BayBFed: Bayesian Backdoor Defense for Federated Learning

Kavita Kumari[†][*], Phillip Rieger[†], Hossein Fereidooni[†], Murtuza Jadliwala[‡] and Ahmad-Reza Sadeghi[†]
[†]*Technical University of Darmstadt,* [‡]*The University of Texas at San Antonio*

2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10179362

Abstract—**Federated learning (FL) is an emerging technology that allows participants to jointly train a machine learning model without sharing their private data with others. However, FL is vulnerable to poisoning attacks such as backdoor attacks. Consequently, a variety of defenses have recently been proposed, which have primarily utilized intermediary states of the global model (i.e., logits) or distance of the local models (i.e., L_2-norm) with respect to the global model to detect malicious backdoors in FL. However, as these approaches *directly* operate on client updates (or weights), their effectiveness depends on factors such as clients' data distribution or the adversary's attack strategies. In this paper, we introduce a novel and more generic backdoor defense framework, called *BayBFed*, which proposes to utilize probability distributions over client updates to detect malicious updates in FL: *BayBFed* computes a probabilistic measure over the clients' updates to keep track of any adjustments made in the updates, and uses a novel detection algorithm that can leverage this probabilistic measure to efficiently detect and filter out malicious updates. Thus, it overcomes the shortcomings of previous approaches that arise due to the *direct* usage of client updates; nevertheless, our probabilistic measure will include all aspects of the local client training strategies. *BayBFed* utilizes two Bayesian Non-Parametric (BNP) extensions: (i) a Hierarchical Beta-Bernoulli process to draw a probabilistic measure given the clients' updates, and (ii) an adaptation of the Chinese Restaurant Process (CRP), referred by us as CRP-Jensen, which leverages this probabilistic measure to detect and filter out malicious updates. We extensively evaluate our defense approach on five benchmark datasets: CIFAR10, Reddit, IoT intrusion detection, MNIST, and FMNIST, and show that it can effectively detect and eliminate malicious updates in FL without deteriorating the benign performance of the global model.**

1. Introduction

A machine learning framework is designed to learn from a single fused data collected from multiple data sources. This trainable data is comparable and homogeneous. However, in practice, data is heterogeneous and segregated across multiple decentralized devices. Learning a single machine learning model by using this scattered data is complex and challenging as it may disclose a user's identifiable and protected information. Federated Learning (FL) overcomes these drawbacks by enabling multiple distributed clients to learn a global model in a collaborative fashion [23], [47]. For instance, multiple hospitals can participate in training a global model for cancer classification without revealing individual patients' cancer records [21], [36], [46]. Similarly, multiple smartphones could train together a word suggestion model without sharing the individually typed texts [24], or detect threats based on risk indicators [12]. In FL, each client locally trains a model on its private dataset and sends the parameters of this local model to a (global) server, which aggregates the different local models from the clients into a global model (see App. A for more details). The server then responds by sending the aggregated model to each client in a single training round. By design, the global server is unaware of the training process being done locally on each client; thus, it is also susceptible to poisoning attacks from malicious clients.

Poisoning Attacks and Defenses. Previous works have shown that FL is prone to poisoning attacks as a malicious client (or clients) can inject malicious weights into the global server model during training [2], [3], [4], [30], [37], [45]. As a consequence, the performance of the global model on all or some subsets of predictive tasks becomes degenerated. In the so-called *targeted* poisoning (or backdoor) attacks, the adversary's goal is to cause well-defined misbehavior of the global model on some trigger data points, i.e., predict a specific class if a particular pattern is present in the input data [3], [31], [42], [45].[1] Our focus in this paper is to mitigate such targeted backdoor attacks.

To detect/mitigate backdoor attacks, existing defenses leverage either the models' outputs (i.e., predictions on some validation data[2]), intermediary states (i.e., logits) of the models, and/or distance of the local models (i.e., L_2-norm or cosine) with regard to the global model, or pairwise distances among the models. However, current defenses have several shortcomings and are not sufficiently robust to defend against different classes of backdoor attacks. For instance, some defenses are bypassed when multiple different backdoors are simultaneously inserted by different malicious clients [37]. Other defenses clip weights and add noise to negate the effect of malicious model updates, which reduces

[*]. Work done while author was affiliated with The University of Texas at San Antonio.

[1]. In contrast, non-targeted poisoning attacks aim to deteriorate the performance of the global model on all test data points [7].

[2]. As pointed out by Rieger *et al.*, it is not realistic to assume validation data to be present on the aggregation server [35].

DOI 10.1109/SP46215.2023.00100

the benign performance of the global model [3], [26], [30], [40], or they make specific assumptions such as (i) the adversary inserts malicious updates (backdoors) in each training round [14], or (ii) the adversary attacks only at the end of the training [1], or (iii) the data of the benign clients having the same distribution [27], [30], [48], or (iv) each benign client must have a similar number of unique labels [35].

Moreover, current state-of-the-art defenses against backdoor attacks make several assumptions about the underlying data and the adversary's adopted strategies, as well as they *directly* employ client weights during detection. In this context, we encountered two main open challenges: First, how can we compute an alternate, more generic, representation of client weights (or updates), such as a probabilistic measure, which will encompass all adjustments made to the updates due to any local training strategy (by the clients). Second, can we design an efficient detection/clustering algorithm that can leverage such a probabilistic measure to effectively filter out malicious updates in FL, without deteriorating the benign accuracy of the global model. We intuitively believe, and later empirically show, that designing a detection algorithm with such a generic probabilistic measure as one of its inputs provides several significant advantages over existing defense solutions. First, different local client training strategies will not affect the detection process at the global server. Consequently, the defense mechanism's detection phase will remain agnostic about an adversary's attack strategies. Second, utilizing distributions over client updates in the defense, instead of directly employing client weights, makes the detection process uninfluenced by the underlying local data distributions used for training.

Our Goals and Contributions. To tackle the challenges outlined above, we present the design and implementation of *BayBFed*, an unconventional and more general backdoor defense for FL that is based on a probabilistic machine learning framework. *BayBFed* comprises of two main modules. The first module computes a probabilistic measure of the client weights that is governed by the posterior of the *Hierarchical Beta-Bernoulli* process [41] (see Sect. 4). The second module implements a detection algorithm which employs this probabilistic measure as an input to differentiate malicious and benign updates. The main idea is to utilize a probabilistic measure to determine the distribution of the incoming local client updates. Additionally, in each FL round, we compute the distribution of existing groups that were assigned client updates (clusters) or a new group (client updates can get assigned to a new group). Then, we compute the (Jensen) divergence of these two distributions to detect malicious updates and compute the selected client's fit to an existing or a new cluster. The detection algorithm (described later) is mainly governed by the *Chinese Restaurant Process (CRP)*, except that it uses *Jensen-Divergence* to compute clients' fit to the clusters.

The only work in the literature that has employed similar Bayesian Non-Parametric (BNP) models in the context of FL is by Yurochkin et al. [49], where BNP models, specifically the Beta-Bernoulli Process and the Indian Buffet Process, are used to reduce the communication overhead between the global server and the clients. They accomplished this by finding the common subset of neurons between the local clients selected in a training round and combining them to form a global model. In contrast to [49], we use BNP models, specifically the Hierarchical Beta-Bernoulli process and CRP, for designing a *defense* mechanism against backdoor attacks in FL. We stress that [49] is vulnerable to backdoor attacks, as malicious training updates can easily be integrated into the global model.

To the best of our knowledge, this is the first work that employs BNP modeling concepts to design an accurate and robust defense against backdoor attacks in FL. Our main contributions can be summarized as:

- We propose *BayBFed*, a novel generic defense framework against backdoor attacks in FL that accurately and effectively detects backdoors without significantly impacting the benign performance of the aggregated model. Our proposed defense is relevant in many adversarial settings as, by design, the malicious update detection functionality utilizes distributions of client updates and, thus, is unaffected by any local client's strategy.
- We take a new approach to the problem of mitigating backdoor attacks in FL by employing non-parametric Bayesian modeling in the design of the defense mechanism. To the best of our knowledge, existing defenses mainly consider the model updates as a set of vectors and matrices, and *directly* administer these weights to filter out the malicious client updates [4], [14], [26], [27], [30], [37]. Given the client weights, *BayBFed* first estimates a probabilistic measure (such as the Beta posterior) that accurately captures the variations in the clients' weights and then uses a novel detection technique based on the Chinese Restaurant Process and Jensen-Divergence for identifying the poisoned models.
- We extensively evaluate our framework on five benchmark datasets: CIFAR-10, Reddit, MNIST, FMNIST, and a real-world IoT network traffic dataset. We show that *BayBFed* effectively mitigates different state-of-the-art as well as adaptive attacks, and accurately and effectively detects the backdoored models so that the benign performance of the aggregated model is not degraded, thus providing a significant advantage over state-of-the-art defenses.

2. Background and Intuition

Our approach is modeled in two steps. First, to determine the probabilistic distributions of clients' updates, we make use of several statistical tools such as Beta Processes (BP), Hierarchical Beta Processes (HBP), and Bernoulli Processes (BeP). Second, to design our detection algorithm, we outline an adaptation of the Chinese Restaurant Process (CRP), called CRP-Jensen, to detect and filter out malicious updates. Below, we briefly discuss the above two steps (see more technical details in the Appendix):

Determining probabilistic measure for client updates. We compute the probabilistic measure for each client selected in an FL round to keep track of the adjustments made during each update. For this, we first draw a baseline probabilistic measure, denoted by the baseline Beta Process (BP), which is computed using the initial global model. Informally, a BP quantifies a subset of points (measure). We use BPs in this work to quantify the client updates and the global model by creating distributions over them.

A BP (A) is a stochastic process defined using two parameters: a concentration function c over some space $\Omega = \mathbb{R}$ and a base measure H; denoted as $A \sim BP(c, H)$. In FL, the base measure H can represent any distribution of the initial global model (see Sect. 4), i.e., before the training starts, and a concentration function c quantitatively characterizes the similarity between the input base measure (H) and the output random measure A (because of the distribution over the random selection of elements in Ω). In this work, c determines the similarity between the input and the output distribution over Ω, and Ω is a space of initial global model weights. The intuition here is to use this baseline BP, called baseline BP *prior*, to form hierarchies of BP, called hierarchical BP prior, for n different clients selected in the first FL round, i.e., create n sub-BP from the baseline BP.

Informally, a *prior* is the previous knowledge of an event before any new empirical data is observed and is typically expressed as a probability distribution or random measure, while a *posterior* is the revised or updated knowledge of the event after considering the new data. Now, an HBP for each client i is denoted as $A_i \sim BP(c_i, A)$. In the subsequent iterations of the FL, these priors (as computed above) will be updated, based on the new client updates, to compute the so-called BP posteriors, i.e., update the c_i and H_i (A_i). In this work, we have assumed the new client updates in each round as the new data to update the previous knowledge of the BP priors, i.e., to compute the BP posteriors.

In this work, we flatten updates for each client i to a one-dimensional vector having l values, denoted as W_i. We assume that each value in this vector is drawn from a Bernoulli Process (BeP), given the client i's BP random measure A_i. Informally, a BeP is a stochastic process with two possible outcomes: success or failure − we use it in this work to show whether a client i's update will have a particular value (or not), given its BP random measure A_i. In FL, each client updates its local model using the common aggregated global model sent by the global server. Hence, we postulate that each client update vector values are drawn from its corresponding BP random measure A_i, using BeP. Thus, a weight vector W_i for client $i \in \{1,...,n\}$ is characterized by a Bernoulli Process, given as $W_i|A_i \sim \text{BeP}(A_i)$. In other words, in $W_i = \{W_{i,1}, W_{i,2}, ..., W_{i,l}\}$, l denotes the independent BeP draws over the likelihood function A_i.

Another reason to use BeP is that it has been shown in the literature that the Beta distribution is the conjugate of the Bernoulli distribution [6]. Hence, we do not have to use the computationally intensive Bayes' rule to compute the posteriors. We keep updating the corresponding HBP (A_i) for client i using the conjugacy of the BP and the BeP, as given in [41]. The posterior distribution of A_i after observing W_i is still a BP with modified parameters:

$$A_i|W_i \sim BP\left(c_i + l, \frac{c_i}{c_i + l}H + \frac{1}{c_i \cdot l}\sum_{l=1}^{l} W_{i,l}\right) \tag{1}$$

Designing the backdoor detection algorithm. Next, we briefly describe how we adapt the Chinese Restaurant Process to detect malicious client updates. The CRP [39], [5], [22] is an infinite (unknown number of clusters) mixture model in which customers (client's updates) are assigned tables (clusters) in a restaurant. In the context of FL, the clusters represent groups of incoming client updates. The customer can either sit at the already occupied tables (existing clusters) or at the new table (a new cluster is created). Our main idea, as discussed earlier, is to utilize a probabilistic measure to determine the distribution of the incoming local client i's update. In addition, we also compute the distribution of the existing clusters of updates plus the new cluster. Then, we compute the Jensen-Divergence between client i's update distribution and each existing plus new cluster's distribution. Informally, Jensen-Divergence (or Jensen-Shannon Divergence) is a measure of how similar two distributions are. In consequence, we obtain a set of Jensen-Divergence values. We take the maximum of this set to determine whether local client i is malicious or not (intuition, as to why use maximum Jensen-Divergence, is shown in Sect. 4). Based on this maximum Jensen-Divergence value, we also determine the client i's update cluster assignment. After the cluster is determined, we append the client i's update to the selected cluster's list of client updates. Finally, we update the cluster's parameters, i.e., mean and standard deviation, using *Chinese Restaurant Process (CRP)*. This adaptation of the CRP is also referred to by us as CRP-Jensen.

