

Federated Learning Based Analysis of Certain Complex Multivariate Functions Given by Sample Only[★]

Nevena Mijajlović^{1,2}, Ajlan Zajmović², Miodrag J. Mihaljević^{1,3},
Wei Shao¹, Lianhai Wang¹, and Shujiang Xu¹

¹ Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center, Qilu University of Technology (Shandong Academy of Sciences), Jinan, China
nevenami@ucg.ac.me

² Faculty of Science and Mathematics, University of Montenegro, Podgorica, Montenegro
ajlan.z@ucg.ac.me (corresponding author)

³ Serbian Academy of Sciences and Arts, Belgrade, Serbia

Abstract. We consider the problem of optimizing a complex function $f(\cdot)$ given a sample consisting of M input-output pairs (*argument*, *function value*), $\mathcal{S} = \{(x_n, y_n)\}_{n=1}^M$, where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. We assume that the sample is available only in a distributed manner at K entities, where each entity controls a sample part $\mathcal{S}_k$ and $\mathcal{S} = \cup_{k=1}^K \mathcal{S}_k$. The problem is addressed employing a machine learning approach. Our goal is to generate a surrogate of the function based on the given sample, and to perform the optimization on the surrogate function. Due to the distributed nature of the sample we employ Federated Machine Learning (FML) to generate the surrogate function by joint efforts of all K entities that possess the sample parts. We call the entities involved in FML clients, and we assume that certain clients participate in FML in a malicious manner by poisoning training data or the model parameters they generate locally. Systematic experiments have been performed to study the characteristics of the surrogate function and its capabilities to provide reliable analysis of the original function depending on the sample available for training, and impacts of the training data poisoning, as well as locally generated model parameters.

Keywords: surrogate functions, neural networks, federated machine learning, algorithms, optimization models and methods, minimum evaluation, systematic numerical evaluation.

1. Introduction

We consider a surrogate $\hat{f}(x)$ of the function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, where $x = [x_1, x_2, \dots, x_d]$ is the input vector, d is the number of dimensions of the problem, and we have a single output y . The upper and lower bounds on the input (or design) vector are known and are denoted as $x_L \leq x \leq x_U$. After determining the input parameters (or the design space)

[★] This is an extended version of a conference paper, presented at the 2025 AI Conference, Serbian Academy of Sciences and Arts, Belgrade, October 8-9, 2025

of the model, we must select the data points to evaluate for building the surrogate model. The number of samples we choose to evaluate and their quality has a direct impact on the precision of the surrogate.

Machine learning models have demonstrated remarkable capabilities in building the surrogate functions by capturing complex relationships between input and output variables. The challenge, however, is that neural-network-based (NN) surrogate models must be trained on a sufficiently large dataset to enable generalization to unseen inputs, [10]. The value of these NN-based surrogate models depends on the assumption that the cost of generating the training dataset does not exceed the cost of conventional ensemble-based approaches. Evaluating the reliability of the constructed surrogate is one of the most important concerns because a flawed surrogate model can result in a misuse of computational resources, and produce negative effects on prediction, optimization, or the resulting analyses. Among the various NN architectures, multilayer feedforward neural network (MFFNNs) are among the most frequently used. In the MFFNN the neurons are organized into three layers: the input layer, the hidden layer, and the output layer. Note that the number of layers in MFFNN can be larger than just one and usually a convergence study is needed to find the right number of hidden layers. Similarly, the number of neurons in the hidden layers needs to be chosen based on a suitable convergence criterion. A number of issues regarding surrogate design and analysis have been reported, including the ones in [12], [17], [9], [1], [2], [4], [21]. Also, the surrogate approaches are of high interest in a number of engineering applications as reported, for example in [16], [3], [29], [6], [18], and may also be of interest for optimization in certain blockchain scenarios (like the ones reported in [13] and [25], for example). On the other hand, to the best of our knowledge, the approach we have employed, based on distributed model training, has not been considered.

Addressed Problem According to the above illustrative discussion of the options for the surrogate functions generation, this paper addresses the following problem: (i) We consider a variant of the approach known as “surrogate-assisted optimization” that replaces the direct optimization of an objective function with its surrogate model representation. The surrogate-assisted optimization algorithm includes an offline training phase prior to the optimization. The offline training aims to build an accurate model of the objective function and the effectiveness of the optimization depends on the initial training set size. In order to evaluate the effectiveness and correctness of the surrogate-assisted optimization, we first create a synthetic dataset by sampling points from a benchmark function within a predefined range of values. This dataset is then used to train, employing federated learning, a feed-forward fully connected neural network (multilayer perceptron) so that it can learn the relationship between the input variables and the function’s output value.

Motivation for the Work Developing surrogate functions is a well-recognized problem, and a number of techniques have been proposed. Neural-network-based surrogate modeling appears to be a promising approach, but there still are numerous issues that have not been entirely addressed, including those related to federated machine learning for training of neural networks for surrogate functions generation. Federated learning (FL) enables multiple data holders to collaboratively train a global model without exchanging raw data, thus preserving data privacy and ownership. However, in practical deployments,

FL systems are often plagued by data heterogeneity such as imbalanced sample sizes, label distribution skew, and incomplete local datasets which significantly degrade model convergence, accuracy, and fairness. It is well known that the distributed nature of FL could be misused by poisoning of training data as well as the parameters of the locally trained models, and these issues need to be addressed for obtaining additional insights in employment of FL for generation of the surrogate functions and optimizations over surrogate functions.

Our Results

The main contributions of this paper are as follows.

- We propose and analyze a distributed method for generating a surrogate function employing a neural network model and training based on FL in the scenarios where the required training data could not be collected in one place because of data sensitivity and/or data ownership rights.
- Consideration of additional insights regarding the impacts of the following issues: (i) training data sample dimension; (ii) heterogeneity of training data; (iii) number of entities that perform local training; and (iv) rate of malicious entities (attackers) that degrade the training process.
- A framework for optimization over the surrogate function under adversarial corruption, including data-poisoning and model-poisoning attacks by malicious FL participants.
- Systematic experimental analysis of the surrogate function characteristics and optimization. The analysis is performed over artificially collected training data in order to provide a controlled framework for comparison of the source and surrogate functions. Accordingly, we have selected certain functions known in the analytical form and evaluate pairs (*input variable, function output*) to obtain a sample for building the surrogate function employing FL. We have analyzed the following issues: (i) dependence of the surrogate function characteristics on the employed sample dimension and data heterogeneity; (ii) impact of training data poisoning and poisoning of locally generated model parameters; (iii) optimization capabilities of the surrogate function depending on the data employed and the training scenarios. The performed systematic experiments have provided additional quantitative insights regarding the considered training settings, the characteristics of the obtained surrogate function, and its capabilities for optimization of the source function.

Organization of the Paper The remainder of the paper is organized as follows. Section 2 provides more details on the background relevant to this paper. Section 3 presents a framework for consideration of additional insights regarding the model training process and the generated surrogate function. Section 4 summarizes the employed approach for finding a minimum over the surrogate function. Systematic experimental analysis of the proposed surrogate function generation and optimization is given in Section 5. Finally, the conclusions and future work are given in Section 6.

2. Background

2.1. Data and Models for Generating a Surrogate Function

Following the discussion in [12], we highlight several important issues related to the construction of surrogate models.

Sampling and Model validation. Since evaluating data points requires computing the true function (in our case, running an expensive simulation), sampling is the source of significant computational cost. Maintaining the quality of the surrogate model without incurring prohibitive sampling cost can be achieved by using appropriate sampling strategies [1]. It is important to select an optimal sample size, since more augmented data does not necessarily improve either local training or the aggregation of the final federated surrogate model. In particular, we point to [20], where an information-theoretic optimal sample size estimation (OSE) is proposed in order to provide reliable OSS estimation for industrial data augmentation, and where the relationship between OSS and dominant factors is theoretically considered. Sampling strategies can broadly be divided into stationary sampling and adaptive sampling. Stationary sampling methods are based on geometry or regular patterns (such as full/half factorial design or grid sampling) and approaches originating from the design of experiments (DOE) literature. In contrast, adaptive sampling starts from a relatively small initial set of samples, usually obtained by applying one of the stationary sampling strategies, and then determines new sample points sequentially. The goal of adaptive strategies is to reduce the number of samples required by evaluating only those points that are expected to most increase the precision of the resulting surrogate. Most adaptive sampling approaches rely on various criteria to balance the trade-off between exploring under-explored regions of the design space (exploration) and refining the regions close to already evaluated samples for improved performance (exploitation). A common use of such approaches is within surrogate-assisted optimization.

Validation of surrogate models is the procedure of evaluating the performance of the resulting surrogate, where, in addition to measuring predictive accuracy, validation techniques are often used to select a suitable surrogate model from a collection of candidate models and to tune its hyperparameters.

Model choice. Following [12], below we list several commonly used models for building surrogate functions.

Response surfaces and linear regression. The classical polynomial response surface model (RSM) remains one of the most frequently employed types of surrogate models in engineering design. In the RSM method, the surrogate is represented as a linear combination of polynomial functions (mostly linear) of the input variables (also called first-order RSM) and it has the form $\hat{f}(x, a) = a_0 + a_1x_1 + \dots + a_mx_m$, where the vector of coefficients $a = (a_0, \dots, a_m)^T$ is obtained by minimizing the sum of squared errors between the value from the expensive-to-compute function and the value predicted by the surrogate model.

Kriging. Kriging refers to a surrogate model based on Gaussian process modelling (and is sometimes called Gaussian process regression). A Kriging surrogate model can be formulated as $\hat{f}(x) = \sum_{j=1}^m a_j g_j(x) + \varepsilon(x)$, where $g_j(x)$ denotes the m independent (and known) basis functions which describe the trend of prediction of the mean at the point x , a_j denotes the unknown parameters, and $\varepsilon(x)$ denotes the random error at the point x which follows a normal distribution with a zero mean.

Radial basis functions (RBFs) Given n different sample points, RBF surrogates are written as $\hat{f}(x) = \sum_{i=1}^n w_i \phi(\|x - x^{(i)}\|_2)$, where w_i denotes the weight which is computed using the method of least-squares, and ϕ is the chosen basis function. There are several (symmetric) radial functions that can serve as a basis function.

Support vector regression (SVR). One of the main advantages of SVR methods is that they allow us to prescribe a margin δ within which errors present in the sampled data are tolerated without having a negative effect on the prediction capabilities of the resulting surrogate models. This can be particularly helpful when the sample data contain random errors, for instance due to a finite mesh size. By performing a mesh-sensitivity analysis, one can determine an appropriate value of δ .

Neural Networks (NN). The structure of a neural network is usually represented as a system of mutually interconnected “neurons”. Neurons in an artificial neural network (ANN) implement primitive nonlinear functions, while the connections between neurons have numerical weights that are computed from the input data. Three main elements guide the construction of an NN: the network topology (single or multiple layers), the primitive functions associated with neurons (activation functions), and the learning algorithm used to compute the corresponding weights.

The most widely used paradigm of an NN is the multilayer perceptron (MLP), which is based on feedforward (supervised) learning and the backpropagation algorithm. The learning capability of an MLP architecture can be improved by using more hidden layers and/or more neurons in the hidden layers (noting that in certain scenarios even a single layer is enough as reported in [24]).

NNs construct differentiable mapping functions through multiple layers of interconnected nodes. In a typical feedforward architecture, each layer transforms its inputs through a composition of linear operations and nonlinear activation functions. For a layer l , the output h_l is computed as

$$h_l = \sigma(W_l h_{l-1} + b_l),$$

where W_l represents the weight matrix, b_l is the bias vector, and σ is a differentiable activation function. The complete network function is a composition of these layer transformations

$$f(x) = h_L \circ h_{L-1} \circ \cdots \circ h_1(x).$$

This architecture ensures end-to-end differentiability, enabling efficient training via backpropagation and making NNs particularly suitable for gradient-based optimization tasks.

