Automatic Adversarial Adaption for Stealthy Poisoning Attacks in Federated Learning

Torsten Krauß
University of Würzburg
torsten.krauss@uni-wuerzburg.de

Jan König
University of Würzburg
jan.koenig@stud-mail.uni-wuerzburg.de

Alexandra Dmitrienko
University of Würzburg
alexandra.dmitrienko@uni-wuerzburg.de

Christian Kanzow
University of Würzburg
kanzow@mathematik.uni-wuerzburg.de

***Abstract*—Federated Learning (FL) enables the training of machine learning models using distributed data. This approach offers benefits such as improved data privacy, reduced communication costs, and enhanced model performance through increased data diversity. However, FL systems are vulnerable to poisoning attacks, where adversaries introduce malicious updates to compromise the integrity of the aggregated model. Existing defense strategies against such attacks include filtering, influence reduction, and robust aggregation techniques. Filtering approaches have the advantage of not reducing classification accuracy, but face the challenge of adversaries adapting to the defense mechanisms. The lack of a universally accepted definition of "adaptive adversaries" in the literature complicates the assessment of detection capabilities and meaningful comparisons of FL defenses. In this paper, we address the limitations of the commonly used definition of "adaptive attackers" proposed by Bagdasaryan et al. We propose AutoAdapt, a novel adaptation method that leverages an Augmented Lagrangian optimization technique. AutoAdapt eliminates the manual search for optimal hyper-parameters by providing a more rational alternative. It generates more effective solutions by accommodating multiple inequality constraints, allowing adaptation to valid value ranges within the defensive metrics. Our proposed method significantly enhances adversaries' capabilities and accelerates research in developing attacks and defenses. By accommodating multiple valid range constraints and adapting to diverse defense metrics, AutoAdapt challenges defenses relying on multiple metrics and expands the range of potential adversarial behaviors. Through comprehensive studies, we demonstrate the effectiveness of AutoAdapt in simultaneously adapting to multiple constraints and showcasing its power by accelerating the performance of tests by a factor of 15. Furthermore, we establish the versatility of AutoAdapt across various application scenarios, encompassing datasets, model architectures, and hyper-parameters, emphasizing its practical utility in real-world contexts. Overall, our contributions advance the evaluation of FL defenses and drive progress in this field.**

I. Introduction

Federated Learning (FL) facilitates the collaborative training of Deep Neural Networks (DNN) across multiple clients [46]. Each client independently trains a DNN using its own data, effectively incorporating data knowledge into model parameters. Notably, only the changes in trained model parameters are communicated to a central server for aggregation. This distinctive approach empowers clients to actively engage in the federation while upholding stringent privacy regulations [3], [1], [2], ensuring that raw data remains safeguarded from third-party access. FL stands apart from centralized learning methodologies by leveraging client-side training efforts, thereby significantly reducing the computational burden on the server. This efficiency has prompted the adoption of FL across a diverse range of application domains [78]. For instance, within the domain of image recognition [44], collaborative model training among hospitals has gained traction [34], [62], [52], [23], [24], [61], [65], [66], [68]. Similarly, in Natural Language Processing, FL has proven instrumental in tasks such as text prediction [35], [59], [80], [19], [47], sentiment analysis [9], and personalization [18]. For a comprehensive overview of additional examples, we refer the reader to [40].

Within federated settings, it is crucial to address the presence of adversaries who control a subset of clients and submit poisoned updates to compromise the integrity of the aggregated model. These attacks can manifest as untargeted endeavors, seeking to diminish the predictive performance of the model [29], [75], [77], [41]. Conversely, targeted poisoning attacks, commonly referred to as backdoor attacks, aim to maintain inconspicuous behavior on regular input while inducing the model to produce adversarial-chosen predictions when presented with input containing a predefined trigger [8], [54], [74], [76], [33], [43], [32], [70], [11], [50], [7], [21], [63], [72], [20], [14], [57]. Backdoors pose a more significant risk as they are challenging to detect, and their unexpected misconduct can have detrimental consequences for users relying on the model in real-world, e.g., for autonomous vehicles [42], [51], [82].

Defenses against poisoning attacks employ three strategies: Influence Reduction (IR), Robust Aggregation (RA), and Detection and Filtering (DF). IR approaches [6], [8], [49], [71] perturb model parameters to cancel malicious behavior, RA-based defenses [81], [46] secure aggregation algorithms even in the presence of poisoned models, and DF-based solutions [13], [48], [67], [53], [31], [60], [84], [16] detect and filter out poisoned models before aggregation. Among the three categories, DF approaches appear to be more prominent solutions, as IR and RA-based systems unavoidably affect benign functionality.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.241366
www.ndss-symposium.org

Yet, the preferable DF methods have to face their own challenges. One particular concern is the fact that adversaries possessing knowledge of defense mechanisms can strategically adapt their poisoned models to evade the defensive strategies in place. This leads to a dynamic competition between the attacker and defender, emphasizing the ongoing pursuit to safeguard the system.

Problem Statement. Upon examining the existing literature on DF defense approaches [13], [48], [67], [53], [31], [60], [84], [16], it becomes apparent that there is no common definition of "adaptive attacker", making it challenging to assess the detection capabilities of defenses and draw meaningful comparisons. While some papers (e.g., [53]) do not explicitly define the capabilities of adaptive attackers, the prevailing state-of-the-art approach is rooted in Bagdasaryan *et al.*'s seminal backdoor paper [8], which introduces an additional objective for adaptation alongside the main objective of model training. More recent papers also consider approaches to detect deviations in more than one single metric [60], which may require consideration of a stonger attacker model in evaluation.

In this paper, we aim to assess if the definitions of adaptive attackers established in existing papers are meaningful. In particular, we establish that their most common definition by Bagdasaryan *et al.* [8] exhibits limitations. While it enables adaptability to detection metrics, it does so while relying on a fixed parameter that allows for a balance between the main training objective and an additional loss function introduced to achieve stealthiness in a detection metric. One needs to invest substantial manual efforts to find a suitable parameter, and even then, there is no guarantee that the found solution is satisfying. Additionally, the method is primarily designed to meet constraints that must equal fixed values. However, the treatment of constraints involving inequalities potentially forming ranges of valid values within some metrics remains unclear. Moreover, the method does not facilitate the training of stealthy malicious models that can successfully evade detection by defense methods that rely on multiple detection metrics (such as, e.g., [60]), which increases the number of constraints to consider.

Contributions. In this paper, we aim to address the limitations of the most commonly used definition of adaptive attackers based on the adaption method by Bagdasaryan *et al.* [8]. In particular, our contributions are as follows:

- We propose *AutoAdapt*, a novel adaption method that can provide a more reasonable alternative for a definition of adaptive attackers. The new method can be used by attackers, but most importantly, it can establish a new baseline in research to evaluate the effectiveness of FL defenses. Our method enhances the adversary's ability to circumvent defenses by leveraging an adaptation of the mathematical optimization technique known as Augmented Lagrangian [10].
- We show that AutoAdapt eliminates the need to search for an optimal balancing parameter between adaption and the main training task, ensuring a solution that meets all demands posed by constraints. This accelerates the adversary's capabilities and expedites research when developing attacks and respective defenses. We show that by leveraging AutoAdapt, the runtime effort required to test FL defenses can be sped up $15\times$.
- AutoAdapt goes beyond the limitations of the existing approach by addressing inequality constraints in a more comprehensive manner. It employs the Augmented Lagrangian method for inequality constraints[10], [55] to go beyond the satisfaction of specific fixed values and facilitate the identification of solutions that *form ranges of valid values within the defense metric*. By considering these ranges, AutoAdapt provides a more flexible and adaptive approach to training malicious models that can successfully evade detection.
- AutoAdapt takes a step further by supporting adaptation to *multiple detection metrics* simultaneously, by effectively incorporating multiple range constraints into the training process. By considering the interplay between these different ranges, AutoAdapt poses a significant challenge to defenses that rely on multiple detection metrics. Overall, it forces defenses to account for a wider range of potential adversarial behaviors and evasion strategies.
- Through a systematic study, we demonstrate that an adversary utilizing AutoAdapt can effectively adapt to one or multiple range constraints. Moreover, we demonstrate the superior performance of AutoAdapt compared to the current state-of-the-art method, successfully bypassing a selection of defense mechanisms. This substantiates its significance as an invaluable tool for both attackers and researchers. Furthermore, we establish the versatility of AutoAdapt, showcasing its applicability across diverse application scenarios encompassing datasets, model architectures, and hyperparameters, thereby emphasizing its practical utility in real-world contexts.

Overall, our proposed method, AutoAdapt, introduces significant advancements in the field of FL defense evaluation and empowers adversaries to enhance their capabilities, ultimately driving the progress of research in this domain.

II. Background

A. Federated Learning

In Federated Learning (FL) [46], [38], [79], multiple clients collaborate to improve a DNN under a central server's guidance. Clients train local DNN models on their own datasets and share the results with the server for aggregation. This preserves data privacy and reduces infrastructure costs on the server side by distributing computational effort. FL follows an iterative process where the server selects a subset of available clients for each round and distributes an initially untrained global model to them. Each client initializes its local model with the global model and trains a new local model using its dataset and predefined algorithm parameters. The client then submits the model updates to the server, which aggregates them into a new global model. Federated Averaging (FedAVG) [46] is commonly used for aggregation, calculating the weighted average of updates U_i^t using a global learning rate δ to get a new global model G^{t+1}, as shown in Eq. 1. After aggregation, the next round is initiated by the server.

$$G^{t+1} = G^t + \delta(\frac{1}{n}\sum_{i=1}^{n} U_i^t) \quad (1)$$

Poisoning Attacks. Untargeted poisoning significantly hampers a model's performance, measured as main task accuracy (MA). This can be achieved through data poisoning, where incorrect labels are assigned to a subset of training dataset samples controlled by the poison data rate (PDR). Alternatively, model poisoning involves the deliberate manipulation of model parameters either during or after training. In targeted attacks, also known as backdoor attacks, a DNN is manipulated to produce specific mispredictions when provided with inputs containing predefined triggers. These triggers, such as a red pixel in the upper left corner of an image, must be inserted into the training dataset via data poisoning. A successful attack is characterized by high prediction performance on triggered data, referred to as backdoor accuracy (BA), while maintaining stealthiness on benign inputs indicated by a high MA. To execute such attacks, an adversary with control over one or more clients within a federation submits poisoned local models to the server. The objective is to align the aggregated model's predictions with those of the local poisoned models.

Defense Adaptive Adversaries. A poisoned model not only needs to maintain a inconspicuous MA but also evade potential defense mechanisms. Commonly, metrics like Cosine and Euclidean distance are used to compare global and local models, filtering out potentially malicious models based on the assumption that more than half of the contributions are benign. Outlier detection methods, both simple and sophisticated, are employed for this purpose. Thus, a poisoned model must also appear inconspicuous in terms of defense metrics. To adapt to defenses while preserving high MA and BA, the classical approach is to include an additional objective ($Loss^{Adaption}$) in the loss function alongside the main objectives ($Loss^{MA/BA}$) as depicted in Eq. 2. This technique, known as "train-and-scale" [8], [28], allows the adversary to balance performance and adaptation intensity, ensuring stealthiness. The weights assigned to these objectives, denoted by $\alpha \in [0, 1]$, determine the priority, whereas large α leads to less adaption. Throughout this paper, this method is referred to as the classical method of Bagdarasyan *et al.*. Furthermore, scaling the updates of a poisoned local model based on their Euclidean distance can enhance their influence on the aggregated model, thus increasing the BA and is called "constrain-and-scale" [8].

$$Loss = \alpha \cdot Loss^{MA/BA} + (1-\alpha) \cdot Loss^{Adaption} \quad (2)$$

The objective of a defense against poisoning attacks is to create a scenario where optimizing both $Loss^{MA/BA}$ and $Loss^{Adaption}$ simultaneously becomes infeasible. This forces the adversary to make a trade-off between executing an effective attack and adapting to the defense, leading to what is known as an adversarial dilemma [60], [30].

