2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

Machine Learning with Reconfigurable Privacy on Resource-Limited Computing Devices

Sana Imtiaz*†, Zannatun N. Tania*, Hassan Nazeer Chaudhry‡, Muhammad Arsalan§,
Ramin Sadre†, Vladimir Vlassov*
*EECS/SCS, KTH Royal Institute of Technology, Stockholm, Sweden
†ICTEAM/INGI, Université catholique de Louvain, Louvain-la-Neuve, Belgium
‡DEIB, Politecnico di Milano, Milan, Italy
§FK EITP, Technische Universität Braunschweig, Braunschweig, Germany
Email: sanaim@kth.se

***Abstract*—Ensuring user privacy while learning from the acquired Internet of Things sensor data, using limited available compute resources on edge devices, is a challenging task. Ideally, it is desirable to make all the features of the collected data private but due to resource limitations, it is not always possible as it may cause overutilization of resources, which in turn affects the performance of the whole system. In this work, we use the generalization techniques for data anonymization and provide customized injective privacy encoder functions to make data features private. Regardless of the resource availability, some data features must be essentially private. All other data features that may pose low privacy threat are termed as non-essential features. We propose Dynamic Iterative Greedy Search (DIGS), a novel approach with corresponding algorithms to select the set of optimal data features to be private for machine learning applications provided device resource constraints. DIGS selects the necessary and the most private version of data for the application, where all essential and a subset of non-essential features are made private on the edge device without resource overutilization. We have implemented DIGS in Python and evaluated it on Raspberry Pi model A (an edge device with limited resources) for an SVM-based classification on real-life health care data. Our evaluation results show that, while providing the required level of privacy, DIGS allows to achieve up to 26.21% memory, 16.67% CPU instructions, and 30.5% of network bandwidth savings as compared to making all the data private. Moreover, our chosen privacy encoding method has a positive impact on the accuracy of the classification model for our chosen application.**

***Index Terms*—Data privacy, optimization, greedy algorithms, machine learning, anonymization, consumer-producer models, edge devices, IoT.**

I. Introduction

With the increasing popularity of the Internet of Things (IoT), a tremendous amount of data is continuously being acquired by a variety of intelligent sensor nodes. This data is then processed over central cloud platforms or is transformed through some edge devices before reaching the central processing node [1]. Although this strategy enables data storage management and big data processing while utilizing a distributed computing paradigm, however, distributed processing and storage also leads to data integrity and privacy concerns. Firstly, cloud-based platforms may misuse data due to monetary goals such as product marketing. In such cases, the privacy contracts are designed underhandedly to conceal such privacy breaches [2]. Secondly, cloud-based platforms undergo security and privacy breaching attacks from time to time [3], [4]. Thirdly, the end-user might not be comfortable sharing his private information or publicly disclose some of his data. In a nutshell, although cloud-based distributed platforms enable big data processing and storage with certain guarantees of the quality of service, however, privacy preservation of such data is vital from the user privacy perspective [5], [6].

Privacy should be preserved on the data acquisition site i.e. mobile device or embedded edge device attached to some sensors nodes (SN) before the data is transmitted to the central cloud-computing resource [7]. In all cases, mobile devices or embedded edge devices have limited battery, memory and processing resources. Moreover, the way devices utilize the available bandwidth effectively, also known as spectrum efficiency, is a critical performance metric in 5G communication with large number of devices [8]. Therefore, privacy preservation in resource constraint data acquisition devices becomes a bottleneck in data processing pipelines.

The SN acquires a particular set of data features depending on the application or usage scenario. For example, in the case of a fire alert system, the features would be temperature, humidity, and presence of smoke. In the best-case scenario, the system would like to preserve the privacy of all possible features. However, preserving some features may not be essential or does not influence the privacy of the user even if they are not preserved, such features are called non essential features (NEF). On the other hand, the application might have some features whose privacy must be preserved at all costs, known as the essential features (EF). In the optimal case, the embedded devices should preserve the privacy of EF as well as the NEF if the resources such as storage, bandwidth, and processing capability of the devices permit. In order to adequately utilize the system resources, the crucial question of which subset of NEF must have privacy preservation should

This work is partially funded by the Erasmus Mundus Joint Doctorate program in Distributed Computing (EACEA of the European Commission under FPA 2012-0030).

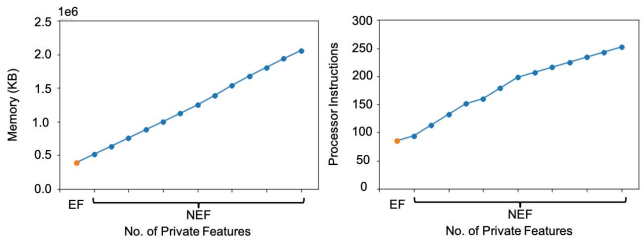
978-1-6654-3574-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00213

be addressed to best utilize the storage, bandwidth, processing and memory resource of the embedded edge device without significantly compromising accuracy of algorithm.

Another important issue concerning the machine learning (ML) applications is the application accuracy, since privacy may affect the performance of the training and utility of the ML system model [9]–[11]. Therefore, the accuracy of the resulting ML training model is a significant constraint while selecting the optimal subset of NEF.

One way to design such a system involves application-specific feature selection by performing different trials or by using application design experience. The handpicked feature selection may work in some application scenarios, however, they are challenging to design, are unscalable and unadaptable. Instead of having handpicked feature selection, a scalable approach would be to dynamically choose NEFs adequately suited to the constrained resources of the devices. To avoid rewriting privacy encoders for varying scenarios, a microservice architecture may provide privacy as a service approach.

Although it might be tempting to apply privacy preservation measures on all the input data features to ensure maximum provision of privacy, privacy comes at the cost of increased resources and very often, a negative impact on accuracy and efficiency of the system [9]–[11]. Figure 1 shows the trend of increasingly private features with device resource constraints for an edge device, Raspberry Pi model A for a simple machine learning application using anonymization techniques for privacy preservation (more details to be presented in Section III-D). As can be seen, adding privacy preservation requires significant amount on increase in resource consumption. Moreover, depending on the privacy preserving technique employed, the system could use up to twice the resources with significant drop in efficiency and accuracy of the system. For example, cryptography-based privacy preserving solutions could consume up to 2-5 seconds per operation [12], [13], which is undesirable for IoT-driven systems.