3. Adversary Model

Attack Objectives. The target system trains a Neural Network (NN) f taking samples from a domain $\mathcal{D}$ as input and returning predictions from the set $\mathcal{L}$. The system realizes a function $f : \mathcal{D} \to \mathcal{L}$. The goal of the adversary $\mathcal{A}$ is to inject a backdoor into the aggregated model making it predict a certain adversary-chosen label $l_{\mathcal{A}} \in \mathcal{L}$ for all samples that contain the backdoor trigger, called the *trigger set* $\mathcal{D}_{\mathcal{A}} \subset \mathcal{D}$. The success of this objective is measured by calculating the accuracy for $\mathcal{D}_{\mathcal{A}}$. The attack needs to be stealthy to prevent the backdoor from being detected. Therefore, $\mathcal{A}$ needs to ensure that the attack does not affect the model's performance on the benign main task, i.e., changing the predictions of samples $d \in \mathcal{D} \setminus \mathcal{D}_{\mathcal{A}}$. For conducting such stealthy backdoor attacks, we assume that $\mathcal{A}$ crafts poisoned model updates. $\mathcal{A}$ also needs to ensure that the poisoned model updates are indistinguishable from the benign model updates in terms of all the metrics that the aggregation server may use to detect poisoned models. As $\mathcal{A}$ knows the defense mechanism deployed on the server side (see below), it suffices to make the poisoned model updates indistinguishable from the benign model updates in

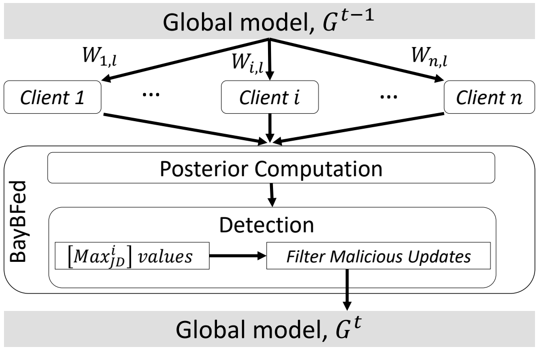


Figure 1: High-level overview of *BayBFed*.

terms of the metrics that are used by the defense mechanism.

Attacker's Capabilities. We assume $\mathcal{A}$ to have the following capabilities to achieve its objectives:

1. Controlling malicious clients: Aligned with existing work [1], [30], [37], we assume $\mathcal{A}$ to fully control $n_{\mathcal{A}} < \frac{n}{2}$ clients where n is the total number of participants. In particular, $\mathcal{A}$ can arbitrarily manipulate the data and training process of the malicious clients. Therefore, besides poisoning the training data, $\mathcal{A}$ can freely adapt the hyperparameters of the training process, and the loss function and can also scale the model updates before sending them to the aggregation server. $\mathcal{A}$ does not control the benign clients. Moreover, it neither knows their training data nor their model updates, although it can make a rough estimation of the benign model updates by training a model using the benign training data (i.e., without backdoors) of the malicious clients.

2. No control over the aggregation server. $\mathcal{A}$ has complete knowledge of the global server's aggregation operations, including the deployed backdoor defenses. However, $\mathcal{A}$ neither controls the server nor knows the parameters that are calculated by the server at runtime and can only interact with the server through the compromised clients. However, an adaptive $\mathcal{A}$ can manipulate the model updates based on the knowledge of the deployed backdoor defense at the global server.

4. Design

In this section, we first discuss the requirements posed on *BayBFed* due to the BNP nature of our defense. Then, we outline the architecture of our *BayBFed* defense mechanism and describe each component in detail.

4.1. Requirements

In BNP models, exchangeability (defined below) is a critical requirement that must be satisfied by a certain sequence of random variables to model different parameters such as priors and posteriors (see Sect. 2). Since, the detection module (CRP-Jensen) takes client updates (W_i) as one of its inputs and its l values are modeled by employing the Hierarchical Beta-Bernoulli Process (HBBP), both the client updates, W_i and it's l values should satisfy the exchangeability property. Informally, the exchangeability property (of a sequence of random variables) states that the joint distribution of all the random variables remains the same for any permutation of random variables. Specifically, we identify the following two key requirements that we will use in the design of *BayBFed*.

Requirement I. *For the posterior computation, a flattened client update vector is a sequence of random variables and should be drawn from an exchangeable set of choices.*

We consider that the l values in a client i's update vector W_i are drawn from an exchangeable set of choices. The reason is, in Eq. 1, we only utilize the summation of client i's l update values to update the base measure H_i. Hence, the order of the l values in client i's update will not affect the computation of H_i. Mathematically, a sequence of random variables $X_1, X_2, ..., X_l$ is called an exchangeable sequence, if the distribution of $X_1, X_2, ..., X_l$ is equal to the distribution of $X_{\pi_1}, X_{\pi_2}, ..., X_{\pi_l}$ for any permutation $(\pi_1, \pi_2, ..., \pi_l)$. We consider $W_i = \{W_{i,1}, W_{i,2}, ..., W_{i,l}\}$ to be an exchangeable sequence for the computation of Beta posterior in *BayBFed*.

Requirement II. *For the detection algorithm employing CRP-Jensen, each client update is a sequence of random variables and should be drawn from an exchangeable set of choices.*

CRP is an infinite mixture model which is used to assign data or samples to the mixtures (or clusters). The data or samples are assumed to be drawn from an exchangeable set of choices. Hence, irrespective of the order in which the data arrives, their assignment to the mixtures or clusters (i.e., their seating arrangement in CRP) is not affected. In this work, we assign client i's update W_i to a cluster by employing CRP and Jensen-Divergence (JD). Thus, we consider W_i to follow the exchangeability property. The reason is that client i's local training does not depend on another client's local training. Thus, permuting the client updates W_i or changing the order of the incoming client updates will not affect the output of the detection module. Thus, in this work, we consider the incoming client updates W_i, where $1 \leq i \leq n$ and n is the number of clients, as an exchangeable sequence.

4.2. *BayBFed* Components

In this section, we describe in detail the two main technical modules of *BayBFed*, i.e., the posterior computation module and the detection module.

4.2.1. Posterior Computation. As briefly explained in Sect. 2, we compute Beta posteriors (using a concentration parameter and a base measure) to have a more generic representation of the client's weights, which can keep track of all the changes made in the client updates. The intuition here is to use the random measure parameters of the previous round $t-1$ (Beta prior), i.e., concentration parameter (c^{t-1}) and the base measure (H^{t-1}), and combine them with client updates (W_i^t) in round t, to compute the Beta posterior c^t and H^t. This is done for each client i selected in round t. Then, the updated base measure H^t

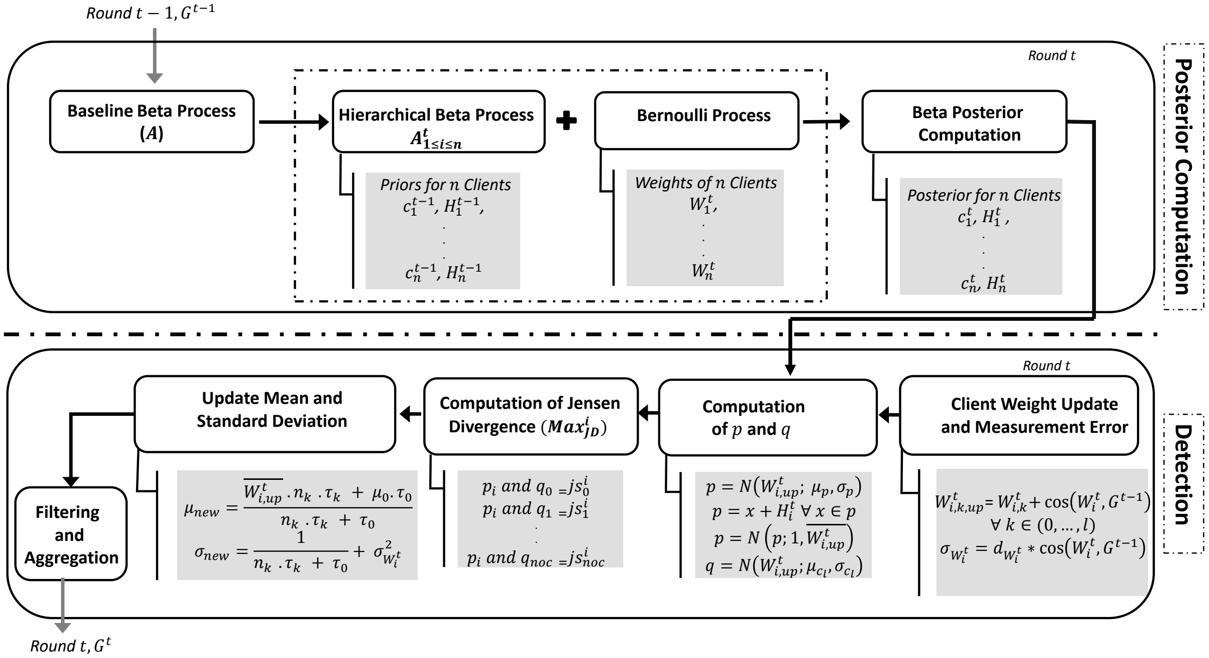

Round $t-1, G^{t-1}$
Round t
Posterior Computation
Baseline Beta Process (A)
Hierarchical Beta Process $A^t_{1 \le i \le n}$
+
Bernoulli Process
Beta Posterior Computation
Priors for n Clients
c_1^{t-1}, H_1^{t-1},
.
.
.
c_n^{t-1}, H_n^{t-1}
Weights of n Clients
W_1^t,
.
.
.
W_n^t
Posterior for n Clients
c_1^t, H_1^t,
.
.
.
c_n^t, H_n^t
Round t
Detection
Update Mean and Standard Deviation
Computation of Jensen Divergence (Max^i_{JD})
Computation of p and q
Client Weight Update and Measurement Error
Filtering and Aggregation
$\mu_{new} = \frac{\overline{W^t_{i,up}} . n_k . \tau_k + \mu_0 . \tau_0}{n_k . \tau_k + \tau_0}$
$\sigma_{new} = \frac{1}{n_k . \tau_k + \tau_0} + \sigma^2_{W_i^t}$
p_i and $q_0 = js_0^i$
p_i and $q_1 = js_1^i$
.
.
p_i and $q_{noc} = js^i_{noc}$
$p = N(W^t_{i,up}; \mu_p, \sigma_p)$
$p = x + H_i^t \; \forall x \in p$
$p = N\left(p; 1, \overline{W^t_{i,up}}\right)$
$q = N(W^t_{i,up}; \mu_{c_l}, \sigma_{c_l})$
$W^t_{i,k,up} = W^t_{i,k} + \cos(W_i^t, G^{t-1})$
$\forall k \in (0, \ldots, l)$
$\sigma_{W_i^t} = d_{W_i^t} * \cos(W_i^t, G^{t-1})$
Round t, G^t

Figure 2: Illustration of *BayBFed*'s design, showing its two modules: Posterior computation and CRP-Jensen.

is utilized in the detection module to filter the poisoned updates. This process is repeated for the subsequent iterations of FL, until the model converges. A high-level overview of BayBFed's architecture is depicted in Fig. 1. Below, we outline a more detailed understanding of the components of the posterior computation as shown in Fig. 2.

Baseline Beta Process (BP). The first step is to create an initial or baseline BP (A) before any FL training starts. The goal is to use this baseline BP to create the sub-Beta priors using the HBP, for the clients selected in the first training round. In our experiments (as discussed later in Sect. 6), we initially choose a random baseline of $c = 5$ and continuously update it based on the posteriors of client updates. Further, we choose a base measure $H = \mathcal{N}(\mu_p, \sigma_p)$ with μ_p equal to the mean of the flattened initial global model and σ_p equal to the standard deviation of the flattened initial global deviation, i.e., populating A with the initial global model weights. We assume that the data points (client updates) are normally distributed for the above mean and standard deviation computation.

