Policy-Based Federated Learning

Kleomenis Katevas
Telefonica Research
kleomenis.katevas@telefonica.com

Eugene Bagdasaryan
Cornell Tech
eugene@cs.cornell.edu

Jason Waterman
Vassar College
jawaterman@vassar.edu

Mohamad Mounir Safadieh
Vassar College
msafadieh@vassar.edu

Eleanor Birrell
Pomona College
eleanor.birrell@pomona.edu

Hamed Haddadi
Imperial College London
h.haddadi@imperial.ac.uk

Deborah Estrin
Cornell Tech
destrin@cs.cornell.edu

ABSTRACT

We are increasingly surrounded by applications, connected devices, services, and smart environments which require fine-grained access to various personal data. The inherent complexities of our personal and professional policies and preferences in interactions with these analytics services raise important challenges in privacy. Moreover, due to sensitivity of the data and regulatory and technical barriers, it is not always feasible to do these policy negotiations in a centralized manner.

In this paper we present *PoliFL*[1], a decentralized, edge-based framework for policy-based personal data analytics. PoliFL brings together a number of existing established components to provide privacy-preserving analytics within a distributed setting. We evaluate our framework using a popular exemplar of private analytics, Federated Learning, and demonstrate that for varying model sizes and use cases, PoliFL is able to perform accurate model training and inference within very reasonable resource and time budgets.

KEYWORDS

Distributed Systems, Privacy Policy, Use-Based Privacy, Federated Learning

1 INTRODUCTION

Increasingly ubiquitous sensing and data collection through our personal devices and our surrounding environment has created the potential to offer innovative services and applications, while also creating significant value from our personal data. The data collected from our smartphone applications and Internet of Things (IoT) devices in our homes and workplace, and the resulting inferences such as our behavioral patterns, preferences, activities, and personal relationships, are often used for a variety of personal and public Machine Learning (ML) models, such as building targeted advertising, content recommendation, health analytics, or mass surveillance.

Today's ML-based systems tend to follow a common pattern: data are collected from various data sources (*e.g.*, mobile sensors or through user interactions), then aggregated and processed centrally, typically on some kind of cloud-hosted service. While successful, this approach creates several challenges: *trust*, as the user requires guarantees that the provided data will not be misused by the centralized service provider; *security*, as unauthorized disclosure of these rich data resources [29] leads governments to bolster consumer privacy protection through regulation (*e.g.*, EU [19], US [48], and Japan [46]); and *scalability*, where the costs of building, maintaining, and managing large data-centers required by current approaches become prohibitive as the volumes of data are increasing dramatically, *e.g.*, some estimates suggest that sensors from individual autonomous vehicles could generate terabytes of data daily [38]. More recent approaches suggest the concept of *edge computing*, where data is processed as close to the source of data as possible, *e.g.*, at the 5G base station or on the users' devices.

While these approaches solve some of the aforementioned problems, decentralization raises new *data-access* challenges as users are still faced with an "all or nothing" data access decision (*i.e.*, either unrestricted access to the data source or strict denial of access which often entails denial of key features). For example, in the mobile ecosystem, currently led by Android and iOS, each app can request unrestricted data access to a specific source (*e.g.*, Location) and the user can either accept or deny this request. Moreover, there is no accountability the system will use the data as intended. For instance, a music provider might want to restrict its services by interpreting location data in a country-level granularity, however the user must simply trust that the provider will not misuse this open-access permission for other needs at the future (*e.g.*, selling the data to other companies interested in personal advertising).

To tackle this challenge, we present *PoliFL*, a scalable, context-based approach to personal data processing. Rather than centrally collecting data from a user population for later processing, PoliFL moves the required computation to the network edge where data logically resides with the data subject. This model is a better fit for current privacy and data processing regulation and allows data subjects to bear more of the cost of providing storage, computation and connectivity. In contrast to traditional data permission management systems where data access is controlled using an "all or nothing" concept, our privacy management component extends Ancile [3], a framework for contextual integrity that reframes privacy as prevention of harmful uses and imposes restrictions on data usage accordingly.

[1] https://github.com/minoskt/PoliFL

Our design addresses the previous issues by deploying the Databox platform [13], a user-controlled platform that provides containerized and access-controlled personal data storage at the edge. We further deploy *Central Service*, a centralized node that coordinates data access at each user-controlled deployment (*i.e.* Databox). The Central Service is responsible for receiving data processing requests from third-party services. It handles these requests by coordinating with the edge devices that contain the data. The Central Service transfers the request along with the associated data policy to the participating user's Databox, where data are accessed and processed locally, in accordance with the data policy. If authorized by the policy, the processed results (*e.g.*, filtered data, inference outcome or locally trained model), are sent back the Central Service, which processes the results from all the users and sends them back to the interested provider.

We evaluate our system under different combinations of distributed training: (i) *centralized training*, (ii) *federated learning*, and (iii) *differentially private federated learning* (Section 4). We show that using an ML-based task, the model converges even when choosing different aggregation methods and policies. We further evaluate the performance of PoliFL in terms of internal (*i.e.*, edge-device) and external (*i.e.*, system) overhead, by simulating three different policy execution tasks distributed to 312 users (Section 5).

Our system can be beneficial to various use cases where third party providers are only interested in specific features from the user data (*e.g.*, a health-related provider might be interested in the broad statistics of the trends in the steps a user has made per week), require data access with less granularity (*e.g.*, a bank might only be interested in the city that a transaction took place to permit it), or use a locally trained model of each user (*e.g.*, a movie provider might integrate users personalized movie recommender models into its global model using federated learning).[2]

This paper makes three key contributions:

- We present PoliFL, an open-source privacy-aware personal data platform that mediates third party services' access to an individual's personal data via verified, audited, and policy-controlled programs.
- We evaluate the platform for managing secure access of users' personal data.
- We extend the Ancile Policy Management framework into a distributed privacy framework for applications.

2 MOTIVATION AND BACKGROUND

Recently, there has been an increasing demand in privacy-preserving content personalization and sensitive data analysis. Examples include the introduction of Federated Learning by Google [11], requests for sensitive image analysis and model building by Facebook against revenge porn,[3] or on-device behavioral intervention against bulling for kids by the BBC *Own It app*.[4] In all these cases, sensitive personal data needs to be used for building personal local models based on individuals' policies. In this section we investigate some potential use cases for our proposed system.

[2]This work does not raise any ethical issues and an ethics approval from our institution was not needed.

[3]https://www.independent.co.uk/life-style/nudes-facebook-revenge-porn-naked-pictures-send-a9207941.html

[4]https://www.bbc.com/ownit/take-control/own-it-app

2.1 Privacy at the Edge

There has been a large body of research demonstrating how *Edge Computing* and local analytics can help in preserving data privacy while enabling accurate analytics [10, 13, 21]. A number of recent industry efforts have also adopted this model for performing accurate analytics while respecting privacy. For example, the Brave browser delivers personalized adverts based on analytics performed locally on users' devices [23, 47]. The BBC has also developed the BBC Box[5] for private and secure aggregation of data and content personalization based on the Databox (discussed in Sec. 3). Recently, large service providers like Google and Apple have adopted techniques such as Differential Privacy [17] and Federated Learning [10, 30] to deploy scalable algorithms for on-device analytics within highly sensitive tasks.

2.2 Privacy Permissions

Typically, mobile and IoT systems have requested that users grant unrestricted access to data sources, such as mobile sensors, location, contacts or calendar. More recently, the concept of app-based permission was introduced, where each application requested permission to access a data source, either before an app was installed (Android until v6.0), or at the moment an app feature requires such access (iOS, Android v.6.0+).

In these access-based systems, consent is typically given at a very coarse granularity (*e.g.*, all-or-nothing). Once an application has access to a sensor or data source, the user has little or no control or visibility into how that data is used. Consequently, some developers were designing attractive features (*e.g.*, customized weather forecasts based on current location) in order to persuade users to grant access to their sensitive data, while at the same time collecting this data for other purposes (*e.g.*, analyzing the data for hedge funds)[6]. Recent versions of Android and iOS do have privacy settings that allow a user to set permissions such that applications can collect data only while in use. However, going forward to support data-rich applications that respect users' privacy, we must look to more robust privacy frameworks, such as contextual integrity and use-based privacy.