Derivative-based optimization. Derivative-based methods iteratively update model parameters by computing gradients and taking small steps in the negative gradient direction. These methods can be broadly categorized into two main approaches: first-order methods (which use only gradient information) and second-order methods (which also exploit curvature information through, e.g., the Hessian or its approximations).

2.2. Federated Machine Learning

Framework Federated learning (FL) (see [27], for example) is a machine learning approach that enables training models across decentralized devices or servers while keeping data localized, thereby ensuring data privacy and security. In particular, FL is a machine learning setup in which multiple clients collaboratively train a model while protecting

data privacy. This privacy-preserving paradigm allows collaborative model training without sharing raw data, thus addressing privacy concerns associated with centralized data storage.

In a typical FL setting, each device or server computes model updates based on its local data and shares only these updates with a central server or aggregator. A key advantage of federated learning is its ability to leverage data from multiple sources without compromising individual privacy. By training models locally on user devices or servers, FL enables personalized model updates while preserving data confidentiality.

According to [11], the basic workflow of FL can be summarized as follows.

- *Setup*: In the setup stage, a central authority, called the orchestrator, defines the learning task and selects the parties that will participate in the training process. The orchestrator then sends the initial model parameters to the selected parties.
- *Local training*: During the local training phase, each participant individually trains their local model using their own local dataset. This local training process resembles traditional machine learning but with a crucial distinction: The data remains localized and is not exchanged with external parties.
- *Model aggregation*: During the model aggregation phase, the local models are consolidated to create a unified global model. This procedure involves the participants transmitting their individually trained models to a central server. Subsequently, the server aggregates these models by averaging the parameters.
- *Model update*: During the model update phase, the updated global model is transmitted back to the participating parties. These parties then use the new parameters to update their respective local models. This iterative process, commonly referred to as the federated learning process, is repeated until the model reaches convergence and achieves the intended level of accuracy.

It is evident that FL eliminates the need for traditional centralized data collection and offloads part of the training workload from the server to the data owners, who also act as trainers in FL. FL preserves the privacy of trainers since they only upload model updates, while sensitive and personal data remain stored locally. Furthermore, FL is often more communication-efficient, since the communication overhead of uploading raw data on both the server and trainer sides is considerably higher than the overhead of transmitting local model updates.

For some additional issues on distributed machine learning and FL approaches we refer the readers to [22] and [7], respectively.

Attacks Against Federated Machine Learning Attacks in federated learning can exploit vulnerabilities in the communication channel, model, or data to gain unauthorized access, modify or steal data, inject noise, or poison the model. These attacks can have severe consequences, including the leakage of sensitive information, the degradation of model performance, and the compromise of the entire system. Here we address data-poisoning and model-poisoning attacks.

Data poisoning attacks (see, for example, [19]) present a significant challenge to the integrity and reliability of FL systems, where model training occurs collaboratively across decentralized devices. Data-poisoning attacks could appear during the training and updating stages. These attacks involve the deliberate injection of malicious data to corrupt the model's training process, ultimately undermining its performance. Given the decentral-

ized nature of FL and the lack of direct access to local data, detecting and mitigating these attacks becomes particularly difficult, especially in unsupervised scenarios where labeled data is unavailable. Such attacks compromise the overall model, leading to unreliable predictions and erroneous outcomes. In another setting, data poisoning produces the poisoned model that works correctly as usual on normal training data, but if the input data is crafted to include the triggering patterns, the model will predict a targeted result that is pre-chosen by the attacker. For an additional illustration we point to the findings recently reported in [15] and [5].

On the other hand, in a model poisoning attack scenario (see, for example, [26]), by uploading malicious models to the central server, attackers can: (i) cause the global model to diverge from the main task; or (ii) inject trigger patterns (backdoors) into the global model to misclassify a targeted class. Attack method (i) can often be detected by observing abnormal loss of the global or local models on a validation set, since the model will not converge to the expected result. Attack method (ii) is more difficult to detect because the global model does not diverge from the main task. At the same time, the global model may learn the backdoor pattern and produce targeted misclassification in image or text tasks.

In a federated learning environment, attackers can inject trigger patterns into the global model by uploading a poisoned local model instead of poisoning all training data, i.e., by performing a model-replacement attack.

A backdoor attack can typically be decomposed into the following three steps:

1. *Trigger definition*: The attacker secretly selects triggers in advance, typically choosing those with low-frequency characteristics that align with their specific objectives.
2. *Poisoned dataset generation*: The attacker selects a subset of the dataset, injects triggers into the samples, and modifies the labels to the target labels. The final training dataset is a combination of clean and poisoned samples.
3. *Model backdoor injection*: The attacker uses the poisoned dataset together with specific attack strategies to jointly train the model on the main task and the backdoor subtask.

Overall, the attacker's objective is to modify the model parameters from θ to θ_p . The parameters θ_p can be formulated as the solution of the following optimization problem:

$$\theta_p = \arg \min_{\theta} \left[\sum_{(x_i, y_i) \in D_c} \mathcal{L}(f(x_i, \theta), y_i) + \sum_{(x_j^*, y_t) \in D_p} \mathcal{L}(f(x_j^*, \theta), y_t) \right],$$

where $\mathcal{L}$ is the loss function, and D_c and D_p represent the clean and poisoned training sets, respectively. The poisoned sample $x_j^* = x_j \oplus \tau$ is obtained by injecting a trigger τ into the clean sample x_j , and y_t is the target output.

The first term in the optimization objective enforces that the backdoored model behaves similarly to the clean model on clean samples. The second term ensures successful backdoor implantation by enforcing the desired (targeted) model behaviour on poisoned samples.

An approach for the defense that is resilient to large-scale model poisoning attacks is reported in [8] that treats the sequence of local models sent by clients in each FL round as a matrix-valued time series. Then, it identifies malicious client updates as outliers in

this time series by comparing actual observations with estimates generated by a matrix autoregressive forecasting model maintained by the server.

Finally, we point out that an in-depth discussion of attacks on FL and corresponding defense techniques is available in [23].

3. Framework for Our Analysis of Surrogate Functions Generation and Certain Corruption Issues

This section introduces a framework for surrogate-based optimization and for evaluating the reliability of learned surrogate models in federated training and adversarial settings.

3.1. Architecture for Surrogate Generation and Challenges

The overall architecture considered for surrogate function generation is shown in Fig. 1.

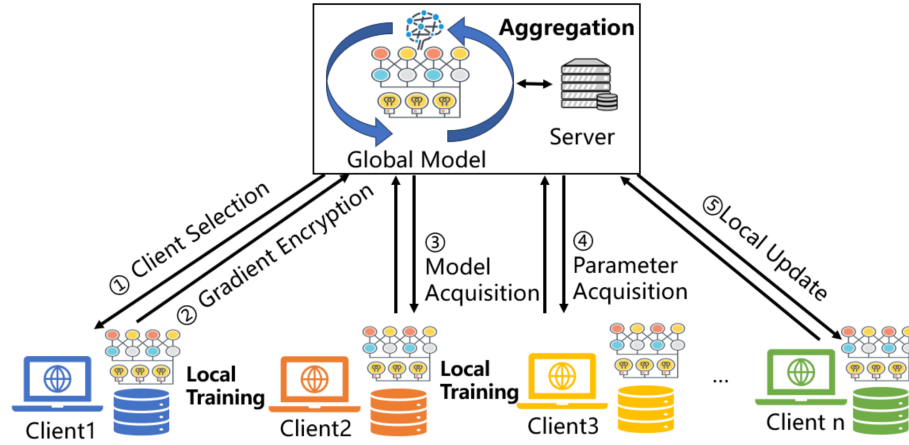


Fig. 1. Architecture for surrogate function generation.

Following Fig. 1, the FL training setup consists of the following main components:

- **Clients:** N entities that perform iterative local training. In communication round t , each client initializes from the global parameters from round $t - 1$ and produces updated local parameters.
- **Communication infrastructure:** a mechanism for exchanging model parameters/updates between clients and the server.
- **Aggregation entity (server):** aggregates client updates and produces the next global model parameters.

The main challenges of federated learning are illustrated in Fig. 2. In this work, we focus on two issues: (i) data heterogeneity and (ii) limited trust, i.e., the possibility of malicious client behavior.



Fig. 2. Main challenges of federated learning.

3.2. Surrogate Generation in the Decomposed Multivariate Space

Many optimization problems involve high-dimensional input spaces where only a subset of variables is actively optimized while the remaining variables are fixed due to constraints, design choices, or partial observability. The methodology used in this work follows this paradigm by decomposing the original d -dimensional problem into two parts: a fixed subvector and a free subvector. Let

$$x = (x_1, \dots, x_d) \in \mathcal{X} \subset \mathbb{R}^d, \quad f : \mathcal{X} \rightarrow \mathbb{R}$$

denote the original objective. The decomposition is defined by selecting an index split $d = d_f + d_v$, where the first d_f coordinates are fixed,

$$x_{1:d_f} = \bar{x}_{1:d_f},$$

and the remaining d_v coordinates are treated as decision variables,

$$z = x_{d_f+1:d} \in \mathcal{Z} \subset \mathbb{R}^{d_v}.$$

This leads to a conditional objective

$$g_{\bar{x}}(z) = f(\bar{x}_{1:d_f}, z),$$

which is the function minimized in the evaluation stage. The surrogate model is trained in the full input space (all d variables), while the decomposition is applied only at the minimum-search stage, where the fixed coordinates are enforced and the optimizer operates in the reduced space $\mathcal{Z}$.

3.3. Surrogate Generation

Collecting Data for the Surrogate The surrogate is trained on a dataset sampled from the domain of the source function. The source function is treated as an oracle that provides exact labels $y = f(x)$ for any queried input x . Data collection uses i.i.d. sampling over the bounded hyper-rectangle $\mathcal{X} = [x_L, x_U]^d$:

$$x \sim \mathcal{U}([x_L, x_U]^d), \quad y = f(x),$$

where $\mathcal{U}$ denotes the uniform distribution over the domain. Each federated client i receives a local dataset

$$\mathcal{S}_i = \{(x_{ij}, y_{ij})\}_{j=1}^m,$$

where m is the number of samples per client. All clients use the same sampling rule but generate independent local datasets through different random draws. This simulates decentralized data ownership where each participant observes only a local subset of the global data distribution.

A separate validation set is created independently of all client datasets:

$$\mathcal{S}_{\text{val}} = \{(x_k^{\text{val}}, y_k^{\text{val}})\}_{k=1}^{n_{\text{val}}}, \quad x_k^{\text{val}} \sim \mathcal{U}([x_L, x_U]^d), \quad y_k^{\text{val}} = f(x_k^{\text{val}}),$$

and is used exclusively for evaluation. This ensures that reported generalization metrics do not reuse training samples.

Global normalization. Since neural surrogates are sensitive to scaling, standardization is applied. A global feature scaler and a global target scaler are fitted using pooled client data and the transformed training quantities are

$$\tilde{x} = \frac{x - \mu_X}{\sigma_X}, \quad \tilde{y} = \frac{y - \mu_y}{\sigma_y},$$

where μ stands for mean and σ for standard deviation.

Training is performed in the normalized space $(\tilde{x}, \tilde{y})$. Evaluation metrics are computed on the original target scale by inverse-transforming predictions. Although the use of global statistics may not be fully consistent with the strict privacy assumptions of federated learning, we assumed them to be available in this study in order to ensure stable surrogate-model training.

Model for the Surrogate Generation The surrogate model is a fully-connected neural network (MLP) that maps the input vector to a scalar function value: $\hat{f}_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$. The deployed architecture follows a sequence of linear layers with a smooth nonlinearity and dropout regularization. For a generic hidden-layer configuration $(h_1, h_2, \dots, h_L)$, the forward mapping is:

$$\begin{aligned} a_0 &= \tilde{x}, \\ a_\ell &= \text{Dropout}(\phi(W_\ell a_{\ell-1} + b_\ell)), \quad \ell = 1, \dots, L, \\ \hat{y} &= W_{L+1} a_L + b_{L+1}, \end{aligned}$$

where $\phi(\cdot)$ denotes the activation function (GELU or SiLU depending on the experimental variant). Dropout is used to reduce overfitting and improve stability under limited

per-client sample sizes. In our implementation, we use an MLP with a small number of hidden layers, where the number of neurons per hidden layer is adapted to the input dimensionality.