Hierarchical Beta process (HBP). The next step is to create hierarchies of the baseline BP for the clients selected in the first round. For a client i selected in round t, HBP is used to define its BP as $A_i^t \sim BP(c_i^t, A)$. In our experiments, before the training starts, we assign the same base measure of H to each client selected in the first training round, and concentration parameters (for each client) are computed as random variables of a Poisson process with parameter c. The Poisson process [18] creates randomized point patterns, and that is why we employ it to compute random concentration parameters (c_i^t) for each client i. After the first round, each client's concentration and base measure gets updated according to Eq. (1).

Bernoulli Process (BeP). In this work, BeP is defined as the draw of an exchangeable sequence of weights, $W_i^t = \{W_{i,1}, W_{i,2}, ..., W_{i,l}\}$, given the concentration parameter c_i^t and the base measure H_i^t, i.e., Beta prior. This means the l-dimensional vector update of a client is considered to be the l independent BeP. Given the client update W_i^t at time t, we use Eq. (1) to obtain the Beta posterior of round t. The computed Beta posterior (c_i^t and H_i^t) over client i's update is integrated into the following detection module to determine whether incoming W_i^t is malicious or not.

4.2.2. Detection Module.

In this module, we design a variation of the CRP, called CRP-Jensen, to filter the poisoned updates sent by malicious clients (see Sect. 2). CRP-Jensen ensures that all malicious updates are detected (and removed) without limiting the benign performance of the target global model. The intuition here is to integrate the updated base measure (H^t) to compute the p distribution of updated client weight, $W^t_{i,up}$, as shown in Eq. (2). Further, we compute a q distribution across the updated client weight, $W^t_{i,up}$, for the existing clusters of the client updates or a new cluster (client update, W_i^t, can get assigned to a new cluster). Then, we compute a set of JD between the client i's p and each cluster's q, obtaining a set (length: number of existing clusters + 1) of JD values for each client i.

Next, we compute the maximum value (Max^i_{JD}) of this set and accordingly decide the cluster assignment for the corresponding clients. In the experiments (see Sect. 6), we show that this value (Max^i_{JD}) varies significantly for malicious and benign updates. Thus, based on these acquired maximum JD values, we filter out the malicious updates and perform the aggregation operation on the remaining benign client updates to obtain the global model G^t. Below, we outline a more detailed understanding of the components of the detection module as shown in Fig. 2.

Client weight update and measurement error. First, we update each client's local model using the cosine angular distance ($cos(W_i^t, G^{t-1})$) between the local model and the global model. The intuition for doing this is to integrate the effect of $cos(W_i^t, G^{t-1})$ into the weights. The reason is that even though an adversary can manipulate the cosine angular distance, the poisoned weights have to differ (slightly) from the benign weights. Otherwise, the poisoned models will predict the correct label rather than the backdoor target label that $\mathcal{A}$ chooses. Therefore, to run an effective attack, $\mathcal{A}$ needs to simulate the weights in the backdoor direction. For the client's model W_i^t with l entries, where $W_{i,k,up}^t$ denotes the element at index k, the updated client weights ($W_{i,up}^t$) are computed as:

$$W_{i,k,up}^t = W_{i,k}^t + cos(W_i^t, G^{t-1}) \quad \forall k \in \{0,\ldots,l\} \tag{2}$$

In CRP, when a new sample is assigned to a cluster, the total error or variance is computed as a combination of two errors: the measurement error because of the new sample, and the errors due to already assigned samples. Thus, we compute the measurement error due to the new client's weight getting assigned to the specific cluster (in each round), given as:

$$\sigma_{w_i^t} = d_{w_i^t} \cdot cos(W_i^t, G^{t-1}) \tag{3}$$

where $d_{w_i^t}$ is the $L_2 -$norm between the global model in the previous round G^{t-1} and client i update in the current round W_i^t. Using the $L_2 -$norm as the measurement error has the same reasoning above for using the cosine angular distance. Even though an adversary can individually manipulate the $L_2 -$norm and cosine distance, there is still a correlation between the two that differs for the malicious and the benign weights. We found the above connections, shown in Eq. (2) and Eq. (3), using our extensive experimental evaluations. We integrate $W_{i,up}^t$ and $\sigma_{w_i^t}$ into the detection module to effectively eliminate all the malicious updates.

Computation of p and q. We then compute the two probability distributions p and q, and use JD to compute their similarity. Here, we integrate the current round base measure H^t to compute p distribution of client updates, such that the detection phase of the defense is not affected by any local client training strategy. Thus, in round t, we compute each client's p and each cluster's q as given in Eq. (4) and Eq. (5), respectively.

$$p = \mathcal{N}(W_{i,up}^t; \mu_p, \sigma_p)$$

$$p = x + H^t \quad \forall x \in p$$

$$p = \mathcal{N}(p; 1, \overline{W_{i,up}^t}) \tag{4}$$

$$q = \mathcal{N}(W_{i,up}^t; \mu_{c_l}, \sigma_{c_l}) \tag{5}$$

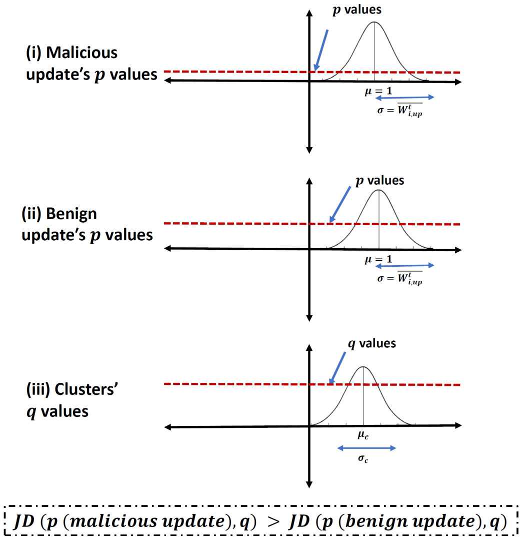

p values
(i) Malicious update's p values
μ = 1
σ = $\overline{W_{i,up}^t}$
p values
(ii) Benign update's p values
μ = 1
σ = $\overline{W_{i,up}^t}$
q values
(iii) Clusters' q values
μ_c
σ_c
$JD(p(malicious\ update), q) > JD(p(benign\ update), q)$

Figure 3: p and q distribution value range for (i) the malicious updates, (ii) the benign updates, and (iii) the clusters.

where, $\overline{W_{i,up}^t}$ is the mean of $W_{i,up}^t$ and μ_p and σ_p are the mean and the variance of the initial global model, respectively. H^t is the updated round t base measure that is given as $\frac{c_i^{t-1}}{c_i^{t-1}+l} H^{t-1} + \frac{1}{c_i^{t-1} \cdot l} \sum_{i=1}^{l} W_i^t$ (see Eq. (1)). μ_{c_l} and σ_{c_l} are the clusters' mean and variance, respectively. Then for each client i, we compute the JD of it's p with each cluster's q.

Computation of Jensen-Divergence (JD). Next, we compute the JD between each client's p and each cluster's q. By computing the JD of each client i's p values with each cluster's q values, we get a set of: $\{(p_i, q_0) : js_0^i, (p_i, q_1) : js_1^i, \ldots, (p_i, q_{noc}) : js_{noc}^i\}$. Then, we compute Max_{JD}^i: $\max(js_0^i, js_1^i, \ldots, js_{noc}^i)$ to output the assigned cluster of client i weights $W_{i,up}^t$ and to decide whether it's a malicious update or a benign update. Here, noc is the total number of clusters formed yet.

Mean and standard deviation update. In the previous step, we computed the client's assigned cluster. Now, we update the mean and the variance of that particular cluster according to the equations:

$$\mu_{new} = \frac{\overline{W_{i,up}^t} n_k \tau_k + \mu_0 \tau_0}{n_k \tau_k + \tau_0} \tag{6}$$

$$\sigma_{new} = \frac{1}{n_k \tau_k + \tau_0} + \sigma_{w_i^t}^2 \tag{7}$$

where, n_k is the number of client updates already assigned to it, τ_k represents the precision of the cluster, μ_0 and τ_0 represent the initial mean and the precision assumed for the new clusters. $\sigma_{w_i^t}^2$ is the variance or the measurement error introduced by the new addition of the client update and is computed according to Eq. (3).

Filtering and Aggregation. Finally, we examine the patterns of the malicious updates based on the computed Max^i_{JD}, which differs significantly from the benign updates. We encountered two patterns of malicious updates Max^i_{JD} in our experiments, as discussed later in Sect. 6: (i) the Max^i_{JD} computed for malicious updates are much greater than that for the benign updates, and (ii) the computed Max^i_{JD} values for the malicious updates are similar to each other. For pattern (i), we observed that the Max^i_{JD} value for benign updates is less than the average of all the clients' Max^i_{JD} values (experimentally evaluated). Therefore, conditioned on this observation, we filtered the malicious updates during the detection phase of *BayBFed*. For pattern (ii), we check if Max^i_{JD} of the incoming update is already present in the set of computed Max^i_{JD} for the n clients; if yes, we do not include the concerned malicious client's update in the final aggregation of updates to output the global model G^t.

Intuition for the filtering step. To understand the relationship between the computed maximum JD values Max^i_{JD} (as outlined above) and the benign/malicious nature of the updates, we conduct experiments utilizing a diverse set of datasets (see Sect. 5). In these experiments, we illustrated the Max^i_{JD} values for malicious and benign updates and observed that Max^i_{JD} values of malicious and benign updates differ significantly. Fig. 3 gives an intuition of why Max^i_{JD} differs for the malicious and benign updates by examining the range of p distribution values for the client updates against the clusters' q distribution values. In Fig. 3, the (i) plot demonstrates the malicious update's p values spanning area, the (ii) plot demonstrates the benign update's p values spanning area, and the (iii) plot demonstrates the clusters' q values spanning area, as observed from the experiments conducted. As seen in these plots, the benign update's p values lie at a larger distance than the malicious update's p values (as the values in p are either equal to or very close to zero). Thus, the distance between the p values of malicious updates, and the q values of the clusters is greater than the distance between the p and q values of benign updates. In other words, JD (p (malicious update), q) > JD (p (benign update), q). Hence, maximum JD is used as a metric to identify malicious updates and assign them to the new cluster.

4.3. *BayBFed* WorkFlow and Algorithms

BayBFed's workflow and detection algorithm have been outlined in Algorithms 1 and 2. Here, μ_0 is the assumed initial mean of the clusters and σ_0^2 is the assumed variance corresponding to mean μ_0. $\sigma^2_{w^t_i}$ is the measurement error of the client update W^t_i and is computed as shown in Eq. (3). Thus, total measurement error or the variance is computed as shown in Eq. (7). If a new cluster is formed, it will have a normal distribution with mean μ_0 and the combined variance of σ_0^2 + $\sigma^2_{w^t_i}$. The set $\{\mu^t_{c_l},\ \sigma^t_{c_l}\}$ represents the mean and standard deviations of the c_l clusters at time t. We start Algorithm 1 by looping through the number of rounds of FL training as shown in line 2. In each round, we initialize an empty array, $Max^{stored}_{JD} = []$, to store the Max^i_{JD} values of the clients. n clients are selected for the training, and we loop through each client i (line 4) to determine its cluster. We then compute $W^t_{i,up}$ and $\sigma_{w^t_i}$ according to equations (2) and (3), respectively (lines 7). If $noc = 0$, then it's the first round, and the first client is assigned to the first cluster (line 9), and accordingly, this new cluster's μ_{new} and σ_{new} are updated (line 10). If $noc \neq 0$, then for each existing cluster plus the new one (line 12), we do the following: first, we compute p and q (line 13), second, we compute the JD of each client's p and each cluster's q (line 14), third, we compute the maximum of the obtained JD set (Max^i_{JD}) and decide the assigned cluster (line 15) according to this value, and finally append it to the array Max^{stored}_{JD} (line 16). Either the client will be assigned to one of the already formed clusters (line 17) or it will be assigned to a new cluster (lines 20, 21). After each FL round, Algorithm 2 ($DetectFilter()$) is called which takes input Max^{stored}_{JD} (line 25) and returns the filtered client updates, ***fcp***. $FedAVG(\boldsymbol{fcp})$ (defined in Appendix A) algorithm is then performed to aggregate the filtered client updates and finally update the global model.

