PubSub-ML: A Model Streaming Alternative to Federated Learning

Lovedeep Gondara
School of Computing Science
Simon Fraser University
Canada
lgondara@sfu.ca

Ke Wang
School of Computing Science
Simon Fraser University
Canada
wangk@cs.sfu.ca

ABSTRACT

Federated learning is a decentralized learning framework where participating sites are engaged in a tight collaboration, forcing them into symmetric sharing and the agreement in terms of data samples, feature spaces, model types and architectures, privacy settings, and training processes. We propose *PubSub-ML*, Publish-Subscribe for Machine Learning, as a solution in a loose collaboration setting where each site maintains local autonomy on these decisions. In PubSub-ML, each site is either a publisher or a subscriber or both. The publishers publish differentially private machine learning models and the subscribers subscribe to published models in order to construct customized models for local use, essentially benefiting from other sites' data by distilling knowledge from publishers' models while respecting data privacy. The term "model streaming" comes from the extension of PubSub-ML to decentralized data streams with concept drift. Our extensive empirical evaluation shows that PubSub-ML outperforms federated learning methods by a significant margin.

KEYWORDS

decentralized learning, differential privacy, federated learning

1 INTRODUCTION

The ability to share information between institutions while respecting data privacy of individuals would lead to more robust and accurate models. Two well-known examples aiding remarkable achievements via access to large data are ImageNet [4] in computer vision and SQuAD [35] in natural language processing. However, the progress is limited in highly regulated domains such as healthcare and finance as regulations and privacy concerns make it impossible for institutions to pool and disseminate their data. The most studied approach in this context is federated learning (FL) [29], a decentralized learning framework that trains a machine learning model on data distributed among multiple sites by training on local data and exchanging parameters between sites at some frequency to generate a global model.

The *general* FL framework is not well-suited to the multi-institut -ional collaboration due to *tight collaboration* of the sites involved, where the sites are forced into symmetric sharing and shared decision making in terms of shared feature space (i.e., horizontal FL [49]) or sample space (i.e., vertical FL [49]), uniform model types and architectures, and training iteration synchronization. These issues are exacerbated for differentially private (DP) FL where all the sites have to agree on a common privacy budget.

For example, consider the healthcare setting, where in FL, all participating hospitals have to contribute their data to receive the final model as the benefit of participation. However, in the real world, the forms of contribution and benefit are diverse. Where, it is possible that some hospitals may pay for the final model instead of contributing their data, whereas other hospitals may contribute their data and receive benefits through monetary rewards or collaboration. Therefore, each site can be a contributor of data, a recipient of final models, or both. Furthermore, collaboration should not cause changes to sites' existing data analysis practices, including features and samples collected, model types and architectures used, how and when to train local models, and how to set site-specific privacy budgets.

1.1 Our Proposal

With the above in mind, we consider a decentralized learning setting under *loose collaboration* with the following core requirements:

Autonomy: Each site should be able to make its local decisions on data samples, feature space, and training processes, as if it were the only site in the decentralized learning setting.

Asymmetric sharing: Each site should be allowed to decide independently whether its participation is through contributing data (i.e., contributor), or receiving the final model (i.e., beneficiary), or both.

Personalized privacy: Each site should be able to set its own privacy budget to preserve local data privacy because a common, one-size-fits-all privacy budget leads to degraded utility and/or privacy.

Model-agnostic: The decentralized learning framework should be independent of the model types used by individual sites. That is, each site should be able to independently select their model types and architectures (for example, one site can use neural network and another site can use random forest, etc.).

We propose **PubSub-ML**, *Publish-Subscribe for Machine Learning*, to achieve the above requirements of loose collaboration for decentralized machine learning. PubSub-ML takes the inspiration from publish/subscribe messaging[1], an asynchronous communication method in which messages are exchanged between applications without knowing the identity of the sender or recipient, selected specifically to enable asymmetric sharing between sites. In PubSub-ML, each site is either a *publisher* or a *subscriber* or both. Instead

[1] https://bit.ly/3gr3jOu

Proceedings on Privacy Enhancing Technologies 2023(2), 464–479
© 2023 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2023-0063

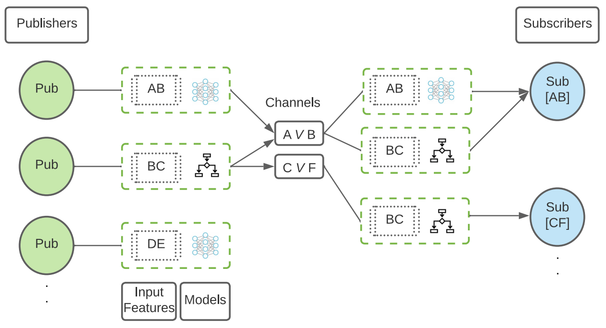

Figure 1: PubSub-ML: Three publishers (Pub) and two subscribers (Sub). A, B, C, D, E, F are individual features. Subscriber 1 with the feature space AB specifies the channel $A \vee B$, which subscribes all published models trained on the input feature space containing either A or B, and Subscriber 2 with the feature space CF specifies the channel $C \vee F$, which subscribes all published models trained on the input feature space containing either C or F.

of publishing and subscribing messages, each publisher publishes locally trained differentially private machine learning models and each subscriber subscribes to the published models through a *channel* that specifies a requirement on the feature space of subscribed models. The models satisfying the requirement are streamed to the subscriber. The subscriber uses the streamed models plus its own model and data to construct the customized model for local use. An example of the subscription for PubSub-ML is shown in Figure 1.

Subscribing other sites' models for customizing the local model is related to teacher-student based knowledge distillation [14], where the subscribed models serve as the teachers for the subscriber to construct the customized model for local use. The model customization offers several advantages in facilitating the subscriber's learning compared to FL.

First, model customization in PubSub-ML takes advantage of distilling knowledge from multiple diverse models, that is, the DP models from the publishers and the non-private, local model from the subscriber. From an ensemble point of view, the individual models trained on limited data act as weak learners, the model customization combines them to create a significantly strong learner using subscriber's local data. In contrast, DP-FL builds a single, global model, using noisy aggregates. This unique property of PubSub-ML leads to superior empirical performance compared to DP-FL, as shown in our evaluation in Section 8. *Second*, in the case where each site has a different data distribution (non-IID data), e.g. a different sub-population, model customization allows the subscriber to bias towards the sites that have a data distribution similar to its local data. In contrast, FL forces the equal use of all sites' data. In this sense, model customization can ignore bad models from publishers that have poor quality data. *Third*, when useful features are distributed across sites, model customization can incorporate all useful features, benefiting from all sites' data. *Fourth*, model customization is not constrained to specific learning scenarios such as classification or regression, instead, it is applicable to all learning problems that can be formulated as an optimization problem with an objective defined by a loss function.

In the real world, decentralized data often comes in the form of continuous data streams originating at multiple related sites (electronic health records from multiple hospitals, credit card transaction flows from multiple banks, etc.), and the properties and patterns of data are subject to change over time, a phenomenon known as concept drift. To be practically useful in such applications, we extend PubSub-ML to decentralized streaming data to allow model publishing and subscribing on a continuous basis. Importantly, our extension ensures a *constant* privacy budget for unlimited model updates for never-ending data streams, which is not possible for existing DP-FL methods where incremental updates to the model deteriorate utility as each update depletes the privacy budget (due to the composition of the privacy loss [9]).

1.2 Contributions

The technical contributions of PubSub-ML are as follows:

- We motivate a decentralized learning setting under loose collaboration and introduce Autonomy, Asymmetric sharing, Personalized privacy, and Model-agnostic property to capture the essential requirements in this setting. This setting requires minimal changes to a site's current practice for participation.
- We propose a publishing/subscribing inspired decentralized machine learning scheme, PubSub-ML, where each site publishes and/or subscribes models. The core component is the customization scheme for a subscriber to construct a customized model utilizing the subscribed models. PubSub-ML satisfies Autonomy, Asymmetric sharing, Personalized privacy, and is Model-agnostic.
- We extend PubSub-ML to time-evolving data streams and show that the same DP guarantee holds after any number of model updates.
- We empirically show that our proposed method significantly outperforms competitors on six real-world and synthetic datasets, for classification and for regression. We make our source code publicly available[2].

Rest of the paper is organized as follows. Section 2 reviews related works. Section 3 gives an overview of PubSub-ML. Section 4 presents the customization algorithm that integrates published models for a subscriber. Section 5 presents the detailed PubSub-ML framework. Section 6 extends the PubSub-ML framework to data streams with concept drift. Section 7 presents our evaluation setup and Section 8 presents evaluation results. Finally, we conclude the paper in Section 9.

2 RELATED WORK

This section reviews the related work in the decentralized learning setting and highlights our differences from existing work.

Model and data heterogeneity. [24, 27, 53] address model autonomy in the decentralized setting using knowledge distillation [14], where they require access to a public dataset with similar distribution as the sites' private data. Assisted learning [48] allows

[2]https://github.com/lgondara/PubSub-ML

different sites to have different features but requires the alignment of samples across sites. Personalized FL [11][50] produces a personalized model at each site by fine-tuning the global model using local data. Other methods in FL have been proposed to support non-IID data across clients and to improve convergence (FedProx [25] and FedAdam [37]). ProxyFL [20] and HeteroFL [5] allow different model architectures, where Proxy FL works via sharing a DP proxy model, and HeteroFL works by constructing clients using a subset of global model parameters. All the above works apart from [20] do not provide any privacy guarantee, do not support full autonomy, and are not model-agnostic as they require the same model type across sites, the alignment of either features or samples across sites, same privacy budget for all sites, and synchronized training.

Meta learning and model fusion. Meta learning and model fusion, e.g., [15] [23] [51], aim to adapt pre-trained models of related tasks to a *new* task where synchronized training is not possible. [23] applies knowledge transfer from black-box models to *new* tasks by learning to capture the correspondence between latent representation that represent the embeddings of different models. [51] develops a framework that learns shared global latent structures by identifying correspondences among local model parameterizations. All of the above works do not provide privacy guarantees and do not support full autonomy.

Clustered and multi-task learning. Clustered FL [12, 38] partitions the clients into several clusters, where each cluster has similar clients, with the client similarity learned during the federated learning process. Clients in each cluster share the same optimal model, which can be different from the optimal models of other clusters. Multi-task learning [28, 41, 45, 52] allows for more nuanced relations among participating sites, where each site can be considered addressing a specific task, allowing the ability to model heterogeneous data distributions across clients. However, all the works above do not guarantee privacy, do not support full autonomy, and are not model-agnostic.

Privacy. Most of the works in FL do not explicitly incorporate privacy guarantees assuming that keeping data locally at the respective sites protects user privacy. We can broadly classify the works that consider formal privacy/security guarantees into two categories. First, the works that use differential privacy as the mechanism to provide privacy guarantees, such as the FL approaches combined with DP [6, 20, 30]. These works only consider uniform privacy guarantees, lacking personalized privacy where sites can independently select their privacy budgets. Second, there is an ongoing effort to leverage security/cryptography based protocols, such as secure multiparty computation [3, 43] or homomorphic encryption [47]. Use of such methods in FL is still in early stages where the progress is hindered by their extensive communication and computational overhead [19]. Please see [18] for a detailed discussion. To our knowledge, no existing work has provided differential privacy for machine learning in the decentralized data streaming setting.