Loss function and optimizer. Training minimizes a robust regression loss in normalized space, typically Smooth L1:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{(\tilde{x}, \tilde{y}) \in \mathcal{B}} \ell(\hat{f}_\theta(\tilde{x}), \tilde{y}),$$

with ℓ the Smooth L1 penalty. Optimization uses AdamW with learning rate η and weight decay λ :

$$\theta \leftarrow \theta - \eta \cdot \text{AdamW}(\nabla_\theta \mathcal{L}(\theta), \lambda).$$

This setup is suitable for continuous-valued function regression where outliers can occur and where robustness is desirable.

Training of the Model for Surrogate Generation Employing Federated Learning

Federated learning is simulated via a synchronous server–client protocol with N clients. Each client holds only its own dataset $\mathcal{S}_i$ and performs local updates. The server coordinates training in R rounds.

At the beginning of round t , the server broadcasts the global model parameters θ_t to all clients. Each client performs local optimization for E epochs using its local data:

$$\theta_t^{(i)} \leftarrow \text{LocalTrain}(\theta_t, \mathcal{S}_i; E, \eta).$$

Clients then send the updated parameters back to the server. Aggregation constructs the next global model θ_{t+1} from the set $\{\theta_t^{(i)}\}_{i=1}^N$.

FedAvg aggregation. The standard baseline aggregation is unweighted Federated Averaging:

$$\theta_{t+1} = \frac{1}{N} \sum_{i=1}^N \theta_t^{(i)}.$$

This treats each client contribution equally when local sample sizes are equal.

Robust aggregation (trimmed mean). To increase robustness against malicious or extreme client updates, robust aggregation can be applied. For each scalar model parameter component $\theta[k]$, the server forms the set of client values $\{\theta^{(i)}[k]\}_{i=1}^N$, sorts them and discards the smallest and largest $\lfloor \beta N \rfloor$ values, where $\lfloor \cdot \rfloor$ denotes the floor function. The trimmed mean is then computed over the remaining indices:

$$\theta_{t+1}[k] = \frac{1}{N - 2\lfloor \beta N \rfloor} \sum_{i \in \mathcal{I}_{\text{keep}}(k)} \theta_t^{(i)}[k],$$

where $\mathcal{I}_{\text{keep}}(k)$ denotes the set of clients retained for component k . This procedure is performed independently for each parameter component.

Centralized baseline. For selected experiments, we additionally train a centralized baseline model as a non-federated reference. The centralized model uses the same MLP architecture, loss function, optimizer, global input and output scalers, and clean validation procedure as the federated model. It is trained on the pooled union of all client datasets.

To ensure a fair comparison, the centralized baseline is trained under the same nominal optimization budget as the federated model, i.e., with the same total number of training epochs corresponding to the federated communication rounds and local updates. When corruption is present, the centralized baseline is trained on the pooled dataset obtained after applying the same corruption mechanism to the corresponding client data. In this way, the comparison isolates the effect of the training regime from that of the underlying data.

Surrogate–oracle comparison. Two complementary evaluation views are used:

- **Predictive accuracy:** the surrogate is evaluated on $\mathcal{S}_{\text{val}}$ using RMSE (Root Mean Square Error) on the original output scale.
- **Optimization fidelity:** the surrogate is used to propose a minimizer under fixed-coordinate constraints, and the quality is measured by a GAP metric (defined in Section 4).

The predictive view assesses the global approximation quality of the surrogate, while the optimization view evaluates whether the surrogate remains reliable in regions relevant to minimization.

For selected experiments, both metrics are reported for the federated model and for the centralized baseline trained on the pooled data. This comparison provides an additional reference for interpreting the effect of federated training under both clean and corrupted conditions.

3.4. Possible Undetected Attacks

The attack analysis studies whether the learned surrogate model can be significantly biased while still appearing to perform reasonably under standard evaluation. The analysis is framed as an *undetected* setting: during training the server has no access to attack labels, and evaluation is always performed using the same global metrics (RMSE and GAP) computed on clean procedures and clean validation data.

Data Poisoning Attacks Data poisoning attacks are modeled as modification of the local training data at a subset of clients in order to bias the aggregated global surrogate. Our approach selects a set of K malicious clients,

$$\mathcal{M} \subset \{1, \dots, N\}, \quad |\mathcal{M}| = K,$$

and replaces each affected local dataset $\mathcal{S}_i$ with a poisoned version $\tilde{\mathcal{S}}_i$:

$$\mathcal{S}_i \rightarrow \tilde{\mathcal{S}}_i, \quad i \in \mathcal{M}.$$

For each malicious client, a fraction ρ of its samples is poisoned using a trigger-based mechanism:

- **Input manipulation:** selected inputs are perturbed by adding a fixed trigger vector Δ and clipping to the feasible domain,

$$x \leftarrow \Pi_{[x_L, x_U]^d}(x + \Delta),$$

- **Label manipulation:** targets for the same samples are overwritten to a constant attack target y_{bd} .

This produces a consistent spurious rule that the surrogate can learn, namely that inputs containing the trigger pattern should be mapped toward the attacker-chosen output. All remaining components of the training pipeline are kept unchanged so that observed degradation can be attributed to the poisoned data.

Model poisoning via additive white noise. In addition to data poisoning, we also consider a simple *model poisoning* variant in which malicious clients perturb their transmitted model parameters. Concretely, after completing local training, each malicious client adds i.i.d. Gaussian white noise to its parameter vector before uploading it to the server,

$$\theta_t^{(i)} \leftarrow \theta_t^{(i)} + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2 I),$$

where the noise is applied component-wise to all trainable parameters. In our experiments we set $\sigma = 0.4$ and vary the number of malicious clients K to quantify the impact of increasingly corrupted updates on both predictive accuracy (RMSE) and optimization fidelity (GAP).

Impact of the Attacks The impact is evaluated without any explicit detection mechanism, using two complementary criteria:

- predictive degradation on clean data, measured as RMSE on a clean, uniformly sampled validation set;
- optimization degradation, measured through the GAP metric obtained by comparing the real function value at the surrogate-suggested conditional minimizer against the true conditional minimum.

To study sensitivity to attacker prevalence, the analysis performs a sweep over the number of malicious clients K while holding all other hyperparameters fixed.

Intentionally Biased Training Sampling We study the effect of non-representative client-side training data by keeping the full federated learning pipeline unchanged and replacing uniform sampling with a deliberately biased distribution. The impact is evaluated using RMSE on a clean, uniformly sampled validation set and the GAP metric for conditional minimization quality, across varying samples per client (SPC).

4. Conditional Minimum Search

We consider a surrogate-assisted optimization approach that replaces direct optimization of an objective function with optimization of its surrogate model. The procedure consists

of an offline training stage followed by a minimum-search stage. First, a synthetic dataset is generated by sampling points from a benchmark function within a predefined input range, and a feed-forward neural network (MLP) is trained to learn the mapping from the input variables to the function value. After training, the neural network acts as a surrogate model, replacing the original function during the optimization stage.

The search for the function’s minimum is performed using the Differential Evolution algorithm. This choice is motivated by its derivative-free nature and its ability to search globally over bounded domains.

True conditional minimum. For a fixed prefix $\bar{x}_{1:d_f}$, the true constrained minimum is defined as:

$$z^* = \arg \min_{z \in \mathcal{Z}} f(\bar{x}_{1:d_f}, z), \quad f^* = f(\bar{x}_{1:d_f}, z^*).$$

Since no analytic minimum is assumed, a derivative-free global optimizer is used to approximate (z^*, f^*) . Differential Evolution (DE) is employed with bounded search:

$$z^* \approx \text{DE}(g_{\bar{x}}(z); \mathcal{Z}),$$

where $\mathcal{Z} = [x_L, x_U]^{d_v}$.

Surrogate-guided conditional minimum. The surrogate $\hat{f}_{\theta}$ defines a proxy objective

$$\hat{g}_{\bar{x}}(z) = \hat{f}_{\theta}(\bar{x}_{1:d_f}, z),$$

and the surrogate-minimizer is approximated similarly:

$$\hat{z} = \arg \min_{z \in \mathcal{Z}} \hat{g}_{\bar{x}}(z) \approx \text{DE}(\hat{g}_{\bar{x}}(z); \mathcal{Z}).$$

To ensure consistent evaluation, DE uses fixed hyperparameters across all runs and a fixed seed. Typical settings include bounds per variable, population size controlled by `popsize`, and a maximum number of iterations `maxiter`. The result is a candidate $\hat{z}$ proposed by the surrogate.

GAP metric. The GAP measures the degradation introduced by using the surrogate for the minimizer proposal:

$$\text{GAP} = f(\bar{x}_{1:d_f}, \hat{z}) - f^*.$$

The key point is that the first term uses the *real* function evaluated at the surrogate-suggested candidate, not the surrogate prediction. This isolates the optimization guidance quality from pure prediction error.

5. Experimental Results over Artificial Data

To investigate the issues discussed in the previous sections, we conduct experiments on artificially generated data as follows: (i) We select an appropriate function known in analytic form. (ii) Using the selected function, we generate a sample for constructing its surrogate. (iii) Based on this sample, we build the surrogate function, evaluate its characteristics, and compare them with those of the original function.

In our experiments, we consider the following two benchmark functions: Ackley and Rastrigin [28].

The experiments were conducted on a 64-bit x64 system equipped with an Intel Core i5-6300U CPU (2.40–2.50 GHz), 8 GB RAM, a 238 GB NVMe SSD (WDC PC SN520), and integrated Intel HD Graphics 520.

In Table 1 we summarize abbreviations used in this section.

Table 1. Essential abbreviations used in this section.

Abbrev.	Full name	Meaning
FL	Federated Learning	Distributed training with server aggregation of client updates.
FedAvg	Federated Averaging	Server averages client parameters/updates each round.
N	Number of clients	Total clients participating (e.g., $N = 50$).
R	Communication rounds	Number of federated aggregation rounds.
E	Local epochs	Local training epochs per round at each client.
SPC	Samples per client	Number of training samples per client.
MLP	Multi-Layer Perceptron	Feed-forward neural network used as a surrogate.
GELU	Gaussian Error Linear Unit	Activation function used in the MLP.
SiLU	Sigmoid Linear Unit (Swish)	Activation function used in the MLP.
RMSE	Root Mean Squared Error	Validation error (reported on original scale).
GAP	Conditional-minimization gap	Difference between true f at surrogate-min point and true conditional minimum.
DE	Differential Evolution	Derivative-free optimizer used for conditional minimization.
K	Number of malicious clients	Number of adversarial clients in the federation.

Unless stated otherwise, Table 2 summarizes the common experimental settings used across all experiments. Function-specific choices such as dimensionality, surrogate architecture, activation, training schedule, samples per client, attack parameters, and aggregation variants are specified in the corresponding subsections. Also, AdamW and differential evolution are used with default values for all hyperparameters not explicitly specified.

5.1. Ackley 4D

We use the standard Ackley function defined for $x \in \mathbb{R}^d$ as

$$f(x) = -a \exp \left(-b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2} \right) - \exp \left(\frac{1}{d} \sum_{i=1}^d \cos(cx_i) \right) + a + e,$$

Table 2. Common experimental settings used across all experiments. Experiment-specific settings are given in the corresponding subsections.