Algorithm 1 *BayBFed*'s workflow.

```
 1: Input: $\mu_0$, $\sigma_0^2$, $\sigma^2_{w^t_i}$, $\tau_0 = \frac{1}{\sigma_0^2}$, $\tau_w = \frac{1}{\sigma^2_{w^t_i}}$, $noc$.
 2: for  each round till the model converges  do
 3:     Initialize an array to store $Max^i_{JD}$ in each round, $Max^{stored}_{JD} =$
        $[]$.
 4:     for  $i$ 1 ← to n  do
 5:         Draw any client update $W^t_i$.
 6:         Compute $d_{w^t_i}$ and $cos(W^t_i, G^{t-1})$ .
 7:         Update $\sigma^2_{w^t_i} \leftarrow d_{w^t_i} \cdot cos(W_{t,i}, G^{t-1})$ and compute $W^t_{i,up}$.
 8:         if $noc$ == 0 then
 9:             Assign $c_0 \leftarrow W^t_i$.
10:             Update $\mu_{new}$ and $\sigma_{new}$ with, $n_k = 1$.
11:         end if
12:         for  $c_l$ ← to $noc$+$1$ do
13:             Compute $p$ and $q$.
14:             Compute the JD by $p$ and $q$ and store the values.
15:             Decide the cluster, $c_i$ according to $Max^i_{JD}$.
16:             Append $Max^i_{JD}$ to $Max^{stored}_{JD}$.
17:             if  $c_{l,i} = c_l$ then
18:                 Update $W^t_i$ assigned cluster, $\{\mu^t_{c_l},\ \sigma^t_{c_l}\}$ according to
                    $c_{l,i} = c_l$.
19:             else
20:                 Increment: $noc = noc + 1$, a new cluster is formed.
21:                 Set $c_{l,i} = noc$ and assign $W^t_i$ to it. Append this new
                    cluster to the vector of non-empty clusters.
22:             end if
23:         end for
24:     end for
25:     Call $DetectFilter()$, $\boldsymbol{fcp}$ = $DetectFilter(Max^{stored}_{JD})$.
26:     Perform $FedAVG(\boldsymbol{fcp})$ and update the global model.
27: end for
```

5. Experimental Setup

We employ the machine learning framework PyTorch to conduct our experiments and use the existing defenses [4], [14], [37], [27], [48], [25], [30] as baseline models to comparatively analyze the performance of *BayBFed*. Aligned with previous work on backdoor attacks [1], [3], [30], we

Algorithm 2 Detection and Filter Algorithm, *DetectFilter*.

```
Input: $Max_{JD}^{stored}$ containing $Max_{JD}^{i}$ values, total clients, $n$
Output: Filtered client updates
Initialize an array to store filtered client updates in each round, $W_{filtered} = []$.
Compute $Max_{JD}^{avg}$ = sum($Max_{JD}^{stored}$)/$n$
for i = 1 to $n$ do
    if $Max_{JD}^{i}$ not in $Max_{JD}^{stored}$ then
        $W_{filtered}$.append($W_i^t$)
    end if
    if $Max_{JD}^{i} < Max_{JD}^{avg}$ then
        $W_{filtered}$.append($W_i^t$)
    end if
end for
Return $W_{filtered}$
```

use the attacks provided by Bagdasaryan *et al.* [3] and Wang *et al.* [42] to implement the Constrain-and-Scale and Edge-Case backdoor attacks. Below, we provide the configurations of the different datasets and the accuracy and precision metrics we use to evaluate the performance of *BayBFed*.

Datasets. To show the generality of our results and the representative nature of *BayBFed* across models/data from different domains, we evaluate the proposed defense mechanism by designing two attacks (see Tab. 1) on three popular FL applications: (i) image classification, (ii) word prediction, and (iii) IoT network intrusion detection. To facilitate an equitable comparison of *BayBFed* with state-of-the-art backdoor attack approaches [3], [30], we align the datasets, setups, and NN architectures employed in our comparative evaluation with the ones used by these research efforts.

Image Classification (IC): We use the popular benchmark datasets MNIST, FMNIST, and CIFAR-10 in our experiments. As these datasets are frequently used for evaluating FL and backdoor attacks and defenses [3], [8], [14], [15], [16], [20], [23], [27], [30], [35], [42], [43], [44], [13], [34], it enables us to perform an equitable comparative analysis of our approach with other state-of-the-art approaches in the literature. All three consist of samples belonging to one out of ten classes, handwritten digits in the case of MNIST, articles of clothing in the case of FMNIST, and objects (airplanes, cars, birds, etc.) in the case of CIFAR-10. The CIFAR-10 dataset consists of 50K training and 10K test images, while MNIST and FMNIST datasets each consist of 60K training and 10K test images. As the NN architecture, a light-weight version of Resnet-18 is used for CIFAR-10 [3], a simple CNN is used for MNIST [8], and a three-layer fully connected NN with *relu* activations is used for FMNIST.

Word Prediction (WP): To evaluate *BayBFed* for a complex Natural Language Processing (NLP) application such as word prediction, we use the Reddit dataset consisting of all posts from November 2017. Aligned with the work of Bagdasaryan *et al.*, we considered each author's posts as a local dataset and only the 50K most frequent words. A Long Short-term Memory (LSTM) model is used to predict the next word [3].

TABLE 1: Backdoor Accuracy (*BA*) and Main Task Accuracy (*MA*) of *BayBFed* compared to two state-of-the-art attacks. All values are represented as percentages.

Attacks	**Dataset**	**No Defense**		***BayBFed***	
		BA	MA	BA	MA
Constrain-and-Scale [3]	Reddit	100.0	22.6	0.0	22.6
	CIFAR-10	100.0	90.5	0.0	92.2
	MNIST	43.0	96.5	0.0	96.0
	FMNIST	71.0	85.5	2.0	85.3
	IoT-Traffic	100.0	100.0	0.0	100.0
Edge-Case [42]	CIFAR-10	33.16	88.42	4.02	82.82

Network Intrusion Detection (NIDS): Further, we evaluate *BayBFed* for the FL-based NIDS DÏoT [29] system using four real-world network traffic datasets, kindly shared with us by Nguyen *et al.* [29], [30] and Sivanathan *et al.* [38]. The datasets consist of network traffic of multiple smart home and office settings. Aligned with previous work [30], [35], we converted the network packets into symbols based on their features, such as source and destination ports, protocols, and flags. To simulate a distributed FL setting, we split the dataset into 100 local datasets, each consisting of symbols between 2K and 3K, which were extracted from the network packets. The NN is trained to predict the next probabilities for each possible symbol (network packet). The NN consists of 2 Gated-Recurrent-Unit layers followed by a fully connected linear layer, as defined by Nguyen *et al.* [29].

Evaluation metrics. We compute four metrics to estimate the accuracy and precision of *BayBFed*.

True Positive Rate (TPR): This metric specifies how accurately the defense is able to detect the poisoned model updates. The total number of correctly identified poisoned updates are called True Positives (TP) and the number of poisoned model updates discerned as benign model updates are called False Negatives (FN). Thus, $TPR = \frac{TP}{TP+FN}$.

True Negative Rate (TNR): This metric determines how accurately the defense is able to detect the benign model updates. The total number of correctly identified benign model updates are called True Negatives (TN) and the number of benign updates discerned as poisoned updates are called False Positives (FP). Thus, $TNR = \frac{TN}{TN+FP}$.

Backdoor Accuracy (BA): This metric is used to measure the accuracy of the model on the triggered inputs. Specifically, it measures the fraction of triggered samples where the model predicts the adversary's chosen label.

Main Task Accuracy (MA): This metric is used to measure the accuracy of the model on its benign main task. It represents the fraction of benign inputs for which the model provides correct predictions.

6. Experimental Results

Next, we empirically illustrate the effectiveness of *BayBFed* against two state-of-the-art attacks [3], [42] and compare its efficacy against various state-of-the-art defense mechanisms. Further, we show how Max_{JD}^{i} varies for the malicious and benign model updates. Finally, we demonstrate the robustness of *BayBFed* for various adversarial attack parameters and sophisticated backdoor injection strategies.

6.1. Overall Performance

Attack Strategies. The effectiveness of *BayBFed* against two state-of-the-art model poisoning attacks, the Constrain-

and-Scale [3] and the Edge-Case backdoor [42] is shown in Tab. 1. As we have assumed that an adversary can fully control the malicious clients (and thus the code on the clients), he is not restricted or constrained in terms of the employed attack strategy. In addition to attacks during training, our adversary can also adopt a runtime strategy to make the attack more stealthy.

As can be seen in Tab. 2, *BayBFed* functions optimally against Constrain-and-Scale attacks by filtering out all poisoned updates (BA = 0%). At the same time, the *MA* remains approximately equal to the benign setting *MA*. It should be noted that if the MA is less than 100%, misclassifications of the model can be counted in favor of the backdoor, especially if the model wrongly predicts the backdoor target. As already pointed out by Rieger *et al.* [35], this phenomenon primarily occurs for image scenarios with pixel-based triggers. It causes the BA to be slightly higher than 0% for backdoor-free models. In the case of an Edge-Case attack, the BA before the attack and after *BayBFed* integration is 11.22% and 4.02%, respectively. However, without defense, the BA achieves 33.16%.

Baseline Models. We compare *BayBFed* against seven state-of-the-art defense mechanisms present in the literature: Krum [4], FoolsGold [14], Auror [37], AFA [27], DP [48], Median [25] and FLAME [30]. We implement the Constrain-and-Scale attack against all the defenses and compare the output statistics in terms of the *BA* and *MA*. As illustrated in Tab. 2, *BayBFed* outperforms all these defense mechanisms. These results show that the existing defense mechanisms either lack the precision in removing all the poisoned updates or limit the *MA* of the global model. Further, these defense mechanisms perform accurately when specific assumptions about the data and attack scenarios are satisfied. For instance, in the case of Krum [4], which selects a single model as an aggregated model, a poisoned model is chosen when an attacker circumvents Krum. Therefore, the aggregated model is entirely replaced by a poisoned model, achieving 100% BA. Similarly, another defense FoolsGold [14], is effective for the highly non-IID Reddit dataset but fails when their clients have similar data. It should be noted that *BayBFed* achieved a TPR and TNR of 100% in all three scenarios.

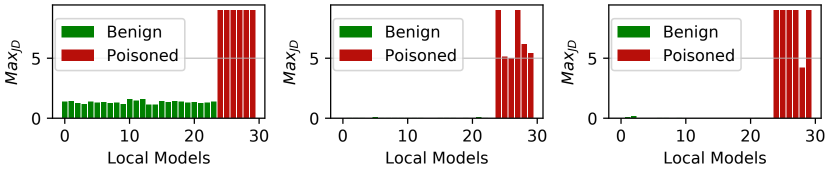
TABLE 2: Backdoor Accuracy (*BA*) and Main Task Accuracy (*MA*) of *BayBFed* compared to state-of-the-art defenses for the Constrain-and-Scale attack. All values are represented as percentages.

Defenses	**Reddit**		**CIFAR-10**		**IoT-Traffic**	
	BA	MA	BA	MA	BA	MA
Benign Setting	-	22.6	-	92.6	-	100.0
No Defense	100.0	22.6	100.0	90.5	100.0	100.0
Krum [4]	100.0	9.6	100.0	56.7	100.0	84.0
FoolsGold [14]	0.0	22.5	100.0	52.3	100.0	99.2
Auror [37]	100.0	22.5	100.0	26.1	100.0	96.6
AFA [27]	100.0	22.4	0.0	91.7	100.0	87.4
DP [3]	14.0	18.9	0.0	78.9	14.8	82.3
Median [48]	0.0	22.0	0.0	50.1	0.0	87.7
FLAME [30]	0.0	22.3	0.0	91.9	0.0	99.8
BayBFed	**0.0**	**22.6**	**0.0**	**92.2**	**0.0**	**100.0**

Impact on the MA. For the IC application CIFAR-10 dataset, we observe that the Constrain-and-Scale attack lowers the MA from 92.6% (FedAVG without attack) to 90.5% (FedAVG). Krum, FoolsGold, Auror, DP, and Median techniques achieve a *MA* of 56.7%, 52.3%, 26.1%, 78.9%, and 50.1%, respectively, which is considerably lower than the benign setting *MA*. In contrast, *BayBFed* has a *MA* of 92.2%, which shows that it works significantly better for IC applications. For the WP application, Krum and DP have a decreased *MA* of 9.6% and 18.9%, compared to the highest *MA* of 22.6%. In this case as well, *BayBFed* performs much better and achieves a *MA* of 22.6%. For the IoT-Traffic dataset, every defense has decreased *MA*. In this case, even small drops in MA need to be avoided due to the nature of this application. The reason is due to the high number of network packets in this scenario; even a small number of false alerts will annoy the user, causing them to ignore the alerts. For example, the defense technique FLAME results in a drop of 0.2%, which causes 2 out of every 1000 packets to be misclassified. As a result, when a high amount of network packets is sent, the user will receive a high number of alerts. It should be noted *BayBFed* recognizes all benign and malicious models correctly ($TPR = 100\%$ and $TNR = 100\%$) in all three scenarios, thus, comparatively performing better than the other defense mechanisms such as FLAME. For example, FLAME excludes benign models; in the NIDS scenario, FLAME wrongly excludes 17 benign models, which might be problematic in the case of highly non-IID data.