2.3 Contextual Integrity

Contextual integrity [39] looks at privacy through the lens of social norms of information flows from one party to another. These information flows are defined along five different parameters: the *data subject*, the *data sender*, the *data recipient*, the *information type*, and the *transmission principle*. For example, in a machine learning system, a smartphone user (the data subject and data sender) might be comfortable with sending samples of their typed text (the information type) to a service provider (the data recipient) for the purpose of improving keyboard prediction (the transmission principle), but not for any other purpose (such as detecting the user's mood). While contextual integrity is a useful tool in analyzing and discussing privacy, the challenge is how to implement such a privacy preserving framework in a system that enforces these policies in real-time.

[5]https://www.bbc.co.uk/rd/blog/2019-06-bbc-box-personal-data-privacy

[6]https://www.nytimes.com/interactive/2018/12/10/business/location-data-privacy-apps.html

2.4 Use-Based Privacy

Use-based privacy [7, 8] defines privacy as a prevention of harmful uses. Data are associated with policies that authorize certain types of data use while denying other uses. These policies are reactive, as they can change as data is transformed. For example, a simple use-based policy around location data is allowing the current smartphone location to be viewed by a weather application only at the granularity of a city. The data in question is location, and the "use" is being able to view the data. This use is only authorized once the precise GPS location has been transformed to a city-level granularity.

Policies may also change due to external events. A use-based location policy might state that a user's location may only be shared while the user is driving a company vehicle. The "use", whether the location data can be shared, changes based on the event of driving the company car. As these policies are reactive, a run-time policy monitor, such as Ancile [3], is needed for enforcement.

2.5 Use Cases

We present the following use case, building a video recommendation system based on user preferences, which illustrates how PoliFL can be used as a secure and distributed solution that protects a user's privacy when dealing with personal data. Traditionally, it would require a central cloud-based system that collects the user's viewing preferences (*e.g.*, watched movies, rankings, watching time, etc.) and either trains a new model, or contributes to existing models for recommending movies to the customers.

In this work, we assume that each *user* (*i.e.*, customer) has deployed an edge device (*e.g.*, a smartphone, or a Databox [13] that stores data related to their movie preferences. The *provider* (*i.e.*, the movie company) does not have direct access to the collected data; rather it uses the Central Service to distribute a use-based policy for execution on the edge device. Under those assumptions, we report on two different approaches that our system could use to protect the user's privacy, while also allowing the provider's recommendations to benefit from the user's viewing preferences:

Data Filtering uses filtering policies to obfuscate/remove sensitive data about user behavior, only allowing the transformed data to be sent to the provider's server. The provider can use the filtered data for training a global model for movie recommendation.

Federated Learning is a novel technique to perform privacy preserving distributed training [10, 30]. It allows a global provider to distribute an ML model to different users for training on local data, and then aggregate the resulting models into a single powerful model without observing the users' private data. We propose a policy that enforces Federated Learning, allowing locally trained models to be released to the provider. Each Databox trains a personalized model locally and our system enforces the policy requiring Differential Privacy [1, 36] to be applied to hide private information from the training set. Finally, a global model is produced in a central node (*i.e.* Central Service) and sent back to the provider.

3 PLATFORM OVERVIEW

Figure 1 gives an overview of PoliFL, which consists of multiple edge devices running the Databox platform and a Central Service used for device coordination. We assume that a third-party service is interested in an outcome (*i.e.*, a generalized ML model) from a collective personal data processing. This service makes a request to Central Service, which is responsible for maintaining the users' data policies and coordinating the communication between the service and the edge devices, where the data is located. We also assume that each device contains one instance of the Databox platform per user and ideally has the form of a physical device (*e.g.*, located at the user's house). Each Databox also includes an Ancile deployment, providing a local use-based policy management system which ensures all edge computation is done in a policy compliant manner. Results computed from the Databox are sent back to the Central Service, which also contains an instance of Ancile for executing policies globally on the received data before sending any results to the third-party service.

3.1 The Databox Platform

The Databox [13] is an open-source platform that enables data subjects to manage controlled access by third parties to their personal data. It mediates access to and, where necessary, collects data from local and remote sources, ranging from online social networks to IoT sensors. It allows data subjects to inspect data gathered from their data sources, and to effect actuation of IoT devices. It provides data processors with specific, limited, logged access to subjects' data by allowing data subjects to select, install, and run applications published by those data processors.

The platform uses Docker containers [16] as a way to bundle application code and associated environment (libraries, etc.) in a portable image format that can be instantiated on a wide range of host operating systems, including macOS and Windows in addition to Linux. Docker configuration enforces isolation between running containers, preventing container access to network connectivity, ensuring that communication between containers must take place using the Databox RESTful APIs over HTTPS using SSL/TLS certificates distributed to each component on launch. This also makes it straightforward to support hybrid models where some components run on the device in the home, and some run remotely in the cloud.

The main components of the Databox architecture (presented in Figure 3) are:

Stores, associated with drivers and apps, that provide the API through which other components interact with drivers and apps. They authenticate and authorize access to data, which they may also retain locally. Request authorization is obtained by the store successfully verifying a bearer token presented by the other party. The API supports a range of data types, including time series and key-value data, accessed via polling or a websocket-based streaming API.

Drivers represent data sources, and are responsible for all interactions with the data source, IoT device or online service. A driver's function is to perform the tasks needed to import data into the Databox, and to provide an API to actuate IoT devices if applicable. Data imported by a driver is written to a store. Drivers provide data in response to reads to their associated store and invoke actuation of devices in response to writes to their associated store.

Apps represent data processors in Databox, obtaining access to data at install time and subsequently processing that data. They may publish processed data through their own store to make it

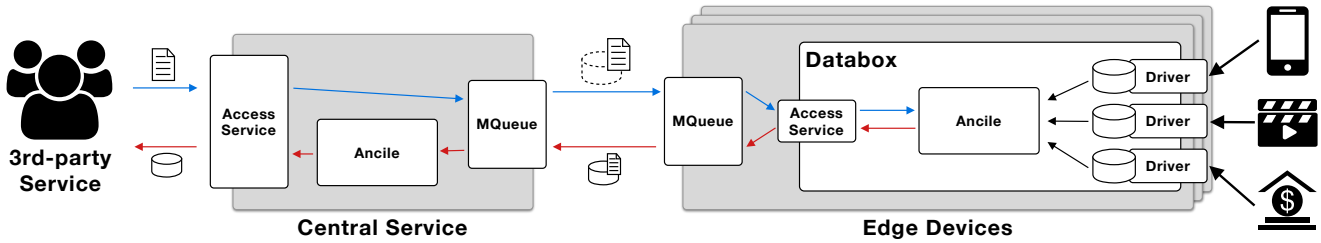


Figure 1: Platform overview: A 3rd-party service sends a policy using the Central Service (*e.g.*, an instruction to receive an ML model trained with multiple users). The policy, together with an initial shared model, is distributed among all registered devices and is executed using the Ancile use-based privacy framework (*e.g.*, train a model and apply differential privacy). The data, certified with the attached policy, is then returned to the Central Service and summarized by Ancile using the post-hoc instruction included in the policy (*e.g.*, average the model in a federated learning approach). The final result is finally returned to the interested 3rd-party service.

available to other apps. Apps cannot communicate with anything other than stores for which they have valid tokens plus the Arbiter from which they may request new tokens.

For code in a container to access personal data requires relaxing a "default deny" configuration, done on the basis of user-authorized permissions represented as bearer tokens granted by the system to the container in question. Each Databox container comes with a manifest that outlines the permissions it requires, which is translated on installation into a *service level agreement* (SLA) encoding the granted data access and export permissions, which the platform then enforces. We address the limitations of such an "all-or-nothing" permission-based system by incorporating Ancile use-based policy management system explained below.