Parameter	Value
Domain of each coordinate	$[-5.12, 5.12]$
Activation	GELU and SiLU
Number of clients	$N = 50$
Learning rate	10^{-3}
Optimizer	AdamW
Weight decay	10^{-4}
Loss function	SmoothL1Loss (Huber)
Batch size	$B = 128$
Input normalization	global standardization (StandardScaler)
Target normalization	global standardization (StandardScaler)
Validation set size	$n_{\text{val}} = 5000$
Validation sampling	clean uniform sampling over the domain
Differential Evolution population size	$\text{popsize} = 30$
Maximum number of DE iterations	$\text{maxiter} = 300$
Random seed	42

with the standard parameters $a = 20$, $b = 0.2$, and $c = 2\pi$. In our experiments we set $d = 4$ and restrict the search space to $x \in [-5.12, 5.12]^4$.

For consistency across all Ackley 4D experiments, we keep the same conditional cross-section $(x_1, x_2) = (0.5, -1.5)$ and optimize only (x_3, x_4) within $[-5.12, 5.12]^2$ (see Section 4).

For the Ackley 4D experiments, we use an MLP surrogate with architecture $4 \rightarrow 32 \rightarrow 16 \rightarrow 1$, dropout rate $p = 0.1$, and both GELU and SiLU activations. Federated training uses $R = 12$ communication rounds and $E_{\text{local}} = 10$ local epochs per round.

Although a lower GAP is more often achieved with the SiLU activation, the overall trends are similar for GELU and SiLU. We therefore use both activations throughout this section to verify whether the choice of activation materially affects surrogate quality and the downstream conditional minimization.

We first evaluate the clean setting with $N = 50$ clients and FedAvg. For each value of SPC we train a federated MLP surrogate and measure two complementary metrics on a clean validation set: RMSE (predictive accuracy on the original scale) and GAP (quality of the surrogate when used for conditional minimization). The results for a sweep of SPC are summarized in Figures 3 and 4.

To better interpret what the surrogate learns geometrically, we additionally visualize 2D cross-sections of the 4D function. Specifically, we fix $(x_1, x_2) = (0.5, -1.5)$ and plot the surface over $(x_3, x_4) \in [-5.12, 5.12]^2$. This allows a direct comparison of the true landscape against the learned surrogate under different conditions. In the following figures we show: (i) the true cross-section, (ii) federated surrogates trained on clean data ($K = 0$), (iii) data-poisoned training with $K = 25$ malicious clients, and (iv) a model-poisoning case where malicious clients perturb parameters before aggregation. The two SPC regimes (450 and 1500) illustrate how data availability interacts with robustness.

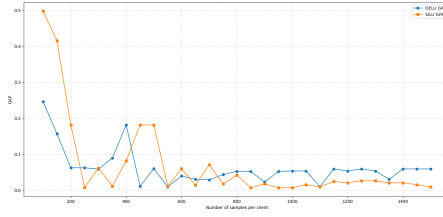


Fig. 3. Clean training: GAP vs. samples per client (GELU vs. SiLU).

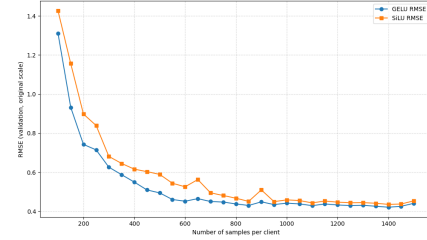


Fig. 4. Clean training: RMSE vs. samples per client (GELU vs. SiLU).

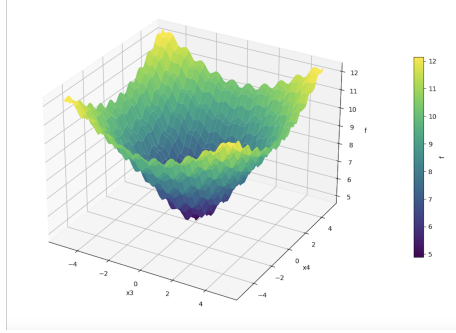


Fig. 5. True Ackley cross-section at $(x_1, x_2) = (0.5, -1.5)$.

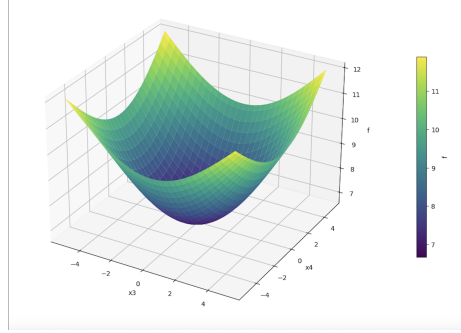


Fig. 6. Surrogate cross-section: SPC= 450, $K = 0$.

The cross-section of the true Ackley function is shown in Fig. 5. The clean surrogates in Figures 6 and 7 indicate how increasing SPC improves the fidelity of the learned surface in this slice. Next, we visualize data-poisoned surrogates in which $K = 25$ clients manipulate a fraction of their local training data. Specifically, we show the surrogate cross-sections for $SPC = 450$ (Fig. 8) and for $SPC = 1500$ (Fig. 9).

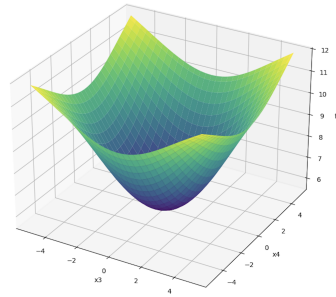


Fig. 7. Surrogate cross-section: SPC= 1500, $K = 0$.

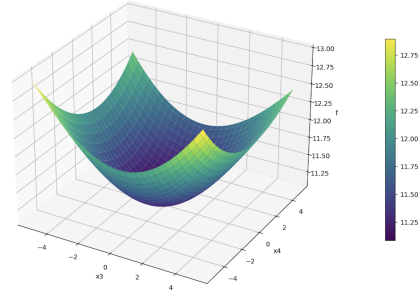


Fig. 8. Surrogate cross-section: SPC= 450, data poisoning, $K = 25$.

Finally, in Fig. 10, we show the surrogate-function cross-section trained with $SPC = 450$ samples per client under a model-poisoning attack, where $K = 25$ clients add i.i.d.

Gaussian white noise to their model parameters before transmitting the updates to the server.

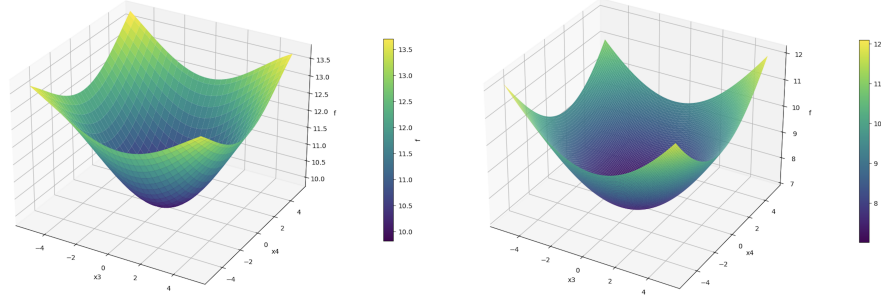


Fig. 9. Surrogate cross-section: SPC= 1500, data poisoning, $K = 25$. **Fig. 10.** Surrogate cross-section: SPC= 450, model poisoning, $K = 25$.

Data poisoning Compared to the baseline setting where all clients train on clean Ackley samples, we introduce a data-poisoning on a subset of clients and study how the number of malicious participants affects the learned surrogate.

We keep the federation size fixed at $N = 50$ and consider two fixed data regimes, $SPC \in \{200, 450\}$, while sweeping the attacker count $K \in \{0, 1, \dots, 25\}$. We selected these regimes because they achieve the lowest GAP while using a reasonable number of samples per client. For each K , we randomly select K clients to be malicious. Each malicious client poisons a fraction $\rho = 0.6$ of its local samples by adding a fixed trigger $\Delta = [3.0, -3.0, 0.0, 0.0]$, with clipping to $[-5.12, 5.12]^4$ and overwriting the corresponding labels with $y_{bd} = 18.0$. All other training components remain identical to the baseline (same model, scaling, optimizer, rounds, and local epochs). Evaluation is performed on a clean validation set, reporting RMSE (original scale) and the GAP metric; we then plot RMSE vs. K and GAP vs. K for both SPC settings. In Figures 11 and 12, we present the obtained results for the GELU activation.

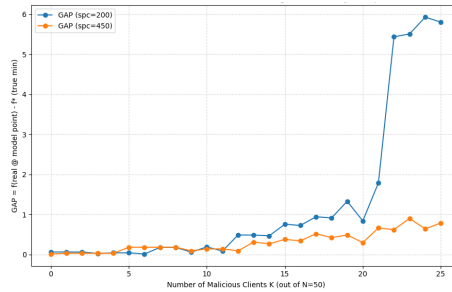


Fig. 11. Data poisoning (GELU): GAP vs. K (SPC 200, 450).

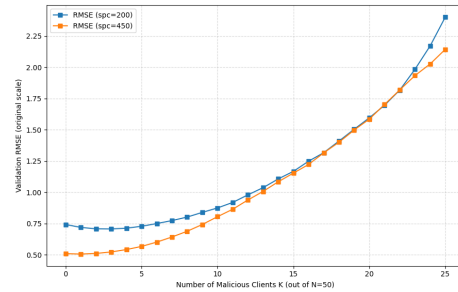


Fig. 12. Data poisoning (GELU): RMSE vs. K (SPC 200, 450).

We perform an analogous sweep using a SiLU-activated surrogate, considering $SPC \in \{200, 250, 350\}$. We keep the same architecture, optimizer, FedAvg setup, and attack definition, and report GAP/RMSE on the same clean validation protocol (Figures 13 and 14).

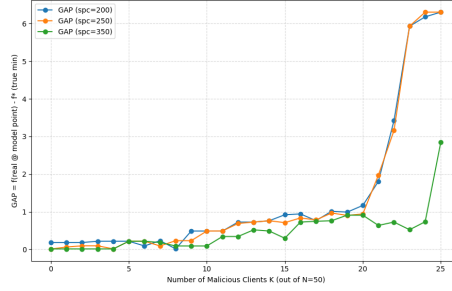


Fig. 13. Data poisoning (SiLU): GAP vs. K (SPC 200, 250, 350).

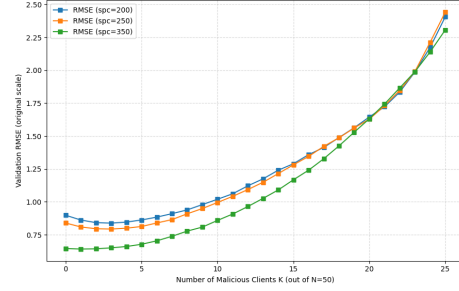


Fig. 14. Data poisoning (SiLU): RMSE vs. K (SPC 200, 250, 350).

To mitigate the impact of adversarial updates, we additionally replace FedAvg with coordinate-wise trimmed mean aggregation using $\beta = 0.3$. Concretely, for each parameter coordinate we discard the smallest and largest 30% of client values and average the remaining central values. This provides a simple robust alternative to plain averaging in the presence of outliers. Results with the GELU activation are shown in Figures 15 and 16, while the results with the SiLU activation are shown in Figures 17 and 18.

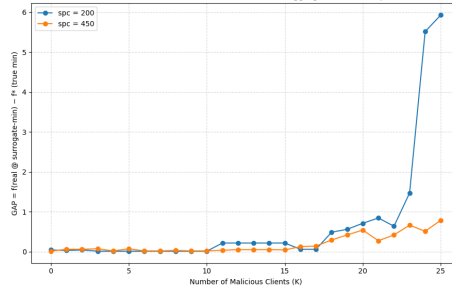


Fig. 15. Trimmed mean (GELU): GAP vs. K (SPC 200, 450).

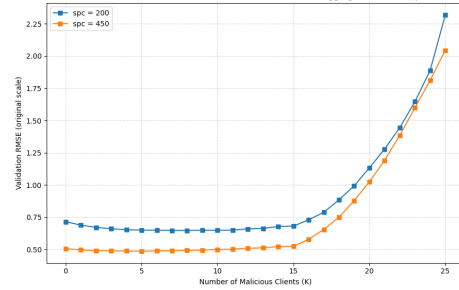


Fig. 16. Trimmed mean (GELU): RMSE vs. K (SPC 200, 450).