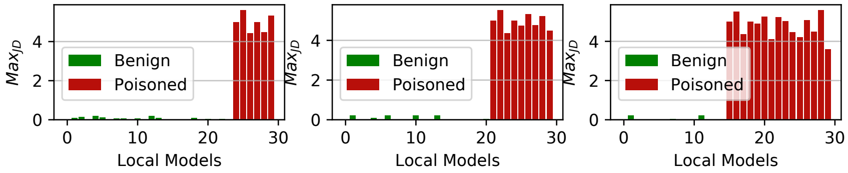
Backdoor updates removal. Krum and Auror fail to remove poisoned updates in all three applications, as these defenses exhibit a *BA* of 100%. FoolsGold eliminates all the poisoned updates in the Reddit dataset ($BA = 0.0\%$). However, it fails to remove them in the CIFAR-10 and IoT-Traffic datasets, as it achieves a *BA* of 100% in those cases. For the AFA defense, it works accurately for CIFAR-10 ($BA = 0.0\%$) but is ineffective for the Reddit ($BA = 0.0\%$) and IoT-Traffic ($BA = 0.0\%$) datasets. In contrast, *BayBFed* significantly outperforms these defenses as it can remove poisoned updates ($BA = 0.0\%$) for all the datasets.

Next, we discuss the impact of two critical experimental parameters on each of the considered applications and datasets in this paper: poisoned model rate (*PMR*) and degree of non-IID data. *PMR* represents the fraction of n_A malicious clients per total clients n. Thus, $PMR = \frac{n_A}{n}$. non-IID represents the percentage of non-IID data at each client. A non-IID value of 0 means that the data is independently and identically distributed, non-IID = 1.0 implies that the data of different clients differ significantly and are distinguishable. For the IC application, we simulate experiments for both non-IID degrees and *PMR* (see Sect. 6.2). However, for the Reddit and the IoT datasets, changing the non-IID degree is not meaningful since this type of data has a natural distribution, as every client obtains data from different Reddit users or traffic chunks from different IoT devices. Thus, we only simulate experiments for different *PMR* for these two datasets. We will also show the impact of these two parameters on Max^i_{JD} for each client (see Sect. 4) and prove that it differs significantly for the benign and poisoned updates.



(a) non-IID = 0.0 (b) non-IID = 0.5 (c) non-IID = 1.0

Figure 4: Effect of different non-IID rates on the maximum Jensen-Divergence (Max^i_{JD}) for CIFAR-10 dataset.



(a) PMR = 20% (b) PMR = 30% (c) PMR = 50%

Figure 5: Effect of different PMR rates on the maximum Jensen-Divergence (Max^i_{JD}) for the CIFAR dataset.

6.2. *BayBFed* Statistics for CIFAR-10

In this section, we evaluate the impact of non-IID rate and *PMR* on the CIFAR-10 dataset. First, we demonstrate the trend of Max^i_{JD} for both the malicious and benign clients with respect to each of these parameters. Then, we illustrate the impact of non-IID rate and *PMR* on *BayBFed*'s performance by quantifying different metrics as stated in Sect. 5 and also compare it against the no defense scenario.

Illustration of Max^i_{JD}. The impact of the degree of non-IID data and *PMR* on Max^i_{JD} for each client is shown in Fig. 4 and Fig. 5, respectively. We select a total of 30 (n= 30) clients for both the non-IID and *PMR* experimental analysis. For non-IID analysis, we test non-IID $\in \{0.0, 0.5, 1.0\}$ and set $PMR = 0.2$. Thus, the number of malicious clients equals 6 ($n_{\mathcal{A}}$ = 6). For *PMR* analysis, we test $PMR \in \{0.2, 0.3, 0.5\}$, i.e., when $n_{\mathcal{A}}$ equals 6, 9, and 15 and set non-IID = 0.7. As illustrated in Fig. 4 and Fig. 5, the Max^i_{JD} value for benign clients differs significantly from that of malicious clients. Hence, *BayBFed* easily filters out all the malicious client updates, achieving a *BA* of zero while keeping the *MA* of the global model intact.

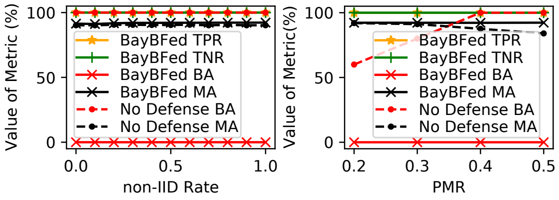
Effect of the degree of non-IID Data. To study the impact of non-IID data on *BayBFed*, we conduct experiments for the Constrain-and-Scale attack on the CIFAR-10 dataset. Following recent work [42], [11], [35], [30], we prepare the non-IID data by varying the number of images assigned to a particular class for each client. Precisely, we form 10 groups corresponding to the ten classes of CIFAR-10. Then, clients in each group are allocated a fixed fraction of images, depending on the non-IID degree of that group's label, while allocating the remaining images to each client randomly. Mainly, for non-IID = 0.0, the samples of all clients followed the same distribution and were chosen randomly from all classes. However, for non-IID = 1.0, the samples of each client were only chosen from the samples belonging to the main class of this client. Fig. 6a compares the impact of the degree of non-IID data in terms of *BA* and *MA* for the plain FedAVG without defense (No Defense *BA*, No Defense *MA*) and the impact on *BayBFed* (*BA*, *MA*). Fig. 6a also shows the computed *TPR* and *TNR* for *BayBFed* in this setting. As one can observe, we obtain $TPR = 100\%$, indicating *BayBFed* achieved *BA* = 0, i.e., all the poisoned models were detected and filtered out before the aggregation. In addition, *BayBFed* achieved $TNR = 100\%$, indicating it correctly identified all the benign updates, thus getting approximately $MA = 92.2\%$ for all the non-IID rates.

Effect of different *PMR* rates. Fig. 6b shows the impact of different *PMR* rates on *BayBFed*. We consider *PMR*s of 0.2, 0.3, 0.4, and 0.5. Hence, $n_{\mathcal{A}}$ equals 6, 9, 12, and 15. We use the same metrics that we used for non-IID rates to evaluate *BayBFed* against different *PMR*s. In this experiment, we achieve results similar to the ones we obtained for different non-IID rates. This demonstrates that *BayBFed* is efficient and accurate in eliminating all the poisoned updates for different data distributions while keeping the benign accuracy of the model intact.

6.3. *BayBFed* Statistics for WP

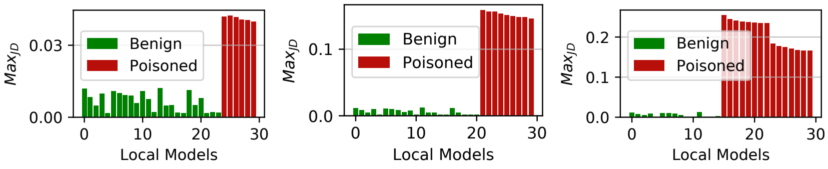
This section evaluates the impact of *PMR* on the Word Prediction application. First, we demonstrate the trend of Max^i_{JD} for both the malicious clients and the benign clients. Then, we illustrate the impact of *PMR* on *BayBFed*'s performance by quantifying different metrics as stated in Sect. 5 and also compare it against the no defense scenario.

Illustration of Max^i_{JD}. In this setting, we also select 30 clients who can participate in each training round, and demonstrate the impact of varying *PMR* values $(0.2, 0.3, 0.5)$ on the clients' Max^i_{JD}. As outlined in Fig. 7, the Max^i_{JD} values of malicious and benign client updates differ signifi-



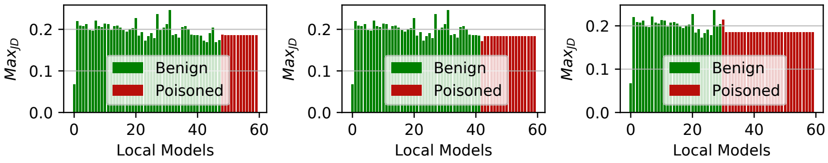
(a) non-IID Rate (b) PMR

Figure 6: Impact of the poisoned model rate $PMR = \frac{n_{\mathcal{A}}}{n}$ and non-IID rate on *BayBFed* for the IC application.



(a) PMR= 20% (b) PMR= 30% (c) PMR= 50%

Figure 7: Effect of different PMR rates on the maximum Jensen-Divergence (Max^i_{JD}) for the Reddit dataset.



(a) PMR = 20% (b) PMR = 30% (c) PMR = 50%

Figure 8: Effect of different PMR rates on the maximum Jensen-Divergence (Max^i_{JD}) for the IoT-Traffic dataset.

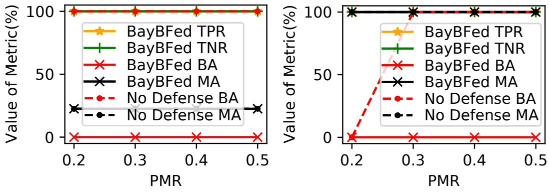
cantly. Thus, *BayBFed* accurately identified all the poisoned updates, achieving a $BA = 0.0\%$ and $MA = 22.6\%$.

Effect of different PMR rates. Next, we evaluate the effectiveness of *BayBFed*, compared against no defense BA and MA, for different PMR values $(0.2, 0.3, 0.4, 0.5)$. The results of this experiment are shown in Fig. 9a. These results indicate that *BayBFed* obtained a $TPR = 100\%$ and a $TNR = 100\%$, for all PMR values. Moreover, it successfully identified all the poisoned and benign updates for different PMR values and achieved a $BA = 0\%$ and the highest possible MA of benign setting, i.e., $MA = 22.6\%$.

6.4. *BayBFed* Statistics for NIDS

This section evaluates the impact of different PMR rates on the NIDS application. Here, we randomly select 60 clients who can participate in each training round. It should be noted that since NIDS models have a lesser number of parameters, training time is reduced. Thus, we evaluated more clients than WP and IC models/applications. However, we set the same PMR in all scenarios. Hence, it did not impact the experimental results (except for the experiments where we considered different PMRs). The number of benign and malicious clients varies based on the selected PMR value, specifically, for PMR values 0.2, 0.3, 0.4, and 0.5, $n_{\mathcal{A}}$ is 12, 18, 24, and 30, respectively. First, we demonstrate the trend of Max^i_{JD} for both the malicious and benign clients and then illustrate the impact of PMR on *BayBFed* compared to the no defense scenario.

Illustration of Max^i_{JD}. Fig. 8 illustrates the impact of different PMR values on Max^i_{JD} for each client. This plot illustrates the sequence of Max^i_{JD} for the poisoned updates, and one can observe that they are equal and different from benign updates. By employing this pattern of the Max^i_{JD}, *BayBFed* was accurately able to filter out all the poisoned updates, thus, attaining a BA of 0%.



(a) WP (b) NIDS

Figure 9: Impact of the poisoned model rate $PMR = \frac{n_{\mathcal{A}}}{n}$ on the evaluation metrics.

Effect of different PMR rates. Next, we compute the TPR, TNR, BA, and MA metrics to evaluate the effectiveness of *BayBFed* compared against the no defense BA and MA, for different PMR values. Results for this set of experiments are shown in Fig. 9b. By using the computed maximum Jensen-Divergence values for each client, *BayBFed* is able to achieve $TPR = 100\%$, $TNR = 100\%$, $BA = 0\%$, and $MA = 100\%$. Hence, *BayBFed* performs optimally for the NIDS application as well.

6.5. *BayBFed* Statistics for FMNIST and MNIST

Further, we evaluate the impact of different non-IID and PMR rates on the FMNIST and MNIST datasets. We used the same setup that we used for the CIFAR-10 dataset (see Sect. 6.2). In all the experiments with FMNIST and MNIST, Max^i_{JD} values of malicious and benign client updates differ significantly, as observed for CIFAR-10. Thus, *BayBFed* accurately identified all the poisoned updates, achieving a BA of 0%. For detailed FMNIST and MNIST results, please refer to App. D and App. E, respectively.

6.6. Effect of Other Factors on *BayBFed*

Next, we conduct additional experiments with *BayBFed* by varying four other parameters: (i) number of clients (hence, the number of malicious clients), (ii) backdoor injection strategies, (iii) poisoned data rates (PDR), and (iv) client order. Additionally, we also assess the trade-off between model accuracy and defense evasion for an adaptive attacker. PDR represents the fraction of injected poisoned data in the overall poisoned training dataset. Our goal in conducting these experiments is to show that *BayBFed* is robust against these factors in detecting backdoor attacks in FL.