3.2 The Ancile Platform

Ancile [3] is a trusted computing environment that allows third-party services to perform policy compliant data computation. It executes submitted programs in a trusted environment on behalf of a service while enforcing use-based privacy policies on data. It achieves policy enforcement by extending programs to operate on policy-tagged values known as a *Data-Policy Pair*.

A Data-Policy Pair contains two restricted fields: `_data` and `_policy`. The `_data` field contains a stored value whose use is governed by the policy represented in the `_policy` field. To prevent programs from directly manipulating data or policies, submitted programs are compiled with *RestrictedPython* [43] before execution. RestrictedPython limits the application's program to predefined Ancile commands and prevents access to internal data structures by transforming the code before compilation and raising errors if a program attempts to access any protected data fields (marked with a leading underscore) or attempts use any unsafe language feature.

In particular, compilation with RestrictedPython guarantees that Data-Policy Pairs are opaque to the submitted program and their internal fields can only be inspected or manipulated by Ancile. Therefore, the only way for a program to interact with a Data-Policy Pair is through a predefined library of Ancile commands, *Ancilelib*. A call to an Ancile command invokes Ancile's reference monitor, which checks for policy compliance before the command is executed. If the existing set of commands is not sufficient, developers may also add their own commands to Ancilelib, once those commands have gone through an approval process by the Ancile Administrator.

Policies are specified as regular expressions over an "alphabet" of commands. Policies define a state transition diagram, specifying how data values and any derived values may be used. Ancile commands operate on Data-Policy Pairs, which transform the data and advance the policy in the Data-Policy Pair. For example, command `fetch_data` has to be followed by the `filter` command before it can be publicly released, thus the policy will look like:

```
fetch_data . filter . return
```

Ancile effectively supports use-based privacy, however, it runs on a centralized server so all computations must be done centrally. While this can be a valid solution for enforcing polices at an institution or enterprise level, it does not scale to more decentralized authority. Our work extends Ancile to support policy-based private analytics by running a deployment of Ancile on each edge device and coordinating via the Central Service.

3.3 Central Service

The Central Service is responsible for maintaining data policies and coordinating the communication between a third-party service provider and all interested devices. It runs as a web service and receives and executes programs on behalf of third-party services. These services communicate with the Central Service by making requests for data through the Access Service module. The Central Service runs an instance of Ancile, which maintains data policies for all users in the system. Users have data policies for each service they are interested in using and have a web-based dashboard on the Central Service to view, add, delete, and modify these policies.

When a third-party service requires users' data from one or more edge devices, the service sends a request with the following elements:

- **Application Token:** A secret token that is used to authenticate the service to the Central Service.
- **Global Program:** This program makes requests to the edge devices for data. It also aggregates and filters the results from the edge nodes. The result of this program, if policy compliant, will return data to either the edge devices or the service who made the request.
- **Local Program:** A piece of computation to be executed on the edge devices and whose result, if policy compliant, will be returned to the Central Service.

The Central Service handles each request by first validating the service's token. Once the service has been validated, Ancile executes the Global Program. The Global Program then selects the participants and sends the Data-Policy Pair along with the Local Program to the selected users on the edge devices.

Once the edge device receives the request from the Central Service, it is executed on the Ancile instance of the edge device. The edge device can then send Data-Policy Pairs back to the Central Service.

To demonstrate the operation of the Central Service, we assume that a service wants to perform a typical federated learning task:

(1) An initial shared model trained with data from public sources is created on the Central Service.
(2) The Central Service picks a subset of all participants and sends them a copy of the shared model.
(3) Each selected participant personalizes the model with their own data.
(4) The updated models are sent back to the Central Service, and combined to create a new shared model.
(5) Steps 2-4 are repeated in as many rounds as needed.
(6) The final model is either distributed to the edge devices for local use or is returned to the third-party service that made the request.

In the example above, a Global Program is used to create the initial model, selecting the participants for each round of training, sending the model to the participants, and combining the results of each round to build a new model. Since the initial model was created from public sources, the policy is permissive, with no restrictions on how the model can be used. The Local Program fetches the user's data from the Databox, updates the local model, and returns the updated model to the Central Service (if authorized by the local version of Ancile running on the edge device).

In this example, the edge devices would send back an updated model Data-Policy Pair trained on the local edge device data. These Data-Policy Pairs from all the participants can be combined on the Central Service using Ancile's aggregation functions. Aggregation functions take in multiple Data-Policy Pairs and return a new transformed Data-Policy Pair with an intersection of the policies from all the source Data-Policy Pairs. If all the Data-Policy Pairs have the same policy, a common case, the intersection of the policies reduces to a single instance of the policy. This new model Data-Policy Pair can be distributed to another subset of participants. These steps are repeated until sufficient accuracy is achieved and the model is released to either the edge nodes for local use, or returned to the third-party that made the request. Note, that if the model is returned to the third-party service, the data are no longer under any policy control.

3.4 Threat model

As demonstrated above, our system operates in the similar setting assumed by the previous work on federated learning [11, 35] where an algorithm is trained among multiple edge machines without exchanging their data samples, but restrict our assumptions about the global provider to enable access to federated learning by third-parties. Our key distinction is that the provider that requests training on the user's data can provide different programs and models but has to satisfy particular policies defined by the participants on their data. Unlike the case of federated learning for Google's keyboard prediction [25], we consider the model where the provider requesting a model does not own the infrastructure, such as BBCBox [5] or OpenMined [40] which aims to open federated learning to external providers. In this work, we make the following assumptions:

- Users trusts their local device (*i.e.*, the Databox) for storing their personal data in raw format.
- We only guard what is protected under the policy.
- Data can only be considered as trusted when a policy accompanies it.
- Once the policy-allowed transformed data leaves PoliFL, we have no control over the data anymore and could be used for any purpose.
- Policies are developed in good faith to protect access to data and the system administrator will inspect and explicitly authorize policies before they are used.

Based on these assumptions, we consider the following threat model:

(1) An attacker can be "honest but curious", meaning that in case they acquire access to the Central Service, they will not be able to manipulate the user policies.
(2) An attacker cannot access the user's device.
(3) An attacker can submit application programs that attempt to use more data or violate policies.

The last point is important in Federated Learning as we assume that participants themselves can access and modify their devices. It's possible that malicious participants can compromise the overall training by injecting backdoors, but this can be mitigated by using differential privacy [36] at expense of losing some accuracy [4]. The application can decide to utilize differential privacy to guarantee both privacy of users' contributions and safety from the backdoor attack.

4 POLICY-BASED FEDERATED LEARNING

To evaluate our system, we pick a domain of training a machine learning model on users' data. We consider a centralized training task at the Central Service that fetches data from the edge devices. However, transferring sensitive data over the internet and storing it in the Central Service adds additional privacy risks. As an alternative to central processing we try Federated Learning [30] that attempts to build a single model by aggregating models obtained by local training on users' data. Besides traditional federated learning,

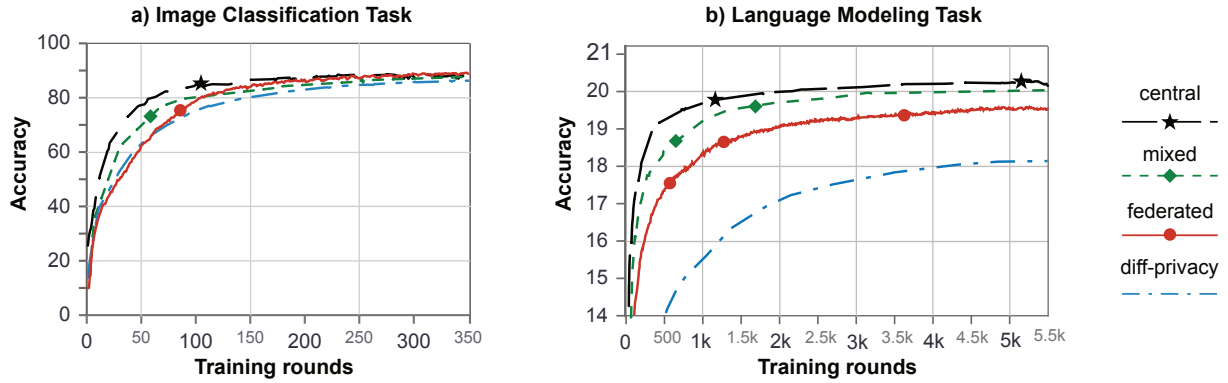


Figure 2: Evaluation of different training strategies. Image classification (a) doesn't suffer from using federated training, whereas language modeling task (b) benefits from centralized training. Differential privacy impacts more the language model than the image model.

