Blades: A Unified Benchmark Suite for Byzantine Attacks and Defenses in Federated Learning

Shenghui Li*, Edith C.-H. Ngai†, Fanghua Ye‡, Li Ju*, Tianru Zhang*, Thiemo Voigt*§
*Uppsala University, Uppsala, Sweden. *{shenghui.li,li.ju,tianru.zhang}@it.uu.se*
†The University of Hong Kong, Hong Kong, China. *chngai@eee.hku.hk*
‡University College London, London, UK. *fanghua.ye.19@ucl.ac.uk*
§Research Institutes of Sweden, Stockholm, Sweden. *thiemo.voigt@angstrom.uu.se*

***Abstract*—Federated learning (FL) facilitates distributed training across different IoT and edge devices, safeguarding the privacy of their data. The inherent distributed structure of FL introduces vulnerabilities, especially from adversarial devices aiming to skew local updates to their advantage. Despite the plethora of research focusing on Byzantine-resilient FL, the academic community has yet to establish a comprehensive benchmark suite, pivotal for impartial assessment and comparison of different techniques. This paper presents Blades, a scalable, extensible, and easily configurable benchmark suite that supports researchers and developers in efficiently implementing and validating novel strategies against baseline algorithms in Byzantine-resilient FL. Blades contains built-in implementations of representative attack and defense strategies and offers a user-friendly interface that seamlessly integrates new ideas. Using Blades, we re-evaluate representative attacks and defenses on wide-ranging experimental configurations (approximately 1,500 trials in total). Through our extensive experiments, we gained new insights into FL robustness and highlighted previously overlooked limitations due to the absence of thorough evaluations and comparisons of baselines under various attack settings. We maintain the source code and documents at https://github.com/lishenghui/blades.**

***Index Terms*—Byzantine attacks, distributed learning, federated learning, IoT, neural networks, robustness**

I. Introduction

Federated learning (FL) [1], [2] has emerged as a compelling paradigm, allowing for collaborative machine learning model construction by leveraging distributed data across a diverse range of client devices, from IoT and edge devices to mobile phones and computers. FL enables on-device machine learning without the need to migrate local data to a central cloud server, which is suitable for distributed IoT devices collecting massive sensor data. FL impacts many existing and future IoT applications, including smart grids, smart transportation, smart health, and augmented reality [3], [4]. The FL process typically involves several iterative steps: Firstly, a central server distributes the current global model to the clients. Subsequently, the clients independently perform one or multiple local steps of stochastic gradient descent (SGD) using their local datasets and transmit the updates back to the server. The server then aggregates these local updates to generate a new global model, which serves as the basis for the next round of training. The FL paradigm allows clients to jointly train a machine learning model without disclosing their private data to the central server. Furthermore, FL exhibits improved communication efficiency compared to traditional distributed learning methods [5], as it capitalizes on multiple local update steps before transmitting the updates [6].

Due to the distributed characteristic of optimization, FL is vulnerable to Byzantine failures [7], [8], wherein certain participants may deviate from the prescribed update protocol and upload arbitrary parameters to the central server. This risk is notable in IoT applications [3], [4], where their open architecture allows diverse interconnectivity, thereby potentially expanding the attack surface. In this context, typical FL algorithms like `FedAvg` [1], which compute the sample mean of client updates for global model aggregation, can be significantly skewed by a single Byzantine client [9]. The server thus requires Byzantine-resilient solutions to defend against malicious clients.

Depending on the adversarial goals, Byzantine attacks in FL can be classified into two categories: *targeted attacks* and *untargeted attacks* [7], [10]. Targeted attacks, such as backdoor attacks, aim to manipulate the global model to generate attacker-desired misclassifications for some particular test samples [11]–[13], while untargeted attacks aim to degrade the overall performance of the global model indiscriminately [14]. Our work focuses on *untargeted attacks*, which is consistent with the majority of Byzantine-resilient research [4], [15]–[20]. Henceforth, any reference to "Byzantine" will imply "untargeted Byzantine".

In recent years, the field of FL has seen the emergence of various Byzantine-resilient approaches. They aim to protect distributed optimization from Byzantine clients and assure the performance of the learned models [8], [21]. For instance, robust aggregation rules (AGRs) are widely used to estimate the global update from a collection of local updates while mitigating the impact of malicious behaviors. Typical AGRs include GeoMed [17], Krum [15], TrimmedMean [16], and Median [16]. Meanwhile, different attack strategies are emerging, striving to circumvent defense strategies [22], [23]. For instance, the A Little Is Enough (ALIE) attack can bypass various AGRs by taking advantage of the empirical variance between clients' updates if such a variance is high enough, especially when the local datasets are not independent and

Edith C.-H. Ngai is the corresponding author.

979-8-3503-7025-6/24/$31.00 ©2024 IEEE
DOI 10.1109/IoTDI61053.2024.00018

identically distributed (non-IID) [9], [22]. Thus, defending against adversarial attacks remains an open problem in FL [24].

Moreover, it has been shown that the experimental evaluation in existing studies may be insufficient to validate their robustness against diverse Byzantine attacks [25], as they were only examined under specific experimental settings (*e.g.,* specific attack types, and hyper-parameter configurations).

The study emphasizes that limited and narrowly focused evaluations might overlook certain vulnerabilities and provide an incomplete picture of their robustness against different threats. This underscores the pressing need for a unified benchmark suite that offers comprehensive assessments and fair comparison across various attacks and scenarios.

Our work: This paper presents *Blades*, our open-sourcebenchmark suite to fill the identified gaps in existing experimental evaluations in Byzantine-resilient FL. Through *Blades*, we conduct comprehensive experimental evaluations, scrutinizing a range of representative attacks and defense techniques. Specifically, we make the following two concrete contributions:

Contribution 1. Blades, a benchmark suite: We introduce Blades, an open-source benchmark suite for **B**yzantine-resilient federated **L**earning with **A**ttacks and **D**efenses **E**xperimental **S**imulation, which is specifically designed to fill the need for studying attack and defense problems in FL. Blades is built upon a versatile distributed framework, Ray, enabling effortless parallelization of single machine code across various settings, including single CPU, multi-core, multi-GPU, or multi-node, with minimal configuration requirements. This makes Blades efficient in terms of execution time, as client and server operations are executed in a parallel manner. In addition, Blades provides a wide range of attack and defense mechanisms and allows end users to plug in customized or new techniques easily. We illustrate the user-friendly nature of Blades through examples and validate its scalability with respect to clients and computational resources. The results highlight that Blades can effectively handle large client populations and computational resources.

Contribution 2. Comparative case studies: Using Blades, we conduct an exhaustive re-evaluation of six representative AGRs, encompassing both three classical and three contemporary methods, against six attacks on three datasets. Additionally, we also inspect key factors and risks that might affect the robustness of AGRs, including data heterogeneity, differential privacy (DP) noise, momentum, and the risk of gradient explosion. The **key findings** from our experiments can be summarized as follows:

1) The effectiveness of adversarial attacks depends on several factors, including the choice of dataset, defensive countermeasure, and the specific FL algorithm employed.
2) The robustness of defenses in existing studies may be overrated owing to their insufficiency in comprehensive evaluation under wide-ranging settings.
3) Additionally, various factors, including data heterogeneity, differential privacy (DP) noise, and momentum, exert considerable influence on the Byzantine resilience of defense strategies.

