Detecting and Mitigating Data Integrity Attacks on Distributed Algorithms for Optimal Power Flow using Machine Learning

Abstract

Using distributed algorithms, multiple computing agents can coordinate their operations by jointly solving optimal power flow problems. However, cyberattacks on the data communicated among agents may maliciously alter the behavior of a distributed algorithm. To improve cybersecurity, this paper proposes a machine learning method for detecting and mitigating data integrity attacks on distributed algorithms for solving optimal power flow problems. In an offline stage with trustworthy data, agents train and share machine learning models of their local subproblems. During online execution, each agent uses the trained models from neighboring agents to detect cyberattacks using a reputation system and then mitigate their impacts. Numerical results show that this method reliably, accurately, and quickly detects data integrity attacks and effectively mitigates their impacts to achieve near-feasible and near-optimal operating points.

Keywords: Cybersecurity, Distributed Optimization, Optimal Power Flow, Data Integrity Attack

1. Introduction

Rapid deployments of distributed energy resources (DERs) motivate the development of new algorithms to optimize the performance of these resources while respecting network limitations. Traditional optimization is performed by a central operator collecting system-wide information and dispatching power from a few large generating plants. This centralized paradigm may be inapplicable to complex networks with widespread DER integration. Distributed optimization algorithms provide an alternative whereby multiple computing agents iteratively solve optimization problems by communicating the values of boundary variables. By enabling parallel computations, these algorithms have potential advantages in scalability [1].

Cybersecurity is a key concern for distributed optimization algorithms which rely on repeated communications between agents. This paper focuses on improving the cybersecurity of distributed optimization algorithms with respect to data integrity attacks. An adversary who takes control of an agent or attacks communication links could disrupt the system or manipulate the system's operating point to profit financially. In this paper, we propose a machine learning method for detecting and mitigating cyberattacks on distributed optimal power flow (OPF) algorithms.

We focus on the popular alternating direction method of multipliers (ADMM) algorithm, but our method can also be applied to other distributed algorithms such as auxiliary problem principle (APP) [2] or analytical target cascading (ATC) [3]. In these algorithms, local computing agents solve OPF subproblems for their region of the network that are augmented with constraints which ensure power flow consistency with neighboring agents. These algorithms alternate between solving augmented local subproblems and sharing boundary variable values to update the consistency constraints. When this shared data is corrupted due to, e.g., communication non-idealities or deliberate attacks, convergence is impaired [4], [5].

Early work on distributed OPF algorithm cyberattack vulnerability includes [6], [7], which develops an attack strategy for distributed primal-dual gradient descent DC OPF algorithms. This work is extended to the AC OPF problem in [8]. The adversary determines a target solution, which is sub-optimal overall but profitable to the attacker, and shares false data which corresponds to that target solution. Building on this work, our prior research in [9], [10] explores two additional cyberattacks on DC OPF problems solved with the APP algorithm and develops a machine learning detection method. The first attack strategy uses PID feedback control principles to gradually approach the target solution, while the second attack strategy uses bilevel optimization to maximize profit.

Some prior work aims to detect and mitigate the impacts of these attacks. The method in [6]–[8] is designed for a component-based distributed algorithm in which every bus has a local controller and agents

communicate with their two-hop neighbors. Agents perform regression on historical shared variable values to predict future values and replace false shared data with these predictions. Another study proposes using physical-layer (power line) communication to verify the cyber-layer communication [11]. Since power line communication bandwidth is limited, this verification increases communication time, but this could be mitigated to some extent by infrequent verification.

Building on this prior work, we propose a cyberattack detection and mitigation method that uses machine learning models to predict neighboring agents' future behavior. Related prior work predicts shared data for distributed DC OPF algorithms. In the context of accelerating convergence, the authors of [12] used a linear interpolation on shared data from previous iterations to estimate values at the next iteration, while the authors of [13] train a recurrent neural network to make a one-time prediction of final shared data values. There has also been recent interest in using neural networks (NNs) to predict AC OPF solutions [14], [15].

While potentially effective in some circumstances, prior work on mitigation in [6]–[8] may fail for certain attack strategies, and the approach in [11] increases solution time. Our prior work on cyberattack detection in [9], [10] requires generating data for specific attack strategies to train the machine learning models and does not mitigate the impacts of cyberattacks.

Leveraging recent developments in machine learning, this paper addresses these gaps by proposing a method for both identifying and mitigating cyberattacks on distributed optimization algorithms. Specifically, we train NN models offline to approximate local AC OPF subproblems and use them during real-time operation to detect corruption or manipulation of shared data. If a cyberattack on shared data is detected, we use the NN predictions to replace the false data. The distributed algorithm can then achieve a near-feasible, near-optimal solution while under a data-integrity cyberattack.

The method for attack detection and mitigation in this paper is more flexible than the method proposed in [6] since the approximation operates on data from the previous iteration only, rather than longer sequences of historical data. Additionally, the method proposed here does not require the defender to train a classifier on data from a specific attack strategy, as in [9], [10], [16], [17]. Thus, our method may better generalize across attack strategies. Building on [14], [15], we enable accurate, repeated predictions of the shared data by training a NN to approximate a local AC OPF subproblem. Unlike prior work on centralized OPF, distributed subproblems depend on shared data from neighbors in addition to local network state. We develop heuristics to identify the relevant sample space during NN training.

We organize the paper as follows. Section 2 discusses the distributed AC OPF formulation and attack strategies. Section 3 presents our detection and mitigation method. Section 4 gives results from several representative test cases that demonstrate our method's ability to reliably detect and mitigate cyberattacks. Section 5 discusses conclusions and future work.

2. Problem Formulation

We next describe the distributed optimal power flow algorithm and attack strategies.

2.1. Optimal Power Flow

The OPF problem optimizes performance while meeting operational limits and obeying physical power flow laws. We use an OPF objective that minimizes generation cost via a quadratic function of the generators' real power outputs. This paper considers an OPF formulation with an AC power flow model, but alternatives like a DC power flow could also be applied.