B. Constrained Optimization

Constrained optimization is the process of finding the best solution to a problem while respecting a set of constraints. It aims to minimize or maximize[1] an objective function while satisfying specific conditions or limitations imposed by the constraints. These constraints can include variable restrictions, relationships between variables, or other specified conditions that must be fulfilled. Generally, such a problem can consist of equality and inequality constraints as formulated in Eq. 3.[2] Thereby, $f(x)$ represents the objective function that must be minimized during the optimization process, while $g_i(x)$ and $h_j(x)$ represent n equality and m inequality constraints, respectively, that must be satisfied for a valid solution.

$$\begin{aligned} \min \quad & f(x) \\ s.t. \quad & g_i(x) = 0 \ \ \forall i = 1, \ldots, n \\ & h_j(x) \leq 0 \ \ \forall j = 1, \ldots, m \end{aligned} \quad (3)$$

Penalty Method. The penalty method is utilized to solve a constrained optimization problem as formulized in Eq. 3 by transforming it into an unconstrained problem and is generally applicable for both equality (Eq. 4) and inequality constraints (Eq. 5). The unconstrained problem for equality constraints $G(x)_k$ is constructed by adding a penalty term $\rho_k \sum_{i=1}^{n} |g_i(x)|^2$ to the objective function $f(x)$.[3] The penalty term incorporates a penalty parameter $\rho_k \in [0, \infty]$ multiplied by a measure of constraint violation represented by the squared sum of all $g_i(x)$. When constraints are violated, the measure of violation is nonzero, but it becomes zero in regions where the constraints are satisfied. Therefore, this term is viable for optimization, e.g. minimization, problem. The method involves solving the transformed problem $G(x)_k$ iteratively for a specified number of iterations (k), which appears to converge toward the solution of the original problem.

$$\min G(x)_k \ \text{ with } \ G(x)_k = f(x) + \rho_k \sum_{i=1}^{n} |g_i(x)|^2 \quad (4)$$

$$\min H(x)_k \text{ with } H(x)_k = f(x) + \rho_k \sum_{j=1}^{m} \max(0, h_j(x))^2 \quad (5)$$

In the first iteration, the penalty parameter ρ_0 starts with a deliberate guess. Then, a solution is computed and the penalty parameter is progressively increased according to a chosen rule in each round k, eventually approaching infinity. Consequently, to achieve convergence towards the optimal solution, the constraints must be satisfied, effectively causing the penalty term to approach zero.

Lagrange Multipliers. This method is capable of finding optimal solutions for constrained optimization under equality constraints. To this end, additional variables, so-called Lagrange multipliers λ, are introduced to incorporate constraints into an optimization problem. Acting as weights, they quantify the impact of constraints $g_i(x)$ on the objective function $f(x)$, similar to ρ_k in the penalty method. The resulting Lagrange function can be constructed for one (Eq. 6) or multiple (Eq. 7) equality constraints. By differentiating the Lagrangian function with respect to the variables and multipliers, critical points are

[1]Maximization equals minimization of the negative objective.

[2]Constraints aiming for a value other than zero like $g_i(x) = 5$ can always be rewritten to the standard form $g_i(x) - 5 = 0$.

[3]The treatment of inequality constraints is similar, but they are formulated as max functions to ensure that all negative values are considered valid and do not affect the optimization process, which aims to minimize the objective.

found as potential solutions. Solving the system of equations formed by equating partial derivatives to zero determines the values of variables and multipliers that satisfy the objective function and constraints, representing the optimal solution.

$$\mathcal{L}(x, \lambda) = f(x) + \lambda \cdot g(x) \tag{6}$$

$$\mathcal{L}(x, \lambda_i) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) \tag{7}$$

Augmented Lagrangian. As there is no direct gradient-based algorithmic approach for a computer to find Lagrange Multipliers, the Augmented Lagrange method (Eq. 8) is employed for more complex problems. Compared to the penalty method (Eq. 4), it prevents the escalation of the penalty parameter ρ_k to infinity by adding a second penalty term, that mimics Lagrange Multipliers. This method aims for the computation of an optimal solution while keeping ρ_k in an acceptable range or even fixed. Through iterative adjustments, the values of the Lagrange multipliers are refined, leading them towards more suitable values, as can be seen in the second line of Eq. 8.

$$\begin{aligned} \mathcal{L}(x, \rho, \lambda)_k &= f(x) + \frac{\rho_k}{2} \sum_{i=1}^{n} |g_i(x)|^2 + \sum_{i=1}^{n} \lambda_i g_i(x) \\ \lambda_i &= \lambda_i + \rho_k g_i(x_k) \end{aligned} \tag{8}$$

KKT Conditions. The Karush-Kuhn-Tucker (KKT) conditions extend the use of Lagrange multipliers to incorporate inequality constraints. The optimization problem is formulated by introducing additional multipliers for inequality constraints, denoted as μ (Eq. 9).

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) + \sum_{j=1}^{m} \mu_j h_j(x) \tag{9}$$

The KKT conditions establish criteria for determining if a candidate solution x^* is optimal. These conditions include primal feasibility (Eq. 10), ensuring that the constraints are satisfied. Stationarity (Eq. 11) requires the first derivative of the Lagrange function to be zero, as commonly done in finding minima. Dual feasibility (Eq. 12) specifies that the multipliers for inequality constraints must be nonnegative. Complementary slackness (Eq. 13) ensures that if an inequality constraint is satisfied but not zero, the corresponding multiplier is zero, truncating the overall objective function (Eq. 9). It can be stated that a point x^* is optimal (for convex programs) if all KKT conditions are satisfied.

- Primal feasibility
$$\begin{aligned} g_i(x^*) = 0 \;\; \forall i = 1, \ldots, n \\ h_j(x^*) \leq 0 \;\; \forall j = 1, \ldots, m \end{aligned} \tag{10}$$
- Stationarity
$$\nabla_x \mathcal{L}(x^*, \lambda, \mu) = 0 \tag{11}$$
- Dual Feasibility
$$\mu_j \geq 0 \;\; \forall j = 1, \ldots, m \tag{12}$$
- Complementary Slackness
$$\mu_j h_j(x^*) = 0 \;\; \forall j = 1, \ldots, m \tag{13}$$

III. Problem Statement

We examine a conventional FL system as described in Sect. II-A. The aggregation server employs FedAVG [46] with a fixed global learning rate (LR) of one. An adversary has control over multiple clients and possesses the capability to carry out various data and model poisoning attacks (cf. Sect. II-A). Further, the adversary has knowledge of the aggregation server's code as well as specifics of any deployed defense mechanisms, enabling them to make adaptation attempts. Following the approach of previous studies [67], [60], [53], [48], [6], [13], we assume that the majority of clients in each training round are benign. To address the uncertainty of adversaries' involvement, the server assigns equal weights to all model updates instead of considering the clients' dataset size, as adversaries may report excessively large datasets.

Limited evaluation of Defenses. To infiltrate a FL system with a backdoor, an attacker faces the challenge of solving a constraint optimization problem. This problem involves optimizing the model regarding MA and BA as the objective function while simultaneously circumventing the applied defenses. In other words, the attacker needs to adapt to benign values within the defense metrics, which can be formulated as constraints. Typically, FL defenses are evaluated using inadequate and unrealistic attacker settings [60], [31], [53], [71], [13], [67], e.g., by applying weak constrained optimization methods that fail to provide satisfying solutions, giving no detailed information about the used method, or do not consider adaptive adversaries at all. In particular, the classical adaption method from Bagdasaryan *et al.* known as "train-and-scale" [8] is commonly used. This method converts the constrained optimization problem into a sum of unconstrained optimization problems. The importance of balancing the main task and the adaptation is determined by an adversarial-chosen fixed value α. This classical adaptation approach shown in Eq. 2 has the following limitations:

Determination of Alpha. Since α is a fixed and manually determined value, extensive experiments need to be conducted to determine the optimal choice. This applies to adversaries attacking a system, but also to researchers testing the effectiveness of FL defenses.

Fixed Alpha. The α value in Eq. 2 is a predetermined constant chosen by the attacker. We notice that Eq. 2 corresponds to a single iteration of the classical penalty method (Eq. 5). If we divide Eq. 2 by α, the result corresponds to a penalty method with fixed penalty parameter $\frac{1-\alpha}{\alpha}$. Similar to ρ_k, this $\frac{1-\alpha}{\alpha}$ term belongs to $[0, \infty[$. Yet, Eq. 2 does not involve any intermediate adjustments to α, and as a result, it fails to explore potentially better solutions. This could be achieved by employing an iterative process similar to that used in the classical penalty method (Eq. 5).

Ill-Conditioning. Merely substituting Eq. 2 with the classical penalty method (Eq. 4) is inadequate, since, as the penalty parameter ρ_k approaches infinity, a minor alteration in the penalty function can have a significant impact on the overall loss – the effect which is called ill-conditioning. This can result in false optimization steps and consequently lead to a suboptimal solution. The same issue arises when α approaches zero, as then the aforementioned penalty term $\frac{1-\alpha}{\alpha}$ will similarly face ill-conditioning due to the division by a value close to zero.

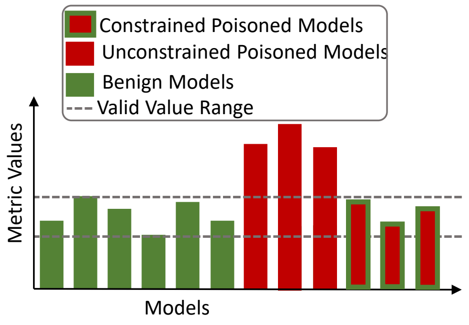

Fig. 1: Visualization of a valid value range for one metric.

Inequality Constraints. In the classical formula by Bagdasaryan *et al.* (Eq. 2), the treatment of inequality constraints is not explicitly defined. Without proper adjustments, a constraint that is fulfilled would inadequately continue to impact the minimization process instead of being disregarded as intended. Consequently, the success of the approach greatly relies on how the different constraints are combined by the attacker or researcher.

Multiple Constraints. In the context of the penalty method (Eq. 4) or the adaption with Bagdasaryan *et al.* (Eq. 2), it is important to note that the main objective and all constraints are balanced by ρ_k and α, respectively. However, there is no weight assigned to the constraints relative to each other. As a result, Eq. 2 may yield suboptimal performance when multiple constraints are involved. This is because constraints with smaller-scale values are treated as already minimized by the optimizer and consequently ignored during the update step.

Range Constraints. Furthermore, the method proposed by Bagdasaryan *et al.* (Eq. 2) lacks guidance on implementing range constraints, which are inherent in backdooring scenarios. The adversary can capture multiple clients and initially train benign models to determine the legitimate values for the defense metric. These benign values then form a range that serves as the target range during the training of poisoned models, which is not natively supported by Eq. 2. Such a scenario is visualized in Fig. 1: The benign (green) models form a valid value range represented by the dotted lines. The constraint models (red with green border) are adapted versions of the unconstrained models (red) so that the metric values reside within the valid value range.

Overall, the state-of-the-art defense adaption method (Eq. 2) is insufficient for testing the efficiency of backdoor defenses. The method does not guarantee to find an satisfying solution for multiple metrics with inequality constraints that form valid ranges for metric values.