Algorithm 1 A `FedAvg`-family Algorithm for FL

Input: K, T, $\boldsymbol{w}^0$, CLIENTOPT, SERVEROPT

for each global round $t \in [T]$ **do**
  Select a subset S_t from K clients at random
  for each client $k \in S_t$ **in parallel do**
    $\boldsymbol{w}_k^t \leftarrow \boldsymbol{w}^t$
    for E_l local rounds **do**
      Compute an estimate $g_k(\boldsymbol{w}_k^t)$ of $\triangledown F_k(\boldsymbol{w}_k^t)$
      $\boldsymbol{w}_k^t \leftarrow \text{CLIENTOPT}(\boldsymbol{w}_k^t, g_k(\boldsymbol{w}_k^t), \eta_l, t)$
    end for
    $\boldsymbol{\Delta}_k^t \leftarrow \boldsymbol{w}_k^t - \boldsymbol{w}^t$
    Send Δ_k^t back to the server
  end for
  $\Delta^{t+1} \leftarrow \text{AGG}(\{\Delta_k^t\}_{k \in S_t})$
  $\boldsymbol{w}^{t+1} \leftarrow \text{SERVEROPT}(\boldsymbol{w}^t, -\Delta^{t+1}, \eta_g, t)$
end for
return $\boldsymbol{w}^T$

The key findings underscore the intrinsic complexities and nuances in ensuring Byzantine resilience in FL and further emphasize the importance of a unified benchmark like Blades that enables comprehensive evaluations on attack and defense techniques.

II. Background and Related Work

A. FL and Optimization

In FL, a collection of clients collaboratively learn a shared global model by leveraging their private datasets in a distributed manner, assisted by the coordination of a central server. The goal is to find a parameter vector $\boldsymbol{w}$ that minimizes the following distributed optimization model:

$$\min_{\boldsymbol{w}} F(\boldsymbol{w}) := \frac{1}{K} \sum_{k \in [K]} F_k(\boldsymbol{w}), \tag{1}$$

where K represents the total number of clients and $F_k(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{z} \sim \mathcal{D}_k}[\ell(\boldsymbol{w}; \boldsymbol{z})]$ denotes the expected risk of the k-th client. Here, $\mathcal{D}_k$ is the data distribution for the k-th client and $\ell(\cdot; \cdot)$ is a user-specified loss function.

The most popular algorithms in the literature that solve (1) are the `FedAvg`-family algorithms [1], [26], [27]. As shown in Algorithm 1, at t-th round of communication, a subset of clients S_t is selected, typically through a random sampling process. The server then broadcasts its current global model parameters $\boldsymbol{w}^t$ to each selected client. Simultaneously, the clients independently perform local optimization on their respective private data, aiming to minimize their empirical loss. This process involves multiple local rounds, denoted as E_l, where the clients compute an estimate $g_k(\boldsymbol{w}_k^t)$ of the gradient $\nabla F_k(\boldsymbol{w}_k^t)$ from their local data. The client model's $\boldsymbol{w}_k^t$ are iteratively updated based on the estimated gradient and a client-specific learning rate η_l. The computed local model

updates, denoted as $\mathbf{\Delta}_k^t$, are then transmitted back to the server. The server aggregates these updates using an aggregation rule, often averaging aggregation [1], to generate a global update. This update represents a direction for the global optimizer, capturing the collective knowledge of the participating clients. Subsequently, the server employs the global optimizer, denoted as SERVEROPT, to update the global model's parameters $\boldsymbol{w}^t$ using the negative of the aggregated updates, denoted by $-\mathbf{\Delta}_k^{t+1}$ (which is called "pseudo-gradient" [26]), and a global learning rate η_g. By iterating this process for multiple rounds, the `FedAvg`-family algorithms refine the global model by leveraging the clients' distributed computing capabilities and decentralized datasets.

Our study adopts the full client participation paradigm in alignment with previous research [28]. As such, every client is actively engaged in each round of local training, ensuring that $|S_t| = K$ as per Algorithm 1. The rationale behind this choice is grounded in a prevailing assumption of Byzantine-resilient studies in FL, i.e., the number of malicious updates for aggregation is less than half during each round. Selecting subsets at random risks contravening this foundational assumption, given the inherent possibility of inadvertently favoring an excessive proportion of adversarial clients over their benign counterparts [9].

B. Scope of our work: Byzantine Attacks and Defenses in FL

Byzantine attacks pose a significant threat to FL due to its distributed optimization nature [7], [8]. In general, the malicious clients may upload arbitrary parameters to the server to degrade the global model's performance. Hence, in Algorithm 1, the `FedAvg`-family algorithm, Line 9 can be replaced by the following update rule:

$$\mathbf{\Delta}_k^t \leftarrow \begin{cases} \star & \text{if } k\text{-th client is Byzantine,} \\ \boldsymbol{w}_k^t - \boldsymbol{w}^t & \text{otherwise,} \end{cases} \tag{2}$$

where $\star$ represents arbitrary values.

As aforementioned, the scope of this work is on untargeted Byzantine attacks, where the adversary's objective is to minimize the accuracy of the global model for any test input [9], [14], [22], [29]. Various attack strategies have been proposed to explore the security vulnerabilities of FL, taking into account different levels of the adversary's capabilities and knowledge [18], [22], [29], [30]. For instance, with limited capabilities and knowledge and without having access to the training pipeline, the adversary can manipulate a single client's input and output data. In more sophisticated attacks, the adversary possesses complete knowledge of the learning system and designs attack strategies to circumvent defenses. Below, we detail here some typical attacks:

LabelFlipping [18]: The adversary simply flips the label of each training sample [14]. Specifically, a label l is flipped as $L-l-1$, where L is the number of classes in the classification problem and $l = 0, 1, ..., L-1$.

SignFlipping [30]: The adversary strives to maximize the loss via gradient ascent instead of gradient descent. Specifically, it flips the gradient's sign during the local updating step.

Noise [4]: The adversary samples some random noise from a distribution (e.g., Gaussian distribution) and uploads it as local updates.

ALIE [22]: The adversary takes advantage of the empirical variance among benign updates and uploads a noise within a range without being detected. For each coordinate $i \in [d]$, the attackers calculate mean (μ_i) and std (δ_i) over benign updates and set malicious updates to values in the range $(\mu_i - z^{max}\delta_i, \mu_i + z^{max}\delta_i)$, where z^{max} ranges from 0 to 1, and is typically obtained from the Cumulative Standard Normal Function. The i-th malicious update is then obtained by $\Delta_{k,i}^t \leftarrow \mu_i - z^{max}\mu_i$.

IPM [23]: The adversary seeks the negative inner product between the true mean of the updates and the output of the aggregation rules so that the loss will at least not descend. Assuming that the attackers know the mean of benign updates, a specific way to perform an IPM attack is

$$\mathbf{\Delta}_1^t = \cdots = \mathbf{\Delta}_M^t = -\frac{\epsilon}{K-M}\sum_{i=M+1}^{K}\mathbf{\Delta}_i^t, \tag{3}$$

assuming that the first M clients are malicious, ϵ is a positive coefficient controlling the magnitude of malicious updates.

MinMax [29]: Similar to ALIE, the adversary strives to ensure that the malicious updates lie close to the clique of the benign updates. The difference is that MinMax re-scales z^{max} such that the maximum distance from malicious updates to any benign updates is upper-bounded by the maximum distance between any two benign updates.

As for defenses, robust aggregation rules (AGRs) are widely applied to make a Byzantine-resilient estimation of the true updates and exclude the influence of malicious updates [4], [15]–[17], [19]. While other research directions, such as trust-based strategies [31]–[33] and variance-reduced algorithms [34], [35], are worth exploring, our study primarily focuses on AGRs. This is because most existing studies predominantly consider AGR-based defenses, and we specifically examine Median [16], TrimmedMean [16], GeoMed [36], DnC [29], ClippedClustering [9], and SignGuard [37] in this work.