Let $\mathcal{N}$, $\mathcal{L}$, and $\mathcal{G}$ denote the sets of buses, lines, and generators, respectively. The OPF formulation is

$$\min_{\substack{\boldsymbol{p}^g,\boldsymbol{q}^g,\boldsymbol{p},\\ \boldsymbol{q},\boldsymbol{\theta},\boldsymbol{v}}} \quad \sum_{i\in\mathcal{N}} f_i(p_i^g) \tag{1a}$$

$$\text{s.t.} \quad (\forall i \in \mathcal{N},\, \forall (i,j) \in \mathcal{L})$$

$$p_i^g - p_i^d = \sum_{(i,j)\in\mathcal{L}} p_{ij} + g_i^{sh} v_i^2, \tag{1b}$$

$$q_i^g - q_i^d = \sum_{(i,j)\in\mathcal{L}} q_{ij} - b_i^{sh} v_i^2, \tag{1c}$$

$$\underline{V}_i \le v_i \le \overline{V}_i, \tag{1d}$$

$$\underline{P}_i^g \le p_i^g \le \overline{P}_i^g, \tag{1e}$$

$$\underline{Q}_i^g \le q_i^g \le \overline{Q}_i^g, \tag{1f}$$

$$p_{ij} = v_i^2 G_{ij} - v_i v_j \left[G_{ij}\cos(\theta_{ij}) + B_{ij}\sin(\theta_{ij})\right] \tag{1g}$$

$$q_{ij} = -v_i^2 (B_{ij}^{sh} + B_{ij}) - v_i v_j \left[G_{ij}\sin(\theta_{ij}) - B_{ij}\cos(\theta_{ij})\right] \tag{1h}$$

$$p_{ij}^2 + q_{ij}^2 \le \left(\overline{S}_{ij}\right)^2, \tag{1i}$$

where f_i is the cost function and p_i^g, q_i^g are the real and reactive power outputs, respectively, of generator $g \in \mathcal{G}$ located at bus $i \in \mathcal{N}$. We define $\theta_{ij} = \theta_i - \theta_j$ for $(i,j) \in \mathcal{L}$. The admittance of line $(i,j) \in \mathcal{L}$ is $Y_{ij} = G_{ij} + jB_{ij}$, while $\overline{S}_{ij}$ denotes the line's

thermal limit. The shunt admittance at bus $i \in \mathcal{N}$ is $g_i^{sh} + jb_i^{sh}$. The state of the buses is defined by the voltage magnitude v_i and voltage angle θ_i for bus $i \in \mathcal{N}$. We denote the real power demand at bus $i \in \mathcal{N}$ as p_i^d and reactive power demand as q_i^d. The OPF problem minimizes the generation cost in (1a) subject to the AC power flow equations (1b)–(1c), (1g)–(1h), the voltage limits and generators' power output limits (1d)–(1f), and the lines' thermal limits (1i).

2.2. Distributed Optimal Power Flow

To decompose the OPF problem, the system is divided into regions under the control of separate computing agents. Consistency constraints ensure valid power flows at the boundaries. We split the system through the tie-lines and give the regions on each side of a split line copies of the variables at their neighbor's bus. Then, we enforce consistency by ensuring that the voltage phasor and power flow variables agree.

Agents seek to minimize generation cost in their region while obeying OPF constraints. Let $\mathcal{G}_m$, $\mathcal{N}_m$, and $\mathcal{L}_m$ denote the sets of generators, buses, and lines in area m, respectively. We denote the set of shared variables in region m with $\mathcal{N}_m^s$. The consistency constraints are relaxed with the augmented Lagrangian technique.

To illustrate the proposed method, we use the ADMM algorithm, but other distributed algorithms could be applied instead with no conceptual changes. In the ADMM algorithm, each agent m solves the following subproblem at each iteration k:

$$\min_{\substack{\boldsymbol{p}^{g,k},\boldsymbol{q}^{g,k},\boldsymbol{p}^{k},\\ \boldsymbol{q}^{k},\boldsymbol{\theta}^{k},\boldsymbol{v}^{k},\boldsymbol{z}_m^{k}}} \sum_{i\in\mathcal{N}_m} f_i(p_i^{g,k}) + (\boldsymbol{\lambda}_m^{k-1})^T \boldsymbol{z}_m^k + \frac{\alpha}{2}||\boldsymbol{z}_m^k - \bar{\boldsymbol{z}}_m^{k-1}||_2^2 \tag{2a}$$

s.t. $(\forall i \in \mathcal{N}_m, \forall (i,j) \in \mathcal{L}_m)$

$$p_i^{g,k} - p_i^d = \sum_{(i,j)\in\mathcal{L}_m} p_{ij}^k + g_i^{sh}(v_i^k)^2, \tag{2b}$$

$$q_i^{g,k} - q_i^d = \sum_{(i,j)\in\mathcal{L}_m} q_{ij}^k - b_i^{sh}(v_i^k)^2, \tag{2c}$$

$$\underline{V}_i \le v_i^k \le \overline{V}_i \tag{2d}$$

$$\underline{P}_i^g \le p_i^{g,k} \le \overline{P}_i^g, \tag{2e}$$

$$\underline{Q}_i^g \le q_i^{g,k} \le \overline{Q}_i^g, \tag{2f}$$

$$p_{ij}^k = (v_i^k)^2 G_{ij} - v_i^k v_j^k \left[G_{ij}\cos(\theta_{ij}^k) + B_{ij}\sin(\theta_{ij}^k)\right] \tag{2g}$$

$$q_{ij}^k = -(v_i^k)^2(B_{ij}^{sh} + B_{ij}) - v_i^k v_j^k \left[G_{ij}\sin(\theta_{ij}^k) - B_{ij}\cos(\theta_{ij}^k)\right] \tag{2h}$$