(a) Memory consumption (b) Processor Instructions

Fig. 1: Increasingly private features vs. resource consumption for an ML-based application with data anonymization

On the other hand, using low levels of privacy may not only violate the rights of users, expose the system to privacy-breaching attacks, but also may violate the data protection laws such as EU's General Data Protection Regulation (GDPR). Therefore, it is important to find optimal operating conditions offering a good trade-off between system performance and the level of privacy preserved. Moreover, in the case of resource constrained devices, it becomes vital to employ efficient privacy preserving practices on selected features to ensure the best functionality and quality of service. In summary, the privacy preservation must be done dynamically in a way that the privacy of all EFs is preserved and the most optimal set of NEFs is selected constrained to memory consumption, processing requirements, bandwidth consumption and accuracy of the algorithm. Moreover, the additional processing time required for privacy preservation should not cause violation of the service-level agreement between the service provider and the clients.

The contributions of this work are as follows.

- We propose and present a novel approach with corresponding algorithms and an end-to-end system pipeline for reconfigurable[1] data privacy in machine learning on resource-limited computing devices.
- We present and have developed a novel greedy search algorithm, DIGS, to find the optimal selection of privacy-preserved input data features provided device resource constraints for a given machine learning algorithm with its input and output data features.
- We propose an end-to-end system pipeline with our proposed DIGS algorithm, as well as injective privacy preservation functions using generalization and anonymity techniques for reconfigurable privacy.
- We have implemented, illustrated and evaluated the results of our proposed approach using a real-world smart health care dataset and machine learning application on a resource-constrained edge device.

Evaluation of our proposed approach for reconfigurable privacy in machine learning on resource-limited devices shows significant resource savings, with up to 26.21% memory, and 16.67% CPU instructions, and 30.5% network bandwidth savings as compared to making all input data features private.

The rest of the paper is organized as follows. Section II presents some preliminaries. The proposed technique and the DIGS algorithm are presented in Section III. Our experiments and results are presented in Section IV and discussed in Section V. Finally, we present related work in Section VI followed by conclusions and future work in Section VII.

II. Preliminaries

A. Privacy by design and default

In accordance with the EU's GDPR, all services employing any collection or usage of user data must provide data protection by design and by default [14]. This implies that by default the system should only collect and process data that is absolutely necessary as well as provide the strictest data protection guarantees. In the context of resource constrained devices, this implies that it is vital to determine the most optimal set of features to make private to provide the strictest possible privacy preservation.

[1]Here we use the words *reconfigurable* and *tunable* data privacy as synonyms.

B. Privacy preservation techniques

When it comes to privacy preservation techniques, a wide variety of solutions is available. Typically, all privacy preservation techniques employ mechanisms like information flow control [15], data or model obfuscation via noise addition [16], use of cryptography [9], [13], anonymization via generalization and suppression of attributes [17]–[19], and use of private computation units. As observed in [6], all these techniques have their respective performance overheads in terms of reduced application accuracy or longer processing times. Anonymization is one of the easy to integrate privacy preservation techniques and also offers the low computational complexity as compared to other techniques.

1) Anonymization techniques: Anonymization techniques commonly employ the principles of generalization and suppression. *Generalization* implies replacing a value with a less specific but semantically consistent value, while *suppression* involves not releasing a value at all [20]. Common practice in anonymization includes removal or modification of sensitive attributes such as names, gender, postal codes, and identification numbers. More sophisticated methods such as *k*-anonymization [17], [20], *l*-diversity [18] and *t*-closeness [19], are employed for better privacy preservation guarantees.

III. Proposed Technique

Given an ML application with its set of input data features (both EF and NEF), we first calculate the cost for the NEF through the cost calculating module, and pass these costs to the optimization algorithm. The algorithm creates a cost matrix using the input and selects the most optimal features within the range of device resource constraints. We then apply privacy preservation to the selected NEF additionally with the EF. This new privacy encoded dataset is used for the ML application. The complete system pipeline is shown in Figure 2. We now explain the system model followed by the explanation for each component of the proposed system pipeline.

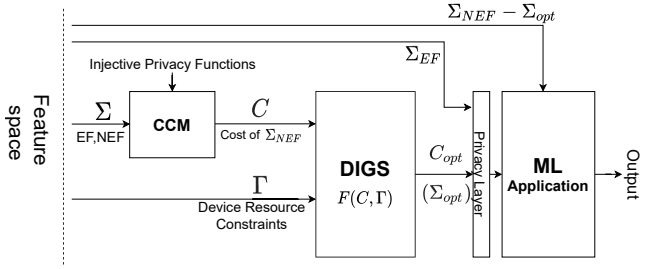


Fig. 2: System pipeline with DIGS

A. System Model

Consider a system of P producers and A consumers (applications and services), where P is set of N producers and A is set of M consumers respectively, as shown in equation (1) and equation (2).

$$\mathbf{P} = \{P_1, P_2, ..., P_N\} \tag{1}$$

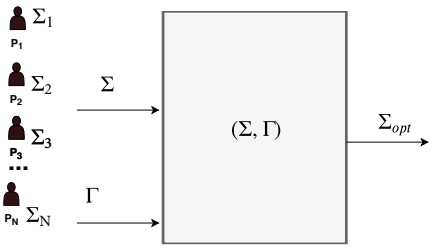


Fig. 3: Producer-consumer system model

$$\mathbf{A} = \{A_1, A_2, ..., A_M\} \tag{2}$$

Each P consists of a certain number of features associated with the producer, for each i^{th} producer the set Σ_i is shown in equation (3). To preserve the privacy of the user, the σ's of each P_i are encoded in a certain format before they reach the consumer, as shown in equation (4).

$$\mathbf{\Sigma_i} = P_{\sigma_i} = \{\sigma_1, \sigma_2, ..., \sigma_x\} \tag{3}$$

where X is the total number of features.

$$\mathbf{\Sigma} = \{P_{\sigma_1}, P_{\sigma_2}, ..., P_{\sigma_N}\} \tag{4}$$

However, encoding all Σ is not efficient in terms of certain constraints Γ such as memory, bandwidth, processing (number of instructions or operations), prediction accuracy, and storage requirements. As shown in equation (5), Γ contains the maximum threshold for all these constraints.

$$\mathbf{\Gamma} = \{\Gamma_m, \Gamma_{bw}, \Gamma_p, \Gamma_a, \Gamma_s\} \tag{5}$$

Therefore, the subset of Σ (Σ_{opt}) is selected to meet the given Γ using a certain optimization function $F(\Sigma, \Gamma)$, where $\Sigma_{opt} \subset \Sigma$. The producer-consumer system model is depicted in Figure 3, where the users are shown on the left hand side.

The major goal of $F(\Sigma, \Gamma)$ is to find a function which maps Σ to Σ_{opt} constrained by Γ. The function $F(\Sigma, \Gamma)$ is tunable in the sense that each time the constraints are updated, new Σ_{opt} can be generated.