C. Unifying Byzantine-resilient FL with Traditional DL

The study of Byzantine-resilient FL can be traced back to traditional distributed learning (DL), where a central server distributes data to workers who perform gradient estimation; the gradients are then aggregated by the server for model update [16], [17], [38]. FL originally emerged as an extension of distributed learning to address the limitations imposed by communication constraints and privacy concerns associated with decentralized data ownership [39]. Although FL and traditional distributed learning are employed in different application domains, they share similar security vulnerabilities stemming from Byzantine attacks due to the distributed nature of optimization. Furthermore, many existing techniques initially proposed for studying Byzantine-resilient distributed learning [15]–[17], [22], [40] have now found extensive application in the defense mechanisms utilized in FL [9], [14],

Table I: Comparing Blades with existing benchmarks for Byzantine-resilient FL

Features	AggregaThor [41]	FedMLSecurity [42]	Blades
Year	2019	2023	2023
ML Backend	TensorFlow	Pytorch	Pytorch
Distributed Backend	MPI	MPI	Ray
Flexible APIs	✗	✓	✓
`FedSGD` Algorithm	✓	✗	✓
`FedAvg`-family Algorithms	✗	✓	✓
Hyperparameter Tuning	✗	✗	✓

[20], [29], [36]. Therefore, it is important to examine FL and traditional distributed machine learning together when it comes to Byzantine resilience.

Benefiting from the generality of Algorithm 1, obtaining traditional distributed learning algorithms is straightforward. For example, by assuming both "CLIENTOPT" and "SERVEROPT" as a gradient descent step and setting $E_l = 1$ and $\eta_l = 1$, Algorithm 1 simplifies to the naive distributed SGD with gradient aggregation [17]. In contrast, setting $\eta_g = 1$ leads to the naive `FedAvg` algorithm. This connection enables the generalization of traditional techniques, such as robust aggregation, from traditional distributed learning to suit the requirements of FL.

D. Benchmarks for Byzantine-resilient FL

Recently, various FL benchmarks have been introduced with different emphases and scopes including scalability (*e.g.,* Fedscale [43]), heterogeneity (*e.g.,* B-FHTL [44] and pFL-bench [45]), privacy (*e.g.,* PrivacyFL [46] and Pysyft [47]), and security (*e.g.,* AggregaThor [41], FedMLSecurity [42], and Backdoor Bench [48]). Concerning Byzantine-resilience, AggregaThor [41], FedMLSecurity [42] are the most relevant benchmarks to our work. However, they both fall short across core dimensions (Table I). AggregaThor is mainly limited in the versatility of attacks/defenses customization and FL algorithms, as it only supports traditional distributed gradient aggregation in `FedSGD`. In contrast, FedMLSecurity predominantly centers on the `FedAvg`-family optimizers but overlooks gradient aggregation techniques. Although FedMLSecurity is compatible with various FL optimizers (*e.g.,* `FedProx` [49] and `FedNOVA` [50]), we believe that `FedSGD` and `FedAvg` represent the foundational algorithms for benchmarking Byzantine-resilient FL, given the substantial volume of attacks and defenses constructed around them. Furthermore, AggregaThor and FedMLSecurity lack user-friendly mechanisms for hyperparameter tuning, resulting in great engineering efforts when benchmarking across varied configurations. In response to these gaps, we have designed and developed a novel benchmark suite, Blades, that addresses the aforementioned limitations and offers a more comprehensive evaluation for Byzantine-resilient FL.

III. The Design of Blades

In this section, we first outline our design goals for Blades. Following that, we delve into the structure of Blades, elucidating its layered architecture and key components. Finally, we demonstrate the implementation of both attacks and defenses with Blades.

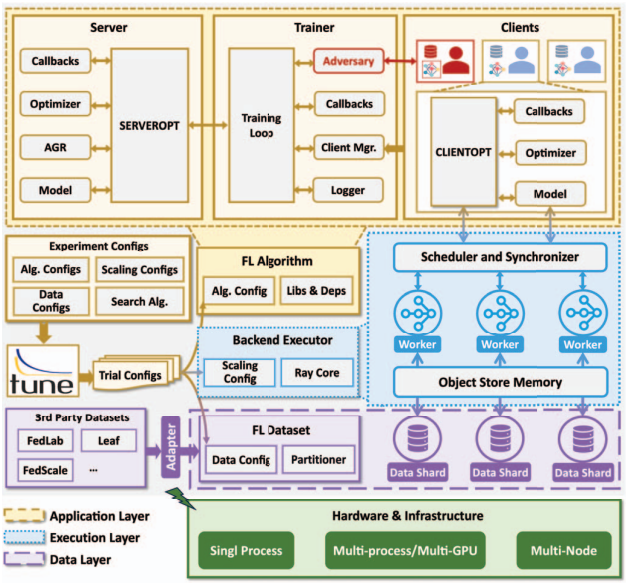

Fig. 1: Overview of the Blades architecture comprising the Application, Execution, and Data layers.

A. Design Goals

We designed and implemented Blades with the following goals in mind:

Usability: The benchmark suite should enable researchers to effortlessly set up experiments, search, and tune hyperparameters. To enable efficient comparisons, it should include implementations of the most representative algorithms (e.g., `FedSGD` and `FedAvg`) with attack and defense strategies.

Extensibility: As new attack and defense strategies are discovered, it should be relatively straightforward to incorporate them into the suite. Additionally, the design should readily accommodate the integration of new datasets, models, and FL algorithms.

Scalability: The suite should exhibit scalability in terms of both clients and computing resources. Scalability with clients refers to the ability to accommodate a large and diverse set of clients participating in the learning process. Scalability with computing resources entails efficiently utilizing and adapting to different hardware setups.

B. Core Framework Architecture

In pursuit of our design goals, particularly extensibility and scalability, we structured the system into three distinct layers: the Application layer, Execution layer, and Data layer. An overview of the architecture of Blades is illustrated in Fig. 1. This tripartite architecture is rooted in our intention to delineate and modularize specific objectives. The Application layer emphasizes extensibility, allowing for straightforward

```yaml
stop: training_round: 2000 # Communication rounds
config:
    global_model: ResNet10
    num_malicious_clients: grid_search: [0, 5]
    # Configuring AGR and optimizer for Server
    server_config:
      AGR: grid_search: [Mean, Median]
      optimizer:           # SERVEROPT
        type: SGD
        lr_schedule: [[0, 0.1], [1500, 0.1]]
    # Specify adversarial attack and parameters
    adversary_config:
      grid_search:
        - type: LabelFlipAdversary
        - type: IPMAdversary
            alpha: grid_search: [0.1, 100]
```

Fig. 2: An example YAML configuration snippet for simulating LabelFlipping and IPM attacks with various hyperparameter settings. Blades is fully configurable and allows grid search for hyperparameter settings and experiment specifications.

configuration and integration of a range of FL-related functionalities and features. The Execution layer is tasked with ensuring system scalability, and adeptly accommodating extensive workloads. Finally, the Data layer streamlines data ingestion and preprocessing in the distributed setting.

1) Application Layer: It is the top layer of Blades and provides a user-friendly interface for designing and deploying FL algorithms. The main abstractions in this layer include:

Server: A server is an object that aggregates model updates from multiple clients and performs global optimization. Once a local training round is finished, it gathers the model updates and takes one iteration step. Defense strategies, such as AGRs, are usually applied here to eliminate the impact of malicious updates.

Trainer: The trainer encapsulates the optimization process for a particular FL algorithm. A trainer manages key aspects of the training loop, including interactions with the server, client manager, and state synchronization between the server and clients. Each trainer corresponds to a specific FL algorithm, such as `FedAvg`, and can be configured with various hyperparameters to control the local training process. It also allows customization with callbacks invoked at specific points during training, such as after each local training round or server optimization step. The "Adversary" component in the Trainer can control a subset of clients to perform malicious operations.

Client: The client acts as a participant in the FL process. We provide the client-oriented programming design pattern [51] to program the local optimization of clients during their involvement in training or coordination within the FL algorithm. This pattern allows end users to specify and execute certain types of attacks easily. Additionally, the interface we offer allows users to tailor the behaviors of Byzantine clients.

The application layer has several dependencies that provide a variety of functionalities. Particularly, we adopt the Tune library[1] [52] for experiment configuration and hyperparameter tuning at any scale. As an example, Fig. 2 shows a configuration file for simulating the LabelFlipping [9] and IPM [22] attacks with different hyperparameter settings. With the help of Tune, Blades reads the file and parses the configurations to generate a series of experimental trials. The trials are then scheduled to execute on the execution layer. This functionality facilitates the creation of multiple combinations of hyperparameters and configurations found within the grid. Specifically, in the given example, the grid searching areas of configuration include `num_malicious_clients`", `AGR`", "`adversary_config`", and "`alpha`". All considered, this culminates in a total of 12 trials generated simultaneously.