$$(p_{ij}^k)^2 + (q_{ij}^k)^2 \le \left(\overline{S}_{ij}\right)^2, \tag{2i}$$

where α is a user-defined penalty parameter. The vector $\boldsymbol{z}_m$ contains all shared variables in $\mathcal{N}_m^s$, and the vector $\bar{\boldsymbol{z}}_m$ is a "central" variable which accounts for all neighbors' copies of the shared variables. In traditional ADMM, this variable is computed by a central coordinator, but for our formulation it simplifies to the average of the two neighboring agents' shared variable values and is thus entirely separable. After solving their subproblem, agents send their computed shared variables and the corresponding dual variables to their neighbors. Then, each agent m updates the $\bar{\boldsymbol{z}}_m$ variables. For every neighbor n of agent m, there is a set of variables $\mathcal{N}_{m,n}^s$ shared between agents m and n. We denote agent m's copies of these shared variables as the vector $\boldsymbol{z}_{m,n}$ and agent n's copies of these shared variables as the vector $\boldsymbol{z}_{n,m}$. Agent m updates the average of local shared variables and shared variables received from neighbor n, $\bar{\boldsymbol{z}}_{m,n}$, as

$$\bar{\boldsymbol{z}}_{m,n}^k = \frac{1}{2}(\boldsymbol{z}_{m,n}^k + \boldsymbol{z}_{n,m}^k). \tag{3}$$

Agent m performs this update for all areas with which it shares variables until there is a complete update for $\bar{\boldsymbol{z}}_m$.

Each agent m updates their Lagrange multipliers as

$$\boldsymbol{\lambda}_m^k = \boldsymbol{\lambda}_m^{k-1} + \alpha(\bar{\boldsymbol{z}}_m^k - \boldsymbol{z}_m^k). \tag{4}$$

Thus, the iterative algorithm alternates between minimizing the agents' subproblems in (2), updating the average copies of variables shared between agents in (3), and updating dual variables in (4). The stopping criterion is the mismatch between local shared variable values $\boldsymbol{z}_m$ and the average shared variable values, $\bar{\boldsymbol{z}}_m$. At iteration k, each agent m records the norm of the mismatches across all its shared variables as

$$||\bar{\boldsymbol{z}}_m - \boldsymbol{z}_m||_2, \tag{5}$$

and the algorithm terminates when the mismatches are below some user-defined tolerance.

2.3. Cyberattack Strategies

This paper considers three possible attack strategies to demonstrate our results. However, we expect the detection and mitigation method to be effective against any data integrity attack. Rather than training a classifier to recognize a specific attack strategy as in [10], the method proposed here flags any deviation from expected algorithm behavior, making it more flexible.

The first two attack strategies consider financially motivated adversaries with access to the network topology and loading. The adversary seeks to steer the solution to some profitable target. To illustrate these attack strategies, we determine the attacker's target operating point by solving a system-wide OPF problem with a certain generator's active power output constrained to the desired value. The adversary saves the results which fall within its region as the target variable values and uses the following strategies, first proposed in [10], to guide the final result to the desired target.

In the first attack strategy, which we refer to as the "simple" attack, the adversary directly sends the desired values for shared variables to neighboring agents. This is simple to detect. The second strategy, "PID feedback," achieves a more subtle attack by employing a proportional-integral-derivative (PID) control method. For each shared variable z^k at iteration k, the attacker computes the error $\boldsymbol{e}^k$ as the difference between the target value and the value received from neighbors:

$$e^k = z^k_{target} - z^k.$$

Then, the attacker computes a correction term using three tuned parameters: the proportional gain k_p, the derivative gain k_d, and the integral gain k_i. After adding this correction term, the updated shared variable value is

$$z^{k+1} = z^k + k_p e^k + k_d(e^k - e^{k-1}) + k_i \sum_{j=1}^{k} e^j.$$

Figure 1 shows how the malicious adversary uses the feedback control method to manipulate the shared data. Agents in charge of each region solve their OPF subproblem to compute the shared data sent to their neighbors. However, the adversary has a target solution in mind corresponding to a shared variable state $\boldsymbol{z}_{target}$. Each iteration, the adversary uses the PID feedback control to compute values for their shared data which will gradually drive their neighbors to the desired target.

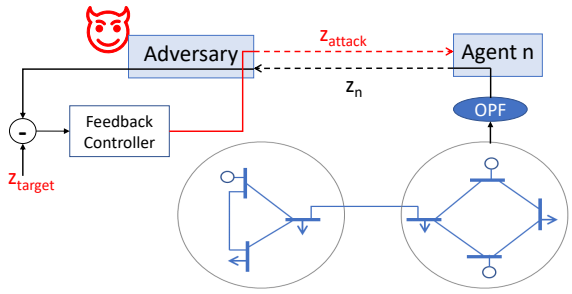

Figure 1: PID feedback threat model

The third attack strategy disrupts the system so that it cannot converge by sending random or infeasible values. An adversary who sends random values requires no knowledge of the system. The adversary may also send infeasible values—for example, power flow values that exceed the export capacity of their region of the network—as proposed in [11]. The disruptive attack strategy is straightforward to detect because the algorithm does not converge. We will show in our results that we can mitigate this attack and replace the random or false data to converge to a reasonable solution.

3. Attack Detection and Mitigation

This section presents our detection and mitigation method for cyberattacks such as those described in Section 2. The method is designed for distributed OPF problems solved repeatedly, every few minutes. Offline, agents train NN models of their local OPF subproblems and share the models with their neighbors. During real-time operation when distributed OPF problems are solved every few minutes, the detection and mitigation method prevents malicious actors from manipulating the solutions. Agents detect attacks by comparing data received from neighbors with the data predicted by the NN and mitigate attacks by replacing corrupted data with the NN approximations. The mitigation method is illustrated in Figure 2.