3 OVERVIEW

3.1 Differential Privacy

Two data sets D and D' are *neighboring* if they differ in the addition or removal of one user's data.

Definition 1 (Differential Privacy (DP) [9]). *A randomized mechanism $\mathcal{M} : \mathbb{D}^n \rightarrow \mathbb{R}^d$ is (ε, δ)-DP if for any pair of neighbouring data sets $D, D' \in \mathbb{D}^n$, and for all sets O of possible outputs:*

$$Pr[\mathcal{M}(D) \in O] \leq e^{\varepsilon} Pr[\mathcal{M}(D') \in O] + \delta \quad (1)$$

The inequality ensures that changing the data of any single user will have a *small* impact, defined by the multiplicative factor e^{ε} and the additive factor δ on the output of a mechanism $\mathcal{M}$ satisfying DP.

Theorem 1 (Parallel Composition [31]). *Let $\mathcal{M}_i$ each provide $(\varepsilon_i, \delta_i)$-DP. Let D_i be arbitrary disjoint subsets of D. The sequence of $\mathcal{M}_i(D_i)$ provides $(max_i \varepsilon_i, max_i \delta_i)$-DP.*

Theorem 2 (Post Processing [9]). *Let $\mathcal{M} : \mathbb{D}^n \rightarrow \mathbb{R}^d$ be a randomized mechanism that is (ε, δ)-DP. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^t$ be a deterministic function. Then $f \circ \mathcal{M} : D^n \rightarrow \mathbb{R}^t$ is (ε, δ)-DP.*

3.2 Overview of PubSub-ML

In PubSub-ML, participating sites preserve their existing data analysis practices with some minor changes. Before beginning, all sites (publishers and subscribers) resolve any ambiguity of feature semantics (i.e., different feature names representing the same semantic and same feature name used to represent different semantics) and feature encodings (i.e., the same values are represented differently at different sites) arising from the distributed nature of sites. For example, all sites agree to label their attribute recording "Age at disease diagnosis" as "Age" and to store the values as number of years. These ambiguities are resolved at the metadata level, while keeping their raw, sensitive data locally.

In PubSub-ML, as in most works on privacy, we assume that the sites are semi-honest, that is, the sites follow the PubSub-ML protocol even if they may try to compute additional information about other sites [7]. We also make the following assumption about local dataset D^i owned by each participating site i.

Assumption 1. *For each site i, the data D^i has its own feature space and is generated independently of other sites.*

In other words, sites are not required to have the same feature space (see Figure 1 and Example 1), and we assume that the data at each site is generated independently. In particular, we do not require the alignment of features or samples across all sites as in [30, 48].

In PubSub-ML, each site is either a *publisher* or a *subscriber* or both. Each publisher p (i.e., the pth publisher) trains and publishes differentially private local model(s) denoted by M^p, where a model, defined by a learning objective and a loss function, generates output for a given input. For example, a model outputs a probability distribution of class labels for classification or the estimated target value for regression. Each subscriber s (i.e., the sth subscriber) subscribes to a *channel* specified in the form $F_1 \vee \cdots \vee F_k$, where F_i is a subset of the features from the subscriber s's feature space. This channel will stream to s all published models that contain all the features in at least one F_i. We denote s's subscribed models by Λ_s. With the models in Λ_s as well as the subscriber's own local model(s) denoted by M^s, the subscriber s constructs a *customized model* for the task at the site s, denoted by C^s. This construction is performed by a *customization algorithm*, detailed in Section 4.

Note that a site i can be both a publisher p and a subscriber s, that is, the pth publisher and the sth subscriber. In this case, M^p denotes the model published by p as the publisher and M^s denotes s's local model used to construct the customized model by the subscriber. As we shall see in Section 4, M^p and M^s may be different.

The overview of PubSub-ML is given in Algorithm 1. $(\varepsilon^p, \delta^p)$ denotes the personalized privacy budget for each publisher p. There are two phases. In the publishing phase, all publishers p train and publish DP models M^p. This can be done by having all publishers running the DP version of their machine learning algorithms. Examples are DP neural networks [1], DP random forest [36], DP Naive Bayes [44], etc. In the customizing phase, all subscribers s construct customized models C^s using subscribed models Λ_s plus local models M^s.

Algorithm 1 Overview of PubSub-ML

```
Require: Data sets $D^i$ for all sites $i$; privacy budgets $(\varepsilon^p, \delta^p)$ for
    publishers $p$.
1: for all publishers $p$ in parallel do
2:     $p$ trains and publishes $(\varepsilon^p, \delta^p)$-DP local models $M^p$;
3: end for
4: for all subscribers $s$ in parallel do
5:     $s$ constructs the customized model $C^s$ using the subscribed
       models $\Lambda_s$ and its own local model(s) $M^s$;
6: end for
```

Example 1. Let A represent features related to intensive care (such as length of stay in ICU, patient vitals, etc.), B represent patient demographics (such as age, ethnicity, etc.), C represent features from the lab system (such as WBC count, hemoglobin, etc.), and D represent features from the pharmacy (such as drugs used, dosage, etc.). Suppose that hospitals 1, 2, and 3 are publishers and have the features AB, BC, BD, respectively, and hospital 4 is a subscriber with features ABC with the task of mortality prediction. We assume that features A and C are most important for this. To benefit from the data on either A or C of the other hospitals, hospital 4 subscribes to the channel $A \vee C$, which streams the models published by hospitals 1 and 2. Note that we make no assumption on the alignment of samples across sites, thus, these hospitals can have different sets of patients.

Discussion. PubSub-ML has many desirable properties. First, since publishers' models are protected by DP, and are used by the subscriber to create a customized model, no trusted server is needed, and the DP guarantees provide protection against model inversion and reconstruction attacks [32, 42]. Second, publishers can train their models in parallel and subscribers can run their customization algorithms in parallel; the only synchronization is that the training of the models is performed before the customization. Therefore, PubSub-ML can scale up to a large number of publishers/subscribers. Compared to the synchronized training of FL, our customization of (published) pre-trained models suits particularly well with resource-constrained IoT devices and bandwidth-limited communications. Third, related to the last property, the communication cost in PubSub-ML is minimum. There is no communication cost on model training (which occurs locally at all sites) and there is one communication round between subscribers and publishers to pass model parameters at end of training, compared to multiple training rounds in FL. Roughly, our communication cost is similar to that of the fully decentralized FL where each site exchanges information with its neighbours in each training round, but we only need one round of communication. Finally, the publishers and subscribers can join and leave the system as desired because the model customization only depends on the models published, not the identity of the publishers.

3.3 Issues

We need to address several issues. *First*, most model fusion schemes assume that sites share the same feature space. In PubSub-ML, each site can have its own set of features, so we need a new way of constructing the customized model from published models. *Second*, the number of models in Λ_s can be potentially large and not all models contribute equally. In such cases, it can be resource intensive for the subscriber to use all models in Λ_s, the subscriber needs a way to select few *good* models. We begin with presenting the customization algorithm in the next section (i.e., line 5 of Algorithm 1), followed by refined PubSub-ML in Section 5. We present PubSub-ML's extension for data streams in Section 6, and empirical evaluation in Sections 7 and 8.

4 CUSTOMIZATION ALGORITHM

The subscriber s uses the subscribed models Λ_s and the subscriber s's own model(s) M^s to construct the customized model C^s for performing the task at s. Note that M^s is never published because it is only used for customization by the subscriber s itself. Traditional ensemble and weighting methods such as [46] do not work because the sites in PubSub-ML are not required to have the same feature space. In addition, traditional methods learn only a linear weighting scheme. Below, we present a general solution.

One question is what data will be used to train M^s and to train the customized model C^s. One option is using the subscriber's whole data, i.e., D^s, to train both M^s and C^s. This would make the customization bias towards M^s trained by the same data, which reduces the benefit of the subscribed models. To avoid this, we split D^s into the *training subset* and the *customization subset*, denoted by $Tra(D^s)$ and $Cus(D^s)$, respectively, and use the former to train M^s and use the latter for customization. In some cases such as sites that are IoT devices or smartphones, the number of publishers, thus, the number of subscribed models in Λ_s, can be very large and it is resource intensive to use all such models for customization. In this case, a small number of models can be selected from $M^s \cup \Lambda_s$ for customization. Let Sel denote the selected models. For example, Sel could be the set of top-m models based on the performance on $Cus(D^s)$.

With the above in mind, we present our customization algorithm, run by the subscriber s, as CUSTOMIZE(s, *loss*) in Algorithm 2. The algorithm has five steps described below.

Step 1: generate local model(s) M^s to be used for customization. This model is trained by s on the data $Tra(D^s)$.

Step 2: select models from $M^s \cup \Lambda_s$ for customization. Sel denotes the selected models.

Step 3: construct the training data for customization. The customization will fuse the selected models Sel to perform the task at the subscriber s, similar to knowledge distillation [14] where a model's output acts as a proxy for the knowledge learned by the model. The challenge is that the subscriber s does not necessarily have the same feature space as publishers, hence, the customization data $Cus(D^s)$ may have different features from the input features of subscribed models. This step unifies their feature spaces by extending the samples x in $Cus(D^s)$ and the inputs of the subscribed models M_i in Sel to the union of their feature spaces, denoted by $all(s, Sel)$. For example, suppose that s has the feature space ABC and one model in Sel has the feature space BD and another has the feature space CF, where each single letter is a feature, then $all(s, Sel) = ABCDF$. The detail is explained below.

For a sample x in $Cus(D^s)$ and a model M_i in Sel, let x' and M_i' denote their extensions to all the features in $all(s, Sel)$, respectively. Let $M_i'(x')$ denote the output vector of M_i' on the input x'. We define

$$m(x) = \{M_i'(x'), \cdots, M_R'(x')\} \tag{2}$$

where M_i's are the models in Sel. Intuitively, $m(x)$ is the concatenation of the output vectors $M_i'(x')$ for all selected models M_i.

Note that we have not specified a specific learning objective for M_i. M_i can be any model such as those for supervised, semi-supervised, or unsupervised learning setting, as long as the M_is are defined with an optimization objective and a loss function, and generate output when presented with appropriate inputs. Examples of M_i are generative adversarial networks, autoencoders, convolutional neural networks, logistic regression, random forest, etc. We have, $|m(x)| = \#out \times R$ where #out is the number of outputs per M_i. For classification, we have $\#out = \#class$ where $\#class$ is the number of classes, and for regression we have $\#out = 1$.