2) Execution Layer: The execution backend is built upon a scalable framework Ray [53] for training and resource allocation. Ray provides two key advantages for Blades: 1) It allows users to customize computing resources (e.g., CPUs and GPUs) to clients and servers conveniently; 2) It enables Blades to easily adapt to in-cluster large-scale distributed training, benefiting from the capabilities of the Ray cluster. The key components in the execution layer are:

Worker: A worker is a proxy that can be allocated with computing resources to execute the training pipeline for clients. Inherited from Ray Actor[2], the worker (essentially a Python process) is the smallest unit for resource management and distributed computing. In a distributed environment, multiple workers collaborate as a group. In case of a large number of clients and limited resources, multiple clients are mapped to one worker, and their local training tasks will be scheduled to run sequentially. This property enables a much larger scale of experiments on common hardware. Moreover, to maximally utilize resources, actors can request fractional GPUs so that multiple actors can be co-located to the same GPU [43]. Such a GPU-sharing technique accelerates FL optimization for lightweight models.

Scheduler and Synchronizer: They facilitate a lightweight and efficient mechanism for model parameter synchronization, worker group coordination, and task scheduling within a distributed training environment. They retrieve the training pipelines submitted by clients from a task queue, allocate them to suitable workers, and synchronize model parameters as required.

Object Store Memory: Following the distributed memory management of Ray, Blades allows storing and caching data objects during distributed computations using object store memory. Remote objects are cached in Ray's distributed shared-memory object store, and there is one object store per node in the cluster for easy access.

By decoupling the execution layer from the application layer, clients remain unaware of the specific implementation details of the backend. As a result, users can concentrate solely

[1] https://docs.ray.io/en/latest/tune
[2] https://docs.ray.io/en/latest/ray-core/actors.html

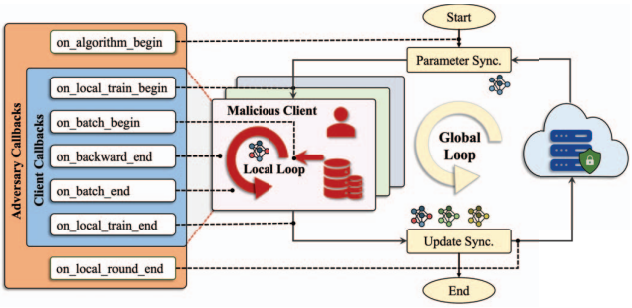

Fig. 3: The pipeline for attack implementation in Blades. We define some time points when users can register executable methods to perform customized attacks. At specific time points, client callbacks and adversary callbacks are triggered to invoke the registered methods.

on the application layer for implementing FL algorithms and submitting client training pipelines to the backend executor.

3) Data Layer: The data layer facilitates data pre-processing and loading for distributed training with the execution backend. It supports both IID and non-IID splitting for studying homogeneous and heterogeneous scenarios, respectively. At the beginning of the training, the dataset is separated into multiple shards and pre-allocated to workers' memory to allow fast data loading. In addition, we provide an adapter to import datasets from well-known 3rd-party benchmarks in FL, such as Leaf [54], FedScale [43], and FedLab [55].

C. Implementation of Attacks and Defenses

1) Implementing Attacks: We note that adversarial attackers may perform some self-defined manipulation before or after specific time points. For instance, LabelFlipping [18] attacks are typically executed at the beginning of batch forward propagation, while SignFlipping [30] attacks are carried out immediately after backpropagation. To address this, we have designed a unified pipeline integrated with a callback mechanism, which enables actions to be performed at various stages of training, as depicted in Fig. 3. This design offers extensibility to facilitate customization, where the minimal pipeline focuses on repetitive local training and server-side optimization while the malicious behaviors are defined through the callback mechanism. Users only need to override specific callback methods to execute a custom attack without modifying the pre-defined logic.

Blades incorporates a dual-tiered callback mechanism: at the client and adversary levels. Elementary attacks (*e.g.,* LabelFlipping and SignFlipping), operate solely on local data or model parameters. Therefore, their implementation is streamlined by registering specific behaviors to client objects, enabling concurrent execution across multiple workers. As an illustrative instance, the upper panel of Fig. 4 shows a code snippet that exemplifies the implementation of a LabelFlipping attack on a classification task encompassing 10 distinct classes. Through a straightforward customization of the "`on_batch_begin()`" callback method, users can easily modify the labels from class i to $9 - i$ during local training.

```python
from blades.clients import ClientCallback
class LabelFlipCallback(ClientCallback):
    def on_batch_begin(self, data, target):
        # Return input data with flipped labels for the current batch (assuming 10 labels).
        return data, 9 - target
```

```python
from blades.adversary import AdversaryCallback
class ALIECallback(AdversaryCallback):
    def on_local_round_end(self, trainer):
        # Compute the dimensional mean and std over client updates, then generate malicious updates by adding std to the mean.
        updates = trainer.get_updates()
        mean = updates.mean(dim=0)
        std = updates.std(dim=0)
        updates = mean + std
        trainer.save_malicious_updates(updates)
```

Fig. 4: Illustration of our callback mechanism used to simulate LabelFlipping (upper) and ALIE (lower) attacks. The design's flexibility enables easy customization by overriding methods associated with both client and adversary callbacks.

For more sophisticated attacks such as colluding attacks, clients in the distributed environment need to exchange data, coordinate actions, and synchronize their activities to make decisions regarding malicious actions. A straightforward solution to simulate such attacks is allowing inter-client communication using the remote function mechanism in Ray. However, this will make the simulation more complex and limit the scalability of the system. One possible consequence is the occurrence of deadlocks, *i.e.,* when two or more clients are waiting for each other to release a resource or respond to a communication, none of them can proceed.

Alternatively, we facilitate the implementation of sophisticated attacks by incorporating additional callbacks tailored for adversary entities. Distinguished from client callbacks, these adversary callbacks are executed within the driver program and allow convenient access to various system components and their states. As a result, this design simplifies the process of acquiring knowledge for high-level attacks. Fig. 3 also shows two of the most essential adversary callbacks, specifically "`on_algorithm_begin()`" and "`on_local_round_end()`". The former is triggered at the start of the algorithm and serves the purpose of initializing the adversary and setting up client callbacks. The latter is triggered upon the completion of a local round and allows for the modification of collected updates from malicious clients before proceeding to server-side optimization and defense operations. During the "`on_local_round_end()`" callback, one can potentially access honest updates and other system states in a read-only manner to launch omniscient attacks. The lower panel of Fig. 4 shows an example of implementing the ALIE [22] attack using this adversary callback.

2) Implementing Defenses: Defenses in the context of our study stem from multiple facets, posing challenges to the establishment of a standardized pipeline akin to the one employed for attacks. Nevertheless, certain indispensable steps are involved in this process, namely update aggregation and global model optimization, although the specific methodology for each step may vary. The update aggregation step combines the locally collected updates, while the global model optimization step performs an optimization procedure based on the aggregated result.

As such, Blades introduces a foundational abstraction of the server entity, encompassing essential components including global model, AGR, and optimizer. This architecture permits the extension of functionalities through the utilization of sub-classing, thereby facilitating the integration of advanced features. It is noteworthy to mention that even in the minimal server implementation, we offer a configurable SGD optimizer and a variety of pre-defined AGRs. Furthermore, all the components are modularized and inheritable, allowing plug-and-play of different configurations. We believe our designs simplify the process of generating benchmark results with minimal effort.

IV. Experimental Evaluations

In this section, we offer a comprehensive reassessment of six representative built-in AGRs, evaluating their performance against six attacks across three datasets with various settings and incorporating other relevant techniques. The results reveal novel insights in the area. Additionally, we test the scalability of Blades, with an emphasis on its adaptability to rising client numbers and computational demands.