as introduced by McMahan *et al.* [35], we consider adding differential privacy [17]. Recently proposed Differentially Private federated learning [36] enhances learning underlying distribution with privacy guarantees. Furthermore, other aggregation techniques such as median and trimmed mean aggregation [9, 14] promise better defenses against privacy and security attacks [4, 6]. However, prior work has not addressed the constraint that users' might have different preferences over their privacy and thus could allow or forbid centralized training. In this section we investigate the combination of multiple distributed training approaches where different participants have policies preferring one way over another. For instance, we consider that some portion of the participants permit centralized training, whereas other part only allows federated learning. We show that a resulting model still converges to the appropriate solution even when there exist multiple different aggregation methods and policies.

4.1 Training modes

We support three initial policies that users can assign to their data on the edge device: (i) *centralized training*, (ii) *federated learning*, and (iii) *differentially private federated learning*. These policies can be extended to support various techniques, such as byzantine tolerant federated learning [9].

Centralized training is the simplest non-federated case, where the single model is trained on the Central Service after the data is pulled from each participant. This setting assumes that the sensitive data will leave the user's device and travel to the Central Service which could introduce additional privacy risks. Although, this approach sends data over the network, it reduces computation on participant's device and offloads it to the cloud. Algorithm 1 shows a simple function that implements central training given some model G^0 and aggregated user dataset $\mathcal{D}_{central}$.

Federated Averaging [35] randomly selects a subset of m participants S_m and sends them the current joint model G^t in each round t. Choosing m involves a trade-off between the efficiency and the speed of training. Each selected participant updates this model to a new local model L^{t+1} by training on their private data and sends the difference $L_i^{t+1} - G^t$ back using Algorithm 2. Communication overhead can be reduced by applying a random mask to the model weights ([31]). The central server averages the received updates to create the new joint model:

$$G^{t+1} = G^t + \frac{\eta}{n}\sum_{i=1}^{m}(L_i^{t+1} - G^t) \quad (1)$$

We use Algorithm 2 for local training to produce the update $L_i^{t+1} - G^t$. Received models in a single round are accumulated in a temporary storage to reduce memory consumption using Algorithm 4. Once the updates are received, Algorithm 5 implements averaging and creates the model G^{t+1}.

Differentially Private Federated Learning – Recent work [36] presented differentially private federated learning on a participant level. Differential privacy establishes that the model with and without the user's contribution would produce similar results, thus providing a user with privacy over participation. The implementation is an extension of Equation 1 where each participant's update is *clipped* to the norm S, *e.g.*, multiplied by $\max(1, \frac{S}{||L_i^{t+1} - G^t||_2})$, to bound the sensitivity of the updates. Additionally, Gaussian noise $\mathcal{N}(0, \sigma)$ is added to the weighted average of updates:

$$G^{t+1} = G^t + \frac{\eta}{n}\sum_{i=1}^{m}(\texttt{Clip}(L_i^{t+1} - G^t, S) + \mathcal{N}(0, \sigma))$$

.

4.2 Policy Controls

Each approach above can be covered by the corresponding policies that enforces the certain type of training without restricting the model architecture or training parameters. The Central Service requires certain functions defined in collaboration with the third-party service, *e.g.*, training, data fetching and aggregation (see Algorithms 1-5). Ancile supports scoping and thus we combine multiple training methods under a single umbrella call **`train`**. We further introduce a function **`get_data(data_type)`** that fetches corresponding data from Databox deployed by the participant. Although the Ancile framework allows to control parameters of the executed functions, we omit function parameters for simplicity.

Algorithm 1 Central training. T - number of training rounds, lr - local learning rate

function TRAIN_CENTRAL(model G^0, data $\mathcal{D}_{central}$)
  for round $t \leftarrow 1, T$ **do**
    for batch $b \in \mathcal{D}_{central}$ **do**
      $G^t \leftarrow G^t - lr \cdot \nabla\ell(G^{t+1}, b)$
    end for
  end for
  return G^T
end function

Algorithm 2 Local training. E - number of local epochs

function TRAIN_LOCAL(model G^t, data $\mathcal{D}_{local}$)
  # Initialize local model L for time t+1
  $L^{t+1} \leftarrow G^t$
  for epoch $e \leftarrow 1, E$ **do**
    for batch $b \in \mathcal{D}_{local}$ **do**
      $L^{t+1} \leftarrow L^{t+1} - lr \cdot \nabla\ell(L^{t+1}, b)$
    end for
  end for
  return L^{t+1}
end function

Algorithm 3 Local training with Differential Privacy. S - clipping bound, σ - amount of added noise

function TRAIN_LOCAL_DP(G^t, $\mathcal{D}_{local}$):
  # Perform normal local training
  $L^{t+1} \leftarrow$ `train_local`$(model, \mathcal{D}_{local})$
  return `Clip`$(L^{t+1}, S) + \mathcal{N}(0, \sigma^2)$
end function

Algorithm 4 Collect received models after one round of training. *tmp_sum* - stores received models and initialized with zeros.

function ACCUMULATE(model L, *tmp_sum*)
  for name n, param $p \in L.parameters()$ **do**
    $tmp\_sum[n] += p$
  end for
  return *tmp_sum*
end function

Algorithm 5 Model update method. $total\_n$ - total number of participants, η - global learning rate

function AVERAGE(*model*, *tmp_sum*)
  for name n, param $p \in model.parameters()$ **do**
    $update = \eta / total\_n \cdot (tmp\_sum[n])$
    $param.add\_(update)$
  end for
  return *model*
end function

Corresponding policies and programs – We propose to use the following policies to control the usage of the participant's data:

```
get_data . train_local . accumulate*
.(train . accumulate* + average)*.return
```

This policy, first, ensures that user's data is only used in local training on the device. Second, it specifies that the result of local training can be combined with other models and the resulting model can participate in future iterations of the federated learning, before being returned to the third-party provider. This policy permits execution of the program in Algorithm 6.

By using iteration in the policy, we support multiple rounds of training using the combination of policies. Function **`train_local`** takes both the current model G^t with policy P_{model} and local user data $\mathcal{D}$ with policy P_{data} and produces a new model L^{t+1} with the policy $P_{model}\&P_{data}$. Similarly, functions **`accumulate`** and **`average`** combine two policy-controlled objects. We prevent explosion of policy size by using simple reduction rules that keep the policy size constant [3].

For the participants who prefer having differential privacy enabled we use method **`train_local_dp`** instead of **`train_local`**. Furthermore, if the participants allow sharing data with the Central Service they can adapt the simpler policy but use a **`filter`** command to enforce removal of sensitive data.

```
get_data . filter . train_central* . return
```

As stated above, the third party can create their own model and distribute it to the participants along with the code that should satisfy the associated policies. The Central Service will fetch the corresponding policies and execute that program. As we stated above, policies are public, and the provider can design programs that satisfy them.

4.3 Experimental Setup

Hardware We perform experiments on two different tasks to demonstrate the feasibility of our approach. We focus on training Federated Learning models for both image and text tasks, similar to the original FL paper [35]. Each experiment is run sequentially on 2 NVidia TitanX GPUs in order to speed up training and obtain the convergence results. As code and library implementation is the same, we expect the same results to hold when run on an edge device. We provide performance evaluation of the edge device in the following Section 5.

Language modeling – Typed text can include sensitive information. Distributed training can help conceal the data by releasing only trained models instead of raw text [25]. We use our system to train a word prediction model using the Reddit dataset [35]. This scenario is reasonable in the Databox setting, where the user might want to train a predictive model not only based on mobile keyboard, but also using activities at the personal computer or laptop.