Our training data for customization is defined by the output vectors $m(x)$ as follows:

$$O^s = \{(m(x), y) \mid (x, y) \in Cus(D^s)\} \tag{3}$$

Algorithm 2 CUSTOMIZE(s, $loss$)

```
Step 1: generate local model(s) $M^s$
Step 2: $Sel \leftarrow$ select top models from $M^s \cup \Lambda_s$ based on performance on $Cus(D^s)$
Step 3: compute $O^s$ by Eqn (3)
Step 4: train $f_c$ using $O^s$ and $loss$
Step 5: return $f_c \circ m$
```

Step 4: model fusion. With O^s as training data, this step learns an appropriate model, f_c, to integrate the models from Sel with respect to a loss function $loss$ determined by the learning task at the subscriber s. For instance, f_c can be a fully connected NN that minimizes $loss$, for classification, f_c has $\#class$ output units, $o_1, \cdots, o_{\#class}$, and an appropriate $loss$ such as the categorical cross-entropy, and for regression, f_c has a single output unit o and $loss$ such as the mean squared error. Note that, with O^s as the training data, the training of f_c does not need to access or modify the internal structure of models in Sel, making the customization algorithm *model agnostic*.

Step 5: return the customized model C^s. The customized model is given by the composition $f_c \circ m$, where m is defined by Eqn (2).

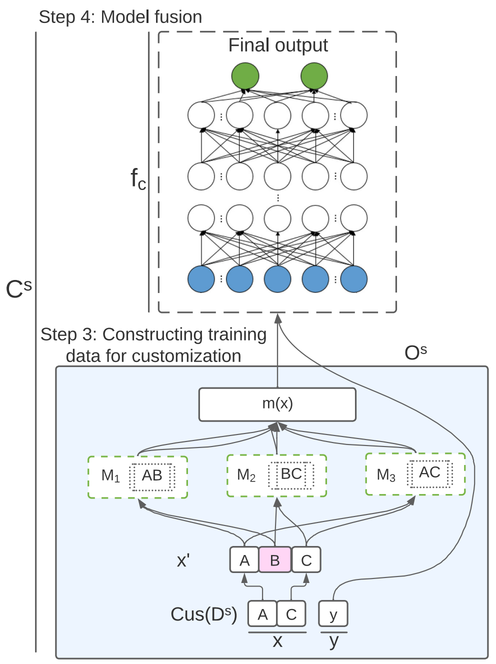

Figure 2: Example for CUSTOMIZE: Subscriber uses its own model $M_3(AC)$ and publishers' models $M_1(AB)$, $M_2(BC)$ to construct the customized model C^s using subscriber's customization subset $Cus(D^s)$. Step 3 (Constructing training data for customization) takes as input $(x, y) \in Cus(D^s)$ and extends x to x' (the feature universe ABC) by adding B (shown in pink), not in subscribers customization subset, which is then used to create the training data for customization (O^s) as defined by Eqn (3).

Example 2. Figure 2 shows the data flow in Step 3 and Step 4 for publisher models $M_1(AB)$ and $M_2(BC)$ and the subscriber model $M_3(AC)$, with binary classes for the classification problem. Step 3 extends all models and subscriber's samples x to the feature universe ABC. After the extension, each extended sample x' is a valid input to the extended models M_1', M_2', M_3' because all are on the same feature space ABC. In particular, x' extends x by adding B (highlighted in pink) and M_1' extends M_1 by adding a never used input for C. M_1' is equivalent to M_1 receiving its valid inputs using x' but ignoring the added input C. Similarly M_2', M_3' extend M_2, M_3. The outputs of M_1', M_2', M_3' create the new training data $O^s = \{(m(x), y)\}$ defined by the samples (x, y) in the customization subset $Cus(D^s)$. Step 4 trains a fully connected NN f_c with O^s as the training data to learn the class distribution for x. Note that M_1, M_2, M_3 are *not* part of the fully connected network f_c; they are only used to create the training data O^s.

Discussion. We use this space to discuss the extension of x to x' by adding B. The subscriber has several options depending on the type of B. One method is to set $B = 0$ if B is categorical and we are using one-hot encoding. This will lead to B being ignored by the models having B in their feature set (i.e., models M_1 and M_2 in the example above). A second, more general approach is treating B as missing data and borrowing from the missing data literature [2].

For example, the subscriber can set the value for B to the dominant value for categorical B and to the average value for continuous B. Note that we consider such dominant values and average values to be publicly available and not shared by any individual site.

The creation of the training data O^s for model customization captures the knowledge of all models in *Sel* and depends only on the input-output mappings of the models, not their internal workings. This setup allows our customization approach to accommodate any type of underlying models (i.e., M_1, M_2, M_3 can be NNs, SVM, etc.) in the model fusion process, making PubSub-ML model-agnostic.

5 PUBSUB-ML

We are now ready to present the full PubSub-ML and its properties. Algorithm 3 refines Algorithm 1 with CUSTOMIZE($s, \mathcal{L}$) being the customization algorithm defined in Algorithm 2.

Algorithm 3 PubSub-ML

```
Require: Data sets $D^i$ for all sites $i$; privacy budgets $(\varepsilon^p, \delta^p)$ for
    all publishers $p$; appropriate loss function $\mathcal{L}$ (such as the cat-
    egorical cross-entropy for classification or the mean square
    error for regression).
1: for all publishers $p$ in parallel do
2:     train $(\varepsilon^p, \delta^p)$-DP models $M^p$ using DP models of their choice
       and publish the models;
3: end for
4: for all subscribers $s$ in parallel do
5:     $C^s \leftarrow$ CUSTOMIZE($s, \mathcal{L}$);
6: end for
```

Theorem 3. *Algorithm 3 is $(\varepsilon^p, \delta^p)$-DP for all publishers p and $(0, 0)$-DP for all subscribers s. For a site that is both a publisher and a subscriber, the DP guarantee for the publisher takes precedence.*

Proof. The publishers release models guaranteeing $(\varepsilon^p, \delta^p)$-DP on their data. In addition, since the data for different publishers is generated independently (see Assumption 1), the access to one publisher's models has no effect on the privacy loss of other publishers' models, thus, the $(\varepsilon^p, \delta^p)$-DP guarantee remains unchanged by accessing all published models. From Theorem 2, this privacy guarantee is unaffected by subsequent processing.

As for the subscribers s, the local data D^s, the local models M^s, and the customized model C^s are only used by the subscriber itself throughout Algorithm 3 and never made public. Therefore, no information about the subscriber is disclosed. □

Discussion. We emphasize the significance of Theorem 3 using an example. Let's consider five sites with individual privacy budgets, $\varepsilon^1 = 0.1$, $\varepsilon^2 = 0.2$, $\varepsilon^3 = 0.3$, $\varepsilon^4 = 0.4$, and $\varepsilon^5 = 0.5$, ignoring δs for simplicity. PubSub-ML provides site-level privacy guarantees, that is, 0.1 for site 1, 0.2 for site 2, 0.3 for site 3, 0.4 for site 4, and 0.5 for site 5. The existing DP-FL based methods, on the other hand, can only provide a uniform privacy guarantee ε across all sites, which has to be set according to the tightest privacy specification among all sites, that is, $\min\{0.1, 0.2, 0.3, 0.4, 0.5\}$; otherwise, the privacy of the site having the tightest privacy requirement will not be preserved. This leads to reduced utility. Note that while we provide site-level privacy guarantees, the notion of differential privacy is record-level as the standard one, instead of site-level.

Theorem 4. *PubSub-ML in Algorithm 3 meets Autonomy, Asymmetric sharing, Personalized privacy, and Model-agnostic requirements proposed in Section 1.*

Proof. *Autonomy* holds because we make no assumption about the local decisions on feature space, samples, and training process, which are made independently by each site. *Asymmetric sharing* holds because each site has the freedom to participate as a publisher (contributor only), as a subscriber (beneficiaries only), or both. *Personalized privacy* holds because each publisher p can set its own privacy budget $(\varepsilon^p, \delta^p)$. *Model-agnostic* holds because all models M_i in *Sel* are treated as black-boxes by the customization algorithm as defined by the mapping function $m(x)$ that depends only on the output $M_i'(x')$ for models M_i. That is, sites have the freedom to choose any model type and architecture for their local models. □

6 HANDLING DATA STREAMS

So far, we assume that at each site i the whole data D^i is available. In the real world, data often arrives in the form of a continuous data stream, such as real-time streaming of electronic health records (EHR) from multiple hospitals, credit card transactional data flows from multiple financial institutions, social media content, etc. For data streams, underlying classification patterns are subject to change with time, a phenomenon known as the concept drift. In this section, we extend PubSub-ML to data streams in the decentralized learning setting where each site i generates a data stream D^i with disjoint data chunks, $D_1^i, \cdots, D_\tau^i$, up to the *current time point* τ, where each data chunk D_u^i is a collection of samples generated during a short time duration represented by the time index u. For data streams, in addition to Assumption 1, we further assume

Assumption 2. *$D_1^i, \cdots, D_\tau^i$ are pairwise disjoint and each D_u^i is generated independently of other data chunks D_v^i.*

The assumption on independent records is a standard assumption made in differential privacy [8]. In case of correlated observations, such as multiple records from the same user, group differential privacy [8, 10] could be used instead of the standard differential privacy. For the rest of this work, Assumption 2 is in place.

To ensure that models' performance stays up-to-date, at each time point, each publisher needs to update published models incorporating the new data becoming available, and each subscriber needs to re-customize its models whenever the subscribed models are updated. At one extreme, a publisher can publish a single model at each time point τ using the union of all data chunks $D_1^i, \cdots, D_\tau^i$. This approach however does not work as *first*, the older, irrelevant data can deteriorate a model's performance in the presence of concept drift and *second*, with differential privacy, retraining and re-releasing the model leads to depletion of privacy budget due to the composition of privacy loss. At the other extreme, a publisher can publish a single model at each time point τ using the most recent data chunk D_τ^i. This approach suffers from the "small data size" of a single data chunk that is collected during a short time period.

To address the above issues, we use the proven *dynamic ensemble* approach [13, 46] consisting of an ensemble of k models, each being trained on one of recent data chunks from D^i, $D^i_{\tau-k+1}, \cdots, D^i_\tau$. As the next data chunk $D^i_{\tau+1}$ becomes available, one model in the ensemble is replaced with the model trained on the new chunk. k is called the *ensemble size* and acts as a trade-off between leveraging recent data and phasing out obsolete data. This approach must address three issues: i) how to generate the initial DP ensemble, ii) as the time progresses from τ to $\tau + 1$, how to update the current ensemble and ensure that the DP guarantee holds after the update, and iii) how to update the customized model. i) is accomplished by line 2 of Algorithm 3 with each of the k local DP models in the ensemble being trained using one local data chunk. There is a cold-start period where the initial ensemble is generated only at time point k when the first k chunks $D^i_1, \cdots, D^i_k$ are available. iii) is accomplished by line 5 of Algorithm 3 but now k models from each publisher will participate in the model selection and model fusion. Below, we focus on ii).