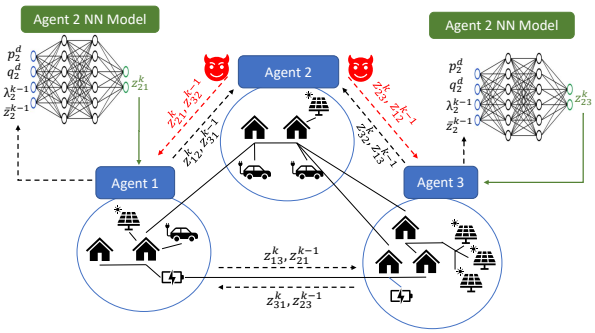

Figure 2: Using NN approximations to replace data from a malicious agent. Note that for dual variables λ shared between neighbors m, n, we have that $\lambda^k_m = \lambda^k_n$.

We next discuss the assumptions underlying our method. The assumptions used solely for detection are less strict than those for mitigation, so we separately discuss each.

3.1. Assumptions for Detection

When detecting cyberattacks on data integrity, we rely on the following four assumptions, A1–A4.

A1, *Trustworthy Training and NN Model Sharing*: Agents train NN models of their local regions offline using accurate data and share these models over a

trustworthy communication link. Thus, agents can trust that the models from their neighbors will not exhibit malicious behavior.

It is nontrivial to determine what information, if any, could be inferred from the trained NN, but it is possible that sharing NN models with neighboring agents could compromise the privacy of local information. Our future work includes ensuring that privacy is preserved, perhaps through secure multi-party computation techniques [18].

A2, *Knowledge of Neighboring Load Demands*: Agents know the load demands at the buses in each neighboring region. This means that loads are not private to local regions, although they are still not known system-wide but communicated only to neighboring regions.

To meet this assumption, a trusted system operator may send load measurements or short-term forecasts for an agent's region to its neighbors. Alternatively, agents may share their load data with neighbors over power line communication (PLC) infrastructure before the algorithm begins, since the physical-layer PLC cmmunications are slower but more difficult for an adversary to infiltrate as described in [11]. In the future, we aim to relax this assumption while preserving privacy by having agents estimate their neighbors' loads or using secure multi-party computation.

A3, *At Most One Malicious Agent*: There is no more than one malicious agent. Our method requires all neighbors of an agent to agree that it is malicious before flagging an attack, so colluding malicious agents could work together to avoid detection. Our future work will address possible collusion between multiple adversaries.

A4, *Communication with Two-Hop Neighbors*: Agents have communication links to their two-hop neighbors (i.e., their neighbors' neighbors). When we decompose the power system into regions, we define as "neighbors" any two regions which share a tie-line in the power system. The NN model for agent m requires as input the shared primal and dual variables from the previous iteration, $k-1$, for each neighbor n: $\left[(\boldsymbol{\lambda}_n^{k-1})^T \quad (\boldsymbol{z}_n^{k-1})^T\right]^T$. These values are used to compute the input variables $\boldsymbol{\lambda}_m^{k-1}, \bar{\boldsymbol{z}}_m^{k-1}$, where $\bar{\boldsymbol{z}}_m^{k-1}$ is the average of $\boldsymbol{z}_m^{k-1}$ and $\boldsymbol{z}_n^{k-1}$. Therefore, at each iteration, agents must obtain the data corresponding to their two-hop neighbors.

There are multiple ways to establish two-hop neighbor communication. One could design the communication infrastructure so that all agents are connected to their two-hop neighbors. If it is important to reduce the number of communication links in the system, another option is to send data through links defined by physical neighbors delayed by some number of iterations. Agents would need to store information to make predictions after the delay. With this approach, it would take longer to detect and mitigate attacks.

3.2. Assumptions for Mitigation

To mitigate cyberattacks, we require the same assumptions as for detection, A1–A4, as well as:

A5, *Adversary Manipulates Data Transmission Only*: The adversary may manipulate data transmission from some agent but cannot affect data reception, internal distributed OPF computations, or the physical control signals sent to system devices. The adversary may also access the communication links to add noise as in the disruptive attack. For instance, a sophisticated, financially motivated adversary may steal or forge Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates to establish encrypted communications and manipulate data to achieve a target solution. Our future work includes defending against adversaries who also control an agent's internal computations or commands to physical system devices.

3.3. Detection and Mitigation Algorithm

We show an overview of our detection and mitigation method in Figure 3. During an offline stage where the agents have access to trustworthy data as discussed above in **A1**, each agent m trains a neural network (NN) which maps the input of its subproblem at iteration k to the output. The input to agent m's subproblem includes the vector of active and reactive power demands at buses in $\mathcal{N}_m$, which we will denote here as $\boldsymbol{p}_m^d$, $\boldsymbol{q}_m^d$. The input also includes $\boldsymbol{\lambda}_m^{k-1}$ (the consistency constraint dual variables) and $\bar{\boldsymbol{z}}_m^{k-1}$ (the average of agent m's copies of shared variables with its neighbors' copies). Thus, the complete input to the NN is $\boldsymbol{x}_m = \left[(\boldsymbol{p}_m^d)^T \quad (\boldsymbol{q}_m^d)^T \quad (\boldsymbol{\lambda}_m^{k-1})^T \quad (\bar{\boldsymbol{z}}_m^{k-1})^T\right]^T$, with length $2|\mathcal{N}_m| + 2|\mathcal{N}_m^s|$. The outputs of the NN are predictions of the values for shared primal variables computed at iteration k, $\boldsymbol{y}_m^k = \left[\hat{\boldsymbol{z}}_m^k\right]^T$. The length of $\boldsymbol{y}_m^k$ is $|\mathcal{N}_m^s|$.