IV. AutoAdapt

This paper focuses on addressing the challenges discussed in Sect. III, particularly the limited applicability of Eq. 2 as a state-of-the-art approach to adapt to metrics of FL defenses when evaluating their performance. We propose an Augmented Lagrangian-based method to replace the adaption method of Bagdasaryan *et al.* [8] (Eq. 2) in the constrained optimization of adaptive adversaries. This method ensures an viable solution within legitimate value ranges for various metrics, considering multiple inequality constraints.

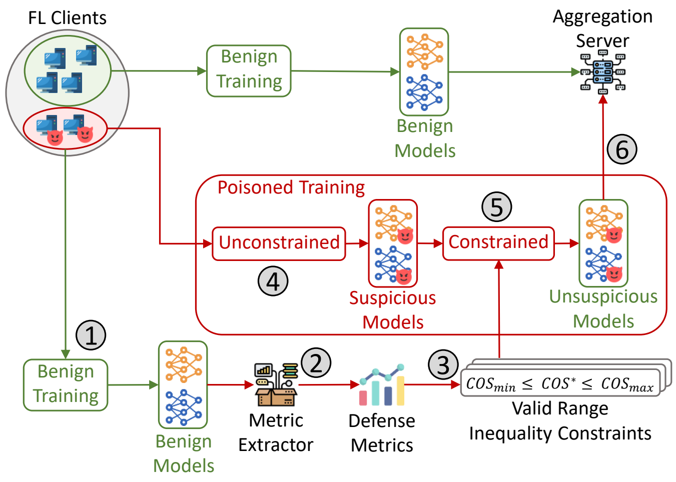

Fig. 2: Overview of adversarial adaption via AutoAdapt.

The adversarial adaptation process of AutoAdapt is depicted in Fig. 2 and consists of the following steps: 1) The adversary initiates the process by training benign models to 2) obtain legitimate values for each metric used by the defense. 3) Subsequently, the minimum and maximum values from these benign models form a valid range for each metric, as visualized in Fig. 1. 4) In terms of malicious training, the adversary begins by training a specific number of unconstrained poisoned epochs.[4] This approach of first training unconstrained models differs from the classical setup of Bagdasaryan *et al.* [8] and is motivated by the fact that for stealthy backdoor embedding, typically the poison data rate (PDR) is low, and, therefore, even a poisoned model evolves in roughly the same direction as benign models.[5] 5) To alleviate the deviations from benign value ranges within metrics of the suspicious models introduced during the poisoned training, the attacker initializes the Augmented Lagrange optimization method respecting the range constraints produced in step 3. The details of this method are described below. Training then proceeds in this constrained optimization mode until a satisfactory solution is achieved. 6) Once all the constraints are adequately fulfilled, the adversary terminates the procedure and submits the adapted poisoned models to the aggregation server. These models are now deemed non-suspicious based on the defense metrics.

Below, we will provide an intuitive explanation and derivation of the Augmented Lagrangian method for backdoor embedding, which is applied in step 5 of Fig. 2. We will start from Eq. 2 and elucidate the modifications we have made to the method, which is summarized in Fig. 3. As an initial step, we convert the problem into a mathematical notation that aligns with our notation introduced in Sect. II. The result is depicted in box 2 of Fig. 3.

Disposing Determination of Fixed Alpha. Instead of investing significant effort in selecting a specific α value by training multiple models and comparing results, one can choose an initial α_k value and gradually increase it over consecutive optimization rounds k. This approach allows for a gradual adjustment of the α value during the optimization process, enabling the model to adapt and "harden" towards the desired

[4]The number of epochs can be either adjusted based on the target BA or set to a fixed value.
56

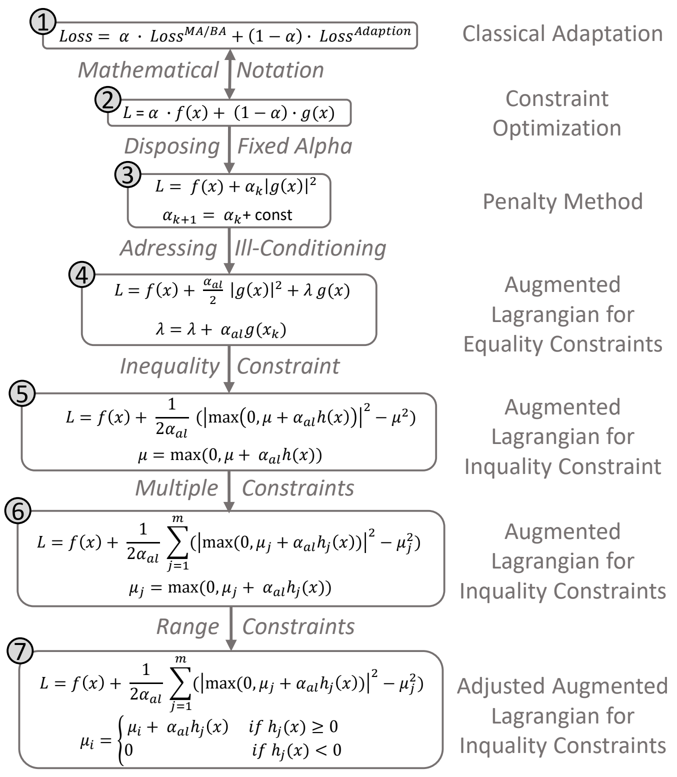
1 $Loss = \alpha \cdot Loss^{MA/BA} + (1-\alpha) \cdot Loss^{Adaption}$
Classical Adaptation
Mathematical Notation
2 $L = \alpha \cdot f(x) + (1-\alpha) \cdot g(x)$
Constraint Optimization
Disposing Fixed Alpha
3 $L = f(x) + \alpha_k |g(x)|^2$
$\alpha_{k+1} = \alpha_k + \text{const}$
Penalty Method
Adressing Ill-Conditioning
4 $L = f(x) + \frac{\alpha_{al}}{2} |g(x)|^2 + \lambda\, g(x)$
$\lambda = \lambda + \alpha_{al} g(x_k)$
Augmented Lagrangian for Equality Constraints
Inequality Constraint
5 $L = f(x) + \frac{1}{2\alpha_{al}} (|\max(0, \mu + \alpha_{al} h(x))|^2 - \mu^2)$
$\mu = \max(0, \mu + \alpha_{al} h(x))$
Augmented Lagrangian for Inquality Constraint
Multiple Constraints
6 $L = f(x) + \frac{1}{2\alpha_{al}} \sum_{j=1}^{m} (|\max(0, \mu_j + \alpha_{al} h_j(x))|^2 - \mu_j^2)$
$\mu_j = \max(0, \mu_j + \alpha_{al} h_j(x))$
Augmented Lagrangian for Inquality Constraints
Range Constraints
7 $L = f(x) + \frac{1}{2\alpha_{al}} \sum_{j=1}^{m} (|\max(0, \mu_j + \alpha_{al} h_j(x))|^2 - \mu_j^2)$
$\mu_i = \begin{cases} \mu_i + \alpha_{al} h_j(x) & if\ h_j(x) \geq 0 \\ 0 & if\ h_j(x) < 0 \end{cases}$
Adjusted Augmented Lagrangian for Inquality Constraints

Fig. 3: Derivation of AutoAdapt.

constraint value for the actual α_k. The constraint violation progressively reaches a minimum, representing the best solution while considering training hyper-parameters such as the learning rate. By incrementally increasing α_k with a constant value, an improved version can be found that further reduces the range violation until an optimal or satisfactory solution is achieved. This involves replacing Eq. 2 with a penalty version similar to Eq. 4, as shown in box 3 of Fig. 3, where α_k is iteratively increased at a constant rate.[7]

Addressing Ill-Conditioning. To prevent α_k from diverging to infinity, we introduce modifications in box 4 of Fig. 3. We replace the penalty method with the Augmented Lagrangian for equality constraints (cf. Sect. II-B), which has the following effects: It avoids the need for α_k to approach infinity to achieve an optimal solution by introducing an additional penalty term with the λ parameter mimicking Lagrange Multipliers. This eliminates concerns about ill-conditioning. α_k can even be fixed to a specific value denoted as α_{al}, which should be sufficiently large to dominate the entire function and ensure convexity. However, the exact α_{al} value is not a sensitive parameter and does not introduce a new hard-to-find hyper-parameter. After each optimization step k, λ is updated based on the constraint violation of the current solution x_k (cf. App. -B).

Inequality Constraint. When augmenting the expression depicted in box 4 of Fig. 3 to account for inequality constraints, one should achieve that the penalty function does not impact the optimization when the constraint is satisfied (negative). This can be done by reformulating an inequality constraint as an equality constraint using squared slack variables s^2 in the following way:

$$h(x) \leq 0 \quad \Longleftrightarrow \quad h(x) + s^2 = 0 \tag{14}$$

This allows to treat inequality constraints as in the standard Augmented Lagrangian approach, but at the cost of an additional variable s which needs to be optimized. Exploiting the special and simple structure in which s occurs within the Augmented Lagrangian, the minimization with respect to the variable s can be carried out analytically. In this way, it is possible to eliminate s from the Augmented Lagrangian, and this leads to the formula given in box 5 of Fig. 3. For the details of the elimination of the s variables, we refer to App. -A or [10], [55]. The transformation results align with the intuitive notion that inequality constraints can be reformulated using min/max functions. The update of μ is thereby motivated similar as for equality constraints and can be retraced in App. -B.

Multiple Constraints. To accommodate multiple inequality constraints, we introduce a sum term and assign individual variables (μ_j) to each constraint, as depicted in box 6 of Fig. 3. Mathematically, this method is equivalent to the Augmented Lagrangian approach for inequality constraints (cf. Sect. II-B), which guarantees optimal convergence of this approach. In the classical method proposed by Bagdasaryan *et al.*, multiple constraints are not explicitly supported. One could assume their implicit support by, e.g., simply summing these constraints. This approach would, however, lead to a situation where a constraint with values on a significantly smaller scale than another constraint would be treated as already minimized and, subsequently, ignored.

Range Constraints. As adversarial training relies on ranges of valid values for each metric (cf. Fig. 1), accommodating multiple inequality constraints that form a range becomes necessary. Therefore, the adversary leverages the maximum and minimum metric value of benign trained models.[8] For instance, a range constraint for the Cosine distance $COS_{min} \leq COS^* \leq COS_{max}$ will be implemented as two inequality constraints $COS_{min} - COS^* \leq 0$ and $COS^* - COS_{max} \leq 0$. However, a challenge arises since those two opposing constraints forming a range compete against each other, potentially resulting in the violation of one while satisfying the other. This situation can occur mathematically when the constraint is fulfilled ($h(x) \leq 0$), but the penalty term $max(0, \mu + \alpha_{al} h(x))$ is not set to zero due to a large value of μ in the max function. To address this issue, we manually set μ to zero when the constraint is satisfied, as visualized in box 7 of Fig. 3. This adjustment is justified by the KKT conditions (cf. Sect. II-B), which state that for an optimal solution, μ must be zero when $h(x) < 0$, as per the complementary slackness condition. By making this adjustment, we enable

[7]The "const" parameter in box 3 of Fig. 3 offers a trade-off between slow-but-safe adaptation and fast adaptation at the risk of neglecting the main-task, which could degrade the main-task-performance.