6.1 Updating the Current Ensemble

[13] presents a DP mechanism for releasing an ensemble $\mathcal{E}_\tau = \{M_{\tau(1)}, \cdots, M_{\tau(k)}, w_{\tau(1)}, \cdots, w_{\tau(k)}\}$ at time τ. $\tau(i)$ denotes the (absolute) time point corresponding to the (relative) position i within $\mathcal{E}_\tau$, $M_{\tau(i)}$ is the noisy model trained on the data chunk at time $\tau(i)$, and $w_{\tau(i)}$ is the noisy weight for $M_{\tau(i)}$. The privacy budget (ε, δ) is split into ε_1 for training models $M_{\tau(i)}$ and ε_2 for computing weights $w_{\tau(i)}$ when both models and weights are released, where $\varepsilon = \varepsilon_1 + \varepsilon_2$. In the context of PubSub-ML, a publisher p publishes only the models $M^p_{\tau(i)} \in \mathcal{E}^p_\tau$, not the ensemble weights $w_{\tau(i)}$, because a subscriber s will use its own data chunk D^s_τ to construct the customized model (line 5, Algorithm 3). Therefore, the whole privacy budget $(\varepsilon^p, \delta^p)$ for p is allocated for training the model $M^p_{\tau(i)}$, i.e., $\varepsilon_1 = \varepsilon^p$ and $\varepsilon_2 = 0$. Also, as the new data chunk $D^p_{\tau+1}$becomes available, we train a new DP model $M^p_{\tau+1}$ to replace the oldest model in $\mathcal{E}^p_\tau$ to get $\mathcal{E}^p_{\tau+1}$. In this case, $\mathcal{E}^p_\tau = \{M^p_{\tau-k+1}, \cdots, M^p_\tau\}$. See the detail of update in Algorithm 4.

Algorithm 4 UpdateEnsemble($\tau, \mathcal{E}^p_\tau, D^p_{\tau+1}$)

```
Require: Publisher $p$; the time $\tau$; the ensemble $\mathcal{E}^p_\tau = \{M^p_{\tau-k+1}, \cdots, M^p_\tau\}$; the data chunk $D^p_{\tau+1}$ at time $\tau + 1$; the privacy budget $(\varepsilon^p, \delta^p)$.
1: Train $(\varepsilon^p, \delta^p)$-DP model $M^p_{\tau+1}$ using the chunk $D^p_{\tau+1}$
2: $\mathcal{E}^p_{\tau+1} \leftarrow \{M^p_{\tau-k+2}, \cdots, M^p_{\tau+1}\}$
3: return $\mathcal{E}^p_{\tau+1}$
```

6.2 DP Guarantees

For a publisher p, to prove the DP-guarantee of the ensemble $\mathcal{E}^p_\tau$, we first extend DP in Definition 1 to an output that is an ensemble (instead of a single model). Consider $X^p_\tau =< D^p_{\tau-k+1}, \cdots, D^p_\tau >$ and $X'^p_\tau =< D'^p_{\tau-k+1}, \cdots, D'^p_\tau >$, where D^p_i and D'^p_i are chunks at time i. We say that X^p_τ and X'^p_τ are *neighboring* if $\cup_i D^p_i$ and $\cup_i D'^p_i$ (duplicates preserved) are neighboring in the sense of Definition 1.

Definition 2 (DP for a Single Ensemble [13]). *A mechanism $\mathcal{M}$ from the domain of X^p_τ to the range of $\mathcal{E}^p_\tau$ is (ε, δ)-DP if for all neighbouring pairs (X^p_τ, X'^p_τ) and for all sets O of possible outputs:*

$$\Pr[\mathcal{M}(X^p_\tau) \in O] \le e^\varepsilon \Pr[\mathcal{M}(X'^p_\tau) \in O] + \delta \tag{4}$$

Theorem 5. *If each model M^p_i in $\mathcal{E}^p_\tau$ is produced by a $(\varepsilon^p, \delta^p)$-DP mechanism, the mechanism that produces $\mathcal{E}^p_\tau = \{M^p_{\tau-k+1}, \cdots, M^p_\tau\}$ is $(\varepsilon^p, \delta^p)$-DP.*

Proof. We note that each model in $\mathcal{E}^p_\tau$ is trained on a *disjoint* data chunk and each data chunk is generated independently (See Assumption 2). Then theorem follows from the parallel composition in Theorem 1. □

As the ensemble is updated repeatedly with the arrival of new data chunks, the adversary observes all the released ensembles $\mathcal{H}^p =< \mathcal{E}^p_1, \cdots, \mathcal{E}^p_\tau >$ up to time τ. To ensure DP on the released ensemble histories, we further extend DP to the global training data $\mathcal{X}^p =< X^p_1, \cdots, X^p_\tau >$ where each X^p_t is the training data for the ensemble $\mathcal{E}^p_t$, $1 \le t \le \tau$. We say that $\mathcal{X}^p$ and $\mathcal{X}'^p$ are *neighboring* if exactly one pair (X^p_t, X'^p_t) is neighboring, and for all other $j \ne t$, $X^p_j = X'^p_j$.

Definition 3 (DP for a History [13]). *A mechanism $\mathcal{M}$ from the domain of $\mathcal{X}^p$ to the range of $\mathcal{H}^p$ is (ε, δ)-DP with respect to history if for any neighbouring pair $(\mathcal{X}^p, \mathcal{X}'^p)$ and for all sets O of history outputs $\mathcal{H}^p$:*

$$\Pr[\mathcal{M}(\mathcal{X}^p) \in O] \le e^\varepsilon \Pr[\mathcal{M}(\mathcal{X}'^p) \in O] + \delta \tag{5}$$

Theorem 6 ([13]). *Assume that the initial $\mathcal{E}^p_1$ is produced by a $(\varepsilon^p, \delta^p)$-DP mechanism, all subsequent ensemble updates are $(\varepsilon^p, \delta^p)$-DP with respect to history.*

The reason for Theorem 6 is the same as Theorem 5. Each model is trained on a disjoint data chunk generated independently. The difference is that each model is published in k consecutive ensembles, but this does not affect applying Theorem 1 (parallel composition). From Theorem 6, even if the adversary has access to all released ensembles, the privacy loss does not accumulate over time. This property is vital because the number of ensemble updates is potentially unbounded for data streams.

Discussion. One method to deploy PubSub-ML in real-world applications is where all sites register as publishers or subscribers via an API, and register their feature spaces, subscription channels, and feature metadata. This information is used to stream trained models directly from publishers to subscribers, and to assist participating sites to resolve feature semantic and feature encoding ambiguities. The API can be managed by either one of the sites or an external entity. Importantly, such registration information is at the metadata level and involves no information about models and raw data. For this reason, the trust level of the managing site of the API is not as high as that of the server for aggregating gradient updates in the FL framework. PubSub-ML is indispensable for institutional collaboration where DP-FL does not work due to the forced tight collaboration. We are collaborating with multiple healthcare organizations to implement PubSub-ML for breast cancer detection in a decentralized learning setting.

As discussed in Section 1.1 and Section 4, PubSub-ML is applicable in all scenarios where the problem can be formulated as an optimization problem with an objective defined by a loss function. For evaluation purpose, we focus on classification and regression, where we evaluate PubSub-ML against eight baselines on six real-world and synthetic datasets. In Section 7 we explain our evaluation setup followed by evaluation results in Section 8.

7 EVALUATION SETUP

Our evaluation is carried out using an Intel Xeon CPU (2.20GHz) with 32 GB RAM and an NVIDIA Tesla P100 GPU. We begin this section by describing our data setup in detail, where to evaluate the utility of PubSub-ML and the other baselines, we need to simulate multi-site scenarios with varying data and feature distributions where each site holds their own data.

7.1 Datasets

NELA-GT-2018. NELA-GT-2018 [16] (NELA) is a publicly available dataset for the study of misinformation and consists of articles from multiple news sources collected between 03/2018-11/2018. Using weak labelling, the articles are labelled as reliable or unreliable based on the originating source. The goal is to classify articles into reliable/unreliable categories. We use the same data and feature extraction as in [16][3]. We combine reliable/unreliable sources to create four distinct sites (Table 1). Similar writing styles within a site and varying writing styles across sites allows us to evaluate PubSub-ML under the IID (Scenario 1, Section 8.1.1) and non-IID (Scenario 2, Section 8.1.2) data distribution settings. Further details on the creation of scenario-specific publishers and subscribers are described in Sections 8.1.1 and 8.1.2.

Site	Rel.	Unrel.	$|D|$ (% Rel.)
1	Reuters NPR	True Pundit Natural News	27063 (35%)
2	USA Today CNN	Infowars Veterans Today	18829 (74%)
3	The NY Times CBS News	Activist Post Mint Press News	14290 (76%)
4	BBC Chicago Sun	News Wars Prison Planet	26078 (71%)

Table 1: NELA-GT-2018 data. Each site is defined as the collection of articles from two reliable sources and two unreliable sources. $|D|$ shows the data size for each site and % Rel. is the average percent of reliable samples within the site.

MIMIC-IV. MIMIC-IV (v1.0) [17] is a publicly available[4] database of de-identified hospital admissions of 382,278 patients in critical care units of the Beth Israel Deaconess Medical Center from 2008-2019. The admission level data are stored in multiple tables, linkable by Admission ID, where each hospital admission is a sample. We consider the problem of in-hospital mortality prediction for patients with Sepsis using the Sepsis-3 criteria [40][5], leading to a total of 34,789 admissions with a mortality rate of 16%. We create three tables A, B, C using original MIMIC-IV tables so that each of A, B, C contains a different type of patient information (Table 2). Table A contains the dynamic patient information stored in an ICU/hospital setting (length of stay, comorbidities, etc.), table B contains patient demographics (age, admission year group, etc.), and table C contains dynamic information from the Lab system (WBC count, hemoglobin, etc.). This allows us to evaluate PubSub-ML under the scenarios when different sites hold different feature sets (Scenario 3, Section 8.1.3 and Scenario 4, Section 8.1.4), where we use A, B, C to create the data for scenario-specific publishers and subscribers.

A	B	C
ICUStay_Detail	Admissions	BG (Blood Gas)
EMAR	Patients	Blood_Differential
Diagnoses_ICD		
Procedures_ICD		
Sofa; Oasis; Charlson		
Services		

Table 2: MIMIC-IV data. We create three tables A, B, and C by joining the original MIMIC-IV tables shown in columns A, B, C, where the join is via Admission ID. Note that Sofa, Oasis, and Charlson are separate tables. The class label mortality is extracted from the ICUStay_Detail table.

Digit-Five. Digit-Five[6] is a collection of five image datasets. MNIST (Train N = 55000; Test N = 10000), MNIST-M (Train N = 55000; Test N = 9001), Synthetic Digits (Train N = 25000; Test N = 9000), SVHN (Train N = 73257; Test N = 26032), and USPS (Train N = 7438; Test N = 1860), where all five datasets are of different size and are from different domains with a common goal of classifying digits. The dataset has been used in prior studies to study federated learning in the non-IID setting [34]. We use this dataset collection to evaluate PubSub-ML under the non-IID setting (Section 8.3.1) involving complex, convolutional neural network based models. Further details around the publishers/subscribers setup is presented in Section 8.3.1.