One of the major challenges in NN training is identifying the relevant input space. In a centralized setting, it is straightforward to bound possible values for the input, which consists of the network's loads. However, for a distributed subproblem, we must also identify bounds on shared variable values, which vary across algorithm iterations and network states. We estimate bounds on the shared primal and dual variable values by running the distributed OPF algorithm with varying load profiles and recording the shared primal and dual variable values at each iteration. We then record the minimum and maximum values for each

variable across all iterations of every run. Once these bounds are in place, we proceed to generate training data. We sample random values for each shared primal and dual variable from a uniform distribution within the variable's bounds. We also generate power demand values by randomly sampling over a uniform distribution from 50% to 150% of the nominal load value. We take the sampled power demand and shared primal and dual variable values to form the input to the agent's local distributed OPF problem. We then record the resulting output, i.e., the agent's shared primal variables for the next iteration. This process is repeated as many times as desired to generate training samples. The loss function which is minimized during NN training is the mean-squared-error (MSE) between predicted and actual primal variable output values:

$$MSE = \sum_i ||\boldsymbol{z}_{m,i} - \hat{\boldsymbol{z}}_{m,i}||_2^2, \tag{6}$$

where $\boldsymbol{z}_{m,i}$ is the true output of agent m's OPF subproblem given input sample i and $\hat{\boldsymbol{z}}_{m,i}$ is the NN's predicted output for input sample i.

After training, agents send copies of their NN models to their neighbors. During real-time operation, as the distributed algorithm is executing, agents track their neighbors' behavior using a reputation index. They verify received data from their neighbors using the NN predictions and increment a neighbor's reputation index if they receive anomalous data from that neighbor. Once the reputation index for some neighbor, agent m, reaches a certain reputation threshold, agents raise a local attack flag for neighbor m and communicate that flag to agent m's other neighbors. If all neighbors agree that agent m is sending anomalous data, they officially detect an attack by agent m. They then replace all data received from agent m with NN predictions. Once an attack is detected, neighbors continue monitoring agent m and decrement the reputation index if they detect valid data. If all neighbors agree that agent m is no longer sending false data, they stop replacing agent m's data with NN predictions.

We now formalize the detection and mitigation method. To perform attack detection, we follow the steps in Algorithm 1. Note that we define $S(n)$ as a function returning all one-hop neighbors of agent n. Before the distributed OPF algorithm begins, we set a detection threshold ϵ; if the L1-norm of the difference between shared and predicted data is above ϵ, that data is marked anomalous. We also set a reputation threshold ζ; when an agent's reputation index for some neighbor reaches ζ, it locally flags that neighbor as malicious. During the distributed OPF execution, at each iteration k, agent n compares the shared data $\boldsymbol{z}_m^k$ received from each neighbor m with the data $\hat{\boldsymbol{z}}_m^k$ predicted by the NN's model of agent m. We measure the difference $\rho_{n,m}^k$ seen by agent n between received and predicted shared data from agent m as the L1-norm of the difference:

$$\rho_{n,m}^k = ||\boldsymbol{z}_m^k - \hat{\boldsymbol{z}}_m^k||_1. \tag{7}$$

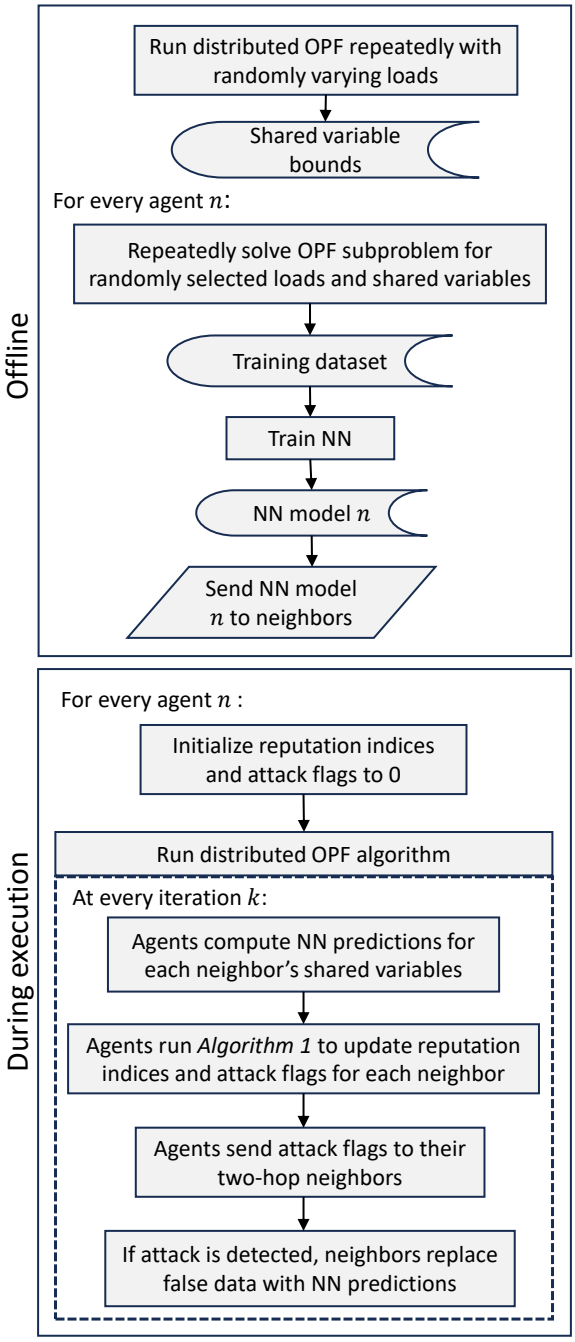

Figure 3: Flowchart showing steps for offline training and online detection and mitigation

Algorithm 1 Detection step for robust distributed OPF

for n = 1,2,. . . ,N **do**
  for $m \in S(n)$ **do**
    $\rho^k_{n,m} = ||\boldsymbol{z}^k_m - \hat{\boldsymbol{z}}^k_m||_1$
    if $\rho^k_{n,m} > \epsilon$ **and** $G_{n,m} < \zeta$ **then**
      $G_{n,m} = G_{n,m} + 1$
    end if
    if $\rho^k_{n,m} \leq \epsilon$ **and** $G_{n,m} > 0$ **then**
      $G_{n,m} = G_{n,m} - 1$
    end if
    if $G_{n,m} \geq \zeta$ **then**
      $A_{n,m} =$ **true**
    else if $G_{n,m}$ **is** 0 **then**
      $A_{n,m} =$ **false**
    end if
  end for
end for