Number of clients. In this experiment, we evaluate the impact on the performance of *BayBFed* by varying the number of clients, thus, the *PMR*. The results are outlined in Fig. 10. In each round, we select a random number of clients ranging from 40 to 90. We conduct this experiment for the IC (Fig. 10a) and NIDS (10b) applications. In both cases, *BayBFed* achieved a *BA* of 0%, thereby showing that it is effective in eliminating all the backdoors compared to the no defense scenario.

Different injection strategies. An adversary ($\mathcal{A}$) can inject multiple backdoors at the same time in order to make the backdoor more difficult to detect, thus making the poisoned models harder to distinguish from benign ones in non-IID scenarios. We perform four experiments for the NIDS application, where each client is trained to inject 1 to 4 backdoors. Existing work [35] has shown that the attack efficiency significantly reduces as the number of backdoors increases, and we observed the same pattern during our experiments. Hence, four backdoors were considered a good number (of backdoors) that provided reasonable attack efficiency. Our evaluations show that *BayBFed* was able to defend against and mitigate all the introduced backdoors effectively, thus achieving a 0% BA.

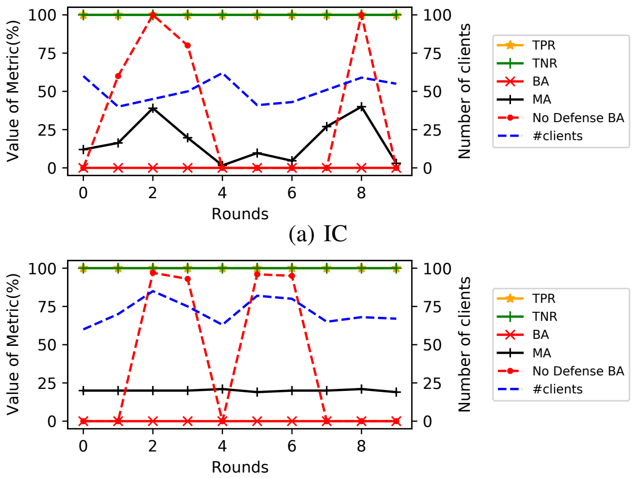
Different Poisoned Data Rates (*PDR*). In this experiment, we consider an adversary that is capable of poisoning the data to launch backdoor attacks. We evaluate this attack on the CIFAR-10 and IoT-Traffic dataset for three different values of *PDR*: 0.05, 0.1, and 0.5, i.e., 5%, 10%, and 50% of the training dataset is poisoned. For the CIFAR-10 dataset, we set $n = 30$ and $PMR = 0.2$, and for the IoT-Traffic dataset, we set $n = 100$ and $PMR = 0.3$. In both these scenarios, *BayBFed* is successful in eliminating all the backdoors, obtaining a *BA* of 0% and achieving an average *MA* of 92.4% for the CIFAR-10 dataset and 100% for the IoT-Traffic dataset.

Client Order. To verify that the client updates are exchangeable, we conducted an experiment for the CIFAR-10 dataset, where the models were randomly shuffled. However, the shuffling did not affect the results, as we got $BA = 0\%$ and $MA = 92.5\%$. These results are intuitive because irrespective of the order in which the client updates arrive at the detection module of *BayBFed*, it does not affect the computation of Max^{i}_{JD}, which is eventually used to identify the poisoned updates.

Adaptive attacks. *BayBFed* assumes that $\mathcal{A}$ knows the backdoor defense deployed at the global server (see Sect. 3). Thus, $\mathcal{A}$ can constrain the training process to make H^t inconspicuous, by using its benign data to estimate a benign model and thus, p and H^t. However, $\mathcal{A}$ cannot estimate q as this requires knowing parameters that the server calculates on run-time. Thus, an adaptive attacker can only work with H^t to launch backdoor attacks against such defense. In this setting, we conducted experiments for the CIFAR-10, by updating the loss function of $\mathcal{A}$ using the base measure for the anomaly evasion loss term [3] according to the equation:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{class}} + (1-\alpha)\mathcal{L}_{\text{BM}} \tag{8}$$

$\mathcal{L}_{\text{class}}$ captures both the BA and the MA, and $\mathcal{L}_{\text{BM}}$ captures the defense mechanism dependency on the base measure. We conducted three experiments with α (determines the trade-off between model accuracy and evasion from defense mechanism) values as 0.0, 0.5, and 1.0. For $\alpha = 0.0$, $\mathcal{A}$ sacrifices the model accuracy to evade the defense mechanism, for $\alpha = 0.5$, $\mathcal{A}$ is equally trading off the model accuracy and defense mechanism evasion, while for $\alpha = 1$, $\mathcal{A}$ is more concerned about the model accuracy than evading the defense mechanism detection. For $\alpha = 1$, *BayBFed* achieved a *BA* of 0%, *MA* of 92.33%, $TPR = 1$ ($TP = 10$ and $FN = 0$), and $TNR = 0.95$ ($TN = 19$ and $FP = 1$). However, as we decreased α to 0.5, *BayBFed* was effective in detecting and filtering an adaptive attacker's model updates. For $\alpha = 0.5$, *BayBFed* obtained a *BA* of 0%, *MA* of 92.25%, $TPR = 1$, and $TNR = 1$. For $\alpha = 0$, *BayBFed* obtained a *BA* of 0%, *MA* of 92.14%, $TPR = 0$ ($TP = 0$ and $FN = 10$), and $TNR = 0.85$ ($TN = 17$ and $FP = 3$). Hence, an adaptive adversary can evade detection at the cost of model accuracy. However, the non-detected models do not have any overall impact on the efficacy of *BayBFed* as the *BA* is always zero. In summary, our experiments show that *BayBFed* is successful in defending against an adaptive adversary who has working knowledge of *BayBFed* deployed at the global server.



(a) IC

(b) NIDS

Figure 10: Impact of the number of clients on *BayBFed* vs No Defense for different datasets.

7. Security Analysis

This section provides a security analysis to corroborate that *BayBFed* can neutralize backdoors by modeling the defense mechanism using BNP modeling concepts. We explain why our defense works and justify its effectiveness. To bypass our defense, an adversarial client ($\mathcal{A}$) has to ensure that *BayBFed* cannot distinguish between malicious and benign model updates. Below, we present three mechanisms through which $\mathcal{A}$ can hide the backdoors from *BayBFed*. First, $\mathcal{A}$ can vary the fraction (PMR) of malicious clients, i.e., $\mathcal{A}$ can either reduce the PMR and make the attack less suspicious, or increase the PMR to keep the attack successful while making the models less suspicious. Second, $\mathcal{A}$ can limit the poison data rate (PDR) for each adversarial client, i.e., instead of poisoning the entire dataset, $\mathcal{A}$ could partially poison the dataset. Finally, $\mathcal{A}$ can utilize an adaptive attack strategy, such as adding regularization terms (i.e., defense evasion) to the objective function of the training process (see Sect. 6.6). A sophisticated $\mathcal{A}$ with the working knowledge of *BayBFed* (has access to the previous round base measure H^{t-1}) could select a sweet spot between the model accuracy and the evasion from *BayBFed*. As a result, the poisoned models are still similar to the benign models.

In all the above cases, we have demonstrated that *BayBFed* successfully detected all the malicious updates. The reason being *BayBFed* computes an alternate, more generic representation of the client updates, i.e., a probabilistic measure that encompasses all the adjustments made to the client updates due to any local client's training strategy. Hence, the detection module that takes this probabilistic measure as one of its inputs correctly identifies all the malicious updates without being affected by any local client training strategies. In addition, we also integrate the effect of $cos(W_i^t, G^{t-1})$ and $L_2 -$norm (Eq. 2 and Eq. 3) in the clients' model updates and the computation of error introduced by the client weight. The rationale is that even though $\mathcal{A}$ makes sure the distribution of malicious updates does not deviate from benign ones, $\mathcal{A}$ cannot fully manipulate the $cos(W_i^t, G^{t-1})$ or $L_2 -$norm. The reason being $\mathcal{A}$ aims to simulate the global model in the backdoor direction. This ensures that any changes the strategic $\mathcal{A}$ makes utilizing advanced hiding techniques cannot bypass *BayBFed*. We also empirically verified the effectiveness of *BayBFed* using state-of-the-art (CIFAR-10, MNIST, and FMNIST) and real-world (IoT) datasets and successfully demonstrated that $\mathcal{A}$ cannot conduct backdoor attacks while simultaneously bypassing our defense mechanism. Therefore, *BayBFed* is robust and resilient against backdoor attacks.

8. Related Works

Defense mechanisms (against backdoor attacks) in the literature can be broadly classified into two categories: *detection-based* defense mechanisms [37], [14], [27], [9], [16], [19], [20] and *mitigation-based* defense mechanisms [48], [15], [33], [40], [43], [44]. Detection-based defenses detect and filter the poisoned updates using similarity measures between the poisoned and benign updates. In contrast, mitigation-based defenses construct aggregation rules or add noise to the updates to mitigate the poisoned updates which are unbeknown to them.

Detecting backdoors. Detection-based defense mechanisms in the literature include: Auror [37], Krum [4], AFA [27], and FoolsGold [14]. However, these defense mechanisms work only when certain conditions are satisfied. For example, Auror and Krum only work for benign IID data. In contrast, FoolsGold overcomes this assumption by assuming the benign data is non-IID and that the manipulated data is IID. In addition, these defense mechanisms can be bypassed if an adversary restricts the malicious updates within the valid range of benign updates distribution. In summary, these defenses only work when certain conditions are satisfied. On the contrary, *BayBFed* does not assume anything about the distribution of local client's data. Thus, it works more effectively against such attacks.

Mitigating backdoors. Mitigation-based defenses include rule-based aggregation mechanisms such as coordinate-wise median and coordinate-wise trimmed mean [48], a two-step aggregation algorithm that combines the Krum and trimmed mean mechanisms [15], and RFA [33]. These defense mechanisms determine a client update to be benign if it lies within the scope of some aggregation rule. These rules, however, can be easily bypassed if an adversary makes sure its update is within the valid range of these rules. In addition, these rules are computationally intensive. Differential privacy (DP) defense mechanisms [40], [43], [44], [28] have also been designed to protect against backdoor attacks. These defense mechanisms follow clipping of the weights and additive noising [10], to limit the impact of the adversarial updates. However, they also decrease the *MA* simultaneously. Nguyen *et al.* [30] designed a defense to limit the impact of noise on *MA*, however, the outlier detection is prone to removing benign models, which reduces the performance in non-IID scenarios. In comparison, the BNP modeling and CRP-Jensen of *BayBFed* allow us to effectively distinguish between benign and poisoned models.

9. Conclusion

This paper proposes *BayBFed*, a novel and more generic probabilistic approach to defend against backdoor attacks in Federated Learning. In contrast to existing defenses that mainly consider models as a set of vectors and matrices [4], [14], [26], [27], [30], [37] and operate *directly* on them, *BayBFed* first computes a probabilistic measure over the clients' updates that encompass all the adjustments made in the updates due to any local client training strategy. Then, *BayBFed* employs a detection algorithm that utilizes this probabilistic measure to detect and filter out malicious updates. Thus, it overcomes several shortcomings of previous backdoor defense approaches. *BayBFed* utilizes two extensions of Bayesian non-parametric modeling techniques: the Hierarchical Beta-Bernoulli Process to draw a probabilistic measure given the clients' model updates (or weights), and a variation of the Chinese Restaurant Process, CRP-Jensen, which is a clustering algorithm that can leverage the probabilistic measure to detect and filter out malicious updates. Our extensive evaluation with benchmark datasets in different domains demonstrates that *BayBFed* can effectively mitigate backdoor attacks in FL while preserving the benign performance of the global model.

Acknowledgements

This work was funded in part by Intel as part of the Private AI center, HMWK within ATHENE project, Huawei as part of the OpenS3 Lab, the Hessian Ministry of Interior and Sport within the F-LION project, as part of the funding guideline for cyber security research and the US NSF under award number 1943351.

References

[1] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. BaFFLe: Backdoor Detection via Feedback-based Federated Learning. In *ICDCS*, 2021.

[2] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11:1803–1831, 2010.

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. In *AISTATS*, 2020.

[4] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *NIPS*, 2017.

[5] David M Blei and Peter I Frazier. Distance dependent chinese restaurant processes. In *ICML*. PMLR, 2010.

[6] Tamara Broderick, Ashia C Wilson, and Michael I Jordan. Posteriors, conjugacy, and exponential families for completely random measures. *Bernoulli*, 24(4B):3181–3221, 2018.

[7] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In *NDSS*, 2021.

[8] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Provably secure federated learning against malicious clients. *AAAI Conference on Artificial Intelligence*, 2021.