B. Greedy Optimization Algorithm - DIGS

We present a greedy approach to selecting a set of optimal features for provided system constraints, *Dynamic Iterative Greedy Search* (DIGS) for privacy preservation.

Assumptions: We make the following assumptions for this algorithm. The features are assumed to be independent as the feature similarity or correlation is not catered to in the algorithm. Moreover, the service provider should specify the important or *EF* and the optional features as *NEF* as this is orthogonal to the functionality of DIGS algorithm.

$$\text{EF} \cap \text{NEF} = \emptyset \tag{6}$$

DIGS: Consider that C represents the cost of all constraints (5) for each feature in Σ, as shown in (7).

$$C = \{C_m, C_{bw}, C_a, C_s, C_p\} \tag{7}$$

Where, each element in Ci is a row which has N elements, the σ corresponding to each producer. For example, the memory cost constraints can be represented as:

$$C_m = \{C_{m\sigma 1}, C_{m\sigma 2}, ..., C_{m\sigma_N}\} \tag{8}$$

In order to select the best features in terms of the provided constraints, we have to check if the cost of each feature is within the the maximum resource consumption allowance and afterwards, we check the combinations of features and their respective costs against the maximum resource consumption allowance. We first append each element of each row in Ci such that it contains the value and the index of that value. Then we sort each row in ascending order to minimize the calculation time. Then for each category of performance constraints (memory, bandwidth and so on) as represented by Γ, we select the set of features Σ_{opt} represented as C_{opt} that are optimal for a particular constraint (*locally optimal*) as well as optimal for the total constraints (*globally optimal*). The current version of DIGS is written in Python. It must be noted that Σ_{opt} contains only the optimal NEF that can be made private under the given device resources constraints for a target ML application.

C. Calculating the feature costs - *CCM*

An important supporting component of DIGS is the cost calculation module, **CCM**. The programming logic consisting of the application code as well as the programming language, and the Σ (containing both EF and NEF) are provided as an input to the CCM. The presented version of CCM currently supports Python programming language and can be easily extended to support other programming languages.

1) Memory consumption: We calculated the memory consumption of each feature in the dataframe after applying the injective privacy function and we also calculated the memory consumed while running the respective injective privacy function. For the dataframe memory, we simply used Python's `memory_usage()` function and derived the memory for all the input data features. For calculating the memory consumed by the functions, we used a Python library called `guppy()` [21] which provides the current heap memory. First, we cleared the heap memory with `setrelheap()`. Then we executed the injective privacy functions with different input data features and printed the heap memory again. We took the difference and added this memory usage (in kilobytes) to the dataframe memory to obtain the overall memory consumption.

2) Bandwidth consumption: The bandwidth cost is calculated by dividing the dataframe memory with the network speed of the selected edge device in a certain network, such as 4G. It is calculated in kbps.

Algorithm 1 The Greedy Algorithm

Function *DIGS(C, Γ) : $opt\_feature$*
  Augment each element x in C with key-value pairs: $x \leftarrow (key : val)$
  $sortedC \leftarrow$ sorted values of C in ascending order
  Initialize empty lists $opt\_feature$ and $blck\_feature$
  $subset \leftarrow$ generate a list of subsets with the keys from $sortedC$
  foreach *y in subset* **do**
    Initialize sum = 0, $flag$ = 0
    $suby$ = $subset[y]$
    foreach *i in Γ* **do**
      forall *key in $blck\_feature$* **do**
        if *$key \subseteq suby$* **then**
          $flag$ = 1
          $blck\_feature$.append($suby$)
          break
      end
      foreach *j in $suby$* **do**
        sum += $sortedC[i][j]$
        if $sum > \Gamma[i]$ **then**
          $blck\_feature$.append($suby$)
          break
      end
    end
    if *$suby$ is not in $blck\_feature$* **then**
      $opt\_feature$.append($suby$)
  end
  return $opt\_feature$
end

3) Computational processing cost: The computational cost is device-specific. We calculated the computational processing cost by translating injective privacy functions into machine instructions via the lookup table of the selected edge device. For each injective privacy function, we tokenize the code to identify instances of operations such as `if, else, and, or` statements. These statements are translated to number of processor instructions using the lookup table from the device instruction set.

All the computed costs are normalized depending on the type of resource constraint. This allows DIGS to make fair comparisons between the resource consumption costs for each data feature for different type of resource constraints.

D. SVM Classification Model - ML application

Another important component of our system pipeline is the target ML application. For this work, we use a supervised classification model using SVM in order to evaluate the effect of our privacy preserved data on the application accuracy and resource consumption. Our dataframe contains these features: nutritional value intake (calories, fat, fiber, protein, carbs, sodium), activity data (lightly active minutes, sedentary minutes, very active minutes, moderately active minutes, calories burned, steps count), heart rate, height, weight, age, and gender. We created custom "labels" based on the data values in

Algorithm 2 The Greedy Algorithm: Additional Features

Function *DIGS_add_feature(*$sortedC, \Gamma, opt\_feature$*) :(*$add\_opt\_feature$*, *$global\_optimal$*)*
    Initialize $add\_opt\_feature$, $add\_blk\_feature$ and $global\_optimal$
    foreach x *in* $range(2, len(opt\_feature) + 1)$ **do**
        $subset \leftarrow$ list of subsets of opt_param of size x
        Initialize gt_sum and gt_cons
        Initialize $flag$ as 0
        foreach y *in* $subset$ **do**
            $suby$ = $subset[y]$
            foreach i *in* Γ **do**
                forall *key in* $add\_blck\_feature$ **do**
                    if *key* $\subseteq$ $suby$ **then**
                        $flag$ = 1
                        $add\_blck\_feature$.append($suby$)
                        break
                end
                foreach j *in* $suby$ **do**
                    sum += $sortedC[i][j]$
                    if $sum > \Gamma[i]$ **then**
                        $add\_blck\_feature$.append($suby$)
                        break
                    else
                        gt_sum += sum
                        gt_cons +=$\Gamma[i]$
                    end
                end
            end
            if $suby$ *is not in* $add\_blck\_feature$ **then**
                $add\_opt\_feature$ += $suby$
                $global\_optimal$.append([$suby, gt\_sum$, gt_cons])
        end
    end
    return $add\_opt\_feature, global\_optimal$
end

Algorithm 3 The Greedy Algorithm: Most Optimal Features

Function *most_optimal(*$global\_optimal$*) :* $most\_opt\_set$
    Select the sets of features with the most number of elements from $global\_optimal$
    $most\_opt\_set$ = Sort the sets by their constraint values in ascending order and select the set that has the least total constraints
    return $most\_opt\_set$
end

the features. We followed the guidelines from [22] and [23] to label each day of the user as healthy or unhealthy. For example, on a specific day if a user was consuming more than 2000 calories and not being active, and if the user's BMI is over the average range, we labeled this day as being unhealthy (1), otherwise healthy (0). We used standard scaler in order to have similar distribution of the dataset as we noticed there were more records with one label than the other. We ran the model on different versions of data, such as all non-private data, all privacy encoded data, dataset with only the EF being private, and the dataset containing DIGS selected private features and essential private features. We compare the accuracy of these datasets to see the effect of added privacy on the performance of ML application.