[8]Adversarial-captured clients can collude and hence communicate valid value ranges (as adversaries have full control over the client, similar to [15]). We can assume that benign models trained by adversaries would at least reside nearby/within the final server-side valid value range and aren't detected as outliers. Therefore, a valid value range is definable with two benign models. Besides range constraints, AutoAdapt allows equality constraints, which can be used if just one client is adversarial. Note, that valid value ranges need to be recalculated each FL round.

the formulation of range constraints that mutually respect each other and can be employed simultaneously.

Implementation Details. The parameter α_{al} is assigned the value of the multiplier that aligns $f(x)$ and the sum of all violations of $h_j(x)$ on the same scale (cf. App. -C). As α_{al} is not a sensitive parameter and just needs to be sufficiently large [10], this ensures a valid value for α_{al}, removing the need for its adjustment as a hyper-parameter.

During constraint training (step 5 in Fig. 2), momentum and decay are disabled for the optimizer of the model training. This decision is based on the rationale that momentum takes into account previous update steps, and if a previous update step satisfies a range constraint from one direction, it can potentially violate the same constraints in the opposite direction due to their competing nature. Decay, on the other hand, serves as a form of parameter regularization and acts as a penalty term. Since we already incorporate our own penalty terms, we want to prevent any external penalty term from undermining our efforts in constructing a non-suspicious model.

Anticipating that a fulfilled constraint in one iteration may be violated in subsequent iterations due to the main objective or other constraints, AutoAdapt employs a memory mechanism to retain the most satisfactory solution. It chooses the solution with the fewest constraint violations, or in cases where violations are equal, it selects the solution with the minimum penalty sum as the final submission.

Summary. Overall, AutoAdapt effectively addresses several shortcomings of the classical adaptation method. It eliminates the reliance on fixed α values, eliminates the challenges associated with determining the optimal α, and mitigates the ill-conditioning issues that arise with extreme α values. Moreover, AutoAdapt allows for the consideration of multiple inequality constraints, enabling the incorporation of valid value ranges. As a result, AutoAdapt surpasses state-of-the-art methods in evaluating the effectiveness of FL defenses against adaptive adversaries.[9]

V. Evaluation

A. Experimental Setup

Hardware and Software. The FL system simulation is conducted on a server utilizing PyTorch [4], [58], a widely recognized Python [73] machine learning library. The client and server code is executed sequentially on the server, which is equipped with an AMD EPYC 7413 24-Core Processor operating on a 64-bit architecture. The server is equipped with 96 processing units and 128GB of main memory. Additionally, a NVIDIA A16 GPU with 4 virtual GPUs, each offering 16GB of GDDR6 memory, is accessible through CUDA [56] from within the PyTorch framework.

Datasets and Models. To ensure comparability with other FL defenses, we have selected settings that are similar to those used in related works. Our primary focus is on image classification tasks using CIFAR-10 [39], GTSRB [69], and MNIST [26]. For the model architectures, we leverage ResNet-18 [36], and SqueezeNet [37], and CNN.

Default Scenario. We establish a default scenario to demonstrate the overall functionality of AutoAdapt, and, subsequently, we explore variations in different parameters to showcase its independence from them. By default, we train a ResNet-18 [36] model with a learning rate (LR) of 0.01 (using the SGD optimizer with a momentum of 0.9 and a decay rate of 0.005) for the CIFAR-10 [39] image classification task, which comprises ten classes. The federation consists of 20 clients, all selected in each FL round. The data follows an independent and identically distributed (IID) pattern, because non-IID scenarios result in models with notable disparities, facilitating adversaries in injecting poisonings due to the expanded valid value ranges within metrics. Each client possesses 2560 samples consisting of 256 samples randomly selected from each class. The adversary targets nine clients, resulting in a poison model rate (PMR) of 45%, the maximum rate achievable with this number of clients. The adversary sets the poison data rate (PDR) to 0.1, α from Eq. 2 to 0.3, and implements a semantic backdoor, whereby green cars are mislabeled as birds, a scenario which is adapted from [8]. The global model is initialized with pre-trained weights from PyTorch[10], with the first and last layers left untrained, as both needed to be modified to accommodate our dataset. The batch size is set to 64, and the models are trained for ten epochs.

B. General Effectiveness of AutoAdapt

In the subsequent analysis, we demonstrate the effectiveness of AutoAdapt in terms of adaptation. Our evaluation employs two metrics: the Cosine distance and Euclidean distance between the local and global models. To obtain an overall assessment, we compute the summed versions of these metrics, which involve calculating the distance on a parameter basis and aggregating the results. These metrics are widely utilized in defense mechanisms as they can effectively detect backdoors introduced through data poisoning, a trend that aligns with our experimental findings. We proceed to adapt to these metrics individually for the entire model and subsequently on a layer-wise basis.

Throughout the following experiments presented in this section, the model performance in terms of MA and BA exhibits no peculiarities or anomalies, thus no individual reports are provided. The MA of the trained local models demonstrates a slight improvement compared to the previous global model, while maintaining an unsuspecting nature even in the presence of poisoning. The BA of the poisoned models ranges from 80% to 100% within the local models, consistently achieving 100% BA in the aggregated global model, effectively embedding a backdoor.

Single Metric. Through the application of AutoAdapt, we successfully adapt the Cosine distance of the poisoned local models to the range of valid values defined by benign models.[11] The left side of Fig. 4 demonstrates the metric without adaptation, indicating that any defense utilizing clustering algorithms based on this metric could identify and filter out the poisoned

[9]Note, that AutoAdapt is independent of deployed defenses, as long as one can adapt to the defenses by constructing corresponding loss functions, which indeed can be very challenging to build. If such a loss function exists, AutoAdapt improves the state-of-the-art adaptation mechanism.

[10]The pre-trained model obtained from PyTorch has different input dimensions and 1000 label classes, since they are trained on ImageNet [25].

[11]The adaption to Euclidean distance only delivered similar results.

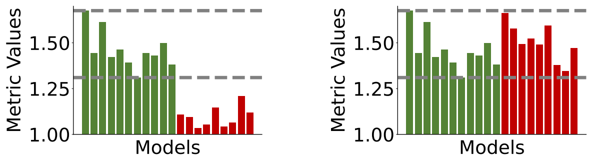

Fig. 4: Unadapted (left) and AutoAdapt-adapted (right) Cosine metric in the default scenario.

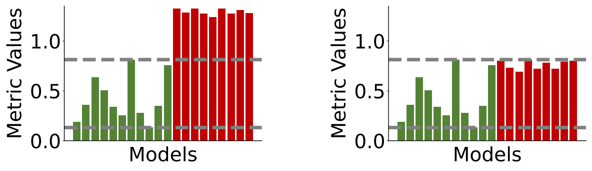

Fig. 5: Unadapted (left) and AutoAdapt-adapted (right) Cosine metric or the last layer in FL round 50.

models. Conversely, on the right side, we observe that the poisoned models blend inconspicuously within the metric after being adapted with AutoAdapt. This confirms the general effectiveness of adaption through AutoAdapt. Notably, we activated the constraint optimization during the eighth epoch out of ten. The constraints were fulfilled within four to seven batches already, resulting in minimal impact on the overall training process, as reflected in the runtime (see Sect. V-E). When the constraints are met, the training continues without the influence of a penalty term, effectively operating unconstrained. Once a constraint is violated again, the constraint mode is automatically reactivated.

Single Metric - Model Subparts. Given that backdoors are frequently embedded in the last layer of a model, certain defenses, e.g., FoolsGold [31], concentrate their analysis on the last layer. Consequently, we conducted an evaluation to determine whether AutoAdapt is capable of adapting exclusively to this layer. To test this, we conducted the same experiment as before in FL round 50 (as the backdoor is not indicative in round 0 in the last layer) and visualized the results in Fig. 5. Remarkably, the adaption was successful in this context as well, with the constraints being met within one to four batches. One can see, that the values reside on the edge of the feasible metric range. If this is a undesired behaviour, the attacker can certainly adjust this by adjusting the value ranges of each client, which we further discuss in Sect. VI. These findings highlight the capability of AutoAdapt to achieve fine-grained adaption within specific parts of the model.

Single Metric - Fine-Grained. Considering the possibility that defenses can analyze the model layer by layer, it is important to address the potential presence of backdoors embedded within the layers, even if the overall model appears benign. To investigate this, we perform an experiment where we adapt all layers of the ResNet-18 [36] model to align with the benign value ranges, following a similar approach as the previous experiment conducted on the last layer. In the case of ResNet-18, this involves incorporating 14 inequality constraints for the seven layers. We can report, that AutoAdapt demonstrates the capability to adapt to all imposed constraints. The adaption already succeeded for all constraints within a remarkably short span of one to five batches. To provide a comprehensive representation of these results, we have included the visualizations in Fig. 8, as per the space limitations.

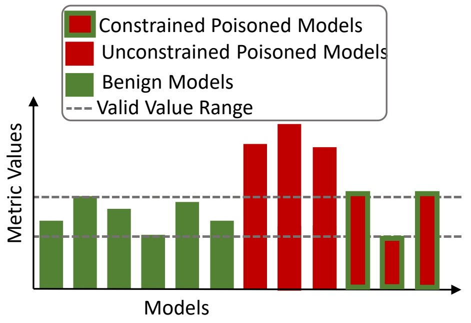

Fig. 6: Visualization of a successful adaption with minimal remaining penalty sum.

Multiple Metrics. Given that defenses commonly employ multiple metrics such as Cosine and Euclidean distances, we replicated the previous experiments by incorporating both metrics. The objective was to demonstrate the ability of AutoAdapt to effectively adapt to and accommodate multiple metrics simultaneously. We started with adapting to Cosine and Euclidean distance with regards to the whole model. Our findings indicate that incorporating a second metric does not pose a challenge for AutoAdapt. However, it was observed that the number of batches required to satisfy the constraints tends to increase as the complexity of the constraints increases. In these experiments, the constraints were fulfilled within a range of five to 15 batches.

Multiple Metrics - Model Subparts. The simultaneous adaptation of the last layer to both Cosine and Euclidean distances yielded favorable results, demonstrating the feasibility of selectively adapting specific model parts based on various metrics. The constraints were successfully met within a range of two to four batches confirming the effectiveness of the adaptation process.

Multiple Metrics - Fine-Grained. In our final extensive experiment, we performed a layer-wise adaption of the poisoned models to Cosine and Euclidean distances, resulting in 28 constraints. The adaption was so successful that the backdoor became completely invisible within these metrics, rendering defenses based on these metrics ineffective. We can confidently affirm that AutoAdapt effectively adapts to all constraints with high efficiency. However, compared to previous experiments, it was necessary to train in a constrained manner for two to four epochs rather than just a few batches to achieve a satisfactory solution. Strictly speaking, two out of the 28 constraints were not completely fulfilled, the quality of their adaption was already so remarkable that the respective penalty sum was in the range 10^{-5}. As a result, the training process continued in a near unconstrained manner, successfully accomplishing the attacker's objective. A depiction of this scenario can be observed in Fig. :6 illustrating the adapted models (depicted in red with a green border) that do not entirely fall within the confines of the valid value range. However, despite this deviation, conventional detection algorithms would be unable to classify them as anomalous. We further discuss such edge cases in Sect. VI.