Each agent tracks the reputation of their one-hop neighbors. At every iteration k, each agent n computes $\rho^k_{n,m}$ for each one-hop neighbor m. Agents update the reputation index $G_{n,m}$ for each neighbor according to Algorithm 1, where $G_{n,m}$ represents the reputation of agent m as seen by agent n. Agent n increments $G_{n,m}$ if it has not yet reached the reputation threshold ζ and $\rho^k_{n,m}$ is above the detection threshold ϵ. Agents decrement $G_{n,m}$ if it is above 0 and $\rho^k_{n,m}$ is below ϵ. We also introduce a flag $A_{n,m}$ which indicates whether or not agent n considers agent m to be malicious. We initialize $A_{n,m}$ to false. If the reputation index reaches its threshold, $G_{n,m} \geq \zeta$, then $A_{n,m} =$ true. If $G_{n,m} = 0$, then $A_{n,m} =$ is set to false. Agents communicate their flags to two-hop neighbors at every iteration. If all an agent's neighbors agree it is under attack, the agent is officially flagged as malicious and its neighbors will rely on NN predictions for its shared data.

4. Experiments and Results

We test the effectiveness of our detection and mitigation method on test cases selected from the PGLib-OPF archive [19]: the IEEE 14-, 30-, 118-, and 300-bus test cases. We first describe details on implementation and then provide results for detection accuracy and the cost and constraint violations from solutions computed while using the mitigation method.

4.1. Implementation Details

We use the PowerModelsADA library [20] to decompose the power networks and run distributed OPF algorithms. We divide the 14-, 30-, and 118-bus cases into three areas and the 300-bus case into 8 areas. We set the penalty parameter $\alpha = 1000$ for the 14-, 30-, and 118-bus cases and $\alpha = 2000$ for the 300-bus case. We run the distributed algorithm until the L2-norm of mismatches between shared variables is below 10^{-4} or until we complete 2000 iterations. We randomly select an agent to be the adversary. Under the simple and PID feedback attack strategies, the adversary attempts to increase power production from generators in its region to their maximum possible value. For the PID feedback attack, we set parameters $k_p = 0.1$, $k_d = 0.1$, and $k_i = 0.01$ for the proportional, derivative, and integral gain respectively. For the disruptive attack strategy, the adversary sends values randomly selected from [0,1] to cause the algorithm to diverge.

To train the neural network (NN) models for each test case, we first generate shared variable bounds for the samples as described in Section 3. We run the algorithm 50 times, each time perturbing the loads by randomly selecting values from a uniform distribution from 50% to 150% of the nominal loads for the 14-, 30-, and 118-bus cases and from 50% to 105% of the nominal loads for the 300-bus case. We record the maximum and minimum values during these algorithm runs for each shared variable and then compute the final bounds by increasing the maximum and decreasing the minimum values by 10% of the difference between maximum and minimum, to account for any input space not covered during the 50 runs. We generate a total of 800,000 samples, using 90% for training and 10% for validation. For each sample, we randomly sample load values from a uniform distribution from 50% to 150% of the nominal load values for the 14-, 30-, and 118-bus cases and from 50% to 105% of the nominal load values for the 300-bus case. We randomly sample shared variable values from a uniform distribution over the bounds. The NNs consist of 16 hidden layers, each with rectified linear unit (ReLU) activation, and a linear output layer. After training the NNs, we record NN errors on the validation datasets as the difference between real and predicted values and compute the mean and standard deviation of the errors.

We set detection tolerances ϵ based on the accuracy of the trained NN models. Shared data is considered anomalous if the average difference from NN predictions is more than 1.5 standard deviations from the NN error on the validation dataset, i.e., $\epsilon = 1.5 \cdot \sigma \cdot N$, where N is the number of shared variables and σ is the standard deviation of the NN prediction errors. The NN error distribution is a bell curve concentrated closer to the mean than a normal distribution of identical mean and standard deviation. After testing several values for $\epsilon = c \cdot \sigma \cdot N$ with c varying in increments of 0.1 from 1.3 to 2.2, we found that $\epsilon = 1.5 \cdot \sigma \cdot N$ was the best detection threshold. Recall that before

an agent is flagged as malicious, all its neighboring agents must see differences between shared data and NN predictions above ϵ for ζ iterations. This approach prevents occasional high NN errors from causing a false positive detection.

We evaluate two criteria for detection results: *accuracy* and *detection delay*. To compute the accuracy, we run the algorithm repeatedly with loads randomly selected from the same distributions used during training, under each of the attack strategies and under no attack. We randomly select attack start times from iterations 1 through 30. Detection is marked correct for an algorithm run if the attack is flagged after it begins and no attack is flagged before. We define detection accuracy as the proportion of runs with correct flags to total runs. We are also interested in the detection delay, which we define as the number of iterations after the attack begins until it is detected. Note that the minimum value for the detection delay is $\zeta + 1$, where ζ is the reputation threshold described in Section 3, or the number of iterations with abnormal shared data required before local agents flag malicious behavior. After waiting ζ iterations for local flags, it takes one additional iteration for agents to communicate these flags to their two-hop neighbors and officially confirm the attack.

We also evaluate two criteria for mitigation results: *feasibility* and *cost*. We run the algorithm repeatedly under each of the attack strategies, performing mitigation by replacing false data with NN predictions once an attack is detected, and save the final solution. Again, with each algorithm run, we perturb the loads by sampling from the distributions used during NN training. Next, we run an AC power flow on the test case with generator dispatch from the distributed OPF solution to find the operating point of the system after applying the solution. We record any constraint violations that occur in system operating point computed from the AC power flow. To evaluate cost, we compute the percent difference between the true optimal cost and the cost of the solution for the attacked algorithm using our mitigation method.