IV. Experiments and Results

A. Dataset

For the evaluation of our proposed system, we used two datasets: a first-hand collected Fitbit dataset and GAN-augmented Fitbit dataset. Table I shows an overview of the datasets in terms of scale. The dataset collection and processing are described in next Sections.

	Fitbit	**FitBit-GAN**
# of Users	25	500
# of Days	60	60
# of raw Records	≈17 M	≈340 M
Size	3.2 GB	≈64 GB

TABLE I: Datasets used for evaluation

1) Fitbit dataset collection and privacy concerns: We have collected a smart health care dataset using *Fitbit Charge 2 HR* devices with 25 subjects. The subjects were distributed across Belgium and Sweden. 12 devices were used for dataset collection with 2 continual participants (male and female) and 10 users in circulation. The data was recorded for 60 days for each user. The missing entries for the nutritional breakdown for meals were imputed using Nutritionix API [24]. The body-mass-index (BMI) was calculated from the user's weight and height using the BMI formula [25]:

$$\mathbf{BMI} = kg/m^2 \tag{9}$$

The calorie intake is calculated by converting the macronutrients (grams) in into calories as:

$$\mathbf{Calories\_in} = fat \times 9 + protein \times 4 + carbs \times 4 + fiber \times 1.5 \tag{10}$$

These conversions are done according to [26] and recommendations by Food and Drug Administration (FDA), USA.

The naming convention used for the features is the same as feature names imported from the Fitbit API, and other features like height, weight, gender and BMI are user-defined. The records are aggregated per day and the complete spectrum of the dataset features is presented in Table II. Here, the less frequently updated variables are termed as static variables.

On one side, the amount of sensitive information is quite huge making the active collection of data hard as it requires fully informed consent for data disclosure. On the other side, experimenting on such dataset demonstrates the capabilities of privacy preservation techniques because of the highly private information present in the dataset. However, we removed the sensitive individually identifying data (using anonymization) for the actual processing.

Feature Name	Type	Description
Date	Static	Data log date
Age	Static	Age of the user
Gender	Static	Male or female
Height	Static	Height of the user
Weight	Static	Weight of the user
Fat	Behavioral	Fat (gm) consumed from each food
Fiber	Behavioral	Fiber (gm) consumed from each food
Carbs	Behavioral	Carbohydrates (gm) consumed from each food
Sodium	Behavioral	Sodium (mg) consumed from each food
Protein	Behavioral	Protein (gm) consumed from each food
Calories_in	Behavioral	Calculated calorie intake
Calories_burned	Behavioral	Calories burnt
Resting_heart_rate	Behavioral	Resting heart rate on the day before Date
Lightly_active_minutes	Behavioral	Minutes of light activity
Moderately_active_minutes	Behavioral	Minutes of moderate activity
Very_active_minutes	Behavioral	Minutes of high activity
Sedentary_minutes	Behavioral	Minutes spent sedentary
Steps	Behavioral	Steps taken
BMI	Static	The BMI of user
labels	Target	Indicates whether the user diet and activity on Date is healthy or unhealthy

TABLE II: Dataset

2) GAN for Data Augmentation:: To extend the collected dataset, we used Bi-directional GANs [27], a type of Generative Adversarial Networks (GAN) [28], to generate additional users based on their location and gender. The augmentation of the data was done separately on the 4 different datasets based on location and gender. We generated total 500 users from the 4 datasets and after combining all the data we get 33120 aggregated daily records in total. All the features of this dataset are exactly the same as the Fitbit dataset and the users also have realistic and similar data distributions as the Fitbit dataset.

B. Device resource constraints

Raspberry pi RPi 1 Model A is employed for the evaluation of the DIGS algorithm. RPi is installed as an edge device which collects data from multiple sensors. RPi has 256 MiB of memory; however, on average, approximately 56 MiB is used by the device itself for its system operations. Besides system memory usage, typically, around 40% of the memory is used by other applications and processes. Therefore, effective available and usable RPi memory is around 120 MiB. Consider a scenario where 100 devices are connected to the edge device, which is a reasonable sensor node size assumption. Our experimentation has revealed that encoding of EF with the load of a hundred sensors consumes almost 20% of the memory. In our experiments, four EF, i.e., weight, height, age, and gender, are always considered private. In a nutshell, the device has limited operational memory left, and the algorithm has to decide the selection of NEF. The situation becomes further complicated with more complex injective privacy functions or more sensors. Similar constraints can be observed for the device's processing capabilities; for the available 700 MHz, mostly 20% of the available computing power is typically used for system operations. Furthermore, in our observations, on an average 40% is utilized for other processes of the machines. So effectively, around 250 MHz clocking power is available for 100 devices. We have employed simple injective privacy functions during the experiments, which worked easily for EF with available device resources. Increasing the complexity of the injective privacy functions revealed that computational resources are much more relaxed than memory in terms of feature encoding and selection. Finally, the bandwidth required to transmit private EF over the network for all devices was accommodated in 40% of the available bandwidth. However, several unnecessary features might cause congestion over a network, or in the case of other traffic bandwidth, over 40% of the bandwidth might not be available. Nevertheless, in our experiments, DIGS could dynamically negotiate required NEF for available bandwidth, memory, and computational power.

C. Results for Fitbit Dataset

We first calculate the cost measures such as memory consumption, bandwidth requirements and the processor instructions on the edge device for the non-private version of the Fitbit dataset. The total memory consumed by the non-private data was 247.0 KB. Due to small size of the dataset, the bandwidth required for this dataset on a 100 Mbps 4G network was 0.0198 Mbps. The additional instructions required for this dataset were 0 since the dataset is not private.