A. Datasets and Model Architectures

UCI-HAR [56] is an IoT dataset comprising observations from 30 volunteers carrying a waist-mounted smartphone and performing six distinct activities: walking, walking upstairs, walking downstairs, sitting, standing, and lying down. Each record is represented by 561 features, extracted from the time and frequency domain. With a total of 10,299 instances, the dataset is naturally non-IID with data distributed across 30 clients. For this dataset, we utilize a CNN architecture, which includes two convolutional layers with 32 and 64 channels, respectively, both followed by ReLU activations and max-pooling operations. The network subsequently channels features through three fully connected layers with dimensions of 1024, 512, and the final output corresponding to the six classes.

Fashion MNIST [57] consists of 50,000 gray-scale training samples and 10,000 test samples. It encompasses 10 different categories of clothing items, such as shoes, T-shirts, and dresses. The images in Fashion MNIST are of size 28×28. For this dataset, we employ a CNN with two convolutional layers. The first layer has 32 channels with a 3x3 kernel, and the second has 64 channels. Following the convolutional operations, the features are passed through three fully connected layers with 600, 120, and 10 neurons respectively.

CIFAR10 [58] contains 50,000 color training samples and 10,000 test samples. It comprises color images of various objects classified into 10 categories, including airplanes and automobiles. The images in CIFAR10 have dimensions of $32 \times 32 \times 3$. For CIFAR10, we employ a modified ResNet architecture [59], termed as ResNet10, which is shallower with only 10 layers. This design makes the model more suitable for resource-constrained IoT configurations while not significantly decreasing performance.

B. Learning and Attack Settings

We split Fashion MNIST and CIFAR10 into 60 distinct subsets and allocate them to 60 clients, utilizing both IID (independently and identically distributed) and non-IID strategies. For the IID approach, we assume homogeneity in data points, with each subset representing a random sampling of the entire dataset, ensuring statistical consistency. For the non-IID partition, we follow prior work [9], [55] and model the non-IID data distributions with a Dirichlet distribution $\boldsymbol{p}_l \sim Dir_K(\alpha)$, in which a smaller α indicates a stronger divergence from IID. Then we allocate a $\boldsymbol{p}_{l,k}$ proportion of the training samples of class l to client k.

In our experiments, the models are trained for 2000 communication rounds using `FedSGD` and 400 rounds for `FedAvg`. By default, the batch size is set to 64. For `FedSGD`, we adopt the learning rates $\eta_l = 1.0$ and $\eta_g = 0.1$, with the latter undergoing a decay to 0.01 commencing from the 1501st round. Conversely, for `FedAvg`, we specify 20 local steps per round with parameters set as $\eta_l = 0.1$, and $\eta_g = 1.0$.

Additionally, in our assessment, we evaluate the six attacks detailed in Section II-B. To rigorously stress-test the defense techniques, we introduce malicious clients, varying their proportion from 0% to 40% in the simulations.

C. Comparison of AGRs under various settings

For the sake of generality, we first examine selected AGRs with the standard `FedSGD` and `FedAvg`. Fig. 5 and Fig. 6 depict the overall comparisons of different AGRs with respect to test accuracy under seven attack configurations. *Overall, traditional and naive AGRs (i.e., Median [16], TrimmedMean [16], and GeoMed [36]) exhibit significant vulnerabilities to several attacks, whereas hybrid strategies (i.e., DnC [29], ClippedClustering [9], and SignGuard [37]) that integrate multiple techniques exhibit superior resilience against various attacks*. This aligns with previous studies [9], [29], which suggest that traditional AGRs, dependent on either dimensional-level filtering or optimization rooted in Euclidean distance, fall short in countering sophisticated attacks. However, our experimental evaluations have unveiled further insights that enrich our comprehension of Byzantine-resilient FL. In what follows, we elucidate these findings and present the crucial pieces of evidence supporting them.

(Finding 1) *The effectiveness of adversarial attacks is contingent upon a confluence of factors, including the choice of dataset, defensive countermeasure, and the specific FL algorithm employed.*

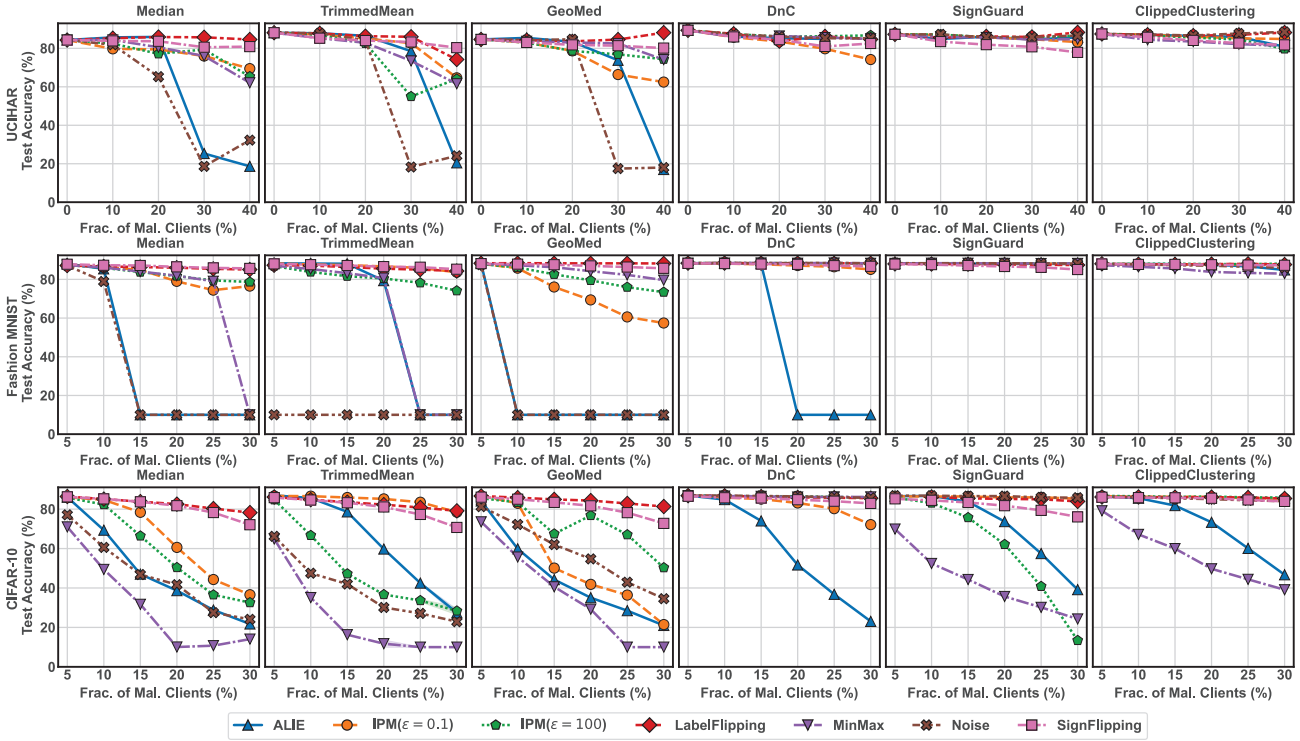

Fig. 5: Comparing AGRs under various attacks with IID partition. Simple attacks like LabelFlipping and SignFlipping are ineffective against most of the AGRs, while more advanced attacks such as ALIE and MinMax result in significant performance degradation across most settings, particularly as the fraction of malicious clients increases. Noticeably, traditional AGRs (i.e., Median [16], TrimmedMean [16], and GeoMed [36]) exhibit vulnerabilities to several attacks, whereas advanced AGRs display greater resilience.