Amazon Reviews. Amazon reviews dataset[7] consists of Amazon customer reviews (input text) and star ratings (output labels) for various products. We use the dataset for sentiment prediction where the ratings of 1 and 2 stars are considered negative and the ratings of 4 and 5 star considered positive. 3 star ratings are considered neutral and excluded from the dataset. For the evaluation, we consider reviews shorter or equal to 25 words in length, leading to a total of 63720 reviews. We use this dataset in addition to Digit-Five to evaluate PubSub-ML under complex modelling scenarios (Bidirectional LSTM in this case, Section 8.3.1). We create multiple sites by splitting the dataset into 10 equal, non-overlapping partitions, simulating the IID data scenario where sites want to benefit from the text data of other sites. Further details on publishers and subscribers are provided in Section 8.3.1.

Housing Market. Housing market [39] dataset contains the property information, with the goal of predicting the continuous

[3]https://pypi.org/project/nela-features/
[4]Access request required, link https://bit.ly/3bR690s
[5]https://github.com/MIT-LCP/mimic-iv/tree/master/concepts

[6]Available for download at https://bit.ly/3OMoZno
[7]Available for download at https://bit.ly/3dcHxQ3

property price. The dataset has 30,473 observations and 292 attributes. This dataset is used to evaluate PubSub-ML for regression problems (Section 8.3.2). To create multiple sites, we divide the dataset into equal, non-overlapping 30 partitions, where we consider each partition to be a separate site, resulting in an average sample size of 1016 observations per site. This simulates the IID data scenario where each site is a separate financial/real-estate institution holding it's own data for a subset of customers. We provide details on the creation of publishers/subscribers using the sites in Section 8.3.2.

Hyperplane. Hyperplane is a synthetic data used extensively in prior studies [13, 46] with the goal to classify points separated by a hyperplane. We generate the dataset using the `HyperplaneGenerator()` function[8] with the hyperparameters shown in Table 3. The dataset provides us the flexibility to generate *large* number of sites with varying data distribution to evaluate PubSub-ML's model selection capabilities and runtime (Section 8.2.3), and the impact of noisy sites (Section 8.2.5). More details on the dataset usage including the creation of publishers/subscribers are provided in Sections 8.2.3 and 8.2.5.

p_1	p_2	p_3	p_4	p_5
100	2	$\mathcal{N}(0,0.4)$	$\mathcal{N}(0,0.2)$	0.0

Table 3: The parameters of the point generating function `HyperplaneGenerator()`. p_1 is the number of features in the data, p_2 is the number of dynamic features, p_3 is the noise percent, p_4 is the magnitude of change on hyperplane rotation, and p_5 is the probability that the direction of change is reversed. As p_3 and p_4 can only be positive, we sample absolute values.

7.2 Models

We standardize continuous features using StandardScaler from scikit-learn [33] and use one-hot encoding for categorical features. For model training involving datasets NELA, MIMIC-IV, Hyperplane, and Housing Market (for both publishers and subscribers), we consider the fully connected neural network. For the dataset Digit-Five, we use a CNN with six hidden layers, the same architecture as used in publicly available repositories[9], and for the dataset Amazon reviews, we use bidirectional LSTMs with two bidirectional LSTM layers and two dense layers (similar to publicly available architecture[10]) and we follow the standard NLP data preprocessing practices using the publicly available code[11].

The architecture and training details for the models are provided in Table 4. For the customization (model fusion) network, we consider the input layer defined by the mapping $m(x)$ fully connected to a new output layer. All models for all datasets, for publishers, subscribers, and customization are trained for 30 epochs with a minibatch size of 100.

[8]https://bit.ly/3o0nPJI
[9]https://bit.ly/3I4Wyiq
[10]https://bit.ly/3JLQKv2
[11]https://bit.ly/3SCOcU3

Data	HL	HL_{act}	OL	OL_{act}
NELA	(80,40)	TanH	2	Softmax
MIMIC-IV	(120,60)	TanH	2	Softmax
Digit-Five	(NA)	ReLU	10	Softmax
Amazon Reviews	(NA)	TanH	2	Softmax
Housing Market	(416,220,60)	TanH	1	Linear
Hyperplane	(100,50)	TanH	2	Softmax

Table 4: Architecture details for the models used. HL and OL specify the number of units in the hidden layers and the output layer with *act* being the activation function. As Digit-Five and Amazon reviews use CNN and bidirectional LSTM based models (with architecture publicly available), the simpler HL size specification is "NA". All sites for all datasets use Adam [22] as the optimizer with the common model parameters as shown, selected via hyperparameter search for fully connected NN models.

7.3 Competitor Methods

We evaluate **PubSub-ML** against the following eight baselines. **PubSub-ML***: The non-private version of PubSub-ML where publishers publish models without privacy constraints. This provides us with the upper-bound on achievable utility for PubSub-ML. All other baselines that follow are privacy-preserving. **Own**: The subscriber's own model trained on local data without the help of subscribed models and without noise addition as it is never published. This helps us study the added utility of using publishers' models. **FedAvg(V)**: The vanilla DP-FL based on FedAvg [30] where clients share their parameters without privacy-preservation, and the server adds noise to guarantee DP [12]. **FedAvg(F)**: This is the personalized DP-FL where a site further finetunes the global model produced by FedAvg(V) using the sites' own data in a non-private fashion [11, 50]. **FedProx(V)**: DP version of FedProx [26], which has been shown to work well with heterogeneous client datasets (non-IID setting) compared to vanilla FL in the non-private setting. **FedProx(F)**: Personalized version of FedProx(V). **FedAdam(V)**: DP version of Federated Adam [37], an adaptive optimizer based approach for FL, proven to have better convergence rates for FL and shown to outperform other FL methods such as SCAFFOLD [21] in the non-private setting. **FedAdam(F)**: Personalized version of FedAdam(V). For FedAvg, FedProx, and FedAdam, we use the hyperparameters as suggested by their respective authors.

Note that the network architecture and other parameters of all baselines are identical to those used for training local models in PubSub-ML with all clients sampled per-round for FL due to the small number of sites unless specified otherwise. We do not consider other variants of FL such as Assisted learning [48] as they require the alignment (or partial alignment) of samples across sites, but our scenarios studied below do not have this restriction. We do not consider the model fusion approach such as [23] because if training synchronization is not an issue, personalized FL (such as FedAvg(F)) outperforms the approach in [23][13].

[12]https://github.com/tensorflow/federated
[13]Confirmed via private communication with the authors of [23]

8 RESULTS

We evaluate models on the scenario-specifically created subscriber's test set, using the F1-score of the minority class for imbalanced datasets (NELA and MIMIC-IV), the classification accuracy for Hyperplane, Digit-Five, and Amazon reviews, and 1 - Mean Squared Error (1 - MSE) for Housing market (to ensure a common comparison, i.e. higher the better). For all evaluation, we consider that a subscriber subscribes to all available models from all publishers. In most of our evaluation, we evaluate PubSub-ML on static datasets (using the data setup as described in Section 7.1), with PubSub-ML on data streams evaluated in Section 8.3.3, and we use all available models for customization, with the impact of model selection studied in Section 8.2.3. We report average results with standard errors using the 5-fold cross-validation. More specifically, for a publisher 80% data is used for training the published models, and for a subscriber 60% data is used for training the local models (i.e., the training subset) and 20% data is used for customization (i.e. the customization subset), and 20% for testing (i.e., collecting the F1-score, classification accuracy, MSE). Note that unless specified, the privacy budget is fixed at ($\varepsilon = 1, \delta = 0.0001$) with the DP guaranteed using DPSGD [1].

Our results are organized as follows. We begin with evaluating PubSub-ML against the baselines for varying data and feature distribution scenarios in Section 8.1. In Section 8.2, we demonstrate the robust nature of PubSub-ML by studying its behavior in detail with respect to the impact of privacy budget, model heterogeneity among publishers, the number of models used by the subscriber out of all available models and runtime, and noisy sites. This is followed by further evaluation of PubSub-ML in Section 8.3 in the case of complex models (CNNs and bidirectional LSTMs), regression, data streams, and in the non-private setting.

8.1 Main Comparison

We begin with evaluating the effectiveness of PubSub-ML claimed in Section 1.1 under several scenarios of data (IID, non-IID) and feature distribution.

8.1.1 Scenario 1 (IID Data). All sites have the same data distribution but small data size.
Setup. Using one site of NELA in Table 1 at a time, we create sites with *similar* data distribution as follows. For a site e under consideration, we divide the dataset within the site into ten non-overlapping partitions. By treating each partition as coming from an individual site, we have ten sites with datasets of similar distribution within each site e. Each of the ten sites takes turn being the subscriber s while the remaining nine sites act as publishers. Let $\mathrm{F1}_{e,s}$ denote the F1 score for the subscriber s when the datasets are created using the site e as above. We report the average of $\mathrm{F1}_{e,s}$ over all e, s pairs.

Results. Figure 3a shows the results. PubSub-ML outperforms all privacy-preserving baselines by a significant margin, that is, on average 40% and 38% over the different flavors of FL (vanilla and finetuned, shown in Figure 3a as different tints of red), and 6% over the subscriber only, non-noisy model (Own), with the performance of PubSub-ML being close to the non-private PubSub-ML* (a difference of 4%). Consistent with [50], we find that the DP-FL models perform much worse compared to the Own models trained by individual sites. One of the main reasons for PubSub-ML's superior performance is the ability of PubSub-ML to leverage multiple weak learners including the non-private model and data from the subscriber to create a strong learner that outperforms all individual and DP-FL based models, as discussed in Section 1.1.
Summary. *PubSub-ML allows the subscriber to integrate multiple weak learners to produce a strong learner.*

8.1.2 Scenario 2 (Non-IID Data). Each publisher has a different data distribution.

Setup. For this scenario, we consider each site of NELA in Table 1 as a publisher with a different distribution due to different writing styles and news content across sites (we observed average F1-score drop of 19% when applying the model trained on one site's data to the content from other sites). The subscriber is created by each site donating 10% of their data, and using the remaining 90% as their full dataset, so the subscriber has a sample of each publisher's data.

Results. Figure 3b shows the results. Similar to Scenario 1, PubSub-ML significantly outperforms Own and all FL models, supporting our claim that PubSub-ML can effectively integrate sub-distributional models for inference on the combined distribution of the subscriber.
Summary. *PubSub-ML allows the subscriber to successfully integrate models trained on non-IID data.*

Scenarios 1 and 2 assume that all sites have the same feature space. In Scenarios 3 and 4, we evaluate PubSub-ML when the sites have different feature spaces with same or different populations.

8.1.3 Scenario 3 (Distributed Features, Same Population). The subscriber's features are distributed among several publishers with same population.