Comparison to State-of-the-art. Lastly in this section, we present the performance of the method proposed by Bagdasaryan *et al.* [8] (cf. Eq. 2) in the aforementioned scenarios. When adapting the basic model to a single metric

	Adaption Level					
	Single Metric			Multi Metric		
	Whole Model	Single Layer	All Layers	Whole Model	Single Layer	All Layers
Bagdasaryan et. al [8]	✓	✓	✗	✓	✓	✗
AutoAdapt	✓	✓	✓	✓	✓	✓

Fig. 7: Comparison of adaption capabilities between Eq. 2 and AutoAdapt. Eq. 2 is capable of adapting for light-weight scenarios (orange), but fails for complex layer-wise adaption with many constraints (red), whereas AutoAdapt provides satisfying solutions in all scenarios.

such as Cosine distance, Eq. 2 also achieved satisfactory results. However, extensive testing with multiple α values was required, resulting in increased computational and time costs. In our case, we tested $\alpha = [0.1, \cdots, 0.9]$ to identify an effective α with minimal side effects. Ultimately, we found a solution that produced adaptations, but the side effects on other metrics, such as Euclidean distance or even Cosine distance within layers, were more pronounced compared to AutoAdapt. This indicates that even with a detailed investigation of α, AutoAdapt outperforms Eq. 2 by finding a superior solution without the need to train multiple models. Similar effects were observed when adapting to a single layer or to both Cosine and Euclidean distances simultaneously. Notably, abnormal spikes in minimal and maximal parameter changes were observed for Eq. 2, further supporting our claim that if Eq. 2 successfully adapts to the metrics, the model becomes more conspicuous in other metrics compared to the model generated by AutoAdapt.

When adapting layer-wise to only Cosine distance or both Cosine and Euclidean distances, AutoAdapt remains successful, as mentioned earlier. In contrast, Eq. 2 fails to fulfill all constraints in our experiments, leaving at least one layer unadapted. Fig. 8 visualizes this layer-wise adaption to the Cosine metric. The first row shows the unadapted version with data poisoning only, whereas the second row is AutoAdapt-adapted. We can observe, that AutoAdapt can adapt to the metric within every layer and fulfills all constraints. The last row depicts classical adaption via the method of Bagdasaryan *et al.* [8] (Eq. 2) and shows a lack in adaption performance in the first convolutional layer (column one). Consequently, a defense relying solely on clustering based on the metric would filter out the poisoned models. As a result, this demonstrates the superiority of AutoAdapt over the state-of-the-art adaptation technique described in Eq. 2 and proves AutoAdapt to be a superior method for both attackers and researchers testing defenses. As depicted in Fig. 7, AutoAdapt achieves superior results under all settings, while Eq. 2 only provides less satisfying results in light-weight adaption settings and fails when adapting all layers seperately.

C. Influence of Parameters on AutoAdapt

To ensure the generalizability of our findings beyond the specific settings of the default scenario, we conducted validation experiments by modifying various parameters and settings. This allowed us to assess the robustness and applicability of our results across different scenarios.

Backdoor Method. To ensure the robustness of our findings beyond the semantic backdoor used in the default setting, we conducted additional evaluations. Firstly, we assessed a pixel trigger approach [33], where a colored pixel square of with size of 1/16 of the sample width is placed in the upper left corner of the sample image. The color selected for the trigger is the maximum color observed in the first image, making it less conspicuous. We observed that the backdoor was detectable in the Cosine metric across all layers. Consequently, we adapted our approach to address this scenario, and we successfully brought all models within the valid value ranges.

Next, we conducted a similar experiment using a label flip backdoor, where samples from one label class are swapped with a target class [12], [15]. In this case, the backdoor was visible in two layers within the Cosine metric. Once again, we adapted our approach to address all layers, and achieved successful results similar to the previous experiment. Hence, we can say that AutoAdapt is independent from the backdoor method.

FL round. In addition to the default scenario executed in round zero, we also examined the behavior of AutoAdapt in round 20 and round 50. This analysis aimed to demonstrate the independence of AutoAdapt from the convergence level of the global model (see Sect. -D). In both round 20 and round 50 settings, the backdoor was observed to be visible across all layers within the Cosine and Euclidean metrics. Consequently, we performed adaptation on all layers based on these metrics. Remarkably, we achieved complete adaption in all cases, affirming that AutoAdapt remains unaffected by the FL round and the convergence level of the global model.

Data Distribution. The underlying data distribution on the local client has minimal impact on the effectiveness of our attack in this study. To demonstrate this, we conducted an experiment using a 1-class non-IID scenario. In this scenario, a client's dataset primarily focuses on one specific label, while the remaining labels contain an equal number of samples. The non-IID ratio, represented by the factor $q \in [0, 1]$, determines the fraction of samples within the main label class compared to the remaining classes.[12] In our experiment, we set the non-IID ratio to $q = 0.5$. We performed layer-wise adaptation based on the Cosine and Euclidean metrics and observed that we could achieve values within the valid range for all constraints. However, the epochs required to achieve a satisfying result indeed increased to up to 3 epochs in this case, which is surprising, since the valid value ranges are bigger in non-IID scenarios. However, also the deviation of those ranges is increased when introducing an effective backdoor. Nevertheless, we can conclude that the data distribution of the local dataset does not significantly influence the effectiveness of AutoAdapt.

Dataset. We replaced the dataset in our default scenario with MNIST [26] and GTSRB [69] to assess the consistency of experimental results across different datasets. Our findings indicate that the results remain stable irrespective of the dataset used. MNIST, being a simpler dataset, allows for faster convergence during training, while GTSRB presents a more complex scenario due to its larger number of label classes. For both experiments, we employed the pixel trigger backdoor [33]

[12]In the case of $q = 1$, all samples exclusively belong to the main label, while $q = 0$ corresponds to the IID scenario.

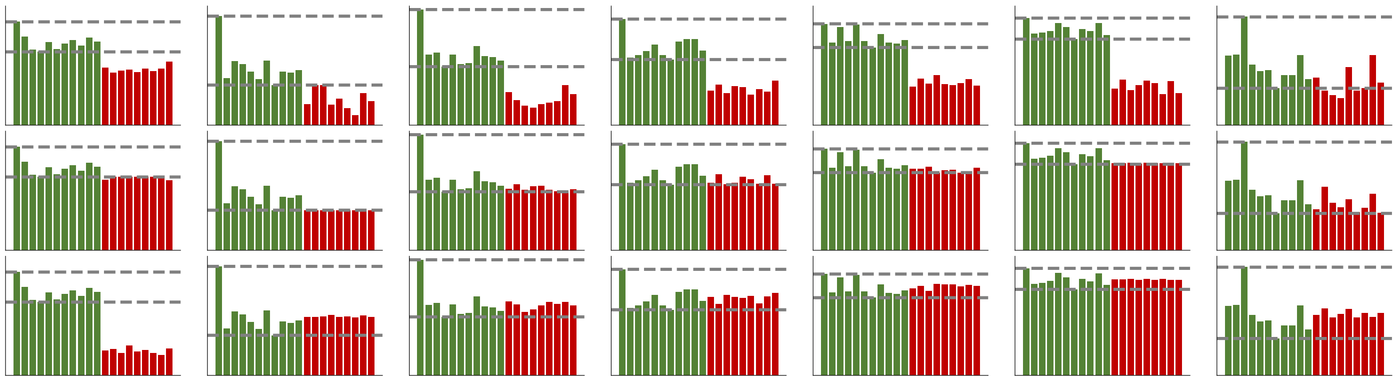
Fig. 8: Visualization of Cosine metric values. Upper row: Unadapted, middle row: AutoAdapt, bottom row: Bagdasaryan *et al.* [8] . The columns represent different layers of ResNet-18 [36]. It can be seen, that the method of Bagdasaryan *et al.* is unsuccessful in adapting to all layers, specifically the first convolutional layer.

since the semantic backdoor is specific to CIFAR-10 [39]. However, the results consistently revealed that the backdoor was present in both the Cosine and Euclidean metrics across all layers, and the adaptation through AutoAdapt was successful within a certain number of batches. These observations support the claim that AutoAdapt is independent of the specific application scenario.

Model Architecture. In addition to ResNet-18, we examined the performance of two additional model architectures: SqueezeNet [37] trained using CIFAR-10 dataset [39] and a CNN with two convolutional layers concatenated with pooling layers and ReLu functions [5], followed by three fully connected layers. The latter one is trained on MNIST [26]. In both cases, we used the pixel trigger as a backdoor [33]. The results obtained from these architectures were consistent with our previous findings. The backdoor was visible within all layers in both Cosine and Euclidean metrics, and the layer-wise adaptation using AutoAdapt was successful within one to two epochs. Consequently, we can confidently assert that AutoAdapt exhibits independence from the specific model architecture within the FL framework.

D. Convergence of AutoAdapt

To examine the behavior of AutoAdapt in relation to the convergence of both, the fulfilled constraints and the overall model performance, we present experimental insights. Fig. 9 illustrates the adaptation of the Cosine metric in the default scenario, while concurrently adapting to the Cosine and Euclidean metrics. On the left side of the figure, we observe that constrained training started in epoch eight, and within a few batches, the constraints are satisfied, as the metric falls within the predefined value ranges indicated by horizontal lines. After a few epochs, the upper (purple) constraint is violated, but AutoAdapt promptly identifies this situation and takes corrective measures. The right side of the figure depicts the corresponding constraint violation values, with the purple line denoting the upper constraint and the yellow line representing the lower constraint. When a constraint is satisfied, the value becomes smaller than zero, as we reformulate all constraints as negativity constraints. In the beginning, the metric values (blue) in the left figure are smaller than the yellow line and hence violates the lower constraint in the left figure. Accordingly, the respective constraint violation on the right is bigger than zero. Likewise, the purple constraint is fulfilled first, hence the constraint violation value is negative. Later, around epoch eleven, one could observe that the upper constraint (purple) gets activated shortly when the metric values (blue) reside slightly above the upper threshold. This demonstrates that our approach functions as intended and converges toward a solution that fulfills the imposed constraints.

To demonstrate that the utilization of AutoAdapt does not hinder model convergence, we conducted a comprehensive training experiment on a client model over a duration of 100 epochs in our default scenario. In this setup, we simultaneously adapted the model to both the Cosine and Euclidean distances with respect to the global model. The constraint optimization mode was activated starting from the eighth epoch. Throughout the training process, we closely monitored two key metrics: The size (L_2 Norm) of the model update gradient and the loss on the training data at the conclusion of each epoch. The desirable outcome is for both the update gradient size and the loss to progressively approach zero, indicating successful training. For the purpose of comparison, we also trained an identical client model without any adaptation, but with momentum and weight decay set to zero from the eighth epoch onwards, ensuring that raw loss values and gradient sizes remain comparable. As depicted in Fig. 10, the results of our experiments affirm that over the course of training, AutoAdapt has negligible impact on model convergence. The lines representing the constrained and unconstrained modes exhibit a high degree of overlap, signifying minimal deviation. With the exception of a few epochs immediately following the activation of constrained optimization, the gradient size and training loss remain virtually identical between the constrained and non-adaptive cases. This experiment serves to demonstrate the inherent stability of AutoAdapt, as it performs on par with non-adaptive training methods, assuring that the model remains intact throughout the training process.

E. Runtime Evaluation

We conducted runtime measurements to evaluate the practical feasibility of AutoAdapt in comparison to the classical method of Eq. 2 and the absence of any adaptation. The average runtime, based on ten experiments, was recorded for a client completing ten training epochs, and the results are presented in Tab. I. The first column represents the baseline

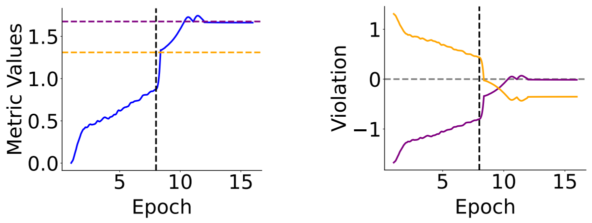

Fig. 9: The left side visualizes the Cosine metric (blue) with the horizontal lines indicating the valid value range. The vertical line indicates the start of the constrained optimization. The right side shows the constraint violation of the two inequality constraints (orange and purple) forming a range, where a value smaller than zero reflects a fulfilled constraint.