4.2. Results

We run the distributed OPF algorithm 1600 times (400 runs under simple attack, 400 runs under PID attack, 400 runs under disruptive attack, and 400 runs under no attack) for each test case and record the accuracy as the proportion of the 1600 runs for which the attack status was correctly classified. We also record the average detection delay for the algorithms under attack.

Table 1 summarizes the results of this experiment. With detection accuracy above 98.6% for all test cases, the algorithm reliably identified malicious agents. The average detection delay is quite close to the ideal, $\zeta + 1 = 4$. Note that it takes small cases such as the 14- and 30-bus systems anywhere from 60 to 500 iterations to converge, while larger cases like the 300-bus system may require more than 1500 iterations. With a four-iteration delay, agents detect an attack soon after it begins and can then start replacing false data with NN predictions for mitigation.

Table 1: Detection Results

Test case	Detection accuracy	Mean detection delay
14	99.9%	4.06 iterations
30	98.6%	4.97 iterations
118	100%	4.12 iterations
300	99.4%	4.03 iterations

For mitigation results, on every test case we run the algorithm 600 times with randomly varying loads, 200 times under each of the attack models, and record the constraint violations as described in Section 4.1. Algorithms running with an agent's shared data replaced by NN predictions do not converge to our typical tolerance of 10^{-4} for the L2-norm of the shared variable mismatches. The final mismatch norms are in the range $[0.005, 0.01]$, where mismatches are in per unit for power flows and voltage magnitudes and in radians for voltage angles. This mismatch allows for some small constraint violations in the final operating point. For the 14-, 30-, and 118-bus cases, the only violations that occur are on reactive power limits. For any mitigated solution, no more than 2% of generators have reactive power limit violations. We display in Table 2 the maximum and mean of any constraint violations that occur for the 600 algorithm runs with mitigated attacks. The 300-bus case mitigated solutions also result in some voltage violations, with maximum value 0.0013 p.u. and mean value $2 \cdot 10^{-4}$ p.u. Part of our ongoing work involves analyzing the impact of shared variable mismatches on the solution feasibility, developing bound tightening algorithms to ensure feasibility for a given mismatch tolerance, and refining the NN training process for more accurate predictions.

Table 2: Reactive Power Limit Violations

Test case	Maximum violation	Mean violation
14	0.57 MVAr	0.15 MVAr
30	0.18 MVAr	0.11 MVAr
118	1.12 MVAr	0.15 MVAr
300	3.22 MVAr	0.42 MVAr

We also evaluate whether mitigation produces solutions that are close to optimal. We run the same test cases for the distributed OPF algorithm under attack with and without mitigation. We compute the percent difference between the true optimal solution and the cost

of the distributed OPF solution as

$$100 \cdot (\hat{z}^* - z^*)/z^*$$

where z^* is the true optimal objective value and $\hat{z}^*$ is the distributed OPF objective value. The goal for financially motivated adversaries with the simple or PID attack strategies is to profit by increasing production from some generator, leading to a sub-optimal solution. Our mitigation method prevents the solution from converging to the adversary's malicious target and instead results in an operating point close to the optimum. On average, the percent difference from optimal for distributed OPF under attack with mitigation is about 0.2%, while adversarial target operating points have percent differences from 5% to 20%. Figure 4 illustrates the results for a representative run using the simple attack strategy on the 118-bus test case, where the operating cost of the current solution is computed at every iteration for unmitigated attack, mitigated attack, and no attack. The mitigation strategy results in a final percent difference from optimal cost of 0.02%, while without mitigation the attacked algorithm converges with a percent difference of 12.7%.

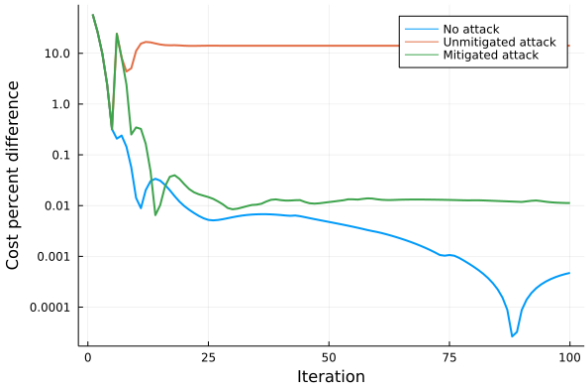

Figure 4: Cost percent difference from optimal as the algorithm evolves for unmitigated attack, mitigated attack, and no attack

The disruptive attack strategy does not seek a specific solution target but rather causes the algorithm to diverge. Table 3 shows the maximum and mean cost percent differences from the optimum across the 600 runs for all three attack strategies with mitigation.

Table 3: Cost Percent Differences

Test case	Maximum difference	Mean difference
14	5.6%	0.24%
30	2.16%	0.31%
118	0.05%	0.02%
300	0.07%	0.06%

We conclude our results with a note about computation time for the detection and mitigation method. In the original distributed OPF algorithm, agents solve local OPF subproblems and share data with their neighbors during every iteration. Our detection and mitigation method additionally requires agents to make one feedforward pass through a NN model for each of their neighbors and check whether the L1-norm of the difference between predicted and received variables is above some threshold. Benchmarking using a computer with a quad-core 2.3 GHz processor and 16 GB of RAM, we find that the detection and mitigation steps only take 0.4% to 0.8% of the total time per iteration. In other words, our method imposes negligible computing overhead relative to solving the OPF subproblems.

5. Conclusion and Future Work

Distributed algorithms for operating the emerging smart grid must be made resilient to cyberattacks. Building on previous literature which developed cyberattack threat models for distributed OPF problems, this paper proposed a detection and mitigation method for these attacks. By training NN models of agents' local OPF subproblems, we can detect false shared data by comparing it to the NN predictions and replace false data with the NN approximations once an attack has been detected. This detection method does not require the defender to train a classifier on data from specific threat models, making our method potentially more flexible to detect many different attack strategies. We use attack strategies proposed in two previous papers to demonstrate the effectiveness of our detection and mitigation algorithms. We achieve very high detection accuracy and show how replacing the false data with NN approximations produces near-feasible, near-optimal solutions to mitigate attacks which would otherwise drive the solution to a sub-optimal target or disrupt convergence.