Comparison with a Centralized Baseline (GELU) For the Ackley 4D setting, we additionally compare the federated surrogate with a centralized MLP baseline. In this comparison, only the GELU activation is considered. The centralized model is trained on the pooled union of all client datasets, using the same network architecture, loss function, optimizer, global input and output scalars, clean validation protocol, and DE-based conditional-minimization procedure as the federated model. To ensure a fair comparison, the centralized baseline is trained under the same nominal optimization budget as the federated model. The comparison is performed under data-poisoning conditions for

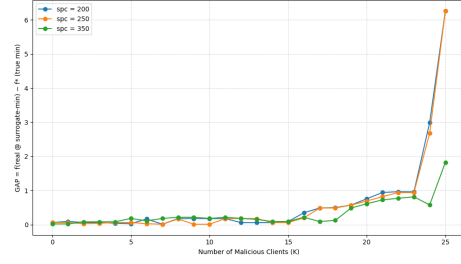
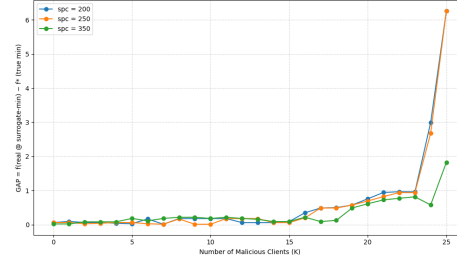


Fig. 17. Trimmed mean (SiLU): GAP vs. K (SPC 200, 250, 350). **Fig. 18.** Trimmed mean (SiLU): RMSE vs. K (SPC 200, 250, 350). $SPC \in \{200, 450\}$, and the resulting RMSE and GAP values are reported as functions of the number of malicious clients K . The corresponding results are shown in Figures 19 and 20.

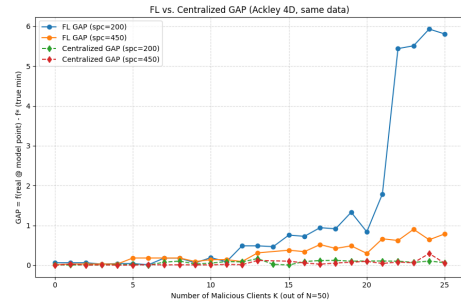
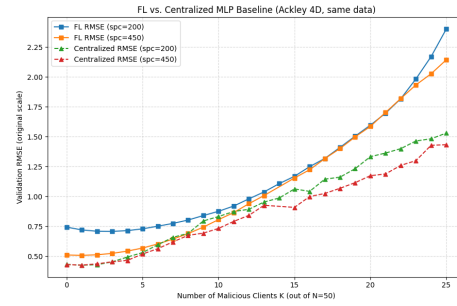


Fig. 19. Ackley 4D, GELU only: RMSE versus the number of malicious clients K for the federated model and the centralized federated model and the centralized baseline, for $SPC = 200$ and $SPC = 450$. **Fig. 20.** Ackley 4D, GELU only: GAP versus the number of malicious clients K for the federated model and the centralized federated model and the centralized baseline, for $SPC = 200$ and $SPC = 450$.

Model poisoning We also consider a model-poisoning setting where training data remain clean and uniformly sampled, but a fixed subset of K clients corrupt their transmitted updates by adding i.i.d. Gaussian white noise to the locally trained parameters before aggregation. This attack directly perturbs the server-side averaging process without altering the local input–output samples (as explained in Section 3). We again report RMSE on a clean validation set and GAP as functions of K , for both GELU and SiLU (Figures 21 and 22 show the results for GELU activation, while Figures 23 and 24 show the results for SiLU activation.)

Intentionally Biased Training Sampling This experiment keeps the full FL pipeline unchanged, but replaces uniform client-side training sampling with a deliberately biased distribution. Each client draws 80% of samples near the domain corners (small Gaussian white noise around the 2^4 corners) and the remaining 20% uniformly while rejecting points with $\|x\|_2 < 1.0$, and we then report RMSE and GAP versus $SPC \in \{100, 150, \dots, 1500\}$ on a clean uniform validation set. The results obtained with the GELU activation under intentionally biased training sampling are shown in Figures 25

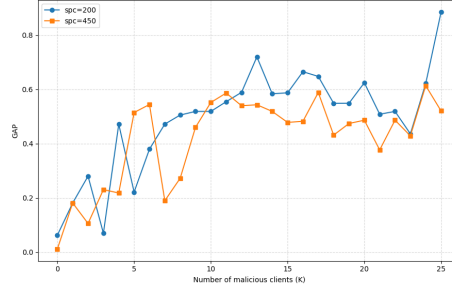


Fig. 21. Model poisoning (GELU): GAP vs. K ($\sigma = 0.4$).

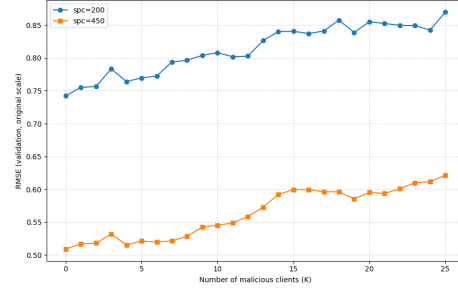


Fig. 22. Model poisoning (GELU): RMSE vs. K ($\sigma = 0.4$).

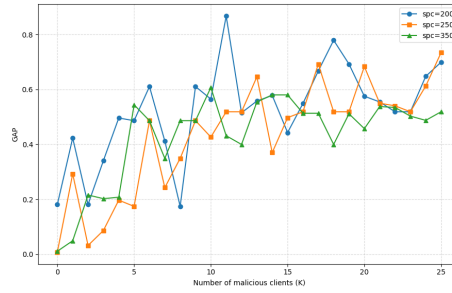


Fig. 23. Model poisoning (SiLU): GAP vs. K ($\sigma = 0.4$).

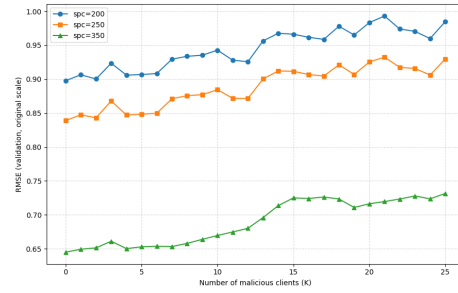


Fig. 24. Model poisoning (SiLU): RMSE vs. K ($\sigma = 0.4$).

and 26, while the corresponding results obtained with the SiLU activation are shown in Figures 27 and 28.

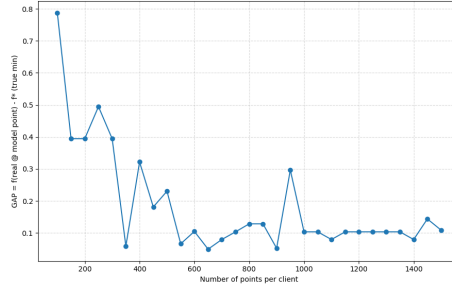


Fig. 25. Biased sampling (GELU): GAP vs. samples per client.

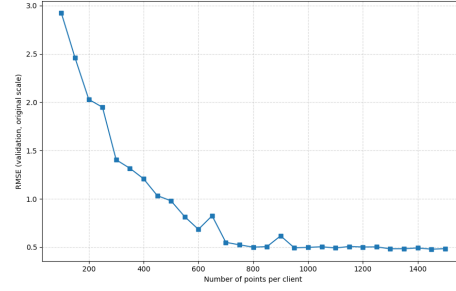


Fig. 26. Biased sampling (GELU): RMSE vs. samples per client.

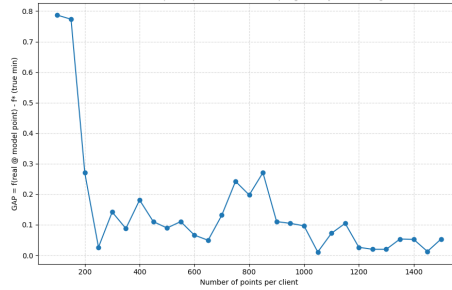


Fig. 27. Biased sampling (SiLU): GAP vs. samples per client.

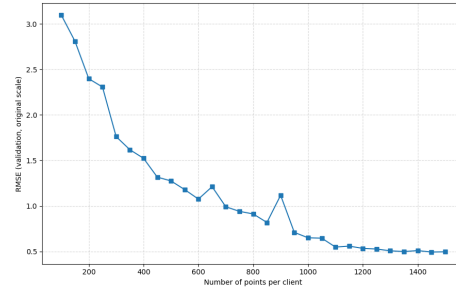


Fig. 28. Biased sampling (SiLU): RMSE vs. samples per client.

5.2. Ackley 10D

In comparison to the Ackley 4D experiment, here we increase the dimensionality to $d = 10$ and generate all training and validation samples uniformly from the hypercube $[-5.12, 5.12]^{10}$. The surrogate architecture is adapted to the higher-dimensional input by using a deeper MLP, $10 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 1$, with dropout rate $p = 0.1$ and either GELU or SiLU activation, while the number of clients remains $N = 50$. For the GAP evaluation, the conditional optimization is performed in a larger free subspace: we fix $(x_1, \dots, x_5) = (0.5, 0.5, 1.0, 1.0, 0.5)$ and optimize $(x_6, \dots, x_{10}) \in [-5.12, 5.12]^5$, which makes the minimum search more challenging than in the 4D case. To compensate for the increased difficulty, we use a longer FL schedule ($R = 24$ rounds and $E = 6$ local epochs) and report GAP/RMSE as functions of SPC using the same evaluation protocol and random seed. The resulting GAP and RMSE trends for this setup are summarized in

Figures 29–32, where we report the performance as a function of SPC for both GELU and SiLU activations.

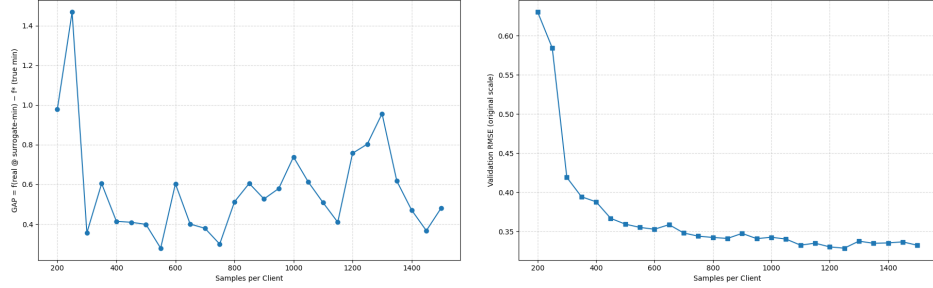


Fig. 29. Clean training (Ackley 10D, GELU): GAP vs. samples per client. **Fig. 30.** Clean training (Ackley 10D, GELU): RMSE vs. samples per client.

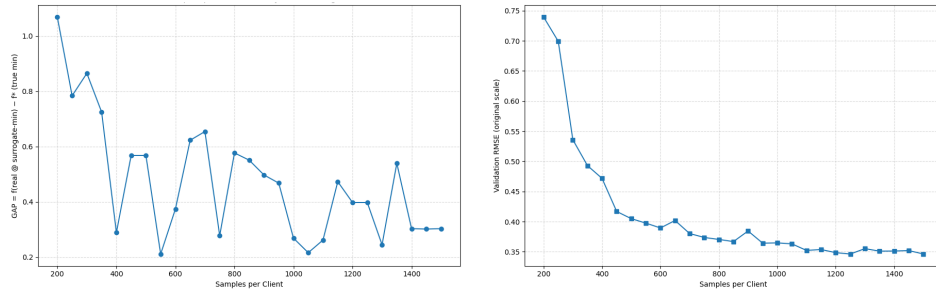


Fig. 31. Clean training (Ackley 10D, SiLU): GAP vs. samples per client. **Fig. 32.** Clean training (Ackley 10D, SiLU): RMSE vs. samples per client.