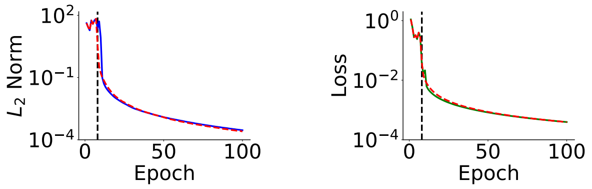

Fig. 10: The figure on the left depicts the model gradient size (blue) during 100 epochs of training. A vertical line indicates the initialization of constrained optimization. A dashed red line (almost identical to the blue line) represents the same experiment without any adaptation. On the right side, the figure displays the training data loss for the constrained case (green) and the non-adaptive case (red).

scenario without any adaptation. The second and third columns show the runtime for a single run with Eq. 2 and for nine runs, which are typically required to determine the optimal α value if one considers $\alpha = [0.1, 0.2, ..., 0.9]$. The third column corresponds to our default scenario, where constraint optimization is activated in the eighth epoch out of ten, resulting in three epochs under the constrained mode. The last column displays the results for only the last epoch being constrained. It is worth noting that in most of our experiments, the constraints were fulfilled within a few batches of one epoch.[13]

In comparison to the scenario without any adaptation, AutoAdapt introduces an additional runtime of 88% for three epochs and only 34% for one epoch. Although this may initially appear significant, it is a notable improvement compared to the 134% increase observed with Eq. 2 for one α. Moreover, when considering different α tests, Eq. 2 could potentially increase the runtime by up to 2007%, whereas with AutoAdapt, we managed to reduce this extreme value by 94% meaning a speed-up of 15.67 (15x faster). By providing a more efficient approach, AutoAdapt enables faster and more comprehensive testing of defenses, benefiting the progress of research in this domain.

F. Application of AutoAdapt

To demonstrate the effectiveness of AutoAdapt against actual defense mechanisms, we conducted experiments aimed at circumventing selected defense methods. Our objective was not to prove that AutoAdapt can bypass every existing defense, but rather to establish AutoAdapt as a valuable tool for adaptation that can be applied to a wide range of defense strategies. We hope that future researchers will adopt AutoAdapt as a fundamental method for evaluating the performance and robustness of FL defenses.

[13]Note, that AutoAdapt introduces an overhead of 34% compared to the "No Adaption" scenario, which is normal, as the adversary needs to compute an additional loss.

TABLE I: Runtime evaluation.

No Adaption	Eq. 2 α =0.9	Eq. 2 α =[0.1,...,0.9]	**AutoAdapt** 3 epochs	**AutoAdapt** 1 epoch
10.87s	25.46s	229.11s	20.50s	14.62s
±0%	+134%	+2007%	+88%	+34%
Saved time		±0%	**-91%**	**-94%**
Speed-up		x0	**11× faster**	**15× faster**

In our experiments, we first focused on a naïve clustering defense that utilized the Cosine metric and employed HDBSCAN [45] clustering on the resulting values. For unadapted models, this approach successfully distinguished two clusters, one for malicious models and one for benign models. However, when we applied AutoAdapt to adapt the models, the poisoned models became inconspicuous within the metric, resulting in the pruning of a single benign model (an outlier). Consequently, we achieved a 100% backdoor accuracy (BA) in the new aggregated model.

Subsequently, we targeted FoolsGold [31], which solely analyzes the last layer in terms of the Cosine distance. Consistent with our previous experiments, we successfully adapted the last layer to an inconspicuous state, aligning it with the global model. Although FoolsGold analyzes the distance between updates, our adaptation to the global model alone was sufficient to bypass this defense.

Lastly, we assessed Krum and M-Krum [13], which select either a single model or the average of a subset of models as the new global model based on the Euclidean distance to the global model. Here, we adapted the models to seemingly benign values in the Euclidean distance. By utilizing AutoAdapt to enforce similar distances across all models, the algorithms, while favoring the density of neighbors within the metric, selected poisoned models only, leading to a 100% BA in the new global model.

In conclusion, we assert that AutoAdapt is a valuable tool for adapting to FL defenses. It not only enhances the state-of-the-art on the attacker side but also empowers researchers to effectively and efficiently test defense mechanisms. By doing so, AutoAdapt enables the validation of claimed security guarantees against real-world adversaries.

VI. Discussion

Edge Case Scenarios. During our experimental analysis, we observed a consistent trend in which the metric values predominantly clustered towards the outer boundaries of the valid value range, as depicted in Fig. 9. While this behavior may be advantageous for the attacker in some cases, there are instances where it is undesired. For example, if the attacker intends for the trained models to exhibit a diverse distribution across the valid value range rather than converging towards a single value, this trend poses a challenge. To address this, the attacker can introduce additional value ranges that are subtly varied from the original range.

Furthermore, it is worth noting that there may be instances where a specific constraint is not completely fulfilled due to the penalty sum being exceedingly close to zero. In our conducted experiments, such scenarios did not present significant challenges, as detection algorithms still considered models as benign even if they slightly deviated from the actual benign range. However, if the adversary aims to strictly adhere to all constraints, they have the option to introduce a surrogate loss. For instance, assuming the true valid value range is defined as 0.4 to 0.5, the adversary could choose a constraint range of 0.42 to 0.48 for optimization purposes. However, for the purpose of retaining the best model, they can utilize the original range. This approach increases the likelihood of the model lying within the original range while satisfying the formal constraints.

An occurrence that may arise is when the penalty sum remains unchanged while not all constraints are satisfied. This situation serves as a clear indication that the value of α_{al} is too small. In such cases, a possible approach would be to dynamically increase α_{al} by, for example, multiplying it by a factor of 1.1. Similar concepts of increasing α_{al} based on certain rules can also be found in literature [55]. However, in our particular experimental setup, we managed to circumvent such scenarios by initializing α_{al} as described in Sect. IV.

Another situation that may arise is when constraint training is activated at the specified epoch, but the model either fully satisfies all constraints or approaches a state of near-perfection. In such cases, the value of α_{al} set by our algorithm tends to approach infinity, which can lead to undesirable outcomes. To prevent this, we assign a maximum value of 10 to α_{al}, ensuring a reasonable upper bound. Alternatively, another option could be to avoid initiating constraint training altogether, possibly by introducing a secondary range that acts as a buffer around the original range, providing some flexibility and preventing the setup from reaching extreme values.

Experiments with Other Optimization Methods Initially, our endeavor involved attempting to establish a logical and predetermined approach for manually weighting different adaption losses. We sought to devise a set of rules derived from the specific application scenario. However, it became apparent rather quickly that such an undertaking was not feasible, as it was highly contingent upon the intricacies of the individual scenario at hand. In pursuing this path, we inadvertently introduced additional hyper-parameters, thereby amplifying the computational burden associated with fine-tuning these parameters appropriately. Consequently, we concluded that such an approach is impractical when it comes to accommodating adaption to multiple constraints.

Motivated by Multi-Objective Optimization (MOO) research, we tried to find a pareto optimal [17] solution for the constraint optimization problem when adapting to metrics. We leveraged the method of Sener *et al.* [64] based on the MGDA algroithm [27]. However, the method did not work and produced broken models regarding the accuracies. We believe, that the reason for this is, that Sener *et al.* consider a system comparable to Multi-Task Learning (MTL) where both, shared and task-specific parameters exist within the model. However, our MOO problem optimizes only shared parameters (the whole model).

VII. Related Work

Considering the lack of directly comparable works to our paper, apart from the paper by Bagdasaryan *et al.* [8], which we have already compared in detail in the main body of our work, we present an overview of the research domain of poisoning FL. We highlight areas where AutoAdapt can serve as a valuable tool to advance the state of the art.

A. Poisoning Attacks in FL

Badnets [33] utilizes data poisoning as a technique by embedding a specific pixel pattern as a trigger into training samples. This method involves labeling samples containing the trigger with the target label associated with the backdoor. The trigger itself can range from a simple colored square to more intricate pixel structures or even a physical sticker.

So-called clean-label backdoors [72] employ a specific trigger, such as a pixel trigger, embedded within training samples. However, only samples from the target class are chosen for the poisoning process, ensuring that the samples retain their correct labels. Nevertheless, during training, the model establishes a correlation between the trigger and the target class, resulting in mislabeling of a sample from a different class that contains the embedded trigger during inference.

A semantic backdoor [8] leverages naturally occurring patterns, such as cars in front of striped backgrounds, as triggers. This approach enhances the stealthiness of the backdoor against data filtering techniques, as the samples themselves do not exhibit any abnormal patterns. During the data poisoning process, only the labels of the samples need to be altered to match the target label.

The edge case backdoor [74] involves utilizing samples that are naturally prone to being misclassified into the target class as triggers. In this method, only the labels of the samples need to be modified. The authors argue that such backdoors are difficult to detect, as they require minimal changes in the model parameters to alter the predictions for these specific samples.

The label flip backdoor technique involves replacing all samples from a specific label class with samples from a target class [12], [15]. While this backdoor is categorized as a targeted poisoning attack, it also has the unintended consequence of functioning as an untargeted attack on the original label class. This occurs because the attack aims to misclassify all samples belonging to the original source class.

Pervasive backdoors [21] are strategically embedded throughout an entire image, intentionally designed to be imperceptible to human observers, such as through the addition of random noise. The underlying rationale is that by perturbing the entire image in a uniform manner, the presence of the trigger goes unnoticed by viewers. One specific implementation of this technique is known as the Blend backdoor [21].

A recent attack paper, Chameleon [22], focuses on creating persistent backdoors by generating poisoned samples using an encoder trained through contrastive learning. This technique aims to align the poisoned samples more closely with the samples from the target class. While Chameleon prioritizes the longevity of the backdoor, AutoAdapt can be employed concurrently to adapt to the existing defense metrics.

Throughout our experiments, we extensively utilize three types of backdoors: the pixel trigger, the semantic backdoor, and the label flip backdoor. However, it is important to note that the focus of our research in this paper is distinct and independent from these approaches. Our proposed method, AutoAdapt, serves as a valuable tool to enhance the stealthiness of all these backdoors. Traditionally, the adaptation to metrics has predominantly relied on Eq. 2. In contrast, AutoAdapt offers superior results while requiring less computational resources, making it a more effective and efficient alternative.

B. Defenses against Poisoning Attacks

Krum [13] operates by utilizing the Euclidean distance metric to evaluate the proximity between local models. It calculates the distances from each local model to its neighboring models and chooses the one surrounded by the densest cluster as the new global model. M-Krum [13] extends this approach to select multiple models simultaneously. However, we showed in our evaluation that AutoAdapt can effortlessly bypass both methods through metric adaptation. Additionally, these methods inherently exhibit a high false negative rate (FNR) even in the absence of adversaries within the system.

AFA [48] employs a straightforward analysis of the Cosine distance between local models as foundation for it's filtering process. This distance metric can be adjusted through the incorporation of an additional loss function. As a result, AutoAdapt can be effectively utilized for adaptation purposes in relation to this defense mechanism as well.

FoolsGold [31] employs a weighting mechanism that assesses the contribution of each local model based on the analysis of cross-wise Cosine distances among model updates in the final layer of the DNN. However, this approach is susceptible to adaptive adversaries who can manipulate the last layers to appear benign. Furthermore, FoolsGold assumes an IID setting and poisoned local models that exhibit similar directions (referred to as Sybils). Additionally, the approach incorporates updates from previous FL rounds to enhance its performance. In our context, AutoAdapt can be utilized to appropriately adjust the parameters of the last layer to ensure that the Cosine distance falls within a valid range.