- When examining the performance on the UCI-HAR and Fashion MNIST datasets, it becomes evident that the majority of AGRs retain a superior accuracy compared to their performance on CIFAR10. A plausible explanation for this observed distinction lies in the inherent characteristics of the datasets. Both UCI-HAR and Fashion MNIST, in comparison to CIFAR10, are considered to have relatively lower levels of complexity. As such, model updates trained on UCI-HAR and Fashion MNIST might face fewer challenges in diversity, leading to better performance under AGRs.
- The effectiveness of attacks varies significantly based on the AGR employed. As an example, for CIFAR10+`FedSGD`, Median, TrimmedMean, GeoMed strategies show gradual declines in test accuracy as malicious clients increase, with MinMax and ALIE attacks being particularly effective. However, DnC remains resilient against MinMax while still showing its vulnerability to ALIE. We note that both ALIE and MinMax leverage the variance of benign updates. However, MinMax typically amplifies the magnitudes to larger values, thereby presenting a mixed set of advantages and disadvantages. On the one hand, larger magnitudes have the potential to push the global model further away if they are not filtered out by the AGRs. On the other hand, these amplified magnitudes are more easily detectable by certain defense mechanisms, such as DnC in our experiments.
- The SignFlipping attack, while exhibiting limited effectiveness under `FedSGD`, leads to substantial performance degradation across various settings when integrated with `FedAvg`. Taking Fashion MNIST as an example, when combined with `FedSGD`, SignFlipping makes little impact on all defenses, even with as many as 30% malicious clients present. However, when associated with `FedAvg`, it disrupts four out of the six AGRs with just 15% malicious clients in the mix. In contrast, both ALIE and MinMax show significant impacts with `FedSGD` but underperform when integrated with `FedAvg`.

(Finding 2) *The robustness of defenses in existing studies may be overrated owing to their insufficiency in comprehensive evaluation under wide-ranging settings.*

- We notice that both DnC and SignGuard, which claimed robustness against several attacks with up to 20% malicious clients for IID datasets, were evaluated with `FedSGD` in their original works. However, the potential vulnerabilities were not detected when applied under the well-known algorithm `FedAvg`. Fig. 6 show that they both become more vulnerable to SignFlipping while utilizing multiple (*e.g.,* $\eta_l = 20$) steps for local updates. Surprisingly, DnC cannot sustain as little as 5% of malicious SignFlipping attack clients when on Fashion MNIST under `FedAvg`.
- In our experiments, SignGuard effectively counters original

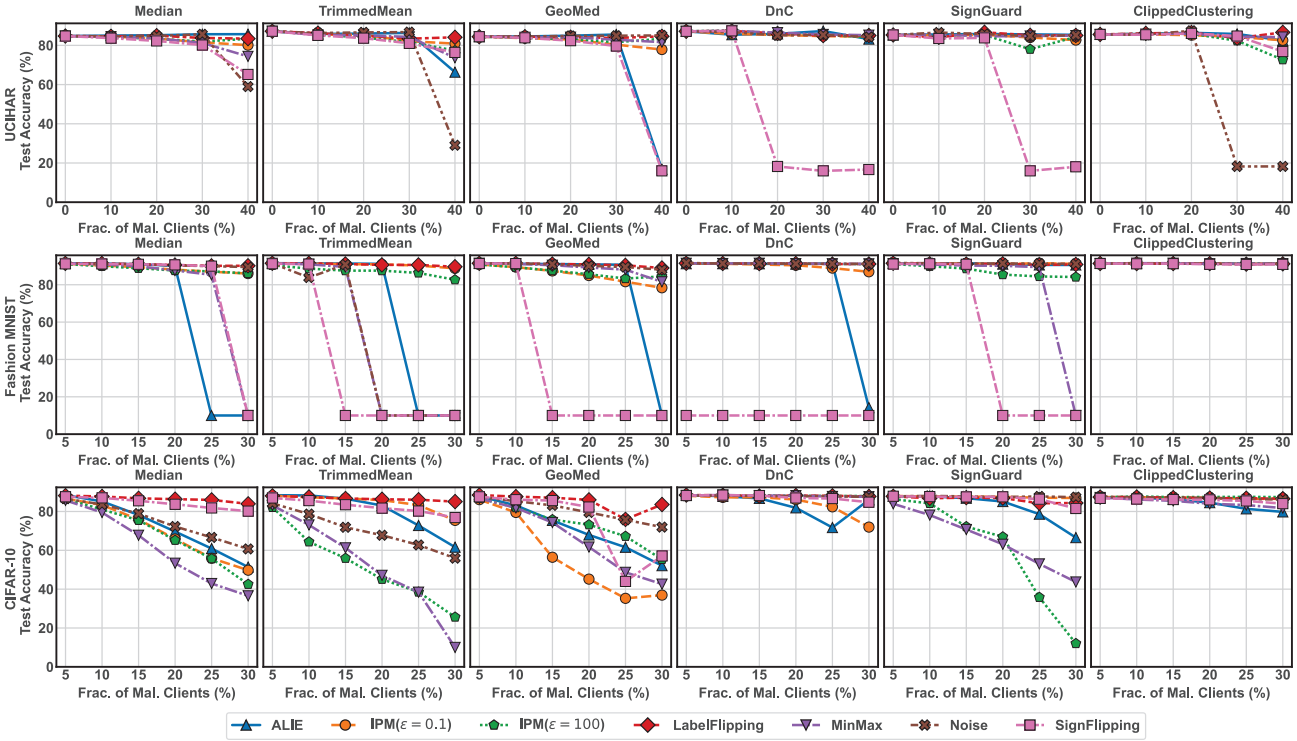

Fig. 6: Evaluation of AGRs while utilizing 20 steps for local updates (i.e., $E_l = 20$). ClippedClustering exhibits effective defense against most examined attacks with minimal performance loss, while others display distinct vulnerabilities depending on the attack type.

ALIE and MinMax attacks by capitalizing on their distinct sign statistics. However, when SignGuard is subjected to a more intensive examination in which we invert half of the signs—while maintaining the sign statistics (*i.e.,* the ratio of positive signs)—before computing the standard deviation for malicious updates, its robustness becomes compromised, making it susceptible to attacker bypass.

- We also evaluated two recent techniques, CC [18] and Bucketing [60] in our work. However, neither of these techniques could match the performance of other examined AGRs under similar experimental settings. We omitted the detailed results due to space limitations.

D. Beyond AGRs: Additional Factors

In addition to AGRs, we further examine a series of factors that may affect the robustness under adversarial attacks.

(Finding 3) *Additionally, various factors, including data heterogeneity, differential privacy (DP) noise, and momentum, exert considerable influence on the Byzantine resilience of defense strategies.*

1) Impact of the degree of non-IID in datasets: Fig. 7 illustrates the test accuracy results of the ALIE attack on CIFAR10, varying the levels of non-IID partitioning. The figure shows that the effectiveness of the attack increases significantly when the dataset is highly non-IID (e.g., $\alpha = 0.1$). This is consistent with prior studies [9], [14], [29]. A commonly

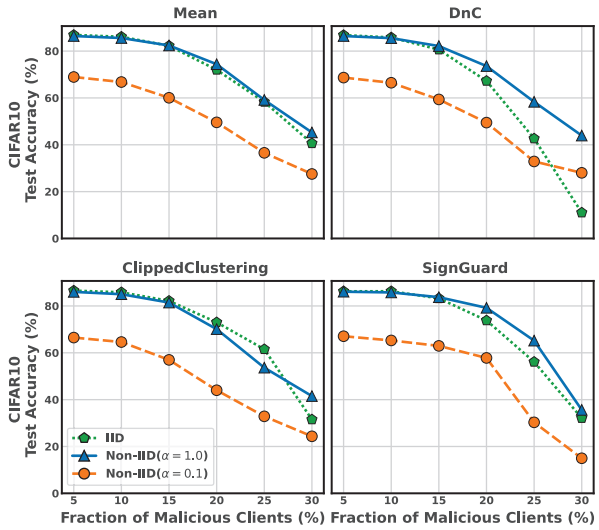

Fig. 7: Impact of various degrees of non-IID on the robustness of AGRs against ALIE attack. A lower α value indicates a higher degree of non-IID.

proposed explanation is that as the local data distributions become significantly different, the model updates diversify, thereby posing an additional challenge for AGRs to perform a proper aggregation.