Afterwards, we transformed every feature from this dataset to private feature using the privacy encoder injective functions customized for each feature. We calculated the memory required for processing of each feature through relevant injective privacy functions. The total memory consumed through this privacy preservation process for all features was 533.378 KB. After converting the features to privacy encoded features the total private dataframe memory was 1750.369 KB. So if we made all the features of the Fitbit dataset private the total additional memory required is 2283.747 KB. The bandwidth required to transfer this data on a 100 Mbps LAN network would be 0.1827 Mbps. The total instructions required to convert the non-private features to private feature was 252 instructions in Raspberry pi RPi 1 Model A.

Due to the sensitive nature of our data there are four features that we decided to make private to protect our users, denoted as EF. These EF are: *Age*, *Gender*, *Height* and *Weight*, which must be private regardless of resource constraints. We then calculated the resources required by the dataset that contained these four EF as private. In this case we only converted these four EF through their own injective privacy encoder functions, and all other features were kept with their original values. The total memory consumption through the injective function for these features was 103.395 KB. The memory consumed by the dataframe after enforcing privacy only on the EF was 528.727 KB. After adding both memory usage through the injective

privacy encoder functions and the dataframe itself, we got a total of 632.122 KB of memory consumption. The bandwidth requirement would be then 0.0506 Mbps and the instructions required to make only the EF private was 85.

In order to provide additional data privacy, now we want to select more features to be private apart from the user-defined EF. Moreover, we would like to not exceed the maximum resources available to us while encoding the data to preserve user privacy. The available resources for our application on the Raspberry Pi device was 2516.584 KB of memory, after deducting the required memory for making the essential features private we had 1884.462 KB of memory available so we can select additional NEF to be private within our allocated memory. We had enough bandwidth and processor to allocate the processing of making our full dataframe private. We applied DIGS to select the additional features that could be private by using the available resources we have in terms of memory, bandwidth and processor instructions. For Fitbit dataset, DIGS selected a set of combination with 9 NEF features that could be private in addition to the 4 private EF. There were 715 different combinations of features from the 13 NEF. After calculating the total resources required by all the 715 combinations, DIGS selected the most optimal set that required the least resources in combination of memory, bandwidth and processor instructions. With these additional 9 NEF we can now encode a total of 13 features including the EF and provide more privacy.

After converting the DIGS selected NEF to private features including the EF, the total memory required by the 13 private features through the injective privacy encoder functions was 409.241 KB. The memory consumed by the dataframe after the privacy encoding was 1340.657. Adding the two required memories, we got a total of 1749.898 KB of memory required to make the DIGS selected features private. The bandwidth requirement was 0.1400 Mbps and the total number of additional processor instructions required were 216.

We name these four different versions of the dataset as:

- Version 1: Non-private data
- Version 2: Only Σ_{EF} private
- Version 3: DIGS selected $\Sigma_{opt,EF}$ private
- Version 4: All $\Sigma_{EF,NEF}$ private

Dataset	Memory (KB)	Bandwidth (Mbps)	Processor (Instructions)
V1	247.0	0.0198	0
V2	632.122	0.0506	85
V3	1749.898	0.1400	216
V4	2283.747	0.1827	252

TABLE III: Additional resources required for different versions of the Fitbit dataset

Figure 4a displays the increase of memory requirement in KB with the increase of more features being private, Figure 4b and 4c represents the bandwidth requirement in Mbps and the processor requirements in terms of instructions respectively.

We assumed total available memory in the Raspberry pi device after all other factors considered was 2516.584 KB, the available bandwidth was 40000 kbps and available instructions were $4.2M$. DIGS had a great impact on memory, as the device's available memory was smaller. However, we noticed that the privacy encoding process did not have a notable impact on network bandwidth and processor instructions. The selected bandwidth speed was very high for the required bandwidth speed to convert the features to privacy encoded features, as well the processing power of the RPi device was also a lot higher than what was required to run all the injective privacy encoder functions. Hence the major resource saved by DIGS here was in terms of memory.

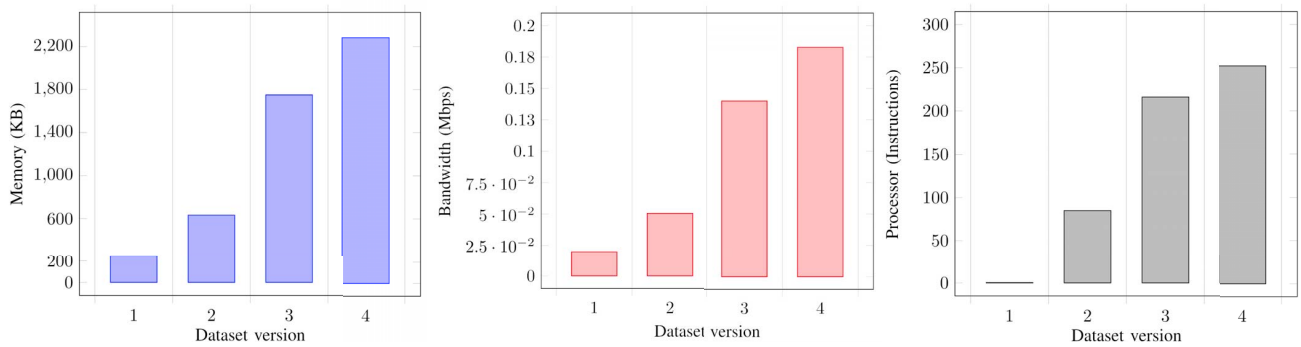
To validate the effect of our privacy preservation method, we ran our ML application on all four versions of the Fitbit dataset. The classification accuracy on each dataset was very close, hence encoding with privacy did not adversely affect the application accuracy. Moreover, for the all private versions of data, we gained a higher accuracy.

Table IV shows the prediction accuracy of different SVM models trained using the four different versions of dataset, the number of private features on each version, the additional memory requirements in order to make the features private, the resources saved when we select the DIGS selected features to be private, and the privacy compromised in terms of number of exposed NEF. We can see that DIGS effectively selected 9 NEF to be private which provides more privacy and also can save 26.21% of memory, 30.5% of network bandwidth, and 16.67% of CPU instructions as compared to making all the features private and overutilizing the resources.

D. Results for GAN Dataset

We discuss the results for GAN dataset in this section. The data processing steps for Fitbit-GAN dataset are the same as that of Fitbit dataset. This dataset is quite large than the Fitbit dataset as it contains 500 users with 33120 aggregated records. The non-private version of the dataset required total 5299.2 KB of memory, the bandwidth required was 0.423 Mbps and the instructions required was 0 as we did not run any injective functions on the non-private dataset. We calculated the additional resources required for this dataset in order to make only the EF private and also to make all the features private. The total memory required for the EF being private only was 11169.24 KB. We ran DIGS with each individual feature's resource cost and noticed that as our current edge device has a low capacity for memory as this large dataset could not be accommodated with the required resource allocation in terms of memory. We still ran our ML application to classify the user's day as "healthy" or "unhealthy" in order to compare the result on this dataset with the original dataset. The accuracy of the SVM on the complete non-private dataset was 98.07% and the accuracy on all private dataset was 98.54%.