Setup. Using MIMIC-IV for this scenario (Table 2), where A and C are most relevant for sepsis related mortality classification and B is least relevant[14], we create three sites, where site 1 contains the data by joining A and B, site 2 contains the data by joining B and C, and site 3 contains the data by joining A and C (All joins are via Admission ID). Site 1 and site 2 are publishers and site 3 is the subscriber. Intuitively, the subscriber's useful features contained in A and C are distributed among site 1 and site 2, and all sites share the admission ids (same population). Further, to simulate the scenario where subscriber has a small number of observations on useful features from A and C, we reduce the data for the subscriber by randomly sampling 10% of the available data for the subscriber.

Results. Figure 3c shows the results. We observe that PubSub-ML performs the best among privacy-preserving methods, with performance closer to the non-private counterpart, despite the severe class-imbalance in the dataset. FL performs worse in the scenario where sites have different sets of features (compared to Scenarios 1 and 2). For FL baselines that require the same feature space across sites, we keep the data setup same as PubSub-ML where each site maintains a conceptual set of all features and sets the missing features in its own data to zero on the input layer.
Summary. *Model customization in PubSub-ML helps the subscriber to successfully integrate publishers' models trained on different feature sets.*

[14]removal of B has only 2% impact on the model performance

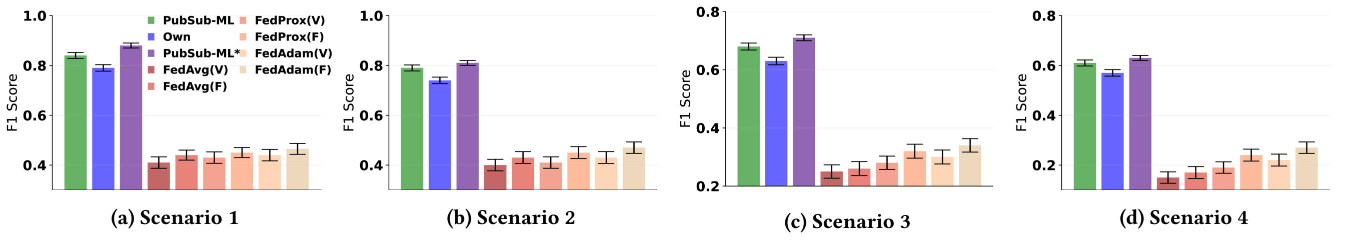

Figure 3: PubSub-ML 🟩 significantly outperforms all privacy-preserving baselines: Own 🟦, FedAvg(V) 🟥, FedAvg(F) 🟥, FedProx(V) 🟧, FedProx(F) 🟧, FedAdam(V) 🟨, and FedAdam(F) 🟨 for Scenario 1 (IID data), Scenario 2 (non-IID data), Scenario 3 (distributed features, same population), and Scenario 4 (distributed features, different population). PubSub-ML provides utility closer to the non-private PubSub-ML* 🟪.

8.1.4 *Scenario 4 (Distributed Features, Different Populations). The data setup is similar to scenario 3, but the subscriber and the publishers have different populations.*

Setup. We use MIMIC-IV for this scenario, with similar setup as in scenario 3. However, the subscriber now keeps the data for patients aged ≤ 40 whereas the publishers keep only the data for patients aged > 40. This leads to the subscriber having a younger/different population with a total of 2,694 admissions and a mortality rate of 10% compared to 17% for publishers.

Results. We observe from Figure 3d that PubSub-ML provides excellent inference for the subscriber's younger patient population by benefiting from the models trained on the publishers' data for different feature sets and populations. With the difference in population compared to scenario 3, the performance is decreased for all methods, but PubSub-ML still performs significantly better than all privacy-preserving baselines by combining the features from all publishers' models and optimizing the utility on the subscriber's data.

Summary. *The subscriber in PubSub-ML equally benefits from the publishers in case of distributed features for same and different populations.*

8.2 Further Analysis

The studies in Section 8.1 show that PubSub-ML significantly outperforms all privacy-preserving competitors under varying conditions of data and feature distribution. Before we evaluate PubSub-ML on additional scenarios involving complex models, regression, data streams with concept drift, and in the non-private setting, we use this section to focus on PubSub-ML to demonstrate its robust nature by showing that a subscriber can expect *good* results under a variety of possible scenarios, such as with respect to the publishers' choice of privacy budget and the choice of model types, with respect to the subscriber selecting a *few* good models out of all available models, minimizing runtime, with respect to varying number of available publishers, and in the presence of noisy sites. Due to the space constraints, we focus on the dataset NELA for studying the impact of privacy budget and model heterogeneity among publishers with the data setup same as in Scenario 2 (Section 8.1.2), and we use the dataset Hyperplane to study model selection, runtime, the impact of number of publishers and subscribers, and the impact of noisy sites with details around the data setup provided in Sections 8.2.3, 8.2.4, and 8.2.5. We begin with exploring the impact of privacy budget.

8.2.1 *Impact of Privacy Budget.*

Setup. We have seen that PubSub-ML performs well with the *tight* privacy budget of $(\varepsilon = 1, \delta = 0.0001)$. Next, we investigate PubSub-ML's behavior when we vary the privacy budget. Keeping other parameters fixed, we investigate two scenarios. First, where all publishers use a uniform privacy budget. Second, to demonstrate the personalized privacy property of PubSub-ML, all publishers use heterogeneous privacy budgets. For the first scenario, we only focus on PubSub-ML's performance where we vary ε from 1 to 0.25 while keeping the δ fixed at 0.0001. For the second scenario, keeping δ fixed at 0.0001, for each publisher, we uniformly randomly sample ε from the interval of [0.25,1.0]. Lets denote the sampled ε values by $\{\varepsilon^1, \varepsilon^2, \varepsilon^3, \varepsilon^4\}$. The competitors, to ensure the same level of DP, use $\varepsilon = \min\{\varepsilon^1, \varepsilon^2, \varepsilon^3, \varepsilon^4\}$. We repeat the procedure 10 times and report the average results with standard error.

Results. Table 5, first row shows the results of first scenario, where as the privacy budget (ε) decreases, the performance decreases. However, even at the tightest privacy setting $(\varepsilon = 0.25)$, the performance of our proposed model fusion decays by only 4%, still outperforming FL baselines trained with $\varepsilon = 1$ (Scenario 8.1.2). For space constraints, the results of second scenario are provided in the Appendix (Section A, Figure 7). Where similar to the first scenario, we observe superior performance of PubSub-ML. This is because our proposed model fusion creates a robust model by leveraging the diversity of multiple models with varying utilities, learning to assign higher importance to high-utility models.

Summary. *PubSub-ML maintains good utility with decreasing privacy budget.*

8.2.2 *Impact of Model Heterogeneity.*

Setup. The model-agnostic PubSub-ML allows models of different types to be integrated. In this experiment, keeping all parameters fixed, i.e., privacy budget=$(\varepsilon = 1, \delta = 0.0001)$, we change the model types. Initially, all four publishers use neural network models, so model heterogeneity is 1. Then we replace publisher 1 with a DP-Logistic Regression, publisher 2 with a DP-Gaussian Naive Bayes, and publisher 3 with a DP-XGBoost, leading to the model heterogeneity of 4. Our implementation of DP-Logistic Regression, DP-Gaussian Naive Bayes, and DP-XGBoost is based on the publicly available source code[15].

Results. Table 5, second row shows the results. There is a slight performance degradation as model heterogeneity increases, likely

[15]https://github.com/IBM/differential-privacy-library
https://github.com/sarus-tech/dp-xgboost

because non-neural network based DP models suffer from worse utility.

Summary. *Model customization in PubSub-ML successfully integrates heterogeneous models from different publishers.*

Privacy Budget	$\varepsilon = 1$	$\varepsilon = 0.5$	$\varepsilon = 0.25$
F1 Score	0.79±0.02	0.77±0.02	0.75±0.02

Model heter.	1	4
F1 Score	0.79±0.02	0.76±0.03

Table 5: The impact of uniform privacy budget (top) and model heterogeneity (bottom). PubSub-ML's performance is robust to the choice of privacy budget and model type.

8.2.3 *Impact of Model Selection and Runtime.*

Setup. Using the synthetic data hyperplane, we generate 1000 *publishers*, each with 500 points, using the parameters specified in Table 3 and we consider 1000 subscribers using the same p_4 as publishers but their $p_3 = 0$ (Table 3), creating an IID data setup and we consider one subscriber at a time. All 1000 publishers release a model, hence the subscriber has 1000 subscribed models plus its own model for integration. We report the average performance (with standard errors) over 1000 subscribers. Further, due to the abundance of participants, FL based approaches take advantage of sampling randomly 10% of the clients for participation at each round.

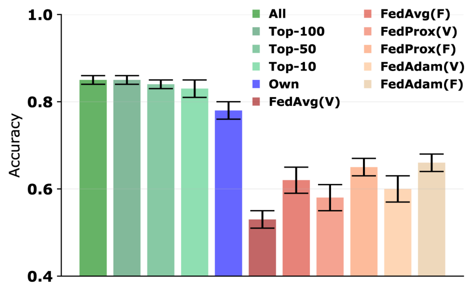

Figure 4: Model Selection - Accuracy when fusing All, Top-100, Top-50, and Top-10 models from PubSub-ML compared to accuracy using just own model and different flavors of FL.

Results. Combining the results from Figure 4 and Table 6, we conclude that PubSub-ML significantly outperforms all FL based methods for utility and runtime (FL(V) and FL(F) capture the average runtime of all FL methods), even when FL enjoys reduced noise due to smaller client sampling ratio. We can further significantly reduce the runtime of the subscriber using top-few models at a marginal loss on accuracy, where top-100 models provide similar accuracy as using all 1001 models. Such model selection would be a preferred choice when the subscriber has limited computational resources.

Summary. *PubSub-ML provides excellent utility using top-few models, with faster runtimes compared to FL based methods.*

Time (s)	All	T-100	T-50	T-10	FL(V)	FL(F)
Load	108	108	108	108	NA	NA
Select	0	6	6	6	NA	NA
Fuse	77	29	24	21	NA	NA
Total	185 ± 5	143 ± 5	138 ± 5	135 ± 5	683 ± 8	689 ± 8

Table 6: Runtime (seconds). T-100 is Top-100, T-50 is Top-50, etc. For PubSub-ML, the time is divided into model load (Load), model evaluation and selection (Select), and model fusion (Fuse). Model training time at publishers is not evaluated because the training can be done in parallel. NA is not applicable for a given cell. The time for Load and Select is same for all number of models selected as the subscriber first needs to load and Select from all available models, unless using All, where the Select is not required.

8.2.4 *Impact of Number of Publishers and Subscribers.*

Setup. Using the same settings as Section 8.2.3 to create publishers and subscribers, we investigate the impact of number of publishers and subscribers in PubSub-ML. Specifically, there are *n* publishers and *n* subscribers, and we vary *n* from 5 to 5000. For each of *n* publishers, we report the average performance over all *n* subscribers.

Results. For space constraints, detailed results are presented in the appendix (Section B). In summary, initial increase in the number of publishers leads to better performance for the subscribers.