Naïve clustering approaches, such as those utilizing HDBSCAN [45], typically require the extraction of a metric such as the Cosine distance from local models in order to reduce dimensionality. Subsequently, these metric values are employed to cluster the models into two groups: One representing benign models and the other representing malicious models. This approach inherently leads to a high FPR or necessitates the introduction of an application-specific hyper-parameter that indicates when both clusters can be assumed benign. However, as demonstrated in our experiments, the use of AutoAdapt enables efficient adaptation to such a defense strategy.

FLAME [53] is a combination of two techniques, namely DF and IR. The approach utilizes pairwise Cosine distances between local models to cluster via HDBSCAN [45], and applies majority filtering to identify and remove adversaries before leveraging differential privacy methods for IR. Specifically, weight clipping is employed, considering the median Euclidean distance of the updates on the remaining local models, followed by the addition of noise to the aggregated model. This step not only aims to decrease the BA but also inherently reduces the MA. When adapting to the Cosine distance, FLAME performs comparably to a straightforward noising mechanism [71] as no models are filtered and only differentaial privacy is applied. The advantage of utilizing AutoAdapt is that it enables layer-wise adaptation to Cosine and Euclidean distances, which an adversary could try to leverage to adapt to the FLAME defense strategy regarding Cosine distance and minimize the noising and clipping mechanisms that consider the Euclidean distance.

DeepSight [60] is an advanced defense strategy, akin to FLAME [53], that integrates filtering techniques with the utilization of differential privacy. This approach incorporates two metrics: the Cosine distance between models and two more complex metrics, called DDif and NEUP, derived from the last layer. Both the Cosine distance and the values obtained from the last layer are candidates for optimization towards benign value ranges through the use of AutoAdapt. This characteristic renders AutoAdapt a valuable tool for adaption attempts of the DeepSight defense mechanism.

FLDetector [83] archives historical client updates, alongside global models, employing advanced mathematical approximation methods to predict forthcoming client model updates. Subsequently, these forecasts undergo a comparison with actual updates. Following an essential initial update phase required for FLDetector, the normed Euclidean distance is used as a metric for comparing real updates with the forecasts. This process empowers adversaries to actively monitor their updates, facilitating the localized application of the FLDetector algorithm. Then, the adversary can try to introduce model adaptations that align with the desired normed Euclidean distance with AutoAdapt.

Similar to the concept of Krum [13], Yin *et al.* [81] uses the coordinate-wise median or mean of the local models to construct the new global model based on the majority assumption. These approaches, called Trimmed-Mean and Trimmed-Median, respectively, are Robust Aggregation (RA) mechanisms, but reduce the MA compared to FedAVG. Especially, the parameters and, thus, the functionality of benign model models lying not centrally within all updates won't be considered. Bagdasaryan *et al.* [8] and Sun *et al.* [71] already proposed update clipping and nosing techniques, but Naseri *et al.* [49] showed, that differential privacy methods not only naturally harm the MA [8], but also can boost the BA when applied to benign FL clients. All of the Influence Reduction (IR) approaches and most RA methods suffer a drop in MA, especially in a setting without an ongoing attack. As one can try to adapt via AutoAdapt to approaches like Trimmed-Mean, it is hard to adapt to Differential Privacy (DF) approaches, as they affect also benign models.

FLTrust [16] operates under the assumption that the server possesses an auxiliary dataset that can be trusted. In each round, the server trains a benign model using this dataset. Subsequently, the Cosine similarity between the model trained by the server and the local models is employed to identify malicious outliers. However, it is important to note that adversaries have the capability to train a benign model and manipulate the Cosine similarity to align with the global model, potentially influencing the metric utilized by FLTrust. In such a scenario, AutoAdapt can be utilized to facilitate adaptation for this particular procedure.

VIII. Conclusion

The production of stealthy poisoned models, aimed at evading detection and filtering defenses, presents a significant challenge for attackers seeking to bypass FL defenses. In order to create such covert models, attackers adapt their models to exhibit benign metric values, which are used by the defense mechanisms to identify and filter out poisoned instances. The state-of-the-art adaptation process involves introducing a secondary optimization objective alongside the primary objective derived from the model's data input. The additional objective serves to impose constraints on the primary objective, and the balance between the two is achieved through a hyper-parameter.

We highlight the limitations of this approach, including the inability to adapt to multiple inequality constraints that define ranges of valid values, as well as the substantial computational effort required to determine the optimal balancing hyper-parameter. Consequently, we propose a novel method called AutoAdapt, which leverages the Augmented Lagrangian method for constraint optimization to automatically adapt to various metrics. Unlike existing approaches, AutoAdapt can establish valid value ranges without making assumptions about the scale of the corresponding objective functions.

We demonstrate the versatility of AutoAdapt across different FL setups and application scenarios, showcasing its effectiveness in generating superior poisoned models compared to state-of-the-art methods. Additionally, we report a notable speedup of being up to $15\times$ faster by applying AutoAdapt in the context of model adaptation. Ultimately, AutoAdapt represents a novel tool that enables attackers to effectively bypass defensive measures, while also serving as a valuable resource for researchers to evaluate the resilience and efficacy of defenses against sophisticated attacks from real-world adversaries.

Acknowledgment

This research was supported by the Private AI Collaborate Research Institute which is co-sponsored by Intel Labs (https://www.private-ai.org) and funded by the European Union's Horizon Europe research and innovation program under grant agreement No 101070537, project CROSSCON (Cross-platform Open Security Stack for Connected Devices).

References

[1] Health Insurance Portability and Accountability Act, 1996. https://www.govinfo.gov/content/pkg/PLAW-104publ191/pdf/PLAW-104publ191.pdf.

[2] California Consumer Privacy Act, 2018. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180SB1121.

[3] General Data Protection Regulation, 2018. https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[4] PyTorch, 2022. https://pytorch.org.

[5] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv preprint arXiv:1803.08375*, 2018.

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

[7] Eugene Bagdasaryan and Vitaly Shmatikov. Blind Backdoors in Deep Learning Models. *USENIX Security*, 2021.

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

[9] Shefali Bansal, Medha Singh, Madhulika Bhadauria, and Richa Adalakha. Federated Learning Approach towards Sentiment Analysis. *ICTACS*, 2022.

[10] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[11] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing Federated Learning through an Adversarial Lens. *ICML*, 2019.

[12] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning Attacks against Support Vector Machine. *ICML*, 2012.

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

[14] Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. Bad characters: Imperceptible NLP attacks. *IEEE S&P*, 2022.

[15] Di Cao, Shan Chang, Zhijian Lin, Guohua Liu, and Donghong Sun. Understanding Distributed Poisoning Attack in Federated Learning. *ICPADS*, 2019.

[16] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. *NDSS*, 2021.

[17] Yair Censor. Pareto Optimality in Multiobjective Problems. *Applied Mathematics and Optimization*, 1977.

[18] Fei Chen, Mi Luo, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated Meta-Learning with Fast Convergence and Efficient Communication. *arXiv preprint arXiv:1802.07876*, 2018.

[19] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated Learning Of Out-Of-Vocabulary Words. *arXiv preprint arXiv:1903.10635*, 2019.

[20] Xiaoyi Chen, Ahmed Salem, Dingfan Chen, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. BadNL: Backdoor Attacks against NLP Models with Semantic-preserving Improvements. *ACSAC*, 2021.

[21] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[22] Yanbo Dai and Songze Li. Chameleon: Adapting to Peer Images for Planting Durable Backdoors in Federated Learning. *arXiv preprint arXiv:2304.12961*, 2023.

[23] Erfan Darzidehkalani, Mohammad Ghasemi-rad, and P.M.A. van Ooijen. Federated Learning in Medical Imaging: Part I: Toward Multicentral Health Care Ecosystems. *Journal of the American College of Radiology*, 2022.

[24] Erfan Darzidehkalani, Mohammad Ghasemi-rad, and P.M.A. van Ooijen. Federated Learning in Medical Imaging: Part II: Methods, Challenges, and Considerations. *Journal of the American College of Radiology*, 2022.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *CVPR*, 2009.

[26] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 2012.

[27] Jean-Antoine Désidéri. *Multiple-gradient descent algorithm (MGDA)*. PhD thesis, INRIA, 2009.

[28] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus Mathematique*, 2012.

[29] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. *USENIX Security*, 2020.

[30] Christopher Frederickson, Michael Moore, Glenn Dawson, and Robi Polikar. Attack Strength vs. Detectability Dilemma in Adversarial Machine Learning. *IJCNN*, 2018.

[31] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The Limitations of Federated Learning in Sybil Settings. *RAID*, 2020.

[32] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor Attacks and Countermeasures on Deep Learning: A Comprehensive Review. *arXiv preprint arXiv:2007.10760*, 2020.

[33] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv preprint arXiv:1708.06733*, 2017.

[34] Gozde N Gunesli, Mohsin Bilal, Shan E Ahmed Raza, and Nasir M Rajpoot. FedDropoutAvg: Generalizable federated learning for histopathology image classification. *arXiv preprint arXiv:2111.13230*, 2021.

[35] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated Learning for Mobile Keyboard Prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CVPR*, 2016.

[37] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

[38] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[39] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. *Citeseer*, 2009.

[40] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 2020.

[41] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. *AAAI*, 2019.

[42] Yijing Li, Xiaofeng Tao, Xuefei Zhang, Junjie Liu, and Jin Xu. Privacy-Preserved Federated Learning for Autonomous Driving. *IEEE T-ITS*, 2022.

[43] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[44] Chih-Ting Liu, Chien-Yi Wang, Shao-Yi Chien, and Shang-Hong Lai. FedFR: Joint Optimization Federated Framework for Generic and Personalized Face Recognition. *AAAI*, 2022.

[45] Leland McInnes, John Healy, and Steve Astels. HDBScan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2017.

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

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

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

[49] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. Local and Central Differential Privacy for Robustness and Privacy in Federated Learning. *NDSS*, 2022.

[50] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 2008.

[51] Anh Nguyen, Tuong Do, Minh Tran, Binh X. Nguyen, Chien Duong, Tu Phan, Erman Tjiputra, and Quang D. Tran. Deep Federated Learning for Autonomous Driving. *IEEE IV*, 2022.

[52] Dinh C. Nguyen, Quoc-Viet Pham, Pubudu N. Pathirana, Ming Ding, Aruna Seneviratne, Zihuai Lin, Octavia Dobre, and Won-Joo Hwang. Federated Learning for Smart Healthcare: A Survey. *ACM Comput. Surv.*, 2022.

[53] 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.

[54] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning Attacks on Federated Learning-Based IoT Intrusion Detection System. *NDSS DISS*, 2020.

[55] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, New York, 2006.

[56] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. CUDA, release: 10.2.89, 2020.

[57] Xudong Pan, Mi Zhang, Beina Sheng, Jiaming Zhu, and Min Yang. Hidden Trigger Backdoor Attack on NLP Models via Linguistic Style Manipulation. *USENIX Security*, 2022.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

[59] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated Learning for Emoji Prediction in a Mobile Keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

[60] Phillip Rieger, Thien Duc Nguyen, Markus Miettinen, and Ahmad-Reza Sadeghi. DeepSight: Mitigating Backdoor Attacks in Federated Learning Through Deep Model Inspection. *NDSS*, 2022.