In order to verify the scalability of our selected edge device's capacity of memory allocation we ran our experiments on three more different sample sizes of the Fitbit-GAN dataset.



(a) Memory (b) Network Bandwidth (c) Processor Instructions

Fig. 4: Additional resource consumption for increasingly private versions of Fitbit dataset.

Dataset version	Accuracy on SVM	Number of private features	Additional memory required (KB)	Resources saved (%)	Number of compromised private features
V1	96.41%	None	0	100%	All
V2	96.35%	4	385.122	81%	13
V3	96.79%	13	1502.898	26.21%	4
V4	99.62%	All	2036.747	0%	None

TABLE IV: The result of SVM on increasingly private versions of Fitbit dataset, number of private features, percentage of resources saved and privacy compromised features

The first dataset we selected was half the size of the GAN dataset. The dataset now contained 16560 rows of records selected from the large GAN dataset. This dataset required 2649.6 KB of memory for the non-private version of the dataset and the available memory we have for our application was 2516.584 KB. So this dataset was also large for the edge device and we did not continue further with our experiments for the half sized of the GAN dataset.

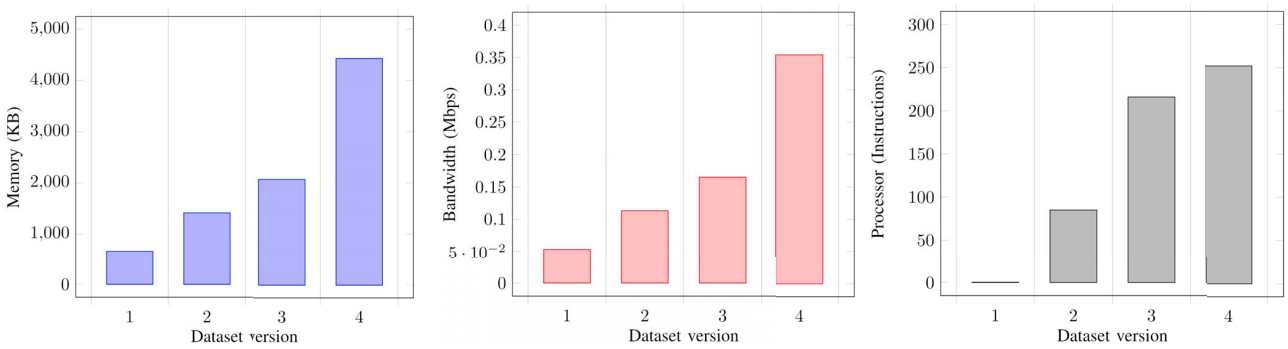
We further downsized to one-quarter of the GAN dataset and the dataset now contained 8280 rows of record that were selected from the large GAN dataset. The memory required for processing this dataset also exceeded the device resource constraints, so we further scaled down to processing one-eighth of the Fitbit-GAN dataset on the edge device.

As can be seen in Figure 5, for the one-eighth size of the GAN dataset, the non-private version required 662.24 KB of memory. This dataset contained 4140 rows of records and was a lot smaller than the original GAN dataset. The additional memory requirement for the injective functions to make the four EF private was 1416.19 KB. Moreover, the required bandwidth for the non-private dataset was 0.0529 Mbps and the additional bandwidth for only the private EF was 0.113. The instructions required for the non-private and essential features being private was 0 and 85 respectively. So now we have 1100.394 KB to allocate the memory requirement for additional features to be private by DIGS. We ran DIGS to select additional features to be private within our allocated resources. DIGS selected three additional features to be private, *Fat*, *Carbs* and *Protein* among the thirteen additional NEF. These three features consumed the lowest resources altogether. We present all the different resource requirements for different versions of this dataset in Table V.

Dataset	Memory (KB)	Bandwidth (Mbps)	Processor (Instructions)
V1	662.24	0.053	0
V2	1416.19	0.113	85
V3	2065.832	0.165	216
V4	4428.896	0.354	252

TABLE V: Resources required for different version of one-eighth sized GAN dataset

We tested the different GAN dataset versions as we did for the original Fitbit data by running the SVM and the results are displayed in Table VI. We notice the trade-off between providing more privacy and resource consumption. If we want to make more private features then we have to allocate more resources such memory in our experiments. If we only make the EF private, then we are compromising privacy for thirteen NEF which would be undesireable as well. By applying DIGS, we were able to select an optimal combination of features to be made private within our available resources while saving extra cost for resources. We were able to save 62.74% memory compared to making all the features private. Also the accuracy on the private versions of the essential features of the data was higher than the non-private version, and the more private DIGS selected private features were a little better than the only EF being private and we got the highest accuracy on the all-private features dataset. One possible reason could be the nature of the supervised classification model combined with the use of one hot encoding for the categorical values, as the SVM only



(a) Memory (b) Network Bandwidth (c) Processor Instructions

Fig. 5: Additional resource consumption for increasingly private versions of Fitbit GAN dataset.

Dataset version	Accuracy on SVM	No. of private features	Additional memory required (KB)	Resources saved (%)	No. of compromised private features
V1	96.83%	None	0	100%	All
V2	97.6%	4	753.95	79.98%	13
V3	97.65%	7	1403.592	62.74%	10
V4	98.80%	All	3766.656	0%	None

TABLE VI: The result of SVM on different dataset for one-eighth GAN dataset, number of private features, percentage of resources saved and privacy compromised features

takes numbers as training and test data. The more privacy we enforced, the more categorical values were created for each feature which give better accuracy for more private data.

V. Discussion

Our proposed algorithm provides a solution for reconfigurable privacy to make more data features private along with the private EF in order to provide maximum privacy to the user. The more privacy, the better it is for any kind of personal data, as data can contain very sensitive information. DIGS performs very well with the real life users' data collected from Fitbit dataset. It has successfully selected 9 NEF to make private in addition to the 4 EF. Out of the 17 features that we have excluding the target feature, "*labels*", we are only compromising privacy for 4 NEF. Moreover, by using the optimal combination of private features selected through DIGS, we can save 26.21% memory, 16.67% processor instructions and 30.5% network bandwidth in comparison with the resources required by all the features being private. In terms of the impact of privacy preservation on ML application accuracy, we noticed that the accuracy of our SVM classification has improved while applying more privacy to the data. This SVM model can help our user to check if the user had a healthy day or unhealthy day and can asses the nutrients value that they consumed and the physical activities that they had done on that specific day to plan a better or healthier lifestyle. Our results prove that our ML application provides more user data privacy while respecting available device resource consumption constraints. Moreover, we observe higher prediction accuracy of the SVM model for the all-private data due to the usage of one-hot encoding on all the input data features.