8.2.5 *Impact of Noisy Sites.*

Setup. In the real world, not all participating sites have *good quality* data. Some sites might have noisy data that can be detrimental to the subscriber's performance. To evaluate the impact of noisy sites in PubSub-ML, we use the dataset Hyperplane as follows. We create a total of 20 sites, with 10 sites acting as publishers, out of which 5 are created using the same parameters as specified in Section 8.2.3, while the other 5 set $p_3 = 1$ (Table 3), generating pure noise. The 10 subscribers are generated using the same parameters as specified for subscribers in Section 8.2.3, and we report results as an average over the 10 subscribers.

Results. Due to space constraints, detailed results are presented in the appendix (Section C). In summary, results show that PubSub-ML can successfully ignore bad quality data/models, providing excellent utility, whereas FL based models suffer from degraded utility.

8.3 More Applications

So far, in Sections 8.1 and 8.2, using simple fully connected NNs, we saw that PubSub-ML significantly outperforms all competitors for varying data distribution and feature distribution scenarios for classification on static datasets. In this section, we provide additional evidence of PubSub-ML's utility for problems related to the use of complex models (CNNs and bidirectional LSTMs, Section 8.3.1), regression (Section 8.3.2), data streams with concept drift (Section 8.3.3), and in the non-private setting (Section 8.3.4).

8.3.1 *Complex Models.*

Setup. For computer vision, we use CNN based models and the Digit-Five dataset under the non-IID setting, where we consider each of the five datasets belonging to a separate site. We create the

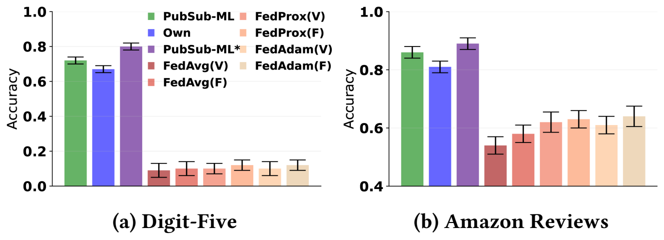

Figure 5: Complex models. In the complex model setting involving image and NLP datasets, PubSub-ML outperforms FL based competitors by an average of $> 40\%$.

subscriber by each site donating 1000 randomly chosen observations from their test set, leading to the subscriber containing 5000 observations of mixed distribution from all publishers. For NLP, we use bidirectional LSTM based model and the Amazon reviews dataset under the IID setting, where each site takes turn being the subscriber while the other nine sites act as publishers.

Results. Figure 5 shows the results, where from Figure 5a and 5b, we observe that PubSub-ML significantly outperforms all FL based approaches with an average accuracy improvement of $> 40\%$, similar to the fully connected NN scenarios in the main comparison (Section 8.1). In the scenario involving CNN models on non-IID data, we observe that the FL based privacy-preserving approaches perform worse compared to the earlier scenarios that use simple, fully connected NNs, likely due to the convergence issues faced by DP-FL for complex models under the non-IID setting.

Summary. *PubSub-ML performs equally well on image and text datasets using complex models (CNNs and bidirectional LSTMs).*

8.3.2 *Regression.*

Setup. For regression, we use the housing market dataset under the IID setting, where each site takes turn being the subscriber while the other 29 sites act as publishers. We report average results using 1-MSE over all such subscribers.

Results. For space constraints, detailed results are presented in the Appendix (Section D). In summary, PubSub-ML performs similarly well for regression as for classification.

8.3.3 *PubSub-ML for Data Streams.*

Setup. To evaluate the PubSub-ML's extension for data streams with concept drift (Section 6), we create data streams using the datasets with a temporal component (NELA, MIMIC-IV, and Hyperplane) in the following way. For the dataset NELA, the data chunks are defined as the news collected bi-weekly, leading to a total of 22 data chunks, with an average chunk size of 1006 observations, with the same data setup for publishers and subscribers as Scenario 2. For the dataset MIMIC-IV, the data chunks are created by splitting each three-year admission interval (only available date in the database) into three disjoint adjacent chunks of equal size based on sorted Admission ID within the interval, resulting in an average of 2742 observations per-chunk for a total of 12 chunks, with the same data setup for publishers and subscribers as Scenario 3. For the dataset Hyperplane, we consider ten sites, with each site producing 10 data chunks of equal size (n=500) from their data stream using the `next_sample` function, and each site takes turn being the subscriber while the other nine sites act as publishers.

We increase the number of most recent models in the ensemble from 1 (similar to the static PubSub-ML setting considered previously) to 8, keeping the other parameters same as in Section 8.1. As a new data chunk becomes available, the ensemble is updated by adding the model trained on the new data chunk and removing the oldest model in the ensemble. Following the standard data stream evaluation practices [13], we collect the performance metrics on the test partition of the subscriber's most recent data chunk (D^s_τ), and report average results over all time points. As DP-FL based models are not designed to work with time-evolving data streams, we only present the results on PubSub-ML.

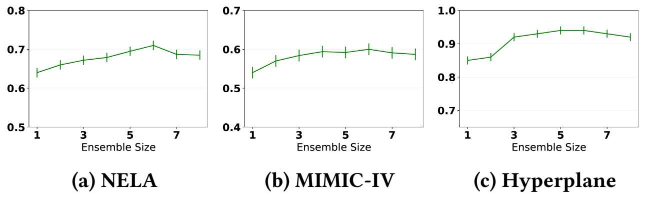

Figure 6: PubSub-ML for data streams. Performance of PubSub-ML increases as we increase the number of models released by each publisher as a part of their ensemble.

Results. Figure 6 shows that the initial increase in the ensemble size improves the utility of PubSub-ML for data streams, which subsequently plateaus after the ensemble size of 5. On one hand, having more models in an ensemble leads to increased model diversity for the model fusion of the subscriber, on the other hand, because of concept drift in the data streams, increasing the ensemble size has the risk of using obsolete models.

Summary. *PubSub-ML's extension for data streams provides good utility, further boosted by publishers releasing larger ensembles.*

8.3.4 *Non-private Scenario.*

Setup. As a large body of work in FL focuses on the non-private setting, in this section we compare non-private PubSub-ML (PubSub-ML*) with the non-private FL baselines.

Results. Due to space constraints, detailed results are presented in the Appendix (Section E). In summary, we observe that PubSub-ML* outperforms FL based method in the non-private setting.

9 CONCLUSION

We presented PubSub-ML as a general solution to decentralized machine learning under loose collaboration. In PubSub-ML, sites benefit from other sites' data through a model publishing/subscribing mechanism that allows a subscriber to construct a customized model by transferring the learning outcomes from publishers while respecting publishers' data privacy. We address several important requirements of this mechanism, including Autonomy, Asymmetric sharing, Personalized privacy, and Model-agnostic, and we extend PubSub-ML to handle data streams with concept drift. PubSub-ML is applicable in all scenarios where the problem can be formulated as an optimization problem with an objective defined by a loss function. To our knowledge, this is the first work to achieve these goals for decentralized machine learning.

ACKNOWLEDGMENTS

This research is in part supported by an Natural Sciences and Engineering Research Council of Canada (NSERC) CGS-D doctoral scholarship for Lovedeep Gondara and an NSERC discovery grant for Ke Wang.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, et al. 2016. Deep learning with differential privacy. In *ACM CCS*.

[2] Stef Buuren and Karin Groothuis-Oudshoorn. 2011. mice: Multivariate imputation by chained equations in R. *Journal of statistical software* 45, 3 (2011).

[3] David Byrd and Antigoni Polychroniadou. 2020. Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications. *Proceedings of the First ACM International Conference on AI in Finance* (2020).

[4] Jia Deng, Wei Dong, Richard Socher, et al. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE CVPR*.

[5] Enmao Diao, Jie Ding, and Vahid Tarokh. 2021. HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients. In *ICLR*.

[6] Fang Dong, Xinghua Ge, Qinya Li, Jinghui Zhang, Dian Shen, Siqi Liu, Xiao Liu, Gang Li, Fan Wu, and Junzhou Luo. 2022. PADP-FedMeta: A personalized and adaptive differentially private federated meta learning mechanism for AIoT. *Journal of Systems Architecture* (2022), 102754.

[7] Yitao Duan, John Canny, and Justin Zhan. 2010. P4P: Practical Large-Scale Privacy-Preserving Distributed Computation Robust against Malicious Users. In *19th USENIX Security Symposium (USENIX Security 10)*.

[8] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, et al. 2006. Our data, ourselves: privacy via distributed noise generation. *EUROCRYPT* (2006).

[9] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* (2014).

[10] Cynthia Dwork and Guy N Rothblum. 2016. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887* (2016).

[11] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2020. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948* (2020).

[12] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. 2020. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 19586–19597.

[13] Lovedeep Gondara, Ke Wang, and Ricardo Silva Carvalho. 2021. Differentially Private Ensemble Classifiers for Data Streams. *arXiv preprint arXiv:2112.04640* (2021).

[14] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *ArXiv* abs/1503.02531 (2015).

[15] Minh Hoang, Nghia Hoang, Bryan Kian Hsiang Low, et al. 2019. Collective Model Fusion for Multiple Black-Box Experts. In *ICML*.

[16] Benjamin D Horne, Jeppe Nørregaard, and Sibel Adali. 2019. Robust Fake News Detection Over Time and Attack. *ACM Transactions on Intelligent Systems and Technology (TIST)* (2019).

[17] Alistair EW Johnson, Tom J Pollard, Lu Shen, et al. 2016. MIMIC-III, a freely accessible critical care database. *Scientific data* (2016).

[18] Peter Kairouz, H Brendan McMahan, Brendan Avent, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[19] Georgios A Kaissis, Marcus R Makowski, Daniel Rückert, and Rickmer F Braren. 2020. Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence* 2, 6 (2020), 305–311.

[20] Shivam Kalra, Junfeng Wen, Jesse C Cresswell, et al. 2021. ProxyFL: Decentralized Federated Learning through Proxy Model Sharing. *arXiv preprint arXiv:2111.11343* (2021).

[21] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *ICML*.

[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[23] Thanh Chi Lam, Nghia Hoang, Bryan Kian Hsiang Low, et al. 2021. Model Fusion for Personalized Learning. In *ICML*.

[24] Daliang Li and Junpu Wang. 2019. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581* (2019).

[25] Tian Li, Anit Kumar Sahu, Manzil Zaheer, et al. 2018. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* (2018).

[26] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems* 2 (2020).

[27] Jiaxin Ma, Ryo Yonetani, and Zahid Iqbal. 2020. Adaptive distillation for decentralized learning from heterogeneous clients. In *ICPR*.

[28] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. 2021. Federated multi-task learning under a mixture of distributions. *Advances in Neural Information Processing Systems* 34 (2021), 15434–15447.

[29] H. B. McMahan, Eider Moore, Daniel Ramage, et al. 2017. Communication-efficient learning of deep networks from decentralized data.. In *AISTATS*.