[61] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletarì, Holger R. Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N. Galtier, Bennett A. Landman, Klaus Maier-Hein, Sébastien Ourselin, Micah Sheller, Ronald M. Summers, Andrew Trask, Daguang Xu, Maximilian Baust, and M. Jorge Cardoso. The future of digital health with federated learning. *npj Digital Medicine*, 2020.

[62] Holger R Roth, Ken Chang, Praveer Singh, Nir Neumark, Wenqi Li, Vikash Gupta, Sharut Gupta, Liangqiong Qu, Alvin Ihsani, Bernardo C Bizzo, et al. Federated learning for breast density classification: A real-world implementation. *MICCAI*, 2020.

[63] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden Trigger Backdoor Attacks. *AAAI*, 2020.

[64] Ozan Sener and Vladlen Koltun. Multi-Task Learning as Multi-Objective Optimization. *Advances in Neural Information Processing Systems*, 2018.

[65] Micah Sheller, Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Federated Learning for Medical Imaging. *Intel AI*, 2018.

[66] 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. *Brain Lesion Workshop*, 2018.

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

[68] Santiago Silva, Boris A. Gutman, Eduardo Romero, Paul M. Thompson, Andre Altmann, and Marco Lorenzi. Federated Learning in Distributed Medical Databases: Meta-Analysis of Large-Scale Subcortical Brain Data. *IEEE ISBI*, 2019.

[69] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012.

[70] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. *USENIX Security*, 2018.

[71] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. Can You Really Backdoor Federated Learning? *arXiv preprint arXiv:1911.07963*, 2019.

[72] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-Consistent Backdoor Attacks. *arXiv preprint arXiv:1912.02771*, 2019.

[73] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[74] 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. *NIPS*, 2020.

[75] Zhaoxian Wu, Qing Ling, Tianyi Chen, and Georgios B. Giannakis. Federated Variance-Reduced Stochastic Gradient Descent With Robustness to Byzantine Attacks. *IEEE Transactions on Signal Processing*, 2020.

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

[77] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Fall of Empires: Breaking Byzantine-tolerant SGD by Inner Product Manipulation. *UAI*, 2020.

[78] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology*, 2019.

[79] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated Machine Learning: Concept and Applications. *TIST*, 2019.

[80] 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.

[81] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. *ICML*, 2018.

[82] Hongyi Zhang, Jan Bosch, and Helena Holmström Olsson. End-to-End Federated Learning for Autonomous Driving Vehicles. *IJCNN*, 2021.

[83] Zaixi Zhang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients. *KDD22*, 2022.

[84] Lingchen Zhao, Shengshan Hu, Qian Wang, Jianlin Jiang, Chao Shen, Xiangyang Luo, and Pengfei Hu. Shielding Collaborative Learning: Mitigating Poisoning Attacks Through Client-Side Detection. *PRDC*, 2021.

Appendix

A. Inequality Constraints

To retrieve the formula in box 5 of Fig. 3, we need to conduct some computations, which are presented here. With Eq. 14, the inequality constraint $h(x)$ is written as an equality constraint $h(x) + s^2$ to the cost of an additional parameter s^2, which needs to be optimized alongside with x. By substituting the equation of box 4 in Fig. 3, we now get Eq. 15.

$$\begin{aligned} L(x,s) =& f(x) + \frac{\alpha_{al}}{2}|h(x)+s^2|^2 + \mu(h(x)+s^2) \\ \mu =& \mu + \alpha_{al}(h(x_k)+s^2) \end{aligned} \tag{15}$$

First, we need to build the derivative of Eq. 15 with respect to s^2 and set the result to zero in order to find the optimal values s^{2^*} for s^2. The first step is visualized in Eq. 16.

$$\begin{aligned}L(x,s)_k =& f(x) + \frac{\alpha_{al}}{2}|h(x)+s^2|^2 + \mu(h(x)+s^2)\\ \frac{\partial L(x,s)_k}{\partial s^2} =& \alpha_{al}(h(x)+s^2)\cdot 1 + \mu\\ =& \alpha_{al}s^2 + \alpha_{al}h(x) + \mu\\ 0 =& \frac{\partial L(x,s)_k}{\partial s^2}\\ 0 =& \alpha_{al}s^2 + \alpha_{al}h(x) + \mu\\ s^2 =& -\frac{\mu}{\alpha_{al}} - h(x)\end{aligned} \tag{16}$$

Since s^2 is always positive, due to the power of two and only exists if $h(x) \leq 0$ is fulfilled, we can modify the solution by leveraging the max function as depicted in Eq. 17.

$$s^{2*} = \max(0, -\frac{\mu}{\alpha_{al}} - h(x)) \tag{17}$$

Since we are now aware of the optimal solution for s^2, this allows us to insert the result into Eq. 15 and get rid of the s^2 parameter. Therefore, we reformulate Eq. 15 to bring it into the desired from in Eq. 18.

$$\begin{aligned}L(x,s) =& f(x) + \frac{\alpha_{al}}{2}|h(x)+s^2|^2 + \mu(h(x)+s^2)\\ =& f(x) + \frac{\alpha_{al}}{\alpha_{al}}\cdot\frac{\alpha_{al}}{2}|h(x)+s^2|^2 +\\ & \frac{2\alpha_{al}}{2\alpha_{al}}\mu(h(x)+s^2)+\\ & \frac{1}{2\alpha_{al}}\mu^2 - \frac{1}{2\alpha_{al}}\mu^2\\ =& f(x) + \frac{1}{2\alpha_{al}}(\alpha_{al}^2|h(x)+s^2|^2+\\ & 2\alpha_{al}(h(x)+s^2)\mu+\\ & \mu^2 - \mu^2)\\ =& f(x) + \frac{1}{2\alpha_{al}}[(\alpha_{al}(h(x)+s^2)+\mu)^2 - \mu^2]\end{aligned} \tag{18}$$

We now insert the optimal s^{2*} into Eq. 18 to retrieve our final formula for box 5 of Fig. 3, which is depicted in Eq. 19 and Eq. 20.

$$\begin{aligned}&\alpha_{al}(h(x)+s^2)+\mu =\\ &= \alpha_{al}(h(x) + \max(0, -\frac{\mu}{\alpha_{al}} - h(x))) + \mu =\\ &= \alpha_{al}h(x) + \max(0, -\mu - \alpha_{al}h(x)) + \mu =\\ &= \max(0, \mu + \alpha_{al}h(x))\end{aligned} \tag{19}$$

$$\begin{aligned}&L(x,s) =\\ &= f(x) + \frac{1}{2\alpha_{al}}[(\alpha_{al}(h(x)+s^2)+\mu)^2 - \mu^2]\\ &= f(x) + \frac{1}{2\alpha_{al}}[(\max(0,\mu+\alpha_{al}h(x)))^2 - \mu^2]\\ &= f(x) + \frac{1}{2\alpha_{al}}(|\max(0,\mu+\alpha_{al}h(x))|^2 - \mu^2)\end{aligned} \tag{20}$$

For more detailed elaboration on this transformations, we refer the reader to [10], [55].

B. Update Rule Deviation

The update process of λ and μ in box 4 and 5 of Fig. 3 is driven by a comparison between the KKT stationarity conditions of the unconstrained Augmented Lagrangian function, as presented in Eq. 21, and the optimality conditions for constrained optimization problems. Specifically, the derivative with respect to x, denoted in Eq. 22 and Eq. 23 for equality and inequality constraints respectively is considered. The update rule is implemented through the Hestenes-Powell update of the Lagrange multipliers [10], [55]. By carefully comparing these formulas, we can identify the update rules for λ and μ as follows: $\lambda = \lambda + \alpha_{al}g(x)$ and $\mu = max(0, \mu + \alpha_{al}h(x))$.

$$\nabla_x\mathcal{L} = \nabla_x f(x) + \lambda\nabla_x g(x) \tag{21}$$

$$\begin{aligned}\nabla_x\mathcal{L} =& \nabla_x f(x) + \frac{\alpha_{al}}{2}\cdot\nabla_x|g(x)|^2 + \lambda\nabla_x g(x)\\ =& \nabla_x f(x) + \frac{\alpha_{al}}{2}\cdot 2g(x)\cdot\nabla_x g(x) + \lambda\nabla_x g(x)\\ =& \nabla_x f(x) + (\alpha_{al}g(x)+\lambda)\nabla_x g(x)\end{aligned} \tag{22}$$

$$\begin{aligned}\nabla_x\mathcal{L} =& \nabla_x f(x) + \frac{1}{2\alpha_{al}}\cdot\nabla_x(|max(0,\mu+\\ & \alpha_{al}h(x))|^2 - \mu^2)\\ =& \nabla_x f(x) + \frac{1}{2\alpha_{al}}\cdot 2\cdot max(0,\mu+\alpha_{al}h(x))\cdot\\ & \alpha_{al}\nabla_x h(x)\\ =& \nabla_x f(x) + max(0,\mu+\alpha_{al}h(x))\nabla_x h(x)\end{aligned} \tag{23}$$

C. Assignment of Fixed Alpha

To automatically assign the parameter α_{al}, it is essential to ensure that its value is sufficiently large. A suitable approach is to choose a value that scales both the main objective loss $f(x)$ and the penalty loss to the same magnitude. To achieve this, we determine the multiplier value during the initiation of the constraint optimization process in step 5 of Fig. 2. Initially, the μ_j values are set to zero. Subsequently, we equate the penalty term and $f(x)$ from box 7 in Fig. 3 and solve the corresponding equation for α_{al}, as illustrated in Eq. 24. The losses from the batch previous to the activation of the constraint training process are then used to determine the concrete value of α_{al}.

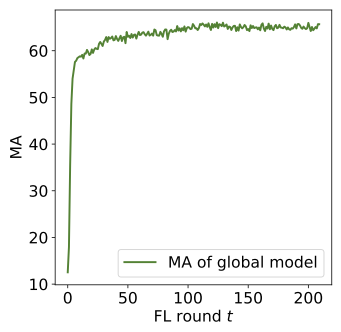

Fig. 11: MA for all FL rounds without attack in the default scenario with CIFAR-10 10 [39], IID distributed data, and 20 clients per round.

$$\begin{aligned}
f(x) =& \frac{1}{2\alpha_{al}} \sum_{j=1}^{m} (|max(0, \mu_j + \alpha_{al} h(x))|^2 - \mu_j^2) \\
=& \frac{1}{2\alpha_{al}} \sum_{j=1}^{m} (|max(0, 0 + \alpha_{al} h(x))|^2 - 0^2) \\
=& \frac{1}{2\alpha_{al}} \sum_{j=1}^{m} (|max(0, \alpha_{al} h(x))|^2) \\
=& \frac{\alpha_{al}}{2} \sum_{j=1}^{m} (|max(0, h(x))|^2) \\
\frac{\alpha_{al}}{2} =& \frac{f(x)}{\sum_{j=1}^{m} (|max(0, h(x))|^2)} \\
\alpha_{al} =& \frac{2f(x)}{\sum_{j=1}^{m} (|max(0, h(x))|^2)}
\end{aligned} \tag{24}$$

D. Quality of Pre-Trained Models

In our experiments (Sect. V), we initialized the global model using the parameters of a pre-trained model. In Fig. 11, we present the accuracy of the main task for the model in the default scenario, considering the following parameters: 2560 samples per client, learning rate (LR) of 0.01 with SGD optimizer with momentum of 0.9 and decay of 0.005, 20 clients per round, and a random seed of 42. After 50 rounds, the model's MA reaches a high and stable level, but continues to increase until round 125, indicating potential overfitting. At this point, it is advisable to adjust the hyper-parameters of the federation. For our experiments, we selected models from round zero, round 20, and round 50 to analyze the performance of AutoAdapt in different stages of convergence.