In order to check the scalability of our application, we experimented with the GAN dataset which contained 500 users and 33120 rows of records. As this dataset was very large as compared to our edge device, Raspberry Pi's available resources were not able to process the privacy preservation for this dataset. The non-private dataset itself was large enough not to fit in the device. Then we decided to test on different sizes of this dataset to check the limit of the device's capacity.

We divided the GAN dataset to its one-eighth size and this dataset was small enough to fit within the resource constraints. However, it was still more than $2\times$ the size of our original Fitbit dataset as it contained 4140 records and the Fitbit dataset had 1663 records. For this dataset, DIGS was able to select 3 additional NEF features to be private in addition to the 4 EF which provided more privacy to the user data. We also could save 62.74% of memory if we compare with all the features being private, even though we are compromising privacy for 10 NEF. But compared to the non-private dataset and only EF being private dataset, with the help of DIGS we were able to make more features private while using the available resources.

The accuracy of the SVM had a similar trend as on our original Fitbit dataset. The SVM trained on the most private dataset version exhibited the highest accuracy due to the one-hot encoding of all the features as they all were categorical after privacy encoding and accuracy gradually improved with more privacy. However in an ideal situation we want all our data to be private and have the highest level of privacy if sufficient device resources were available.

The reason that DIGS could not be applied on a large dataset like Fitbit-GAN was not due to the ML application, but rather it was device specific. We have selected a low-

memory edge device and it was not able to handle large memory consumption required for processing the Fitbit-GAN dataset. For edge devices with more resources, our algorithm would work fine as it worked on the two smaller dataset Fitbit dataset and the one-eighth size of the GAN dataset.

VI. Related Work

Fitness trackers are gaining popularity as they help to maintain user's health and well-being. But in order to provide a fully personalized user experience, these trackers require the users to share their personal information. Sharing personal information becomes a huge concern for the users, as they need to decide whether the tracking devices are a safe platform to share sensitive personal information or not [29]. Data privacy in similar applications and devices has become the biggest concern due to the pervasive nature of huge data collection through different IoT devices and the lack of data security [30]. Various data privacy techniques have been applied to address these concerns and researchers are continuously working on inventing new techniques to provide better protection for the user data throughout the data mining process [29].

Commonly used data protection techniques in the health care sector include k-anonymity [17], l-diversity [18] and t-closeness [19]. The k-anonymity method involves arranging specific columns of quasi-identifiers that are altered or removed, resulting in k rows in the dataset with similar attributes [17]. l-diversity [18] and t-closeness [19] are extensions of the same concept with stronger privacy guarantees. The required modifications are implemented before publishing the data to tackle privacy threats. This is also known as privacy-preserving data publishing. These privacy-preserving techniques work well in general, however, with the increased learning abilities of artificial intelligence-based algorithms, these data protection methods are not enough to reduce various privacy-breaching attacks and threats, as noted in [6], [31].

In the study [32], [33] by Horchidan et al., differential privacy [16] was applied to add noise to the dataset in order to provide privacy to a similar Fitbit dataset. This technique was effective when applied on a large dataset, whereas on a smaller dataset it might produce incorrect results [32], [33]. Orlosky et al. [34] studied accelerometer and pulse rate of 24 users from Fitbit Blaze devices. Their study showed that the accuracy on the Fitbit Blaze is not good compared to the medical grade devices and the users' concern about data privacy is valid due to the intervention of third-party applications.

Providing data privacy has some trade-offs. Dong et al. [35] conducted a research to measure the trade-off between smart grid operations and adversarial inferences about consumer conduct, by considering direct load monitoring of thermostatically regulated loads and investigating how its output degrades as it receives less samples. By providing less samples they wanted to protect the privacy of the data. Their work provided a framework to evaluate the trade-off between utility of the collected data and its privacy preservation.

Reconfigurable or tunable privacy provides user control over the trade-off between the user privacy and other factors such as: access to services [36], data sharing to trusted parties in collaborative computing environments [37], system performance [38] and efficiency [39], [40], model accuracy [41], and data utility [35], [42] and deniability-utility (in case of location-based services) [39].

In our study, we used users' food and activity data and our goal was to make user data as private as possible constrained to device resource consumption. We used generalization techniques in order to provide user data privacy. We also studied the trade-off between provision of privacy preservation and the required additional resource consumption for privacy preservation in a resource constrained environment.

VII. Conclusions and Future Work

We propose DIGS (Dynamic Iterative Greedy Search), a novel mechanism for reconfigurable privacy preservation for ML features on the resource constrained edge devices. DIGS provides reconfigurable privacy by choosing an optimal set of data features to make private provided the device resource constraints. DIGS employs user-defined privacy injective functions to make the data private. We demonstrate DIGS using privacy injective functions employing the anonymization based privacy preservation solutions. Moreover, our privacy preservation mechanism based on user-defined privacy injective functions is flexible as it can cater to any privacy preservation solution as long as each data point is processed individually and the cost for each operation can be computed. Results of our experiments on health care datasets show that for the studied ML application with 17 data features, DIGS is able to select up to 9 additional (non-essential) features apart from the 4 user-defined essential features that must be private and provided additional privacy to the user data, with significant memory savings as well as CPU instructions and network-bandwidth savings as compared to making all the features private. Moreover, the privacy encoded data used in ML applications provides minimal impact on the performance of the model, and in our case, provides even better prediction accuracy due to the use of the one-hot encoding mechanism.

In this work, we have implemented and evaluated a proof-of-concept prototype of our proposed mechanism for reconfigurable privacy in ML. Our research can be extended in multiple directions as it is a novel approach towards privacy provision. The algorithm can be tested on other edge devices such as micro-controllers like Arduino Uno, smartphones, and also advanced variants of Raspberry Pi that have more resources. We can also extend our system by applying ML-based optimization solutions for selecting the most optimal feature set to be private provided the device resource constraints. Furthermore, we can incorporate other techniques for privacy preservation in the injective functions such as k-anonymity, l-diversity, t-closeness, and even stronger techniques for privacy preservation such as differential privacy. Lastly, we can demonstrate the impact of using our system for other kinds of ML applications performing regression or classification while employing different versions of the same dataset with different user privacy preservation levels.

ACKNOWLEDGMENT

The authors are truly grateful to the anonymous participants who volunteered to provide their data for this study.

