to pursue the following research directions in future work:

- *MPC-based fall detection using deep learning.* Existing fall detection systems rely heavily on traditional machine learning classifiers; a small selection of recent work, however, has begun to explore the use of deep neural networks (DNNs), such as RNNs [34]. As such, a future avenue lies in exploring MPC-based DNNs and the overhead imposed by evaluating large numbers of intermediate layers with potentially hundreds or thousands of nodes.
- *Video-based fall detection.* A separate paradigm of fall detection systems, described in Section 2, relies upon video camera data from devices situated in the user's ambient environment. This data, however, is of significantly greater dimension than the IMU data explored in this work; a future direction is investigating the extent to which MPC-based fall detection can be performed with high-frequency, high-dimensionality data.

REFERENCES

[1] Mark V Albert, Konrad Kording, Megan Herrmann, and Arun Jayaraman. 2012. Fall classification by machine learning using mobile phones. *PloS one* 7, 5 (2012).
[2] Amazon. 2019. Amazon EC2 T2 instances. https://aws.amazon.com/ec2/instance-types/t2/.
[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.* 110, 3 (June 2008), 346–359.
[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. ACM.
[5] Yabo Cao, Yujiu Yang, and WenHuang Liu. 2012. E-FallD: A fall detection system using android-based smartphone. In *9th International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE, 1509–1513.
[6] Ping Chen. 2012. *Secure Multiparty Computation for Privacy Preserving Data Mining*. Master's thesis. TU Eindhoven.
[7] Andrea Continella, Mario Polino, Marcello Pogliani, and Stefano Zanero. 2018. There's a hole in that bucket!: A large-scale analysis of misconfigured S3 buckets. In *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 702–711.
[8] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 886–893.
[9] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. 2009. Asynchronous multiparty computation: Theory and implementation. In *International Workshop on Public Key Cryptography*. Springer, 160–179.
[10] Sebastiaan de Hoogh. 2012. Design of large scale applications of secure multiparty computation: secure linear programming. *Ph. D. dissertation* (2012).
[11] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. 2014. Practical secure decision tree learning in a teletreatment application. In *International Conference on Financial Cryptography and Data Security*. Springer.
[12] Yueng Santiago Delahoz and Miguel Angel Labrador. 2014. Survey on fall detection and fall prevention using wearable and external sensors. *Sensors* 14, 10 (2014), 19806–19842.
[13] Charalampos Doukas and Ilias Maglogiannis. 2011. Managing wearable sensor data through cloud computing. In *3rd IEEE International Conference on Cloud Computing Technology and Science*. IEEE.
[14] Charalampos Doukas, Ilias Maglogiannis, Philippos Tragas, Dimitris Liapis, and Gregory Yovanof. 2007. Patient fall detection using support vector machines. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 147–156.
[15] Alex Edgcomb and Frank Vahid. 2012. Automated fall detection on privacy-enhanced video. In *Engineering in Medicine and Biology Society (EMBC), Annual International Conference of the IEEE*. IEEE, 252–255.
[16] Curtis S Florence, Gwen Bergen, Adam Atherly, Elizabeth Burns, Judy Stevens, and Cynthia Drake. 2018. Medical costs of fatal and nonfatal falls in older adults. *Journal of the American Geriatrics Society* 66, 4 (2018), 693–698.
[17] Giancarlo Fortino, Daniele Parisi, Vincenzo Pirrone, and Giuseppe Di Fatta. 2014. BodyCloud: A SaaS approach for community body sensor networks. *Future Generation Computer Systems* 35 (2014), 62–79.
[18] Davrondzhon Gafurov. 2007. A survey of biometric gait recognition: Approaches, security and challenges. In *Annual Norwegian Computer Science Conference*.
[19] Davrondzhon Gafurov, Einar Snekkenes, and Patrick Bours. 2007. Gait authentication and identification using wearable accelerometer sensor. In *IEEE Workshop on Automatic Identification Advanced Technologies*. IEEE, 220–225.
[20] Raffaele Gravina, Congcong Ma, Pasquale Pace, Gianluca Aloi, Wilma Russo, Wenfeng Li, and Giancarlo Fortino. 2017. Cloud-based Activity-aaService cyber-physical framework for human activity monitoring in mobility. *Future Generation Computer Systems* 75 (2017), 158–171.
[21] Yu-Jin Hong, Ig-Jae Kim, Sang Chul Ahn, and Hyoung-Gon Kim. 2010. Mobile health monitoring system based on activity recognition using accelerometer. *Simulation Modelling Practice and Theory* 18, 4 (2010), 446–455.
[22] Felix Juefei-Xu, Chandrasekhar Bhagavatula, Aaron Jaech, Unni Prasad, and Marios Savvides. 2012. Gait-ID on the move: Pace independent human identification using cell phone accelerometer dynamics. In *Fifth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*. IEEE, 8–15.

[23] Wonho Kang and Youngnam Han. 2015. SmartPDR: Smartphone-based pedestrian dead reckoning for indoor localization. *IEEE Sensors Journal* 15, 5 (2015).

[24] Bogdan Kwolek and Michal Kepski. 2014. Human fall detection on embedded platform using depth maps and wireless accelerometer. *Computer methods and programs in biomedicine* 117, 3 (2014), 489–501.