[9] Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *ICML*. PMLR, 2019.

[10] Cynthia Dwork, Adam Smith, Thomas Steinke, Jonathan Ullman, and Salil Vadhan. Robust traceability from trace amounts. In *FOCS*, 2015.

[11] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security*, 2020.

[12] Hossein Fereidooni, Alexandra Dmitrienko, Phillip Rieger, Markus Miettinen, Ahmad-Reza Sadeghi, and Felix Madlener. FedCRI: Federated Mobile Cyber-Risk Intelligence. In *NDSS*, 2022.

[13] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. Safelearn: Secure aggregation for private federated learning. In *IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021.

[14] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *RAID*, 2020.

[15] Rachid Guerraoui, Sébastien Rouault, et al. The hidden vulnerability of distributed learning in byzantium. In *ICML*. PMLR, 2018.

[16] Youssef Khazbak, Tianxiang Tan, and Guohong Cao. Mlguard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning. In *International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020.

[17] Yongdai Kim. Nonparametric bayesian estimators for counting processes. *Annals of Statistics*, pages 562–588, 1999.

[18] Günter Last and Mathew Penrose. *Lectures on the Poisson process*, volume 7. Cambridge University Press, 2017.

[19] Suyi Li, Yong Cheng, Yang Liu, Wei Wang, and Tianjian Chen. Abnormal client behavior detection in federated learning. *arXiv preprint arXiv:1910.09933*, 2019.

[20] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

[21] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[22] Yuelin Li, Elizabeth Schofield, and Mithat Gönen. A tutorial on dirichlet process mixture modeling. *Journal of mathematical psychology*, 91:128–144, 2019.

[23] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, 2017.

[24] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative Machine Learning without Centralized Training Data. Google AI, 2017.

[25] Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[26] Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning Differentially Private Language Models Without Losing Accuracy. In *ICLR*, 2018.

[27] Luis Muñoz-González, Kenneth T. Co, and Emil C. Lupu. Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging. In *arXiv preprint:1909.05125*, 2019.

[28] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. Local and central differential privacy for robustness and privacy in federated learning. *NDSS*, 2022.

[29] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *ICDCS*, 2019.

[30] Thien Duc Nguyen, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. FLAME: taming backdoors in federated learning. *USENIX Security*, 2022.

[31] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning Attacks on Federated Learning-Based IoT Intrusion Detection System. In *Workshop on Decentralized IoT Systems and Security*, 2020.

[32] John W Paisley, Aimee K Zaas, Christopher W Woods, Geoffrey S Ginsburg, and Lawrence Carin. A stick-breaking construction of the beta process. In *ICML*, 2010.

[33] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 2022.

[34] Phillip Rieger, Torsten Krauß, Markus Miettinen, Alexandra Dmitrienko, and Ahmad-Reza Sadeghi. Close the Gate: Detecting Backdoored Models in Federated Learning based on Client-Side Deep Layer Output Analysis. *arXiv preprint arXiv:2210.07714*, 2022.

[35] Phillip Rieger, Thien Duc Nguyen, Markus Miettinen, and Ahmad-Reza Sadeghi. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. *NDSS*, 2022.

[36] Micah Sheller, Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation. In *Brain Lesion Workshop*, 2018.

[37] Shiqi Shen, Shruti Tople, and Prateek Saxena. Auror: Defending Against Poisoning Attacks in Collaborative Deep Learning Systems. In *ACSAC*, 2016.

[38] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. In *TMC*, 2018.

[39] Richard Socher, Andrew Maas, and Christopher Manning. Spectral chinese restaurant processes: Nonparametric clustering based on similarities. In *AISTATS*. JMLR Workshop and Conference Proceedings, 2011.

[40] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

[41] Romain Thibaux and Michael I Jordan. Hierarchical beta processes and the indian buffet process. In *AISTATS*. PMLR, 2007.

[42] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*, 2020.

[43] Chen Wu, Xian Yang, Sencun Zhu, and Prasenjit Mitra. Mitigating backdoor attacks in federated learning. *arXiv preprint arXiv:2011.01767*, 2020.

[44] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. Crfl: Certifiably robust federated learning against backdoor attacks. In *ICML*. PMLR, 2021.

[45] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. DBA: Distributed Backdoor Attacks against Federated Learning. In *ICLR*, 2020.

[46] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1), 2021.

[47] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[48] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *ICML*. PMLR, 2018.

[49] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *ICML*. PMLR, 2019.

Appendix A. Federated Learning

Federated learning (FL) collaboratively learns a global model G by iteratively aggregating the local model updates sent by the n clients selected during each training round. In each training round t, the global model server selects a total of n clients and sends them a common aggregated global model, G^{t-1}. Then, each client $i \in \{1,\dots,n\}$ locally trains a local model W_i^t, using its own local dataset D_i. After training the local model, each client i, sends the updated model parameters to the global server to compute the next stage global model, G^t. In this paper, we assume the global server aggregates the local updates by utilizing *Federated Averaging* (FedAVG) function [23], given as:

$$G^t = \sum_{i=1}^{n} \frac{s_i}{s} W_i^t, \quad \text{where } s_i = ||D_i|| \text{ and } s = \sum_{i=1}^{n} s_i$$

FedAVG utilizes each client's training dataset to compute the latest global model. A malicious client (or clients) can take advantage of this requirement by sending the erroneous dataset size [42]. To eliminate this scenario, we assume equal weights ($s_i = 1/n$) for all the clients. Such an approach has also been adopted in previous research efforts [3], [37], [45], [35].

Appendix B. Background and Preliminaries

In this section, we provide brief technical background knowledge on Bayesian non-parametric modeling and related concepts, which will be critical in understanding the design of *BayBFed*.

B.1. Hierarchical Beta-Bernoulli Process (HBBP)

Next, we describe the concepts of the baseline Beta Process, the Hierarchical Beta Process, and the Bernoulli Process, which are used to compute the probabilistic measure.

Beta Process (BP). A Beta Process is a random discrete measure on the countable infinitely drawn set of weights, where each weight has a mass in the range (0,1) such that the total mass sums to 1. A Beta Process uses a concentration function c over some space $\Omega = \mathbb{R}$ and a base measure H to produce some random measure A, i.e., $A \sim BP(c,H)$. Given the set Ω, informally, a *measure* is any consistent assignment of *sizes* to (some of) the subsets of the set. Depending on the application, the size of a subset may be interpreted as either its physical size or the probability that some random process will yield a result within the subset. Formally, a measure is a function $\mu : \Sigma \mapsto [0,\infty]$, where Σ is the σ-algebra (collection of subsets) of Ω. A concentration function (c) quantitatively characterizes the scatter of the values of a random variable. Thus, it indicates the similarity between the input base measure (H) and the output random measure (A). The base measure H can represent any initial distribution (see Sect. 4). We also call $\gamma_0 = H(\Omega)$ the mass parameter. So, if H is a normal distribution, then γ_0 is a normal distribution of the complete space Ω. Alternatively, A is a discrete measure (discrete weights), represented by $A = \sum_j a_j \delta_{w_j}$, where δ is an indicator function. Thus, in order to draw an infinitely countable set of points $(a_j, w_j) \rightarrow [0,1] \times \Omega$ needs to be drawn. The probabilistic weights $\{a_j\}_{j=1}^{\infty}$ are distributed by a stick-breaking process: $d_j \sim Beta(\gamma_0, 1)$, $a_j = \prod_{k=1}^{j} d_k$. In a stick-breaking process [32], there is a stick of length of 1 and a_j represents the probabilistic weight taken from the remainder of the stick every time. w_j are drawn independently and identically distributed (IID) from the normalized base measure $w_j \sim H/H(\Omega)$ with domain Ω. Here, $Beta(\cdot)$ represents the beta distribution that is used to model the continuous random variables in the range [0, 1]. This work assumes that Ω is simply a space of weights. The objective here is first to draw a baseline Beta prior (random discrete measure) using a Beta Process and then use this prior (in a Hierarchical Beta Process, as explained next) to draw the corresponding Beta priors for the n entities.

Hierarchical Beta Process (HBP). A Hierarchical Beta Process (HBP) is used to create hierarchies of the baseline Beta process when certain conditions are satisfied. Alternatively, from a pool of countable infinite sets of weights of the BP, a subset of weights (under some conditions) are drawn for each sub-Beta Process, creating hierarchies. We can employ HBP to draw a discrete random measure corresponding to each of the n entities based on the baseline Beta Process prior [41]. Let us consider the following rationale for the construction of the HBP. Suppose that W^{Prior} is a list of the n entities' weight vectors, Now, we assume that the prior for each entity i, W_i^{Prior} is generated by including weights, which have a specific cosine angular distance, with respect to some base weight, w_b (different for every entity). Thus, each W_i^{Prior} is generated by including h weights (w) independently with a probability p_w^h specific to the entity i. These probabilities form a discrete measure $A_{i,h}$ over the space of weights Ω, and we put a Beta Process $BP(c_i, A)$ prior on $A_{i,h}$ (Note: $A_{i,h}$ is the same as A_i defined in Sect. 2 of main paper). In summary, we have the following Hierarchical Beta model:

$$\text{Baseline Beta prior:} \quad A \sim BP(c, H)$$

$$\text{Hierarchical Beta prior:} \quad A_{i,h} \sim BP(c_i, A) \quad \forall 1 \leq i \leq n$$

The random measure A, thus, $A_{i,h}$, encodes the probability that each entity possesses each particular weight.

Bernoulli Process (BeP). A Bernoulli Process (BeP) is a draw of weights from a space of total weights, given the Beta Process random measure that encodes the probability of selecting the weight in the draw. BeP is employed to draw weights given the Hierarchical Beta priors computed earlier. Thus, the subsets of points in the HBP prior $A_{i,h}$ are drawn using a BeP with input as the random measure $A_{i,h}$. Each subset W_i for entity in $i \in \{1, ..., n\}$, having l weights, is characterized by a Bernoulli Process such that $W_i | A_{i,h} \sim \text{BeP}(A_{i,h})$. Each subset can also be represented by a discrete measure such that the points $(b_{i,l}, w_l) \rightarrow [0,1] \times \Omega$, forms $W_i = \sum_l b_{i,l} \delta_{w_l}$, where $b_{i,l}$ is the probabilistic weight (success probability) given to w_l, i.e., $Pr(b_{i,l} = 1) = p_w^h$, if they are included in the subset W_i.

Conjugacy. It has been shown in the literature that the Beta distribution is the conjugate of the Bernoulli distribution [6]. Hence, we do not have to use the computationally intensive Bayes' rule to compute the posterior distribution of hierarchical random measures. It can be computed as follows: Let $A_{i,h} \sim BP(c_i, A)$, and let $W_i | A_{i,h} \sim \text{BeP}(A_{i,h})$. In $W_i = \{W_{i,1}, W_{i,2}, ..., W_{i,l}\}$, l denotes the independent BeP draws over the likelihood function, $A_{i,h}$. By using the results for HBP and BeP in [17], the posterior distribution of $A_{i,h}$ after observing W_i is still a Beta process with modified parameters:

$$A_{i,h} | W_i \sim BP\left(c_i + l, \frac{c_i}{c_i + l} H + \frac{1}{c_i \cdot l} \sum_{l=1}^{l} W_i\right) \tag{9}$$

Our motivation is to first draw a baseline Beta Process random measure by drawing a countable infinitely set of weights, such that their probabilistic weight sums to 1. Then, we use this baseline Beta Process to form hierarchies of Beta Process for n different entities. We do so by selecting a subset of h weights from the total weights space for each of the n entities. Then, for each of the n entities, we use Bernoulli Process to draw l weights from the corresponding hierarchical Beta Process weights space. Finally, we keep updating the corresponding hierarchical Beta Process for entity n using the conjugacy of the Beta Process and the Bernoulli Process [6].

B.2. Mixture modeling: Chinese Restaurant Process (CRP)

The CRP [39], [5], [22] is a discrete-time stochastic process in the probability theory that resembles the situation of seating customers at tables in a Chinese restaurant with an infinite number of circular tables, each with infinite capacity. The first customer that arrives sits at the first table. The following customers can either sit at the already occupied tables or can choose to sit at the new table. This process partitions the customers among tables. The results of this process are *exchangeable*, meaning that the order in which the customers arrive and sit does not affect the probability of the final distribution. In CRP, we compute two probabilities for table (cluster) assignment. The first probability is the probability of the customer entering the restaurant and sitting at the already occupied tables (clusters). The second probability is the predictive probability of how well this new customer fits the mean of already occupied tables (clusters).

