Detecting Shared Data Manipulation in Distributed Optimization Algorithms

Mohannad Alkhraijah, Rachel Harris, Samuel Litchfield, David Huggins, and Daniel K. Molzahn

***Abstract*—This paper investigates the vulnerability of the Alternating Direction Method of Multipliers (ADMM) algorithm to shared data manipulation, with a focus on solving optimal power flow (OPF) problems. Deliberate data manipulation may cause the ADMM algorithm to converge to suboptimal solutions. We derive two sufficient conditions for detecting data manipulation based on the theoretical convergence trajectory of the ADMM algorithm. We evaluate the detection conditions' performance on three data manipulation strategies we previously proposed: simple, feedback, and bilevel optimization attacks. We then extend these three data manipulation strategies to avoid detection by considering both the detection conditions and a neural network (NN) detection model in the attacker's decision process. We also propose an adversarial NN training framework to detect shared data manipulation. We illustrate the performance of our data manipulation strategy and detection framework on OPF problems. The results show that the proposed detection conditions successfully detect most of the data manipulation attacks. However, a bilevel optimization attack strategy that incorporates the detection methods may avoid being detected. Countering this, our proposed adversarial training framework detects all the instances of the bilevel optimization attack.**

***Index Terms*—Cybersecurity, Data manipulation, Distributed optimization, Optimal power flow.**

I. Introduction

Distributed optimization algorithms allow multiple agents to collaboratively solve large-scale optimization problems. Agents using distributed optimization solve subproblems iteratively and send the solutions of the shared variables with their neighbors at each iteration. If the solutions are correct and accurate, the algorithm converges to the optimal solution under mild technical assumptions. Using distributed algorithms to solve power system optimization problems such as optimal power flow (OPF) has the potential to scale computations, increase reliability, and enhance data privacy [1].

Since distributed algorithms must communicate shared variable data, these algorithms are vulnerable to communication nonidealities and data manipulation. Most existing literature on distributed optimization assumes that the participating agents are trustworthy and the shared data are accurate. Nonetheless, communications are prone to errors. Furthermore, agents may deliberately share inaccurate solutions, e.g., to increase their profit by changing their local generators' outputs.

M. Alkhraijah, R. Harris, and D.K. Molzahn are with the School of Electrical and Computer Engineering, Georgia Institute of Technology. S. Litchfield and D. Huggins are with the Cybersecurity, Information Protection, and Hardware Evaluation Research Laboratory, Georgia Tech Research Institute. Support from NSF AI Institute for Advances in Optimization (AI4OPT), #2112533.

Communication nonidealities significantly impact the performance of distributed algorithms [2]. Even with a low probability of occurrence, large errors can prevent the algorithm from converging [2]. Although large random errors rarely occur in modern communication networks, they may be used by malicious agents to preform "denial-of-service" attacks that cause the algorithm to diverge. However, making the algorithm converge to a suboptimal solution while being stealthy is challenging and requires deliberate manipulations.

There is limited research on cyberattacks that manipulate data shared between agents in distributed algorithms. Reference [3] investigates false data injection attacks on a primal-dual gradient descent distributed algorithm that solves OPF problems which use the DC power flow approximation (DCOPF problems). Reference [3] also proposes a detection method that estimates values for the shared variables using data communicated in previous iterations. The agents then update a reputation index for neighboring agents based on the deviation from the expected shared data values. An extension in [4] considers OPF problems for radial networks using a second-order cone programming (SOCP) relaxation. Other work in [5] proposes the use of power line communication and encryption to prevent "man-in-the-middle" attacks that compromise the communication between the agents. However, the method in [5] only works when a malicious agent compromises communications between the agents but not one of the agent's controllers. Moreover, the methods in [3]–[5] consider a simplistic attack scenario where the attacker repeatedly shares data corresponding to a malicious target solution.