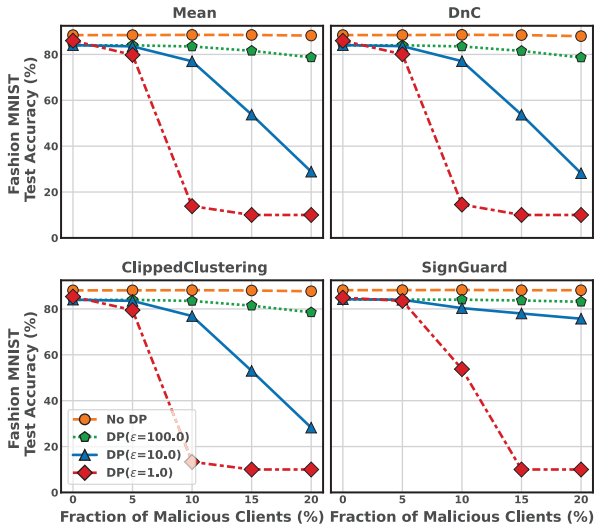

Fig. 8: Impact of various DP levels on the robustness of AGRs against ALIE attack. A lower ϵ value indicates a lower budget for DP. All AGRs become more vulnerable to adversarial attacks with a lower privacy budget.

2) Impact of noise-adding for DP: Next, we assess the influence of introducing Gaussian noise for DP on the resilience of defenses utilizing Fashion MNIST. Fig. 8 illustrates that as the privacy budget diminishes, the test accuracy across all AGRs declines more rapidly as the number of malicious clients increases. With a high budget (e.g., $\epsilon = 100.0$), all AGRs result in comparable accuracy levels to those without the incorporation of DP noise. In contrast, when the privacy parameter ϵ equals 1.0, the AGRs yield extremely low accuracy (nearly the same as random guessing) when 10% of the clients are malicious.

3) Impact of Momentum: Momentum is considered a supplementary technique aiming at bolstering the robustness of Byzantine-resilient FL. To evaluate its effectiveness, we conduct experiments by training a ResNet10 on CIFAR10 under the ALIE attack. The obtained results are presented in Table II. Interestingly, the integration of server momentum demonstrates minimal enhancement in the robustness of the AGRs, particularly in situations involving a limited ratio (i.e., 10%) of malicious clients, while leading to lower accuracy in other situations. In contrast, the AGRs consistently exhibit significant improvements when client momentum is employed.

4) The Risk of Gradient Explosion: Another interesting observation in Fig. 5 is that four AGRs fail to handle Noise attacks on Fashion MNIST even when only 10% of the clients are malicious. Upon closer examination of the gradients and loss values, we observe significant gradient explosions among the benign clients, which prevent the model from converging to the optimal solution. We believe that the decline in performance is attributed not only to substantial deviations but also to the potential detrimental impact of gradient explosions caused by malicious attacks.

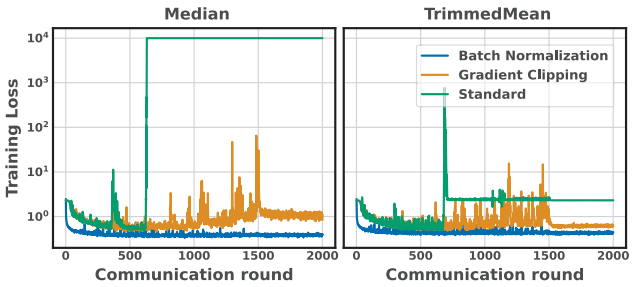

Fig. 9: Training loss on Fashion MNIST with noise attacks where 20% of clients are malicious. The loss values are clamped to $[0, 10^4]$. The attacks lead to gradient explosions, which are mitigated by gradient clipping and batch normalization.

To validate our findings, we employ two measures to mitigate gradient explosions, i.e., gradient clipping and batch normalization, and show the comparison in Fig. 9. When no measures are taken, the training loss shows sudden spikes and ends up with a large value, indicating that the model cannot converge to an appropriate solution. With gradient clipping and batch normalization, the training process becomes more stable. Batch normalization not only prevents gradient explosions but also accelerates the training process, facilitating faster convergence.

E. Scalability Evaluation

We evaluate the scalability of Blades along two lines: First, we evaluate the training time for a global round with an increasing number of clients and computing resources. Second, we evaluate the training time per round with an increasing number of GPUs.

To assess the scalability with respect to the number of clients, we conduct simulations with a fixed set of resources and vary the number of clients. As shown in Table III, the number of clients increases from 16 to 512, which greatly outnumbers the available CPUs and GPUs. The results show that with a linear increase in the number of clients, the average training time of each global round increases linearly with little standard deviation, indicating the stable and efficient communication and task distribution implementation in Blades.

To evaluate the resource scalability of Blades, we design a simulation task involving 480 clients to be executed on GPUs. Fig. 10 shows the time cost of each global round with different numbers of GPUs and the associated speedups. The time cost for each global round reduces from 82.5s to 49.5s when the number of GPUs increases from 1 to 2, and the time cost reduces more if more GPUs are added. The speedup achieved does not align precisely with the number of GPUs utilized, primarily because FL is not an entirely parallelizable algorithm. Consequently, a substantial portion of serial work, such as the communication of local updates and model aggregation, hinders the potential speedup. Nonetheless, the increase in computing resources significantly reduces the time cost of the simulation, and Blades is scalable with computing resources.

Table II: Test accuracy of CIFAR10 on different momentum settings. The numbers with the highest accuracy are in bold. Client momentum significantly improves the robustness.

AGR	10% malicious clients			15% malicious clients			20% malicious clients		
	No Momentum	Server Momentum	Client Momentum	No Momentum	Server Momentum	Client Momentum	No Momentum	Server Momentum	Client Momentum
Mean [1]	86.05 (0.20)	88.46 (0.24)	**89.51 (0.22)**	82.89 (0.46)	80.72 (0.53)	**88.18 (0.34)**	73.50 (0.91)	59.92 (1.36)	**85.88 (0.22)**
DnC [29]	85.84 (0.19)	87.92 (0.45)	**89.40 (0.15)**	81.45 (0.29)	76.58 (0.08)	**87.89 (0.45)**	68.63 (0.40)	50.04 (3.88)	**83.60 (0.40)**
Median [16]	69.38 (0.58)	48.39 (2.41)	**83.95 (0.31)**	45.95 (3.05)	27.52 (3.97)	**68.20 (0.63)**	27.85 (10.7)	14.29 (6.23)	**52.22 (3.73)**
TrimmedMean [16]	85.98 (0.29)	87.81 (0.14)	**89.57 (0.18)**	79.98 (0.47)	72.40 (0.49)	**87.47 (0.12)**	66.69 (0.47)	41.32 (4.75)	**81.72 (0.44)**
ClippedClustering [9]	85.76 (0.16)	87.86 (0.19)	**89.47 (0.20)**	82.25 (0.25)	73.23 (1.23)	**87.94 (0.36)**	73.19 (0.98)	41.52 (5.63)	**83.06 (0.62)**

Table III: Experiment for client scalability: average and standard deviation of training time per global round with increasing numbers of clients.

# Clients	16	32	64	128	256	512
Avg (seconds)	1.41	2.48	4.51	9.10	17.58	34.53
Std (seconds)	0.05	0.07	0.11	0.18	0.38	0.73

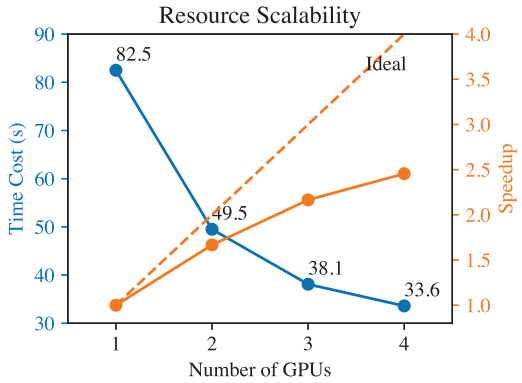

Fig. 10: Resource scalability of Blades. Blue: Average time cost per global round decreases with the number of GPUs. Orange: The speedup increases with the number of GPUs.