We use 80,000 posts from the public Reddit dataset considering users that have between 150 and 500 posts with 247 posts each on average. The task is to predict the next word given a partial word sequence. Each post is treated as a training sentence. We use a two-layer, 10M-parameter sample LSTM model [42] with 200 hidden units. An input is split into a sequence of 64 words. For participants local training, we use a batch size 20, learning rate of 20, and the SGD optimizer. We run 5,500 rounds of training and picking 100 participants every round. For differential private training we follow

parameters from [36] and set the clipping bound $S = 15$ and noise $\sigma = 0.01$.

Image classification – A user might decide to store photos or videos on the local device but still decide to contribute to the global recognition model. We use CIFAR-10 image recognition [32] task as another scenario of use for Federated Learning. We split the data among 100 participants using Dirichlet distribution with $\alpha = 0.9$. We use ResNet18 model [27] with 1.2mln parameters, batch size 32, and learning rate 0.1. We run 350 rounds of training selecting 10 participants each round. For differentially private federated learning we similarly set $S = 15$ and $\sigma = 0.01$.

4.4 Convergence results

We want to check whether the combination of multiple different modes of training is capable of improving the performance of the model. We perform training of three models: centralized, federated, and do a combination of two of them. We observe that the centrally trained model is capable of achieving higher performance on the Reddit dataset, but does not outperform the FL model on image classification data.

We then perform training of the mixed model, where some participants can have a policy permitting data release, and others only allow federated training on their data. For the mixed model we assign half of the users a federated policy and another half a centralized training policy. We implement it by sampling each round of training from the group of users who are either locally trained or centrally trained. Thus, we mix the rounds of central and federated training.

Figure 2 demonstrates that the language task on the Reddit data can greatly improve by using some participants who are willing to train on their data centrally, thus achieving improvement over the federated model. The Reddit dataset effect is profound, but it does not outperform the FL model on image classification data. We speculate that this could be caused by the fact that the Reddit data is unique to each participant whereas CIFAR dataset is split by randomly taking subparts of the dataset.

We ran training of a differentially private model and achieved accuracy lower than non-differential approach. The effect of clipping and noise as methods to restrict update contributions by a single user could significantly diverge or slow down training and affect. We do not combine the DP training to the centralized or federated models, as this combination might break the privacy guarantees of the final model. However, combination of protected and less-protected data with meaningful guarantees could be a promising direction.

We demonstrate that the policy-controlled machine learning can converge for different modes of training and even in the mixed scenario when participants have different policies preferring centralized and federated scenarios. Provided policies are compact and control the programs submitted by the third-party allowing only the trained model to be returned back.

5 SYSTEM PERFORMANCE

In this section, we seek to answer the following questions:

- What is the internal overhead of the edge components of the system?

Algorithm 6 Sample program sent to Central Service. *users* - list of participant's names and their addresses. `RemoteCall` - asynchronous remote execution service.

```
function REMOTE_PROG(dpp, model)
    data = get_data(dpp, model)
    model = train_local(model, data)
    return model
end function

model = create_model()
rpc = RemoteCall()
for round r ∈ rounds do
    users = sample(total, m)
    for (user u, address addr) ← users do
        # create new DPP with user's policy
        dpp = get_dpp(u)
        msg = pack(dpp, REMOTE_PROG, model)
        # distribute the request
        rpc.sendto(to = addr, msg = msg)
    end for
    tmp_sum = rpc.join(callback = accumulate)
    model = average(model, tmp_sum)
end for
return model
```

Table 1: Device Specification

Type	Specification
Model	Intel NUC NUC7i3BNB
RAM	3GB DDR4 2,133 MHz
Swap	16 GB
Processor	Intel Core i3-7100U (2-Cores, 2.4 GHz)
OS	Ubuntu Server 64-bit 18.04.3 LTS
Linux	4.15.0-74

- What is the external overhead imposed by the platform in terms of network traffic?
- To what extent can we scale our system, in terms of number of supported users and amount of data that could be analyzed?

In order to answer these questions we conducted three evaluations using the dataset from the *language modeling* task described in Section 4. First, we evaluate the performance of training and filtering on the edge device. Next, we measure round trip times for Central Service requests executed on an edge device. In particular, we measure the times shown in Figure 3. *Time to Distribute* (TTD) is the time to send the policy program and Data-Policy Pair to the edge device; *Time to Execute* (TTE) is the time to execute the policy program on the edge device; *Time to Receive* (TTR) is the time to transmit the updated Data-Policy Pair back to the Central Service; and *Time to Process* (TPP) is the time to process the received Data-Policy Pair on the Central Service. Finally, we evaluate the scalability of the system by performing one round of training on 312 simulated edge devices and measuring TTD, TTR, and TTP for all devices.

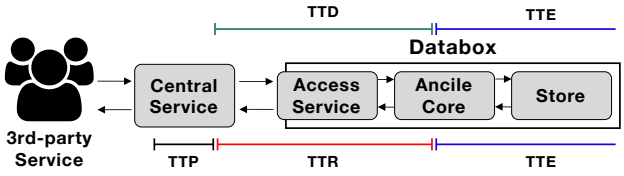


Figure 3: PoliFL Evaluation: We consider measures for the Time to Distribute (TTD), Time to Execute (TTE), Time to Receive (TTR) and Time to Process (TTP).

5.1 System Configuration

Our evaluation experiments were run with several hardware and software configurations described below. The edge-device overhead evaluations were carried out using an Intel NUC, a small factor mini PC with the specifications in Table 1 and running PoliFL on Databox v0.5.2.

To evaluate network and system overheads, we install the Central Service on an AWS instance with a Xeon E5-2686 8-core processor running at 2.3 GHz with 64 GB of memory. For our scalability evaluation, we run the Central Service on an Intel Core i7-4790 4-core processor running at 3.6 GHz with 64 GB of memory. This server is located on a 1 Gigabit Ethernet Local Area Network (LAN) with 52 edge devices, each of which has Intel Core i5-8500 6-core processor with running at 3.0 GHz with 16 GB of memory. Each of the 52 machines have the same software configuration as the Intel NUC used in the previous experiments.

The Central Service of PoliFL runs on Ubuntu Server v18.04.3 using Ancile v0.3 and communicates with the edge devices using the RabbitMQ[7] message broker v3.8.2.

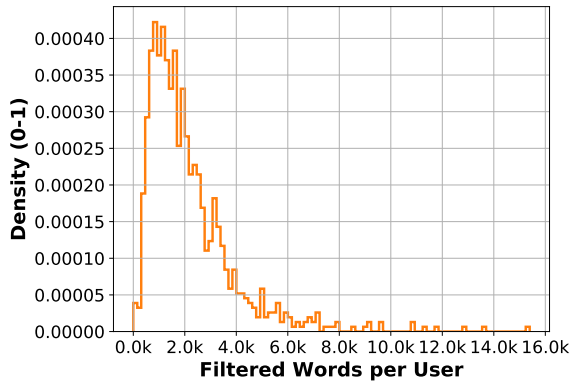


Figure 4: Density histogram of all filtered words (*i.e.*, words with valence ranking > 5.0) per user.

5.2 Edge-Device Overheads

This section reports the performance of an edge device conducting the two Ancile policy enforcement operations: *(i)* a text filtering policy in a centralized training context, and *(ii)* training two Deep Neural Networks (DNNs) of different sizes, in a federating learning context, as previously discussed in Section 4. Specifically, we report the *Time to Execute (TTE)*, as the time this operation takes to execute, together with the CPU utilization and memory overhead of the edge device. We run the referenced task on the Intel NUC for 1,000 users in the dataset, simulating how an edge device with different user data would execute these operations.

For the text filtering task, we applied a policy that identified in text those words with high valence (valence ranking > 5.0; $N = 1,459$ words) and filtered them out. We used the Affective Norms for English Words (ANEW) dataset [12] as our word reference dictionary. Figure 4 reports the filtered words per user in the Reddit dataset, demonstrating the weight of the text filtering task, with a mean of $2.17k$ word removals per user ($\pm 1.74k$).