We previously presented and analyzed various data manipulation strategies that drive distributed optimization algorithms for OPF problems to suboptimal solutions [6]–[8]. We consider a more stealthy attack model in [6] that uses a feedback loop to reduce the deviation of the manipulated data from the expected value. Our prior work also proposes an attack model that involves solving a bilevel optimization problem with the neighboring agents' subproblems in the lower level. The bilevel optimization attack model finds a solution that optimizes the attacker's malicious objective and ensures the convergence of the algorithm in two iterations. Due to the fast convergence of the bilevel optimization attack, reputation-based detection methods may fail since these methods require multiple iterations before flagging an attack.

Using neural network (NN) models for anomaly detection is a promising approach for detecting data manipulation [9]. Our prior work in [6], [8] shows that NN detection models have the potential to detect a variety of data manipulation strategies in the context of distributed optimization algorithms. However,

using NN models for anomaly detection is challenging due to the nature of anomalies, as they are rare and heterogeneous, and hence difficult to collect on a large scale for training [9].

The detection models discussed above are data-driven with no detectability guarantees. Conversely, [10] proposes an analytical detection method for convex problems via checking the convexity of the attacker's subproblem in the ADMM algorithm. The method in [10] estimates the Hessian matrices for the objectives of the neighboring agents' local subproblems. If the estimated Hessians are not positive semidefinite, which is necessary for convexity, then there is data manipulation. This method requires shared data from a large number of iterations. Moreover, numerical issues may arise due to poor matrix conditioning when estimating the Hessian matrices, especially for subproblem solutions that are close to the optimal solution. Thus, [10] uses additional shared data iterates to improve conditioning at the expense of more computations. Nevertheless, these challenges may cause inaccurate estimations, potentially resulting in false negatives. Moreover, the method in [10] is only applicable to convex problems.

This paper investigates the vulnerability of the ADMM algorithm to elaborate data manipulation attacks. To check the correctness of the shared data, we verify that the shared data corresponds to a solution for an optimization problem that is consistent with the solutions from previous iterations. We derive two conditions that must be satisfied if the shared data correspond to solutions of the same optimization problem for the ADMM algorithm. These conditions use the shared data and can be calculated by neighboring agents.

The first condition is based on the convergence proof of the ADMM algorithm presented in [11]. This condition uses the shared and dual variables in addition to the final solution to check if the convergence trajectory is valid for the ADMM algorithm. The second condition is based on the optimality conditions of the ADMM subproblems. This condition uses the shared variable values to verify the consistency of the agents in solving the same subproblem across iterations. Unlike the first condition, which uses the final solution, the second condition uses shared data from two consecutive iterations. Thus, agents can evaluate the second condition during any intermediate iterations of the algorithm. Moreover, the second condition is suitable for both convex and non-convex problems.

This paper then presents data manipulation attacks that bypass the machine learning detection model we previously proposed in [6] and also account for the two detection conditions we propose in this paper. These attacks embed the trained NN and detection conditions into the attacker's problem. Finally, we propose an adversarial NN training framework that improves the detectability of data manipulation even if the attacker has access to the detection models.

This paper is organized as follows. Section II presents the OPF problem and the ADMM algorithm. Section III proposes two detection conditions. Section IV describes three data manipulation strategies that bypass the detection methods. Section V presents a framework to detect data manipulation using an adversarially trained NN. Section VI shows results from numerical simulations. Section VII gives conclusions.

II. Background

This section presents the background information and notation for the OPF problem and the ADMM algorithm along with a sketch of the ADMM convergence proof.