Data poisoning Compared to the clean baseline, we introduce K malicious clients whose local datasets are partially poisoned: a fraction $\rho = 0.40$ of their samples is modified by adding a fixed trigger vector $\Delta = [1.2, -1.2, 0.8, 0.8, -1.0, 0, 0, 0, 0]$ (clipped to $[-5.12, 5.12]^{10}$) and forcing the corresponding labels to the target value $y_{bd} = 12.0$. Consequently, the global input/target standardizers are fitted on the contaminated pooled data, so the server aggregates updates learned from a mixture of honest and poisoned client distributions. The impact of the backdoor attack is shown in Figures 33–36, which plot GAP and validation RMSE versus the number of malicious clients K for several SPC regimes, with GELU evaluated for $SPC \in \{300, 500, 550\}$ and SiLU for $SPC \in \{400, 500, 550\}$.

In addition to standard FedAvg, we also evaluate a robust aggregation rule based on the coordinate-wise trimmed mean with parameter $\beta = 0.3$. For each parameter coordinate, we discard the smallest and largest 30% of client values and average the remaining central values. We report the resulting GAP and RMSE versus K for both activations and for the selected SPC regimes in Figures 37–40.

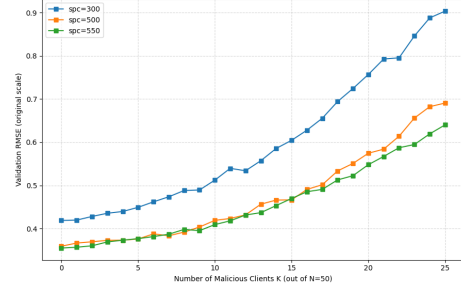
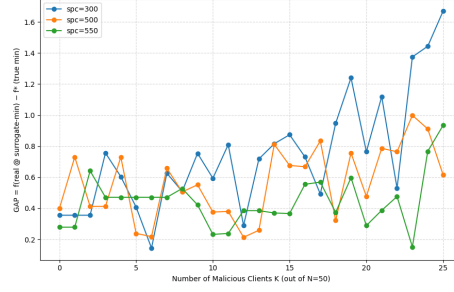


Fig. 33. Data poisoning (Ackley 10D, GELU): GAP vs. K (SPC = 300, 500, 550). **Fig. 34.** Data poisoning (Ackley 10D, GELU): RMSE vs. K (SPC = 300, 500, 550).

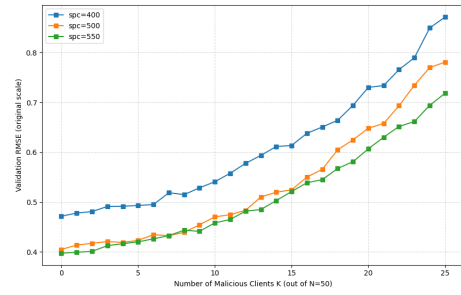
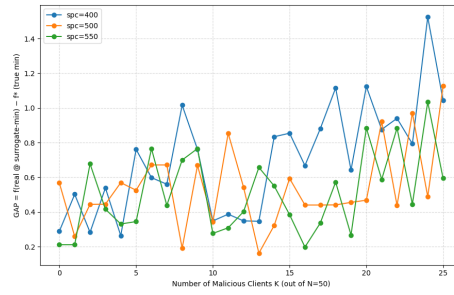


Fig. 35. Data poisoning (Ackley 10D, SiLU): GAP vs. K (SPC = 400, 500, 550). **Fig. 36.** Data poisoning (Ackley 10D, SiLU): RMSE vs. K (SPC = 400, 500, 550).

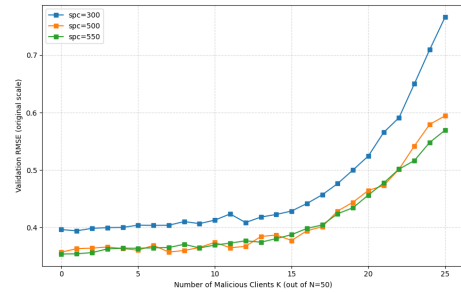
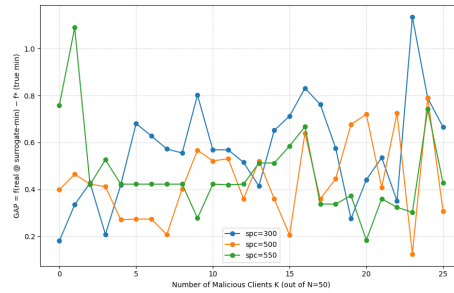


Fig. 37. Trimmed mean (Ackley 10D, GELU): GAP vs. K (SPC = 300, 500, 550). **Fig. 38.** Trimmed mean (Ackley 10D, GELU): RMSE vs. K (SPC = 300, 500, 550).

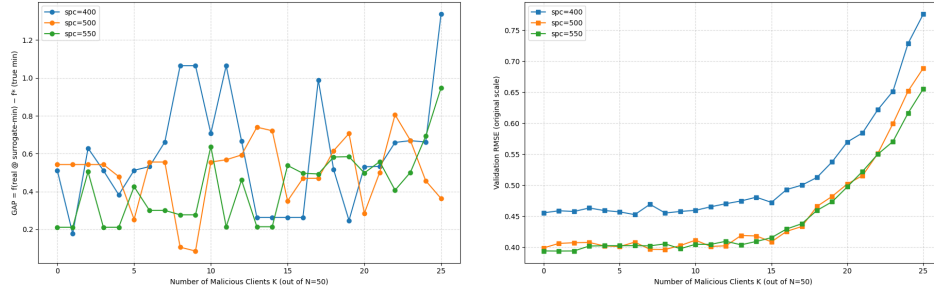


Fig. 39. Trimmed mean (Ackley 10D, SiLU): GAP vs. K (SPC = 400, 500, 550). **Fig. 40.** Trimmed mean (Ackley 10D, SiLU): RMSE vs. K (SPC = 400, 500, 550).

Model poisoning (Ackley 10D). We next study a *model-poisoning* scenario in the 10D setting. The surrogate is still trained on clean, uniformly sampled client data; however, in each round we designate a fixed subset of K clients as malicious and corrupt their transmitted updates by adding i.i.d. Gaussian white noise with standard deviation $\sigma = 0.4$ to the locally trained parameters before sending them to the server. The server then aggregates the (mixed) updates using FedAvg, and we quantify the impact of this attack by reporting the validation RMSE (on a clean uniform validation set) and the GAP metric as functions of K . Results for GELU and SiLU activations are shown in Figures 41–44.

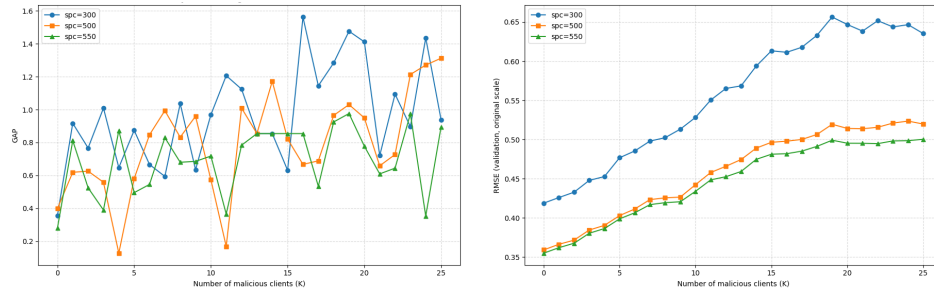


Fig. 41. Model poisoning (Ackley 10D, GELU): GAP vs. K ($\sigma = 0.4$). **Fig. 42.** Model poisoning (Ackley 10D, GELU): RMSE vs. K ($\sigma = 0.4$).

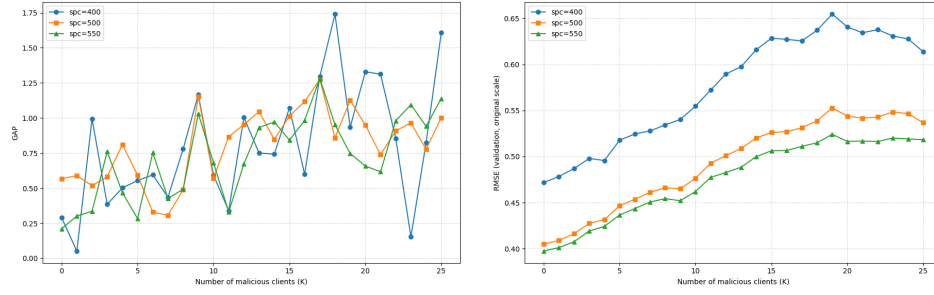


Fig. 43. Model poisoning (Ackley 10D, SiLU): GAP vs. K ($\sigma = 0.4$). **Fig. 44.** Model poisoning (Ackley 10D, SiLU): RMSE vs. K ($\sigma = 0.4$).

5.3. Rastrigin 4D

We consider the Rastrigin function in $d = 4$ dimensions defined by

$$f(x) = 10d + \sum_{i=1}^d (x_i^2 - 10 \cos(2\pi x_i)),$$

on the hypercube domain $x \in [-5.12, 5.12]^4$.

The surrogate model is a feed-forward MLP with architecture $4 \rightarrow 64 \rightarrow 32 \rightarrow 1$, using a dropout layer with $p = 0.05$ after each hidden layer and either GELU or SiLU activation. Federated training uses FedAvg with $N = 50$ clients, 12 communication rounds, and 10 local epochs per round.

For the GAP evaluation we fix $(x_1, x_2) = (0.5, -1.5)$ and perform conditional minimization over $(x_3, x_4) \in [-5.12, 5.12]^2$. Figures 45–48 summarize the results.

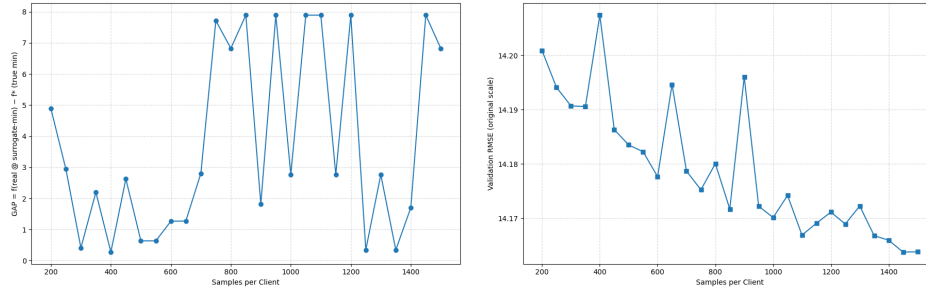


Fig. 45. Clean training (Rastrigin 4D, GELU): GAP vs. SPC. **Fig. 46.** Clean training (Rastrigin 4D, GELU): RMSE vs. SPC.