For the DNN task, we used the *language modelling task* described in Section 4 for the two models of different sizes: *(i)* 1 layer LSTM with 20-dimensional embedding vectors (small DNN), and *(ii)* 2 layers with 200-dimensional (large DNN). We ran a single round of Federated Learning [35] training with 1,000 participants per round, each performing 100 local epochs before submitting model weights to the Central Service.

Figure 5 summarizes the edge device performance for these tasks. As expected, text filtering was the least expensive for the edge device, with mean TTE $0.08s$ (± 0.07), CPU utilization 9.02% (± 7.91) and almost no memory overhead. The DNN training task of the small model reported a mean TTE of $31.79s$ (± 26.55), CPU utilization 61.94% (± 14.51) and a memory overhead 3.51% (± 4.05). Finally, the larger DNN training task was the most resource demanding of all, with a mean TTE of $72.80s$ (± 63.85), CPU utilization 50.02% (± 18.41) and memory overhead 2.65% (± 2.93). The results show that a small factor mini PC like the Intel NUC could be used as an edge device in our system, even with more demanding task such as training a DNN model, with reasonable execution times and resource overheads. While not tested in this work, we expect that cheaper devices that have similar device specifications, such as the new Raspberry Pi 4[8], could also be used in the same context.

5.3 Network and System Overheads

Having evaluated the performance of local execution on the edge device, we now look at the performance of the Central Service by evaluating networking and processing overheads on the Central Service. We evaluate the system-wide performance for the three tasks (text filtering, small DNN training, and large DNN training) with the Central Service hosted on an AWS instance communicating to a single edge device. As we previously measured TTE, these experiments only measure TTD, TTR, and TTP.

For text filtering, we performed 100 rounds of filtering on the edge device, which sends the filtered text back to the Central Service. The results are shown in Figure 6a. The TTD, $0.37s$ (± 0.0), for this task is low, as only the policy program is distributed. The TTR, $3.11s$ (± 0.03), for this task takes a little longer. Although the filtered text is not very large ($46kb$), there is a small amount of fixed overhead in creating the data connection from the edge device back to the Central Service. The TTP was measured in section 4.3 and was $0.2s$ (± 0.01), which is a negligible part of the total task.

[7] https://www.rabbitmq.com

[8] https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

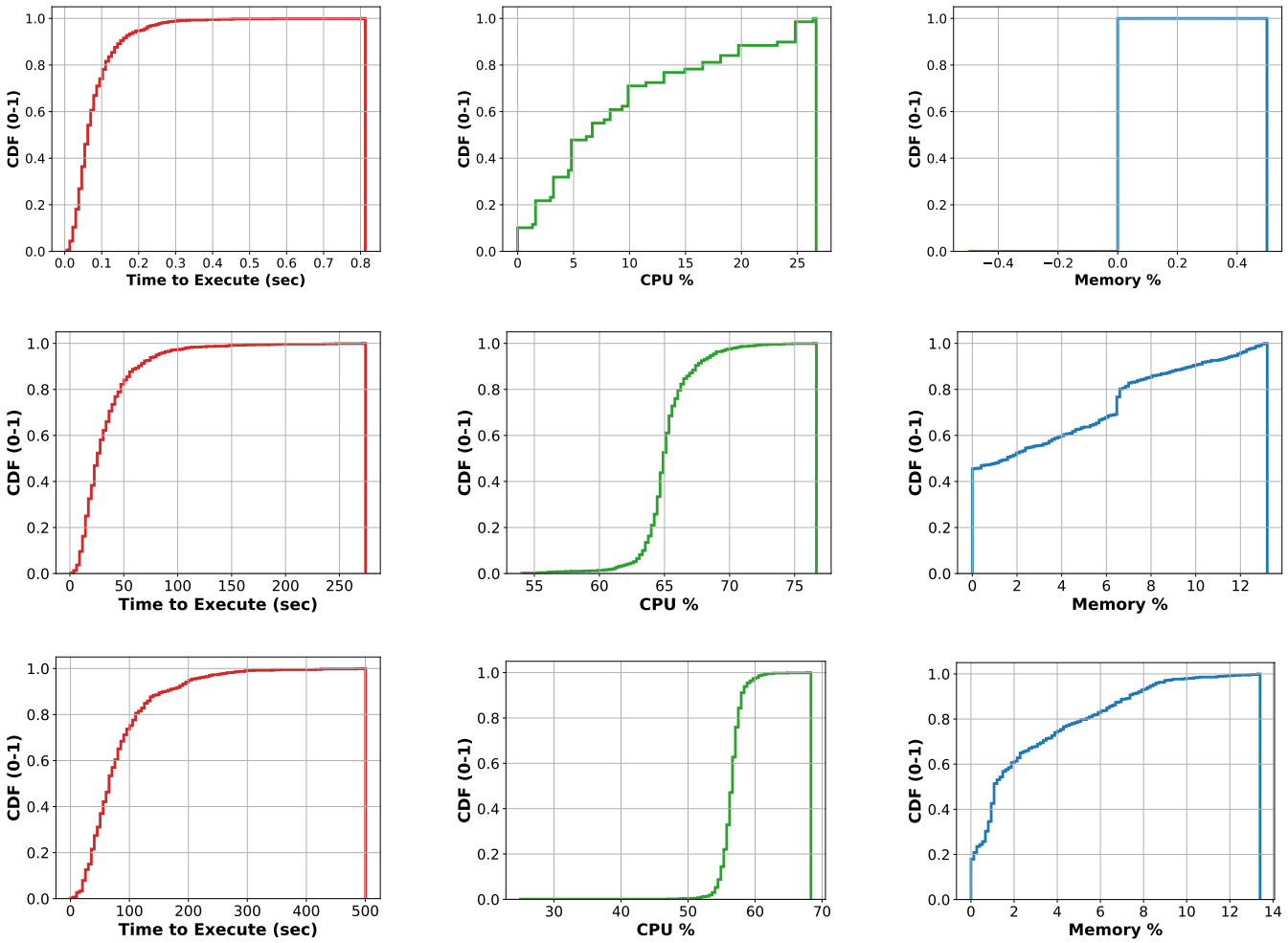


Figure 5: Cumulative Distribution Function (CDF) representation of the performance metrics (duration, CPU utilization and memory overhead) for the following use cases: *(i)* text filtering (first row), *(ii)* Small DNN training task (second row) and *(iii)* Large DNN training task (third row).

For the DNN training tasks, we evaluated the round-trip performance for 100 rounds of training on an edge device. We did two experiments, one with a small DNN (12 MB) model and one with a large DNN (120 MB) model. Figures 6b and 6c show the range of times for one round of training for these models.

For both the small and large DNN, it took longer to receive the model than to distribute it. There are two reasons for this. The first is the model sent from Central Service to the edge device is smaller than the one that is returned back to the Central Service due to PyTorch's implementation of running statistics. In our experiments, the small DNN model sent to the edge device was 8 MB in size, while the model returned to the Central Service was 12 MB. Similarly, the large DNN model sent to the edge device was 80 MB, while the model returned to the Central Service was 120 MB, 50% larger. The other reason is that the networking bandwidth to the AWS server was asymmetrical. The bandwidth for sending data to the edge device was measured at 17.5 MB/s, while the bandwidth from the edge device to the Central Service was only 10.2 MB/s, so it is not surprising the TTR times were larger than TTD times.

For both experiments, the overhead to send and receive the model is reasonable, relative to the time to compute the model. For the small DNN model, the mean time to distribute the model (TTD) was 2.57s (±0.18), and the mean time to receive the model (TTE) was 6.03s (±0.05), compared to the mean TTE of 31.8s (±26.55). Results for the large DNN model are similar. The mean TTD was 6.9s (±1.74) and the mean TTR 17.35s (±1.98), compared to the mean TTE of 72.8s (±63.85).

The last measurement, TTP, is the time to accumulate and average the model once it is received at the Central Service. This time is negligible, averaging 0.29s (±0.01) for the small DNN model, and 0.7s (±0.01) for the large DNN model.