V. Conclusion and Future Work

In this paper, we introduced an open-source benchmark suite, namely Blades, to address the research gap concerning attack and defense problems in FL. The Blades framework offers usability, extensibility, and scalability that facilitate the implementation and expansion of novel attack and defense techniques. By utilizing Blades, we conducted a comprehensive evaluation of prominent defense techniques against state-of-the-art attacks, yielding insightful findings regarding the resilience of Byzantine-resilient FL. Furthermore, we showed that Blades is scalable in terms of both the number of clients and computing resources. In the future, we will continue our efforts to address more security threats with our new releases of Blades, and integrate more cutting-edge methods for comprehensive benchmarking.

Acknowledgments

This research was supported by the RGC General Research Funds No. 17203320 and No. 17209822 from Hong Kong, the Swedish Research Council project grant No. 2017-04543, and HKU-TCL joint research centre for artificial intelligence seed funding.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017.

[2] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[3] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for internet of things: Recent advances, taxonomy, and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1759–1799, 2021.

[4] S. Li, E. Ngai, and T. Voigt, "Byzantine-robust aggregation in federated learning empowered industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1165–1175, 2023.

[5] Y. Arjevani and O. Shamir, "Communication complexity of distributed convex learning and optimization," *Advances in neural information processing systems*, vol. 28, 2015.

[6] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[7] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," *arXiv preprint arXiv:2003.02133*, 2020.

[8] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara, "Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges," *Information Fusion*, vol. 90, pp. 148–173, 2023.

[9] S. Li, E. C.-H. Ngai, and T. Voigt, "An experimental study of byzantine-robust aggregation schemes in federated learning," *IEEE Transactions on Big Data*, 2023.

[10] M. S. Jere, T. Farnan, and F. Koushanfar, "A taxonomy of attacks on federated learning," *IEEE Security & Privacy*, vol. 19, no. 2, 2020.

[11] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *International Conference on Learning Representations*, 2020.

[12] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2938–2948.

[13] S. Andreina, G. A. Marson, H. Möllering, and G. Karame, "Baffle: Backdoor detection via feedback-based federated learning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 852–863.

[14] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1605–1622.

[15] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[16] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*. PMLR, 2018.

[17] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2017.

[18] S. P. Karimireddy, L. He, and M. Jaggi, "Learning from history for byzantine robust optimization," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021.

[19] F. Sattler, K.-R. Müller, T. Wiegand, and W. Samek, "On the byzantine robustness of clustered federated learning," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8861–8865.

[20] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1354–1371.

[21] S. Hu, J. Lu, W. Wan, and L. Y. Zhang, "Challenges and approaches for mitigating byzantine attacks in federated learning," *arXiv preprint arXiv:2112.14468*, 2021.

[22] G. Baruch, M. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[23] C. Xie, O. Koyejo, and I. Gupta, "Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 261–270.

[24] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[25] M. A. Khan, V. Shejwalkar, A. Houmansadr, and F. M. Anwar, "On the pitfalls of security evaluation of robust federated learning," in *2023 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2023, pp. 57–68.

[26] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=LkFG3lB13U5

[27] L. Ju, T. Zhang, S. Toor, and A. Hellander, "Accelerating fair federated learning: Adaptive federated adam," *arXiv preprint arXiv:2301.09357*, 2023.

[28] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in *International Conference on Learning Representations*, 2020.

[29] V. Shejwalkar and A. Houmansadr, "Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning," in *NDSS*, 2021.

[30] L. Li, W. Xu, T. Chen, G. B. Giannakis, and Q. Ling, "Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 1544–1551.

[31] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *ISOC Network and Distributed System Security Symposium (NDSS)*, 2021.

[32] C. Xu, Y. Jia, L. Zhu, C. Zhang, G. Jin, and K. Sharif, "Tdfl: Truth discovery based byzantine robust federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, 2022.

[33] J. Park, D.-J. Han, M. Choi, and J. Moon, "Sageflow: Robust federated learning against both stragglers and adversaries," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 840–851.

[34] E. Gorbunov, S. Horváth, P. Richtárik, and G. Gidel, "Variance reduction is an antidote to byzantines: Better rates, weaker assumptions and communication compression as a cherry on the top," in *International Conference on Learning Representations*, 2023.

[35] Z. Wu, Q. Ling, T. Chen, and G. B. Giannakis, "Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4583–4596, 2020.

[36] K. Pillutla, S. M. Kakade, and Z. Harchaoui, "Robust aggregation for federated learning," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1142–1154, 2022.

[37] J. Xu, S.-L. Huang, L. Song, and T. Lan, "Byzantine-robust federated learning through collaborative malicious gradient filtering," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 1223–1235.

[38] D. Alistarh, Z. Allen-Zhu, and J. Li, "Byzantine stochastic gradient descent," *Advances in Neural Information Processing Systems*, 2018.

[39] X. Yin, Y. Zhu, and J. Hu, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[40] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signsgd with majority vote is communication efficient and fault tolerant," *arXiv preprint arXiv:1810.05291*, 2018.

[41] G. Damaskinos, E.-M. El-Mhamdi, R. Guerraoui, A. Guirguis, and S. Rouault, "Aggregathor: Byzantine machine learning via robust gradient aggregation," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 81–106, 2019.

[42] S. Han, B. Buyukates, Z. Hu, H. Jin, W. Jin, L. Sun, X. Wang, C. Xie, K. Zhang *et al.*, "Fedmlsecurity: A benchmark for attacks and defenses in federated learning and llms," *arXiv preprint arXiv:2306.04959*, 2023.

[43] F. Lai, Y. Dai, S. Singapuram, J. Liu, X. Zhu, H. Madhyastha, and M. Chowdhury, "Fedscale: Benchmarking model and system performance of federated learning at scale," in *International Conference on Machine Learning*. PMLR, 2022, pp. 11 814–11 827.

[44] L. Yao, D. Gao, Z. Wang, Y. Xie, W. Kuang, D. Chen, H. Wang, C. Dong, B. Ding, and Y. Li, "A benchmark for federated hetero-task learning," *arXiv preprint arXiv*, vol. 2206, 2022.

[45] D. Chen, D. Gao, W. Kuang, Y. Li, and B. Ding, "pfl-bench: A comprehensive benchmark for personalized federated learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 9344–9360, 2022.

[46] V. Mugunthan, A. Peraire-Bueno, and L. Kagal, "Privacyfl: A simulator for privacy-preserving and secure federated learning," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3085–3092.

[47] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose *et al.*, "Pysyft: A library for easy federated learning," in *Federated Learning Systems*. Springer, 2021, pp. 111–139.

[48] Z. Qin, L. Yao, D. Chen, Y. Li, B. Ding, and M. Cheng, "Revisiting personalized federated learning: Robustness against backdoor attacks," *arXiv preprint arXiv:2302.01677*, 2023.

[49] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[50] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, 2020.

[51] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu *et al.*, "Fedml: A research library and benchmark for federated machine learning," *arXiv preprint arXiv:2007.13518*, 2020.

[52] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[53] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *et al.*, "Ray: A distributed framework for emerging {AI} applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018.

[54] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[55] D. Zeng, S. Liang, X. Hu, H. Wang, and Z. Xu, "Fedlab: A flexible federated learning framework," *Journal of Machine Learning Research*, vol. 24, no. 100, pp. 1–7, 2023.

[56] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. Reyes-Ortiz *et al.*, "A public domain dataset for human activity recognition using smartphones," in *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. CIACO, 2013.

[57] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[58] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[60] S. P. Karimireddy, L. He, and M. Jaggi, "Byzantine-robust learning on heterogeneous datasets via bucketing," in *International Conference on Learning Representations*, 2022.