REFERENCES

[1] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[2] M. Wedel and P. Kannan, "Marketing analytics for data-rich environments," *Journal of Marketing*, vol. 80, no. 6, pp. 97–121, 2016.

[3] H. Dev, T. Sen, M. Basak, and M. E. Ali, "An approach to protect the privacy of cloud data from data mining based attacks," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 2012, pp. 1106–1115.

[4] V. N. Inukollu, S. Arsi, and S. R. Ravuri, "Security issues associated with big data in cloud computing," *International Journal of Network Security & Its Applications*, vol. 6, no. 3, p. 45, 2014.

[5] D. Puthal, S. Nepal, R. Ranjan, and J. Chen, "Threats to networking cloud and edge datacenters in the internet of things," *IEEE Cloud Computing*, vol. 3, no. 3, pp. 64–71, 2016.

[6] S. Imtiaz, R. Sadre, and V. Vlassov, "On the Case of Privacy in the IoT Ecosystem: A Survey," in *2019 International Conference on Internet of Things (iThings)*. IEEE, 2019, pp. 1015–1024.

[7] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," 2015.

[8] F. Kulsoom, A. Vizziello, H. N. Chaudhry, and P. Savazzi, "Joint sparse channel recovery with quantized feedback for multi-user massive mimo systems," *IEEE Access*, vol. 8, pp. 11 046–11 060, 2020.

[9] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci, "Enhancing privacy and preserving accuracy of a distributed collaborative filtering," in *Proceedings of the 2007 ACM conference on Recommender systems*, 2007, pp. 9–16.

[10] R. Shokri, P. Pedarsani, G. Theodorakopoulos, and J.-P. Hubaux, "Preserving privacy in collaborative filtering through distributed aggregation of offline profiles," in *Proceedings of the third ACM conference on Recommender systems*, 2009, pp. 157–164.

[11] I. Mazeh and E. Shmueli, "A personal data store approach for recommender systems: enhancing privacy without sacrificing accuracy," *Expert Systems with Applications*, vol. 139, p. 112858, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419305688

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[13] A. Padron and G. Vargas. Multiparty homomorphic encryption. Online: https://courses.csail.mit.edu/6.857/2016/files/17.pdf.

[14] Privacy by Design — General Data Protection Regulation (GDPR). [Online]. Available: https://gdpr-info.eu/issues/privacy-by-design/

[15] D. Hedin and A. Sabelfeld, "A perspective on information-flow control." *Software safety and security*, vol. 33, pp. 319–347, 2012.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[17] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, pp. 557–570, 2002.

[18] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, p. 3, 2007.

[19] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *23rd International Conference on Data Engineering*. IEEE, 2007, pp. 106–115.

[20] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 571–588, 2002.

[21] guppy3.PyPI. [Online]. Available: https://pypi.org/project/guppy3/

[22] How many calories should you eat per day to lose weight? [Online]. Available: https://www.healthline.com/nutrition/how-many-calories-per-day#average-calorie-needs

[23] How many steps do I need a day? [Online]. Available: https://www.healthline.com/health/how-many-steps-a-day

[24] Nutritionix API. [Online]. Available: https://developer.nutritionix.com/

[25] About Adult BMI:Healthy Weight, Nutrition, and Physical Activity | CDC. [Online]. Available: https://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html

[26] How many calories are in one gram of fat, carbohydrate, or protein? | Food and Nutrition Information Center. [Online]. Available: https://www.nal.usda.gov/fnic/how-many-calories-are-one-gram-fat-carbohydrate-or-protein

[27] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," *CoRR*, vol. abs/1605.09782, 2016. [Online]. Available: http://arxiv.org/abs/1605.09782

[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 27, 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[29] J. Vitak, Y. Liao, P. Kumar, M. Zimmer, and K. Kritikos, "Privacy attitudes and data valuation among fitness tracker users," in *International Conference on Information*. Springer, 01 2018, pp. 229–239.

[30] E. Bertino, "Data privacy for IoT systems: Concepts, approaches, and research directions," in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 3645–3647.

[31] P. Zhao, H. Jiang, C. Wang, H. Huang, G. Liu, and Y. Yang, "On the performance of k-anonymity against inference attacks with background information," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 808–819, 2019.

[32] S.-F. Horchidan, "Real-time forecasting of dietary habits and user health using federated learning with privacy guarantees," 2020.

[33] S. Imtiaz, S.-F. Horchidan, Z. Abbas, M. Arsalan, H. N. Chaudhry, and V. Vlassov, "Privacy preserving time-series forecasting of user health data streams," in *IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 3428–3437.

[34] J. Orlosky, O. Ezenwoye, H. Yates, and G. Besenyi, "A look at the security and privacy of fitbit as a health activity tracker," in *Proceedings of the 2019 ACM Southeast Conference*, 04 2019, pp. 241–244.

[35] R. Dong, L. J. Ratliff, A. A. Cárdenas, H. Ohlsson, and S. S. Sastry, "Quantifying the utility–privacy tradeoff in the internet of things," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 2, May 2018. [Online]. Available: https://doi.org/10.1145/3185511

[36] L. M. Aiello and G. Ruffo, "LotusNet: Tunable privacy for distributed online social network services," *Computer Communications*, vol. 35, no. 1, pp. 75–88, 2012.

[37] G. Skinner, "Dynamic user reconfigurable privacy and trust settings for collaborative industrial environments," in *INDIN'05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.* IEEE, 2005, pp. 761–766.

[38] G. Ghinita, K. Nguyen, M. Maruseac, and C. Shahabi, "A secure location-based alert system with tunable privacy-performance trade-off," *GeoInformatica*, vol. 24, no. 4, pp. 951–985, 2020.

[39] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen, "Billion-scale federated learning on mobile clients: a submodel design with tunable privacy," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[40] R. Narendula, T. G. Papaioannou, Z. Miklós, and K. Aberer, "Tunable privacy for access controlled data in peer-to-peer systems," in *2010 22nd International Teletraffic Congress (lTC 22)*. IEEE, 2010, pp. 1–8.

[41] E. Duriakova, E. Z. Tragos, B. Smyth, N. Hurley, F. J. Peña, P. Symeonidis, J. Geraci, and A. Lawlor, "PDMFRec: a decentralised matrix factorisation with tunable user-centric privacy," in *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019, pp. 457–461.

[42] J. Liao, O. Kosut, L. Sankar, and F. du Pin Calmon, "Tunable measures for information leakage and applications to privacy-utility tradeoffs," *IEEE Transactions on Information Theory*, vol. 65, no. 12, pp. 8043–8066, 2019.