B.3. Jensen-Shannon Divergence

Jensen-Shannon Divergence or Jensen-Divergence is used to realize the distance between two distributions. Specifically, it is computed by estimating the relative entropy between two distributions. The entropy of a random variable X having probability mass function $P(x)$ is given as:

$$H(X) = -\sum_{x \in X} P(x) log_b P(x) \tag{10}$$

Jensen-Divergence is estimated using the Kullback-Leibler divergence (KL-divergence). KL-divergence also measures the distance between two distributions. However, it is not symmetric and does not satisfy the triangle inequality. Jensen-Divergence is an approach that improves upon the KL-divergence, as it is symmetric and a smoothed version of KL-divergence. KL divergence between two distributions p and q is given as:

$$D_{KL}(p||q) = \int_{-\infty}^{\infty} p(x) log\left(\frac{p(x)}{q(x)}\right) dx \tag{11}$$

Jensen-Divergence between two distributions p and q is given as:

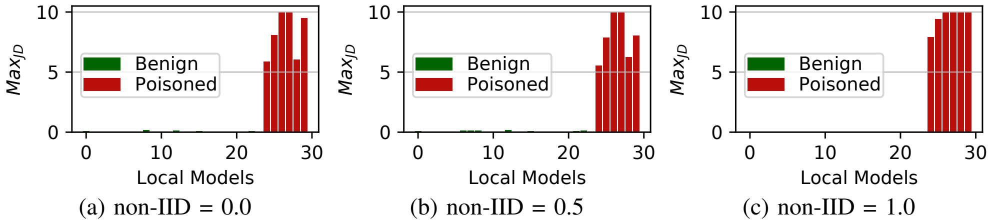
$$JSD(p||q) = \frac{1}{2}\left(D_{KL}(p||m) + D_{KL}(q||m)\right) \tag{12}$$

Appendix C. Overview of Used Symbols

Table 3 contains an overview of the used symbols.

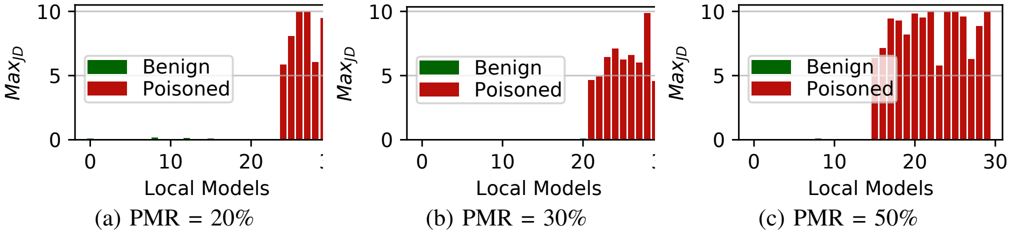
Appendix D. *BayBFed* statistics for FMNIST

In this section, we evaluate the impact of non-IID rate and *PMR* on the FMNIST dataset. First, we demonstrate



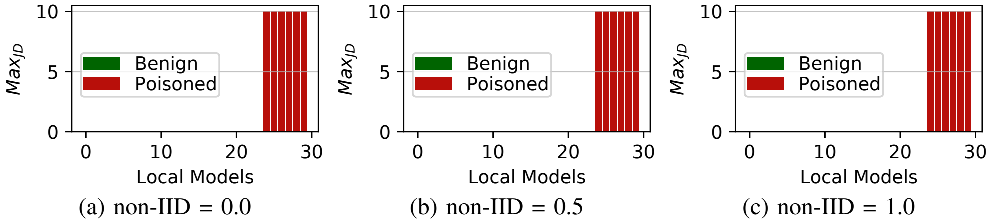
(a) non-IID = 0.0 (b) non-IID = 0.5 (c) non-IID = 1.0

Figure 11: Effect of different non-IID rates on the maximum Jensen-Divergence (Max^{i}_{JD}) for FMNIST dataset.



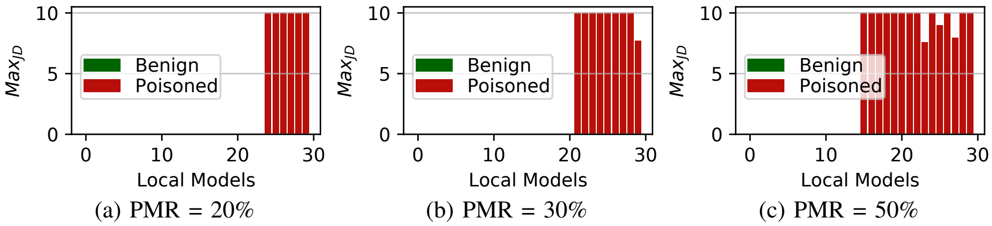
(a) PMR = 20% (b) PMR = 30% (c) PMR = 50%

Figure 12: Effect of different PMR rates on the maximum Jensen-Divergence (Max^{i}_{JD}) for the FMNIST dataset.



(a) non-IID = 0.0 (b) non-IID = 0.5 (c) non-IID = 1.0

Figure 13: Effect of different non-IID rates on the maximum Jensen-Divergence (Max^{i}_{JD}) for MNIST dataset.



(a) PMR = 20% (b) PMR = 30% (c) PMR = 50%

Figure 14: Effect of different PMR rates on the maximum Jensen-Divergence (Max^{i}_{JD}) for the MNIST dataset.

the trend of Max^{i}_{JD} for both the malicious and benign clients with respect to each of these parameters. Then, we illustrate the impact of non-IID rate and *PMR* on *BayBFed*'s performance by quantifying different metrics as stated in Sect. 5 and also compare it against the no defense scenario.

Illustration of Max^{i}_{JD}. The impact of the degree of non-IID data and *PMR* on Max^{i}_{JD} for each client is shown in Fig. 11 and Fig. 12, respectively. We select a total of 30 (n= 30) clients for both the non-IID and *PMR* experimental analysis. For non-IID analysis, we test non-IID $\in \{0.0, 0.5, 1.0\}$ and set $PMR = 0.2$. Thus, the number of malicious clients equals 6 ($n_{\mathcal{A}}$ = 6). For *PMR* analysis, we test $PMR \in \{0.2, 0.3, 0.5\}$, i.e., when $n_{\mathcal{A}}$ equals 6, 9, and 15 and set non-IID = 0.7. As illustrated in Fig. 11 and Fig. 12, the Max^{i}_{JD} value for benign clients differs significantly from that of malicious clients. Hence, *BayBFed* easily filters out all the malicious client updates, achieving a *BA* of zero while keeping the *MA* of the global model intact.

Appendix E. *BayBFed* statistics for MNIST

In this section, we evaluate the impact of non-IID rate and *PMR* on the MNIST dataset. First, we demonstrate the trend of Max^{i}_{JD} for both the malicious and benign clients, with respect to each of these parameters. Then, we illustrate the impact of non-IID rate and *PMR* on *BayBFed*'s performance by quantifying different metrics as stated in Sect. 5 and also compare it against the no defense scenario.

Illustration of Max^{i}_{JD}. Impact of the degree of non-IID data and *PMR* on Max^{i}_{JD} for each client is shown in Fig. 13 and Fig. 14, respectively. We select a total of 30 (n= 30)

TABLE 3: Overview of symbols.

Symbol	Description
BNP	Bayesian non-parametric
HBBP	Hierarchical Beta-Bernoulli Process
BP	Beta Process
HBP	Hierarchical Beta Process
BeP	Bernouli Process
CRP	Chinese Restaurant Process
f	Neural Network (NN)
$Beta(\cdot)$	Beta distribution
G^t	Global Model at time t
W_i^t	Local Model of client i at time t
D_i	Local dataset of client i
n	Number of clients
$n_{\mathcal{A}}$	Number of malicious clients
t	FL training round
c	Concentration function
$\Omega(\mathbb{R})$	Space of weights
H	Base measure
A	Beta Process random measure
γ_0	Mass parameter
δ	Indicator function
w_j	IID weights drawn from $\Omega(\mathbb{R})$ or from BP
a_j	Probabilistic weight assigned to w_j
d_j	A stick-breaking process
W^{Prior}	List of the n client weight vectors
W_i^{Prior}	client i's weight vector
h	Weights drawn from BP to be included in W_i^{Prior}
p_w^h	Probability with which h weights are drawn from BP to be included in W_i^{Prior}
$A_{i,h}$ or A_i	HBP random measure for each client i having weights h
c_i	client i's concentration function
$BP(c_i,A)$	HBP $BP(c_i,A)$ prior on $A_{i,h}$
w_l	IID weights drawn from HBP
$b_{i,l}$	probabilistic weight given to w_l. Equal to p_w^h
$D_{KL}(p \Vert q)$	KL divergence between two distributions p and q
$JSD(p \Vert q)$	Jensen-Divergence between two distributions p and q
$\mathcal{L}$	Labels for samples from domain $\mathcal{D}$
$\mathcal{A}$	Adversary
$l_{\mathcal{A}}$	$\mathcal{A}$ chosen labels
$\mathcal{D}_{\mathcal{A}}$	*trigger set* of $\mathcal{A}$
$\mathcal{N}$	Normal distribution
μ_p	Mean of the flattened initial G^t
σ_p	Standard deviation of the flattened initial G^t
$W_{i,up}^t$	Updated client weight at time t
Max_{JD}^i	Maximum Jensen-Divergence
$cos(G_{t-1}, W_i^t)$	Cosine angular distance between local model W_i^t and global model G_{t-1}
$d_{w_i^t}$	L_2-norm between G_{t-1} and W_i^t
$\sigma_{w_i^t}$	Measurement error due to the new client's weight
$\overline{W_{i,up}^t}$	Mean of $W_{i,up}^t$
μ_{c_l}	Mean of the clusters
σ_{c_l}	Variance of the clusters
noc	Total number of clusters formed yet
p_i	p distribution of client i
q_{noc}	noc cluster q distribution
js_{noc}^i	Jensen-Divergence of noc cluster q distribution with p distribution of client i
μ_{new}	Updated cluster's mean
σ_{new}	Updated cluster's standard deviation
n_k	Number of clients update already assigned to a particular cluster
τ_k	Precision of the cluster
μ_0	Initial mean for the new cluster
τ_0	Initial Precision assumed for the new cluster
$Max_{JD}^{stored} = []$	Array of stored Max_{JD}^i

clients for both the non-IID and *PMR* experimental analysis. For non-IID analysis, we test non-IID $\in \{0.0, 0.5, 1.0\}$ and set $PMR = 0.2$. Thus, number of malicious clients equals 6 ($n_{\mathcal{A}}$ = 6). For *PMR* analysis, we test $PMR \in \{0.2, 0.3, 0.5\}$, i.e., when $n_{\mathcal{A}}$ equals 6, 9, and 15 and set non-IID = 0.7. As illustrated in Fig. 13 and Fig. 14, the Max_{JD}^i value for benign clients differs significantly from that of malicious clients. Hence, *BayBFed* easily filters out all the malicious client updates, achieving a *BA* of zero while keeping the *MA* of the global model intact.

Appendix F. Additional adaptive attack

To implement an adaptive attack in which an adversary makes small changes to client updates to keep the JD divergences small, we increment the Poisoned Data Rate (PDR) to demonstrate the increment in the client updates for the CIFAR-10 dataset. We choose PDR to implement such an adaptive attack because arbitrary increments in PDR will also reflect the random increments in the client updates. Then, we compute the *TPR*, *TNR*, *BA*, and *MA* metrics to evaluate the effectiveness of *BayBFed* compared against the no-defense *BA* and *MA*, for different PDR values. We conduct experiments for three cases: a) PDR $\in$ (0.1,0.75) with an increment of 0.1, b) PDR $\in$ (0.01,0.1) with an increment of 0.01, and c) PDR $\in$ (0.1,0.2) with an increment of 0.01. Case a) demonstrates the impact of large PDR increments on the metrics mentioned above. The *BA* with no defense remains at zero for the initial PDR values of 0.01 and 0.1, and after that, it starts to increase. *BayBFed* easily identified all the malicious updates, thus filtering out all the malicious client updates, achieving a *BA* of zero while keeping the *MA* of the global model intact. As we observed in case a), the no-defense *BA* remains zero at PDR = 0.1 and starts to increase after that; we conducted two more experiments for PDR $\in$ (0,0.1) and PDR $\in$ (0.1,0.2). The reason is to analyze the *BayBFed* performance when PDR increases. Case b) demonstrates the impact of very small increments in PDR, and the no defense *BA* remains at zero in this case. Nevertheless, *BayBFed* was correctly able to identify all the malicious updates. In the case of c), the no defense *BA* starts to increase from PDR = 0.11, and *BayBFed* again correctly identified all the malicious updates, thus achieving a *BA* of zero while keeping the *MA* of the global model intact. This experimental analysis demonstrates that even when a shrewd adversary makes small iterative changes to PDR (in consequence, client updates), *BayBFed* works efficiently by identifying all the malicious updates.