Data poisoning We next evaluate robustness to *data poisoning* in the Rastrigin 4D setting. The federation size remains fixed at $N = 50$, while we sweep the number of malicious clients $K \in \{0, 1, \dots, 25\}$. For each K , we randomly select K clients to be malicious. Each malicious client poisons a fraction $\rho = 0.30$ of its local samples by injecting the

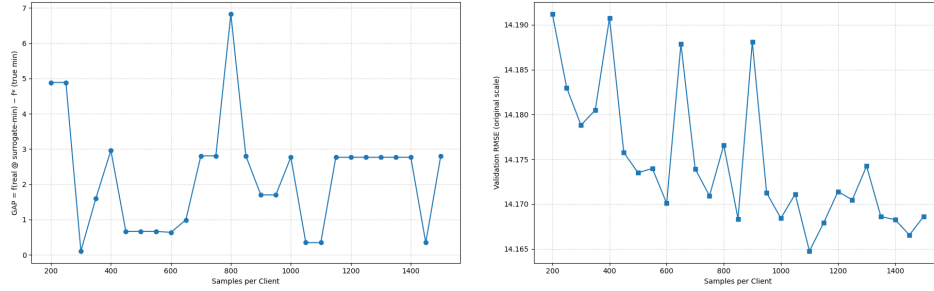


Fig. 47. Clean training (Rastrigin 4D, SiLU): GAP vs. SPC. **Fig. 48.** Clean training (Rastrigin 4D, SiLU): RMSE vs. SPC.

trigger $\Delta = [0.8, -0.8, 0.6, -0.6]$, with clipping to $[-5.12, 5.12]^4$, and overwriting the corresponding labels with the target value $y_{bd} = 60.0$. All other training components remain unchanged (same model, scaling, optimizer, R rounds, and E local epochs). Evaluation is always performed on a clean, uniformly sampled validation set. We report RMSE and GAP as functions of K for multiple SPC regimes and for both activations; the resulting curves are shown in Figures 49–52.

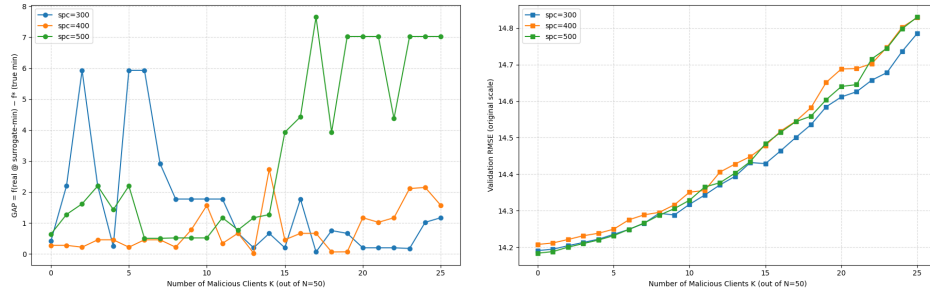


Fig. 49. Data poisoning (Rastrigin 4D, GELU, FedAvg): GAP vs. K (SPC = 300, 400, 500). **Fig. 50.** Data poisoning (Rastrigin 4D, GELU, FedAvg): RMSE vs. K (SPC = 300, 400, 500).

To mitigate the influence of poisoned clients, we also replace FedAvg with coordinate-wise trimmed mean aggregation using $\beta = 0.3$. In each round, the server discards the lowest and highest 30% of client parameter values per coordinate and averages the remaining values. We report GAP and RMSE versus K under trimmed mean for both activations in Figures 53–56.

Model poisoning Finally, we evaluate a *model-poisoning* attack on the Rastrigin 4D benchmark. All clients train on clean, uniformly sampled data; however, in each communication round a fixed subset of K malicious clients corrupts their transmitted updates by adding i.i.d. Gaussian white noise with standard deviation $\sigma = 0.4$ to the locally

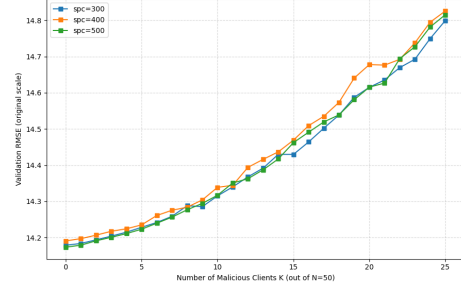
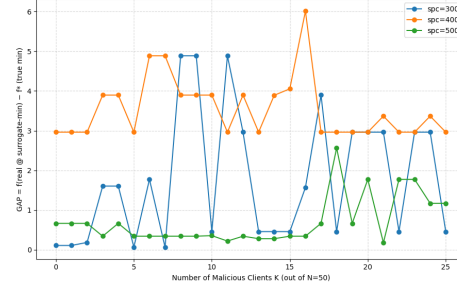


Fig. 51. Data poisoning (Rastrigin 4D, SiLU, FedAvg): GAP vs. K (SPC = 300, 400, 500). **Fig. 52.** Data poisoning (Rastrigin 4D, SiLU, FedAvg): RMSE vs. K (SPC = 300, 400, 500).

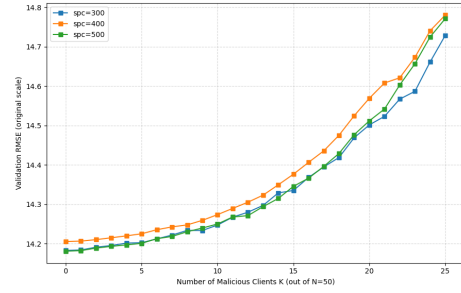
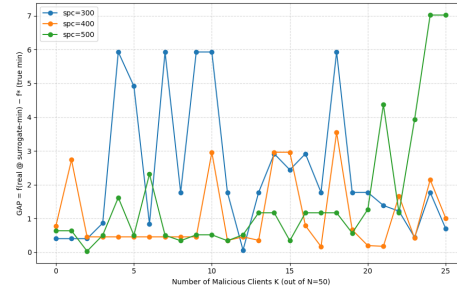


Fig. 53. Trimmed mean (Rastrigin 4D, GELU): GAP vs. K (SPC = 300, 400, 500; $\beta = 0.3$). **Fig. 54.** Trimmed mean (Rastrigin 4D, GELU): RMSE vs. K (SPC = 300, 400, 500; $\beta = 0.3$).

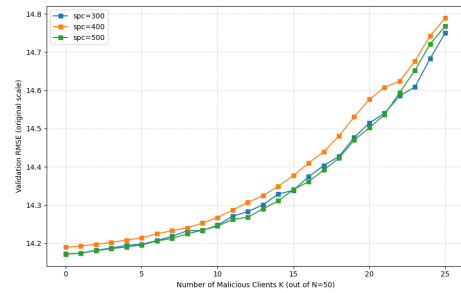
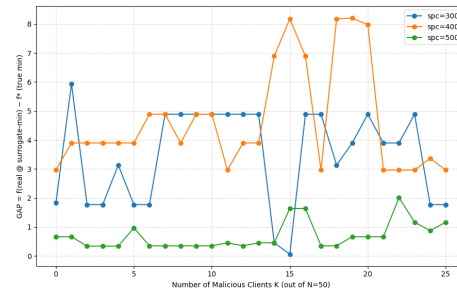


Fig. 55. Trimmed mean (Rastrigin 4D, SiLU): GAP vs. K (SPC = 300, 400, 500; $\beta = 0.3$). **Fig. 56.** Trimmed mean (Rastrigin 4D, SiLU): RMSE vs. K (SPC = 300, 400, 500; $\beta = 0.3$).

trained parameter vector prior to transmission. The server aggregates the resulting mixture of honest and corrupted updates using FedAvg. We report validation RMSE (clean uniform validation set, original scale) and GAP as functions of K for both GELU and SiLU activations; the corresponding curves are shown in Figures 57–60.

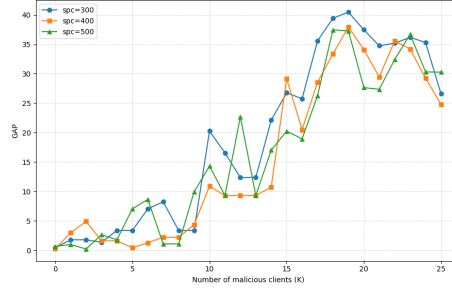


Fig. 57. Model poisoning (Rastrigin 4D, GELU, FedAvg): GAP vs. K ($\sigma = 0.4$).

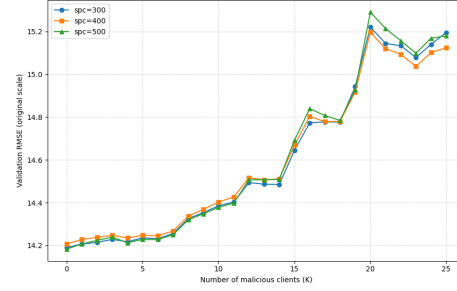


Fig. 58. Model poisoning (Rastrigin 4D, GELU, FedAvg): RMSE vs. K ($\sigma = 0.4$).

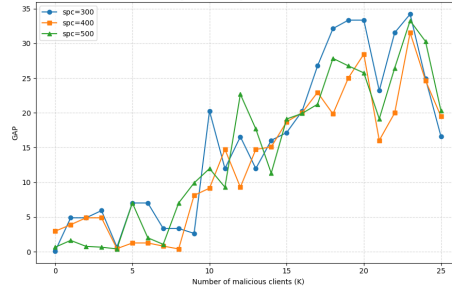


Fig. 59. Model poisoning (Rastrigin 4D, SiLU, FedAvg): GAP vs. K ($\sigma = 0.4$).

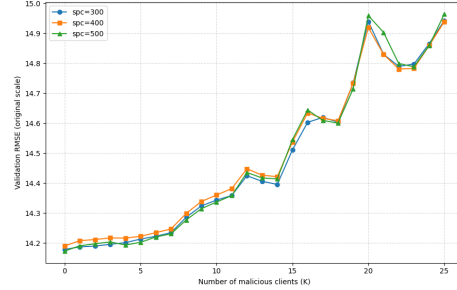


Fig. 60. Model poisoning (Rastrigin 4D, SiLU, FedAvg): RMSE vs. K ($\sigma = 0.4$).

6. Conclusions and Future Work Directions

This paper presents an analysis of Federated Machine Learning for the generation of surrogate functions and provides additional insights into the addressed issues. The analysis was performed on artificially generated training data in order to provide a suitable framework for comparing the source and surrogate functions. Accordingly, we selected benchmark functions known in analytical form and evaluated input–output pairs (*input variable, function output*) to obtain samples for building the surrogate function using machine learning. We analyzed Federated Learning with respect to the following issues: (i) the dependence of surrogate-function characteristics on the sample dimension and data heterogeneity; (ii) the impact of training-data poisoning and model poisoning on the surrogate function; and (iii) the optimization capabilities of the surrogate function depending on the employed data and the training scenario.

Across our experiments, we did not observe a meaningful difference between the SiLU and GELU activation functions in terms of either the root mean squared error or the conditional-minimization gap. In both cases, model poisoning and data poisoning generally increased the root mean squared error, indicating a degradation in predictive accuracy. For the conditional-minimization gap, however, the trends differed: increasing the number of malicious clients under model poisoning typically increased the gap, whereas under data poisoning this was not consistently the case. The similar performance of SiLU and GELU is not entirely surprising, since both are smooth, non-monotonic activation functions with comparable gradient behavior around the origin and without the sharp saturation effects typical of older activations. In several runs, a smaller gap was obtained even with a larger malicious fraction, which suggests that backdoor perturbations can sometimes distort the surrogate in a way that does not necessarily worsen the particular conditional minimization task being evaluated. These findings indicate that backdoor perturbations may degrade global predictive accuracy, as reflected by higher RMSE, while not necessarily deteriorating the specific conditional minimization task under certain settings.

Regarding the considered benchmark functions, we note that the Rastrigin function is notably more *rugged* than Ackley, yet the surrogate often still captured the global tendency toward the true global minimum. This indicates that, when the objective has a clearly dominant global basin compared to its local minima, even a relatively small neural network and a moderate amount of training data can be sufficient to localize the optimum-relevant region, despite a comparatively large root mean squared error. Replacing simple averaging with coordinate-wise trimmed mean aggregation stabilized both the root mean squared error and the conditional-minimization gap for a reasonable number of malicious clients relative to the total number of clients, which is consistent with the expected robust behavior of trimmed mean under moderate adversarial contamination.

The additional comparison with a centralized baseline provides a useful reference point for the Ackley 4D case. The centralized baseline consistently achieved lower RMSE and lower GAP than the federated model for both $SPC = 200$ and $SPC = 450$. This advantage was especially pronounced for the smaller sample regime and became even more visible in the GAP metric, where the federated model showed a substantial degradation as the number of malicious clients increased. Overall, these results suggest that, for Ackley 4D under the considered poisoning setup, centralized training was more robust than federated training both in predictive accuracy and in surrogate-based optimization quality.