[25] Shing-Hong Liu and Wen-Chang Cheng. 2012. Fall detection with the support vector machine during scripted and continuous unscripted activities. *Sensors* 12, 9 (2012), 12301–12316.

[26] David G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110.

[27] Mitja Luštrek and Boštjan Kaluža. 2009. Fall detection and activity recognition with machine learning. *Informatica* 33, 2 (2009).

[28] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. 2017. EPIC: Efficient Private Image Classification (or: Learning from the Masters). Cryptology ePrint Archive, Report 2017/1190. https://eprint.iacr.org/2017/1190.

[29] Jani Mantyjarvi, Mikko Lindholm, Elena Vildjiounaite, S-M Makela, and HA Ailisto. 2005. Identifying users of portable devices from gait pattern with accelerometers. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE.

[30] Georgios Mastorakis and Dimitrios Makris. 2014. Fall detection system using Kinect's infrared sensor. *Journal of Real-Time Image Processing* 9, 4 (2014).

[31] Taylor Mauldin, Marc Canby, Vangelis Metsis, Anne Ngu, and Coralys Rivera. 2018. SmartFall: a smartwatch-based fall detection system using deep learning. *Sensors* 18, 10 (2018), 3363.

[32] S Mellone, C Tacconi, L Schwickert, J Klenk, C Becker, and L Chiari. 2012. Smartphone-based solutions for fall detection and prevention: the FARSEEING approach. *Zeitschrift für Gerontologie und Geriatrie* 45, 8 (2012), 722–727.

[33] Muhammad Mubashir, Ling Shao, and Luke Seed. 2013. A survey on fall detection: Principles and approaches. *Neurocomputing* 100 (2013), 144–152.

[34] Mirto Musci, Daniele De Martini, Nicola Blago, Tullio Facchinetti, and Marco Piastra. 2018. Online fall detection using recurrent neural networks. *arXiv preprint arXiv:1804.04976* (2018).

[35] Kartik Palani, Emily Holt, and Sean Smith. 2016. Invisible and forgotten: Zero-day blooms in the IoT. In *Pervasive Computing and Communication Workshops*. IEEE.

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

[37] Azkario Rizky Pratama, Risanuri Hidayat, et al. 2012. Smartphone-based pedestrian dead reckoning as an indoor positioning system. In *International Conference on System Engineering and Technology*. IEEE.

[38] José Antonio Santoyo-Ramón, Eduardo Casilari, and José Manuel Cano-García. 2018. Analysis of a smartphone-based architecture with multiple mobility sensors for fall detection with supervised learning. *Sensors* 18, 4 (2018), 1155.

[39] Berry Schoenmakers. 2018. MPyC—Python package for secure multiparty computation. In *Workshop on the Theory and Practice of MPC*. https://github.com/lschoe/mpyc.

[40] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[41] Maja Stikic, Tâm Huynh, Kristof Van Laerhoven, and Bernt Schiele. 2008. ADL recognition based on the combination of RFID and accelerometer sensing. In *Pervasive Computing Technologies for Healthcare*. IEEE, 258–263.

[42] Angela Sucerquia, José David López, and Jesús Francisco Vargas-Bonilla. 2018. Real-life/real-time elderly fall detection with a triaxial accelerometer. *Sensors* 18, 4 (2018), 1101.

[43] Shuai Tao, Mineichi Kudo, and Hidetoshi Nonaka. 2012. Privacy-preserved behavior analysis and fall detection by an infrared ceiling sensor network. *Sensors* 12, 12 (2012), 16920–16936.

[44] Hoang Minh Thang, Vo Quang Viet, Nguyen Dinh Thuc, and Deokjai Choi. 2012. Gait identification using accelerometer on mobile phone. In *Control, Automation and Information Sciences*. IEEE, 344–348.

[45] Thomas Vilarinho, Babak Farshchian, Daniel Gloppestad Bajer, Ole Halvor Dahl, Iver Egge, Sondre Steinsland Hegdal, Andreas Lønes, Johan N Slettevold, and Sam Mathias Weggersen. 2015. A combined smartphone and smartwatch fall detection system. In *IEEE International Conference on Ubiquitous Computing and Communications*. IEEE, 1443–1448.

[46] Boyuan Wang, Xuelin Liu, Baoguo Yu, Ruicai Jia, and Xingli Gan. 2018. Pedestrian dead reckoning based on motion mode recognition using a smartphone. *Sensors* 18, 6 (2018), 1811.

[47] Gokhan Yavuz, Mustafa Kocak, Gokberk Ergun, Hande Ozgur Alemdar, Hulya Yalcin, Ozlem Durmaz Incel, and Cem Ersoy. 2010. A smartphone based fall detector with online location support. In *International Workshop on Sensing for App Phones*. 31–35.

[48] Chenyang Zhang, Yingli Tian, and Elizabeth Capezuti. 2012. Privacy preserving automatic fall detection for elderly using RGBD cameras. In *International Conference on Computers for Handicapped Persons*. Springer, 625–633.

[49] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. 2014. IoT security: ongoing challenges and research opportunities. In *7th International Conference on Service-Oriented Computing and Applications*. IEEE, 230–234.