[30] H Brendan McMahan, Daniel Ramage, Kunal Talwar, et al. 2018. Learning differentially private recurrent language models. In *ICLR*.

[31] Frank McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*. ACM.

[32] Cheolhee Park, Dowon Hong, and Changho Seo. 2019. An attack-based evaluation method for differentially private learning against model inversion attack. *IEEE Access* 7 (2019), 124988–124999.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* (2011).

[34] Xingchao Peng, Zijun Huang, Yizhe Zhu, and Kate Saenko. 2019. Federated adversarial domain adaptation. *arXiv preprint arXiv:1911.02054* (2019).

[35] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, et al. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *EMNLP*.

[36] Santu Rana, Sunil Kumar Gupta, and Svetha Venkatesh. 2015. Differentially private random forest with high utility. In *IEEE ICDM*.

[37] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. 2020. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295* (2020).

[38] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. 2020. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems* 32, 8 (2020), 3710–3722.

[39] Sberbank. 2017. *Sberbank Russian Housing Market Dataset.* https://www.kaggle.com/c/sberbank-russian-housing-market/data

[40] Mervyn Singer, Clifford S Deutschman, Christopher Warren Seymour, et al. 2016. The third international consensus definitions for sepsis and septic shock (Sepsis-3). *JAMA* (2016).

[41] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated multi-task learning. *Advances in neural information processing systems* 30 (2017).

[42] Pierre Stock, Igor Shilov, Ilya Mironov, and Alexandre Sablayrolles. 2022. Defending against Reconstruction Attacks with R\'enyi Differential Privacy. *arXiv preprint arXiv:2202.07623* (2022).

[43] Stacey Truex, Nathalie Baracaldo, Ali Anwar, et al. 2019. A hybrid approach to privacy-preserving federated learning. In *ACM Workshop on Artificial Intelligence and Security*.

[44] Jaideep Vaidya, Basit Shafiq, Anirban Basu, et al. 2013. Differentially private naive bayes classification. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*.

[45] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. 2017. Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*. PMLR, 509–517.

[46] Haixun Wang, Wei Fan, Philip S Yu, et al. 2003. Mining concept-drifting data streams using ensemble classifiers. In *ACM SIGKDD*.

[47] Febrianti Wibawa, Ferhat Ozgur Catak, Murat Kuzlu, Salih Sarp, and Umit Cali. 2022. Homomorphic Encryption and Federated Learning based Privacy-Preserving CNN Training: COVID-19 Detection Use-Case. In *Proceedings of the 2022 European Interdisciplinary Cybersecurity Conference*. 85–90.

[48] Xun Xian, Xinran Wang, Jie Ding, et al. 2020. Assisted Learning: A Framework for Multi-Organization Learning. In *NeurIPS*.

[49] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[50] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. 2020. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758* (2020).

[51] Mikhail Yurochkin, Mayank Agarwal, Soumya Shubhra Ghosh, et al. 2019. Statistical Model Aggregation via Parameter Matching. In *NeurIPS*.

[52] Valentina Zantedeschi, Aurélien Bellet, and Marc Tommasi. 2020. Fully decentralized joint learning of personalized models and collaboration graphs. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 864–874.

[53] Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. 2021. Data-Free Knowledge Distillation for Heterogeneous Federated Learning. *ICML* (2021).

A IMPACT OF PRIVACY BUDGET

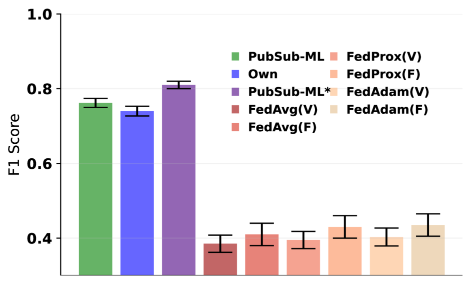

Figure 7: Impact of heterogeneous privacy budget. PubSub-ML outperforms all privacy-preserving baselines by a significant margin.

B IMPACT OF NUMBER OF PUBLISHERS AND SUBSCRIBERS

Setup. Using the same settings as Section 8.2.3 to create publishers and subscribers, we investigate the impact of number of publishers and subscribers in PubSub-ML. Specifically, there are n publishers and n subscribers, and we vary n from 5 to 5000. For each of n publishers, we report the average performance over all n subscribers.

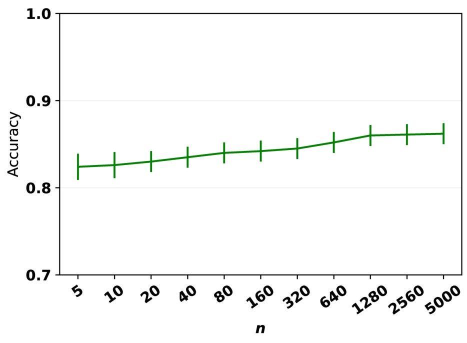

Figure 8: Impact of number of publishers and subscribers (n). Performance initially improves with increasing n.

Results. Figure 8 shows the results. We observe that initial increase in the number of publishers leads to improved performance for subscribers which subsequently plateaus. This is because an initial increase in the number of publishers increases the training data dimensionality (O^s) and the model diversity, leading to a performance boost, which reaches its maximum after a certain number of publishers.

Summary. *Using more publishers can lead to better performance for the subscriber.*

C IMPACT OF NOISY SITES

Setup. To evaluate the impact of noisy sites in PubSub-ML, we use the dataset Hyperplane as follows. We create a total of 20 sites, with 10 sites acting as publishers, out of which 5 are created using the same parameters as specified in Section 8.2.3, while the other 5 set $p_3 = 1$ (Table 3), generating pure noise. The 10 subscribers are generated using the same parameters as specified for subscribers in Section 8.2.3, and we report results as an average over the 10 subscribers.

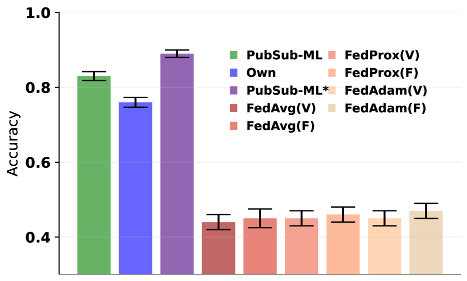

Figure 9: Impact of Noisy Sites. We observe that PubSub-ML provides excellent utility ignoring the noisy sites, whereas FL based methods suffer from degraded utility.

Results. Figure 9 shows the results. Where we observe that PubSub-ML provides excellent utility because our model fusion process can ignore the bad quality models, whereas FL based models suffer from worse degradation (compared to Section 8.2.3) because FL has no mechanism of ignoring bad quality data.

Summary. *PubSub-ML can successfully ignore bad quality data/-models whereas FL based models suffer from degraded utility.*

D REGRESSION.

Setup. For regression, we use the housing market dataset under the IID setting, where each site takes turn being the subscriber while the other 29 sites act as publishers. We report average results using 1-MSE over all such subscribers.

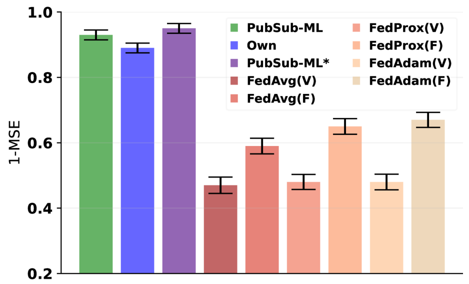

Figure 10: Regression. Similar to the other scenarios, PubSub-ML outperforms all privacy-preserving baselines by a significant margin.

Results. Figure 10 shows the results. We observe that PubSub-ML significantly outperforms all privacy-preserving baselines for regression, where the difference between the non-private PubSub-ML* and PubSub-ML is *small* (MSE of 0.07 for PubSub-ML vs MSE of 0.04 for PubSub-ML*). This provides evidence for our earlier claim that our model fusion process works well for regression.

Summary. *Model fusion in PubSub-ML works well for regression.*

E NON-PRIVATE SCENARIO

Setup. As a large body of work in FL focuses on the non-private setting, in this section, we compare non-private PubSub-ML (PubSub-ML*) with the non-private FL baselines.

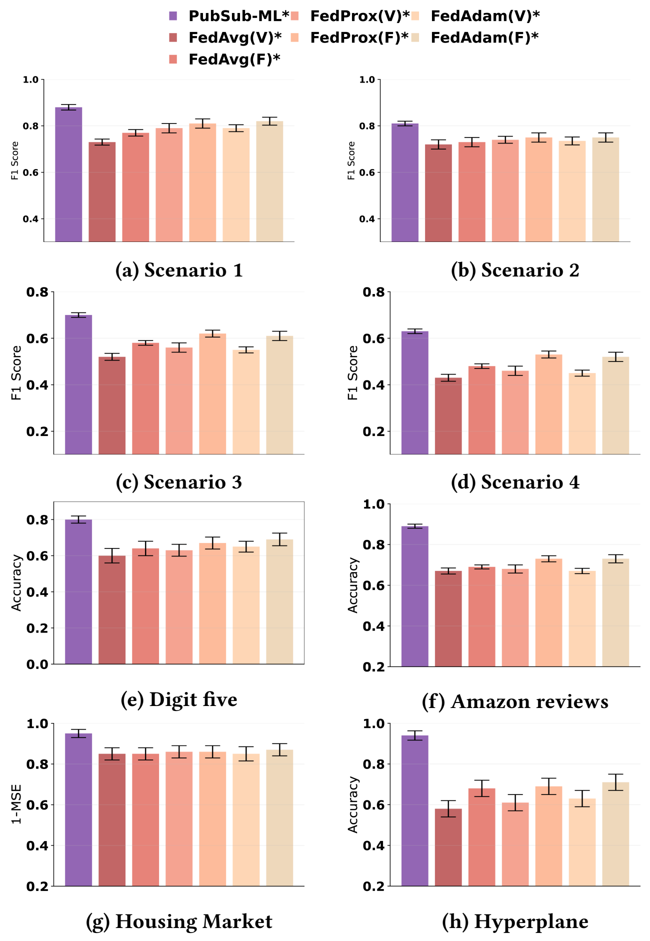

Figure 11: Non-private comparison of PubSub-ML (PubSub-ML* 🟪) with FL (FedAvg(V)* 🟥, FedAvg(F)* 🟥, FedProx(V)* 🟧, FedProx(F)* 🟧, FedAdam(V)* 🟨, and FedAdam(F)* 🟨). PubSub-ML outperforms FL in the non-private setting.

Results. Figure 11 shows the results for the non-private comparison for scenarios 1-4, for the digit-five (CV) and Amazon reviews (NLP) datasets, for regression (Housing Market), and for the dataset hyperplane. We observe that PubSub-ML outperforms FL based methods in the non-private scenario as well where there is no privacy constraint enforced. This provides evidence that PubSub-ML is equally useful in the scenarios where we require privacy-preservation and where privacy is not a concern, for example, a scenario where all sites trust each other.

Summary. *PubSub-ML is a viable alternative to FL based approaches when privacy is not a concern.*