Several directions for future work naturally follow from the present study. First, it would be of interest to consider stronger adversarial strategies, such as model-replacement, scaling-based, or stealthier backdoor attacks. Second, the experimental part could be extended by including additional robust aggregation rules, such as coordinate-wise median, Krum, or Bulyan. Third, since the current normalization procedure relies on global statistics for training stability, it would be useful to investigate local normalization or federated/statistics-preserving alternatives. Fourth, although benchmark functions provide a controlled and reproducible setting, it would be important to validate the proposed framework on concrete application-oriented optimization problems. Finally, while the present study adopts an MLP as a simple and well-established baseline for federated regression, more advanced surrogate architectures, such as variational autoencoders (VAEs) or diffusion models, as well as issues related to optimal sample size and data aug-

mentation, represent promising directions for further research. In particular, such models are especially promising in the context of generative surrogate modelling and search-space exploration. As an illustration, we refer to [14], which demonstrates the use of image transforms and diffusion models for the generative modelling of short and long time series, and which may serve as a guideline for leveraging diffusion models to improve search-space exploration in surrogate-assisted optimization.

Acknowledgement. This work was supported in part by the projects "Advanced Synergy of Wireless Information Security, Blockchain, Machine Learning and Optimization for 6G Networks" of the Faculty of Electrical Engineering, University of Montenegro, funded by the Ministry of Education, Science and Innovations of Montenegro, and "Mathematical analysis, optimization and machine learning" of the Faculty of Science and Mathematics, University of Montenegro, funded by the Ministry of Education, Science and Innovations of Montenegro. Also, the work was partially supported by the Shandong Provincial Key Research and Development Program (2025CXPTWZ003, WSR2024004), the Shandong Natural Science Foundation of China (ZR2024MF104, ZR2023QF129), the Taishan Scholars Program (tsqn202312231), the New 20 project of higher education of Jinan, China (202228017), the Pilot Project for Integrated Innovation of Science, Education, and Industry of Qilu University of Technology (Shandong Academy of Sciences) (2024ZDZX08).

Fundings. This work was funded by the Government of Montenegro, Ministry of Education, Science and Innovations, under grant number 07-604/25-11592/11.

References

1. Bhosekar, A., Ierapetritou, M.: Advances in surrogate-based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering* 108, 250–267 (2018)
2. Blik, L., Guijt, A., Karlsson, R., Verwer, S., de Weerd, M.: Benchmarking surrogate-based optimisation algorithms on expensive black-box functions. *Applied Soft Computing* 134, 110744 (Nov 2023)
3. Carpio, R.R., Giordano, R.C., Secchi, A.R.: Enhanced surrogate assisted framework for constrained global optimization of expensive black-box functions. *Computers & Chemical Engineering* 118, 91–102 (Oct 2018)
4. Chen, S., Huang, Z., Zheng, W.: Optimization of non-smooth functions via differentiable surrogates. *PLOS ONE* 20(5), e0321862 (2025)
5. Chong, C., Siquan, H., Leyu, S., Yanhong, L., Ying, G.: Distributed clustering meets federated learning: A clustering-based approach to data poisoning mitigation. *Cybersecurity* 8, 108 (2025)
6. Danes, L.H., Avansi, G.D., Schiozer, D.J.: Using surrogates as faster objective function estimators for oil production strategy optimization in a real-case reservoir. *SPE Journal* 30(5), 2757–2778 (2025)
7. Formery, Y., Mendiboure, L., Villain, J., Deniau, V., Gransart, C.: A framework to design efficient blockchain-based decentralized federated learning architectures. *IEEE Open Journal of the Computer Society* 5, 705–723 (2024)
8. Gabrielli, E., Belli, D., Matrullo, Z., Miori, V., Tolomei, G.: Securing federated learning against extreme model poisoning attacks via multidimensional time series anomaly detection on local updates. *IEEE Transactions on Information Forensics and Security* 20, 9610–9624 (Sep 2025)

9. Haftka, R.T., Villanueva, D., Chaudhuri, A.: Parallel surrogate-assisted global optimization with expensive functions - a survey. *Structural and Multidisciplinary Optimization* 54, 3–13 (2016)
10. de Hoop, M.V., Huang, D.Z., Qian, E., Stuart, A.M.: The cost-accuracy trade-off in operator learning with neural networks. *Journal of Machine Learning* 1(3), 299–341 (2022)
11. Kaushal, V., Sharma, S.: Securing the collective intelligence: a comprehensive review of federated learning security attacks and defensive strategies. *Knowledge and Information Systems* 67, 3099–3137 (Jan 2025)
12. Kudela, J., Matousek, R.: Recent advances and applications of surrogate models for finite element method computations: A review. *Soft Computing* 26, 13709–13733 (2022)
13. Mihaljevic, M.J.: A blockchain consensus protocol based on dedicated time-memory-data trade-off. *IEEE Access* 8, 41258–41268 (2020)
14. Naiman, I., Berman, N., Pemper, I., Arbiv, I., Fadlon, G., Azencot, O.: Utilizing image transforms and diffusion models for generative modelling of short and long time series. In: *Advances in Neural Information Processing Systems*. vol. 37, pp. 121699–121730 (2024)
15. Nowroozi, E., Haider, I., Taheri, R., Conti, M.: Federated learning under attack: Exposing vulnerabilities through data poisoning attacks in computer networks. *IEEE Transactions on Network and Service Management* 22(1), 822–831 (Feb 2025)
16. Papathanasopoulos, A., Apostolopoulos, P.A., Rahmat-Samii, Y.: Optimization assisted by neural network-based machine learning in electromagnetic applications. *IEEE Transactions on Antennas and Propagation* 11, 160–173 (Jan 2024)
17. Propp, A.M., Tartakovsky, D.M.: Transfer learning on multi-dimensional data: A novel approach to neural network-based surrogate modeling. *Journal of Machine Learning for Modeling and Computing* 6(2), 13–27 (2025)
18. Queipo, N., Haftka, R., Shyy, W., Goel, T., Vaidyanathan, Tucker, P.: Surrogate-based analysis and optimization. *Progress in Aerospace Sciences* 41, 1–28 (Jan 2005)
19. Sun, G., Cong, Y., Dong, J., Wang, Q., Lyu, L., Liu, J.: Data poisoning attacks on federated machine learning. *IEEE Internet of Things Journal* 9, 11365–11374 (2022)
20. Sun, M., Zhao, R., Huang, Z., Ding, S., Liu, J.: It-ose: Exploring optimal sample size for industrial data augmentation. *IEEE Transactions on Industrial Informatics* 22 (2026)
21. Tong, H., Huang, C., Minku, L.L., Yao, X.: Surrogate models in evolutionary single-objective optimization: A new taxonomy and experimental study. *Information Sciences* 562, 414–437 (2021)
22. Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., Rellermeyer, J.S.: A survey on distributed machine learning. *ACM Computing Surveys* 53(2), 1–33 (2020)
23. Wan, Y., Qu, Y., Ni, W., Xiang, Y., Gao, L., Hossain, E.: Data and model poisoning backdoor attacks on wireless federated learning, and the defense mechanisms: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 26(3), 1861–1897 (2024), third Quarter
24. Wang, Q., Wang, L., Xu, S., Zhang, S., Shao, W., Mihaljević, M.: Single-layer trainable neural network for secure inference. *IEEE Internet of Things Journal* 12, 2968–2978 (2025)
25. Xu, S., Dong, S., Wang, L., Mihaljević, M., Zhang, S., Shao, W., Wang, Q.: Blockchain-based secure data sharing with overlapping clustering and searchable encryption. *Computer Standards & Interfaces* 93, 103979 (2025)
26. Yang, L., Miao, Y., Liu, Z., Liu, Z., Li, X., Kuang, D., Li, H., Deng, R.H.: Enhanced model poisoning attack and multi-strategy defense in federated learning. *IEEE Transactions on Information Forensics and Security* 20, 3877–3892 (2025)
27. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol* 10, 1–19 (2019)
28. Yang, X.S.: *Test Problems in Optimization*. Wiley, Hoboken, NJ, USA (2010)
29. Zhang, J., Feng, F., Jin, J., Zhang, W., Zhao, Z., Zhang, Q.J.: Adaptively weighted yield-driven em optimization incorporating neurotransfer function surrogate with applications to microwave filters. *IEEE Transactions on Microwave Theory and Techniques* 69, 518–528 (Jan 2021)

Nevena Mijajlović was born in 1985. She received the B.Sc., M.Sc., and Ph.D. degrees in mathematics from the University of Montenegro, Podgorica, Montenegro, in 2007, 2009, and 2015, respectively. Since 2008, she has been with the University of Montenegro, initially as a Teaching Assistant, later as an Assistant Professor, and currently as an Associate Professor with the Faculty of Natural Sciences and Mathematics, Podgorica. Her research interests include optimization methods, variational inequalities, equilibrium problems, and blockchain technology. Prof. Mijajlović is a member of the Committee for Mathematics and Physics of the Montenegrin Academy of Sciences and Arts (MASA) and the Center for Young Scientists and Artists of MASA. She is also a member of the University of Montenegro's Board of Directors, representing the academic staff.

Ajlan Zajmović was born in Bijelo Polje, Montenegro, on January 2, 2001. He received the B.Sc. degree in Mathematics and Computer Science from the Faculty of Natural Sciences and Mathematics, University of Montenegro, Podgorica, in 2023, and the M.Sc. degree in Mathematics from the same faculty in December 2025. Since 2023, he has been a Teaching Assistant with the Faculty of Natural Sciences and Mathematics, University of Montenegro. He has published two scientific papers and presented his research at five international scientific conferences. He has also been involved as a junior researcher in several scientific projects. His research interests include mathematical optimization and related areas of applied mathematics.

Miodrag J. Mihaljević has been holding long-term visiting positions with various universities and research institutes in Japan, including The University of Tokyo, Sony Research Laboratories, and the National Institute AIST, Tokyo, since 1997. He is currently a Distinguished Professor with Shandong Computer Science Center, Qilu University of Technology (Shandong Academy of Sciences), Jinan, China, as well as a member of the Serbian Academy of Sciences and Arts, Belgrade. His main research interests include cryptology, information security, blockchain technology, and machine learning. Since 2014, he has been an Elected Member of Academia Europaea. Since 2021, he has been an Elected Member of the Serbian Academy of Sciences and Arts. In 2013, he received the National Award of the Serbian Academy of Sciences and Arts for ten-year achievements. In the six consecutive years 2020–2025, he was included in the ranked list colloquially known as World's Top 2% Scientists by Elsevier and Stanford University, regarding his career achievements. For more information, visit <http://www.mi.sanu.ac.rs/cv/cvmihaljevic.htm>.

Lianhai Wang received the B.S. degree in computer software and theory, and the Ph.D. degree in computer science and technology from Shandong University, Jinan, China, in 2003 and 2014, respectively. He is currently a Research Professor with the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center, Qilu University of Technology (Shandong Academy of Sciences), China. His current research interests include artificial intelligence security and privacy protection, blockchain, and digital forensics.

Shujiang Xu received the M.S. degree in applied mathematics and the Ph.D. degree in cryptography from Beijing University of Posts and Telecommunications, China, in 2006 and 2014, respectively. He is currently a Research Professor with the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong

Computer Science Center, Qilu University of Technology (Shandong Academy of Sciences), China. His main research interests include data security and privacy protection, blockchain, and digital forensics.

Wei Shao received the M.S. degree in information security from University College London, U.K., and the Ph.D. degree from the School of Cyber Science, Nankai University, Tianjin, China, in 2021. She is currently with the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center, Qilu University of Technology (Shandong Academy of Sciences), China. Her research interests include information security, cryptography, and blockchain security.

Received: December 15, 2025; Accepted: June 7, 2026.