For the DDN training tasks, the majority of the time is spent doing computation on the edge device and distributing and receiving the models. The time to transfer the models is dependent on the available bandwidth and is unavoidable when using federated learning.

5.4 Scaling

For our last evaluation, we measure the Central Service performance on the language modeling task with multiple edge devices. We demonstrate the scalability of our system by performing one round of training with 312 participants. This number was chosen as we had access to 52 machines, each with a six-core processors, allowing us to run six instances of Databox and Ancile, one on each core, and giving us a total of six edge devices per machine. These machines were all on the same 1 Gigabit Ethernet Local Area Network (LAN) along with a server running the Central Service.

The Central Service distributed the initial model to all 312 devices simultaneously. To match the performance of the edge device, we took TTE times from the Section 5.2 (Figure 5), and sent back a model after waiting. We ran the experiment for both the small and large DNN tasks and the results are shown in Figure 7.

This figure shows the total time to process each device request, shown in the order of arrival at the Central Service. The total time for each request is shown, broken down as TTD, TTE, and TTR. TTP was measured ($0.29s$ (± 0.01) for the small DMM and $0.7s$ (± 0.02) for the large DMM) but is not shown as it is negligible compared to the other times.

For both tasks, the majority of the overhead is the time it takes to distribute the model (TTD) to all 312 devices. The network is at capacity and many nodes have to wait awhile before they can start running the policy program. In the case of the small DNN task, the models are small enough that there was not much of a delay in sending the models back. In the large DNN task, the network is not able to keep up with receiving models as well, adding extra delay. However, in both cases, there are outlier devices with a relatively long TTE. This is especially noticeable in the small DNN task. As the total time to complete a round of training is dependent on the slowest policy program execution time, the overall time to complete a round of training is not much longer than the TTE of the slowest device. This shows that the longer the local computation on the device, the less important the networking overheads become at scale.

6 RELATED WORK

Personal Information Management Systems. Also known as *personal data banks*, PIMS are generalized solutions that help individuals manage their personal data in secure, usually local storage systems and control the data sharing process. While this work was based on the Databox [13] platform, other similar open-source systems exist such as the OpenPDS [15]. At this moment, none of these platforms have the ability to control data sharing with third-parties using use-based privacy policies. Instead, they are used as platforms for securing the data, where third-party apps can be installed that can process the data locally.

Examples of these approaches are appearing in the consumer domain. For instance, BBC Box [5], an upcoming digital media hub from the BBC will allow users to enjoy personalized recommender services without sharing personal data. The BBC Box recommender system aggregates the user data from a range of media services and processes it at the device to create a user profile, without sharing data to a centralized cloud service.

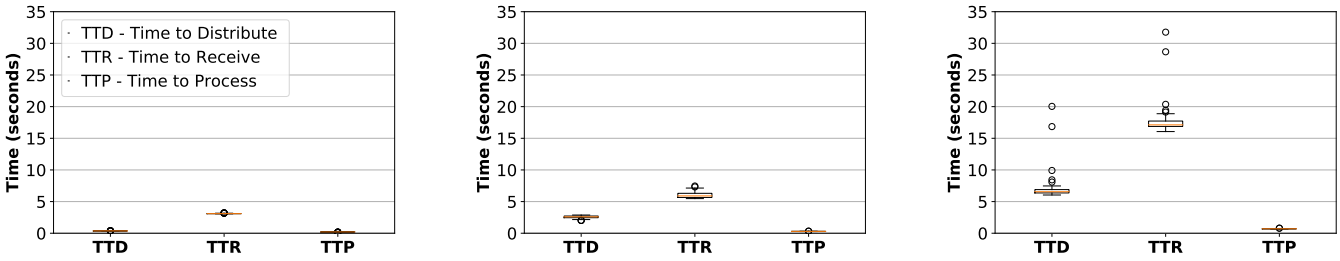
Privacy Preserving Machine Learning. One of the first techniques that was used for protecting the user's privacy while running joint computation is Secure Multi-Party Computation (MPC) [22], a cryptographic technique that allows mutually distrusting parties to run joint computations without revealing private data. One of the main limitations of such technique is that it performs badly with large datasets, even with only a few parties getting involved. Volgushev *et al.* [49] recently presented a scalable solution, based on MPC, that is applicable to large datasets, using query rewriting to minimize expensive processing under MPC. Other approaches for secure computations exist, such as Opaque [53], where they use a Trust Execution Environment (TEE) to run most of the computation under a secure environment.

More focused on the machine learning context, Servia-Rodriguez *et al.* [45] presented an approach of downloading a shared model trained with a small number of users and personalize it at the edge device for protecting the user's privacy. Federated Learning [30, 31, 35], aim to keep users' data on their devices while contributing into a global model without sharing the data. Federated learning assumes only a single consumer that runs training on the user's data. To further protect user data recent solutions extend federated learning with differential privacy [36] or encryption [11, 26]. These approaches are relevant to our work, however the privacy policies cannot be dynamically negotiated and are uniform across users. Instead, we investigate a policy-controlled execution of federated learning, where each user has a personal policy that specifies how to handle their data depending on the application specifics.

Privacy in Ubiquitous Systems. User's passively generated data could reveal sensitive details such as behavioral patterns [24, 28] and physical presence [51] can lead to stalking or disparate treatment [24, 52]. PrivacyStreams [34] focuses on controlling Android applications. However, it lacks a policy enforcement component and does not support FL. The TaintDroid [18] and FlowDroid [2] project do not enforce policy restrictions, similar to other works [33, 37, 41, 50].

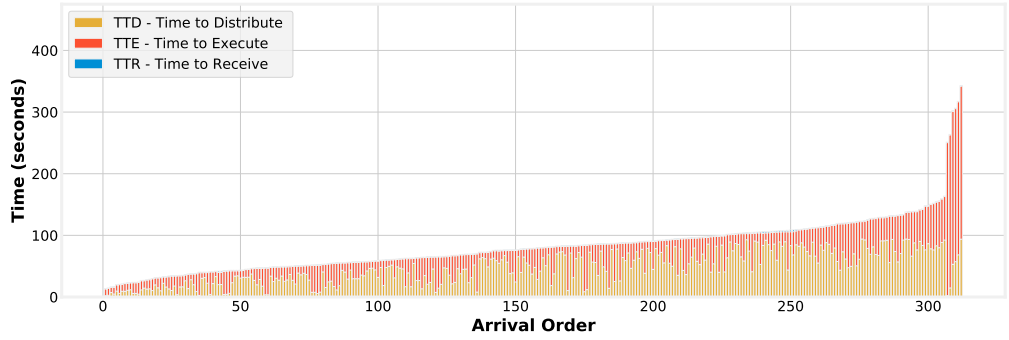
Policy-based Privacy. Language-level information flow control techniques [44] is a popular approach on tracking data propagation through the system. The recently proposed Ancile framework [3] combines this concept with use-based privacy [8] that introduces policies that "react" to data transformation. However, that framework is limited to centralized data processing and does not extend to federated setting.

Recent work [20], proposed data flow authentication control using homomorphic encryption and SGX enclaves that prevents an adversary from modifying the program. However, this approach requires a fixed program to be authorized, whereas our framework supports flexible programs that satisfy the corresponding policies.

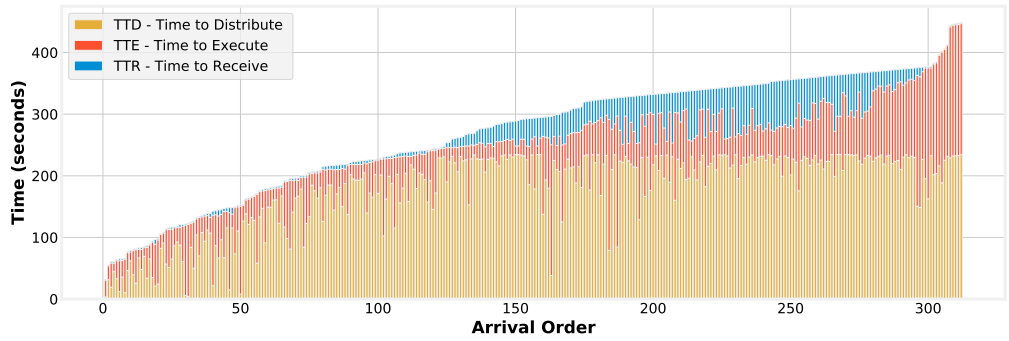


(a) Text Filtering Task (46 KB) (b) Small DNN Model (12 MB) (c) Large DNN Model (120 MB)

Figure 6: Time to Distribute (TTD), Time to Receive (TTR) and Time to Process (TTP) for the three tasks on a single Databox.