Our future work includes the following tasks:

1) We plan to extend the detection and mitigation results to larger systems with more agents.

2) We are studying ways to relax the assumptions listed in Section 3.1. In particular, we aim to detect and mitigate attacks while keeping load demand information completely private, address attacks where multiple adversaries collude, and reduce communication requirements. We also intend to address the assumption for mitigation in Section 3.2 by finding safe operating points for the system in the event that an adversary gains complete control of a computing agent.

3) We will further investigate constraint violations that may occur from a solution computed while mitigating

an attack, in which small shared variable mismatches remain. We are developing constraint tightening algorithms to ensure feasible solutions given some bound on the mismatches.

4) We are assessing threat models which might be explicitly designed to bypass our NN detection method, perhaps by embedding our NN detection model into their attack calculation step.

References

[1] D. K. Molzahn, F. Dörfler, H. Sandberg, *et al.*, "A Survey of Distributed Optimization and Control Algorithms for Electric Power Systems," *IEEE Trans. Smart Grid*, vol. 8, no. 6, pp. 2941–2962, 2017.

[2] B. Kim and R. Baldick, "A Comparison of Distributed Optimal Power Flow Algorithms," *IEEE Trans. Power Syst.*, vol. 15, no. 2, pp. 599–604, 2000.

[3] A. Mohammadi and A. Kargarian, "Accelerated and Robust Analytical Target Cascading for Distributed Optimal Power Flow," *IEEE Trans. Industr. Inform.*, vol. 16, no. 12, pp. 7521–7531, 2020.

[4] M. Alkhraijah, C. Menendez, and D. K. Molzahn, "Assessing the Impacts of Nonideal Communications on Distributed Optimal Power Flow Algorithms," *Electr. Power Syst. Res.*, vol. 212, p. 108 297, 2022, presented at *22nd Power Systems Computation Conference (PSCC)*.

[5] A. Kargarian, M. Mehrtash, and B. Falahati, "Decentralized Implementation of Unit Commitment with Analytical Target Cascading: A Parallel Approach," *IEEE Trans. Power Syst.*, vol. 33, no. 4, pp. 3981–3993, 2017.

[6] J. Duan, W. Zeng, and M.-Y. Chow, "Resilient Distributed DC Optimal Power Flow Against Data Integrity Attack," *IEEE Trans. Smart Grid*, vol. 9, no. 4, pp. 3543–3552, 2018.

[7] J. Duan, W. Zeng, and M.-Y. Chow, "An Attack-Resilient Distributed DC Optimal Power Flow Algorithm via Neighborhood Monitoring," in *IEEE Power and Energy Society General Meeting*, 2016.

[8] Z. Cheng and M.-Y. Chow, "Resilient Collaborative Distributed AC Optimal Power Flow Against False Data Injection Attacks: A Theoretical Framework," *IEEE Trans. Smart Grid*, vol. 13, no. 1, pp. 795–806, 2022.

[9] R. Harris, M. Alkhraijah, D. Huggins, and D. K. Molzahn, "On the Impacts of Different Consistency Constraint Formulations for Distributed Optimal Power Flow," in *Texas Power and Energy Conference*, 2022.

[10] M. Alkhraijah, R. Harris, S. Litchfield, D. Huggins, and D. K. Molzahn, "Analyzing Malicious Data Injection Attacks on Distributed Optimal Power Flow Algorithms," *4th North American Power Symposium (NAPS)*, Oct. 2022.

[11] Y. Yang, G. Raman, J. C.-H. Peng, and Z.-S. Ye, "Resilient Consensus-Based AC Optimal Power Flow Against Data Integrity Attacks Using PLC," *IEEE Trans. Smart Grid*, vol. 13, no. 5, pp. 3786–3797, 2022.

[12] A. Mohammadi and A. Kargarian, "Learning-Aided Asynchronous ADMM for Optimal Power Flow," *IEEE Trans. Power Syst.*, vol. 37, no. 3, pp. 1671–1681, 2022.

[13] D. Biagioni, P. Graf, X. Zhang, A. S. Zamzam, K. Baker, and J. King, "Learning-Accelerated ADMM for Distributed DC Optimal Power Flow," *IEEE Contr. Syst. Lett.*, vol. 6, 2022.

[14] F. Fioretto, T. W. Mak, and P. V. Hentenryck, "Predicting AC Optimal Power Flows: Combining Deep Learning and Lagrangian Dual Methods," in *AAAI Conference on Artificial Intelligence*, vol. 34, Apr. 2020, pp. 630–637.

[15] R. Nellikkath and S. Chatzivasileiadis, "Physics-Informed Neural Networks for AC Optimal Power Flow," *Electr. Power Syst. Res.*, vol. 212, Nov. 2022.

[16] Q. Zhou, M. Shahidehpour, A. Alabdulwahab, and A. Abusorrah, "A Cyber-Attack Resilient Distributed Control Strategy in Islanded Microgrids," *IEEE Trans. Smart Grid*, vol. 11, no. 5, pp. 3690–3701, 2020.

[17] J. Shi, S. Liu, B. Chen, and L. Yu, "Distributed Data-Driven Intrusion Detection for Sparse Stealthy FDI Attacks in Smart Grids," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 68, no. 3, pp. 993–997, 2021.

[18] L. L. Ng and S. M. Chow, "SoK: Cryptographic Neural-Network Computation," in *IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 497–514.

[19] IEEE PES PGLib-OPF Task Force, "The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms," Aug. 2019, arXiv:1908.02788.

[20] M. Alkhraijah, R. Harris, C. Coffrin, and D. K. Molzahn, "PowerModelsADA: A Framework for Solving Optimal Power Flow using Distributed Algorithms," *arXiv:2304.00639*, 2023.