(a) Small DNN Model (12 MB)



(b) Large DNN Model (120 MB)

Figure 7: Time to Distribute (TTD), Time to Execute (TTE), and Time to Receive (TTR) for a small and a large DNN model for one round of training with 312 simultaneous users. The time to process the request is dominated by the time it takes to send the DNN model to all the edge devices (TTD) and a small number of edge devices that require a longer time to execute. The Time to Process (TTP) on the Central Service is negligible relative to the other times and is not shown here.

7 CONCLUSIONS

We presented PoliFL, a decentralised policy-based framework for personalized and private data analytics. Using a number of privacy-sensitive use cases including image and text-based models, we evaluated PoliFL based on Federated Learning as an exemplar application scenario and demonstrated its feasibility and scalability.

One limitation of our system is that the overall performance depends on the network bandwidth. In future work we will investigate techniques such as DNN model compression or use of alternative queuing system to help PoliFL scale better.

ACKNOWLEDGMENTS

The authors would like to thank Nate Foster, Fred B. Schneider, and Eleanor Birrell for the initial productive discussions and ideas. This work was supported in part by the NSF Grant 1642120. Haddadi and Katevas were partially funded by the EPSRC Databox project EP/N028260/1 and the EPSRC DADA project EP/R03351X/1. This work was done while Katevas was at Imperial College London.

REFERENCES

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *CCS*, 2016.

[2] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[3] E. Bagdasaryan, G. Berlstein, J. Waterman, E. Birrell, N. Foster, F. B. Schneider, and D. Estrin. Ancile: Enhancing privacy for ubiquitous computing with use-based privacy. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 111–124, 2019.

[4] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459*, 2018.

[5] BBC. Introducing the BBC Box. https://www.bbc.co.uk/rd/blog/2019-06-bbc-box-personal-data-privacy.

[6] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo. Analyzing federated learning through an adversarial lens. *arXiv preprint arXiv:1811.12470*, 2018.

[7] E. Birrell, A. Gjerdrum, R. van Renesse, H. Johansen, D. Johansen, and F. B. Schneider. Sgx enforcement of use-based privacy. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, pages 155–167. ACM, 2018.

[8] E. Birrell and F. B. Schneider. A reactive approach for use-based privacy. Technical report, Cornell University, 2017.

[9] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NIPS*, 2017.

[10] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. M. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander. Towards federated learning at scale: System design. In *SysML 2019*, 2019.

[11] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[12] M. M. Bradley and P. J. Lang. Affective norms for english words (anew): Instruction manual and affective ratings. Technical report, Technical report C-1, the center for research in psychophysiology, University of Florida, 1999.

[13] A. Chaudhry, J. Crowcroft, H. Howard, A. Madhavapeddy, R. Mortier, H. Haddadi, and D. McAuley. Personal data: Thinking inside the box. In *Proceedings of The Fifth Decennial Aarhus Conference on Critical Alternatives*, CA'15, pages 29–32, Aarhus N, 2015. Aarhus University Press.

[14] X. Chen, T. Chen, H. Sun, Z. S. Wu, and M. Hong. Distributed training with heterogeneous data: Bridging median and mean based algorithms. *arXiv preprint arXiv:1906.01736*, 2019.

[15] Y.-A. De Montjoye, E. Shmueli, S. S. Wang, and A. S. Pentland. openpds: Protecting the privacy of metadata through safeanswers. *PloS one*, 9(7), 2014.

[16] Docker Inc. What is a container? https://www.docker.com/what-container, Apr. 2017.

[17] C. Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[18] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[19] EU General Data Protection Regulation. Proposal for a regulation of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2012:0011:FIN:en:PDF, 2012.

[20] A. Fischer, B. Fuhry, F. Kerschbaum, and E. Bodden. Computation on encrypted data using dataflow authentication. *Proceedings on Privacy Enhancing Technologies*, 2020(1):5–25, 2020.

[21] 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. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42, Sept. 2015.

[22] O. Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[23] S. Guha, A. Reznichenko, K. Tang, H. Haddadi, and P. Francis. Serving ads from localhost for performance, privacy, and profit. In *ACM Workshop on Hot Topics in Networks (HotNets-VIII)*. ACM, 2009.

[24] P. Händel, J. Ohlsson, M. Ohlsson, I. Skog, and E. Nygren. Smartphone-based measurement systems for road vehicle traffic monitoring and usage-based insurance. *IEEE systems journal*, 8(4):1238–1248, 2013.

[25] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[26] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] P. Holley. Wearable technology started by tracking steps. soon, it may allow your boss to track your performance. *Washington Post*, 2019.

[29] Identity Theft Resource Center. ITRC Breach Statistics 2005—2015. http://www.idtheftcenter.org/images/breach/2005to2015_20160828.pdf, 2016.

[30] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[31] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop*, 2016.

[32] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[33] T. Li, Y. Agarwal, and J. I. Hong. Coconut: An ide plugin for developing privacy-friendly apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):178, 2018.

[34] Y. Li, F. Chen, T. J.-J. Li, Y. Guo, G. Huang, M. Fredrikson, Y. Agarwal, and J. I. Hong. Privacystreams: Enabling transparency in personal data processing for mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):76, 2017.

[35] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pages 1273–1282, 2017.

[36] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. In *ICLR*, 2018.

[37] S. Narain and G. Noubir. Mitigating location privacy attacks on mobile devices using dynamic app sandboxing. *Proceedings on Privacy Enhancing Technologies*, 2019(2):66–87, 2019.

[38] P. Nelson. Just one autonomous car will use 4,000 GB of data/day. NetworkWorld, http://www.networkworld.com/article/3147892/internet/one-autonomous-car-will-use-4000-gb-of-dataday.html, Dec. 7 2016.

[39] H. Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004.

[40] OpenMined. https://www.openmined.org/, 2019.

[41] E. Pournaras, I. Moise, and D. Helbing. Privacy-preserving ubiquitous social mining via modular and compositional virtual sensors. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 332–338. IEEE, 2015.

[42] PyTorch examples. https://github.com/pytorch/examples/tree/master/word_language_model/, 2019.

[43] A restricted execution environment for python to run untrusted code. https://github.com/zopefoundation/RestrictedPython, 2006.

[44] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on selected areas in communications*, 21(1):5–19, 2003.

[45] S. Servia-Rodríguez, L. Wang, J. R. Zhao, R. Mortier, and H. Haddadi. Privacy-preserving personal model training. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 153–164. IEEE, 2018.

[46] Strategic Headquarters for the Promotion of an Advanced Information and Telecommunications Network Society. Policy outline of the institutional revision for utilization of personal data. http://japan.kantei.go.jp/policy/it/20140715_2.pdf, 2014.

[47] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings Network and Distributed System Symposium*, 2010.

[48] US Consumer Privacy Bill of Rights. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. www.whitehouse.gov/sites/default/files/privacy-final.pdf, 2012.

[49] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros. Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019*, page 3. ACM, 2019.

[50] H. Wang, J. Hong, and Y. Guo. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 1107–1118. ACM, 2015.

[51] S. B. Wicker. The loss of location privacy in the cellular age. *Communications of the ACM*, 55(8):60–68, 2012.

[52] D. Woodlock. The abuse of technology in domestic violence and stalking. *Violence Against Women*, 23(5):584–602, 2017.

[53] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 283–298, 2017.