A. Optimal Power Flow

OPF is a fundamental optimization problem in power systems that finds the controller setpoints that minimize operational cost subject to the power flow equations and engineering constraints. There is a wide variety of OPF formulations, including various approximations and relaxations [12]. We present a general OPF formulation with AC power flow constraints for illustrative purposes, but the results in this paper apply to other OPF formulations. The OPF problem is

$$\min \quad \sum_{g\in\mathcal{G}} c_{g_2}\Re(S_g^G)^2 + c_{g_1}\Re(S_g^G) + c_{g_0} \tag{1a}$$

subject to:

$$\sum_{g\in\mathcal{G}_i} S_g^G - \sum_{l\in\mathcal{L}_i} S_l^L = \sum_{(i,j)\in\mathcal{E}} S_{ij}, \qquad \forall i\in\mathcal{N}, \tag{1b}$$

$$S_{ij} = Y_{ij}^* V_i V_i^* - Y_{ij}^* V_i V_j^*, \qquad \forall (i,j)\in\mathcal{E}, \tag{1c}$$

$$V_i^{min} \le |V_i| \le V_i^{max}, \qquad \forall i\in\mathcal{N}, \tag{1d}$$

$$S_g^{min} \le S_g^G \le S_g^{max}, \qquad \forall g\in\mathcal{G}, \tag{1e}$$

$$|S_{ij}| \le S_{ij}^{max}, \qquad \forall (i,j)\in\mathcal{E}, \tag{1f}$$

$$\angle V_r = 0, \tag{1g}$$

where $\mathcal{N}$, $\mathcal{E}$, $\mathcal{G}$, and $\mathcal{L}$ are the sets of buses, branches, generators, and loads, respectively. The subsets $\mathcal{G}_i \subset \mathcal{G}$ and $\mathcal{L}_i \subset \mathcal{L}$ are the corresponding elements connected to bus $i \in \mathcal{N}$. The decision variables are the buses' voltage phasors $V_i \in \mathbb{C}$, $\forall i \in \mathcal{N}$, the generators' complex power outputs $S_g^G \in \mathbb{C}$, $\forall g \in \mathcal{G}$, and the branches' complex power flows $S_{ij} \in \mathbb{C}$, $\forall (i,j) \in \mathcal{E}$. We denote the complex load demands with $S_l^L \in \mathbb{C}$, $\forall l \in \mathcal{L}$ and branch series admittances with $Y_{ij} \in \mathbb{C}$, $\forall (i,j) \in \mathcal{E}$. Generator $g \in \mathcal{G}$ has a quadratic cost function with coefficients c_{g_2}, c_{g_1}, and c_{g_0}. We use $\Re(\cdot)$, $|\cdot|$, $\angle(\cdot)$, and $(\cdot)^*$ to denote the real part, magnitude, phase angle, and conjugate of complex variables.

The objective (1a) minimizes the generation cost. The equalities (1b) enforce power balance, and (1c) are the branches' power flow expressions. The inequalities (1d)–(1f) bound bus i voltage magnitude between V_i^{min} and V_i^{max}, generator g power output between S_g^{min} and S_g^{max}, and branch (i,j) apparent power flow below S_{ij}^{max}. The bounds (1e) are element-wise on the real and imaginary parts. Constraint (1g) sets the reference angle at the chosen reference bus r.

The formulation in (1) is commonly called the *ACOPF* problem because of the inclusion of the AC power flow equations in (1b)–(1c). Since the ACOPF problem (1) is non-convex, local non-linear solvers are frequently used to find good solutions. Other OPF formulations use various approximations, e.g., DCOPF, and convex relaxations, e.g., SOCP relaxation [12]. For notational simplicity, we group the ACOPF variables in a vector $x = [V \;\; S^G \;\; S]^T$ and use $[x]_j$ to

denote the j-th entry of x. We denote the equality and inequality constraints as $h^E(x) = 0$ and $h^I(x) \leq 0$. We further define feasible solutions as the set $\Omega = \{x | h^E(x) = 0,\, h^I(x) \leq 0\}$.

B. Alternating Direction Method of Multipliers

ADMM is a well-known distributed optimization algorithm based on the augmented Lagrangian method. We first present the general form of the ADMM algorithm. We then describe a special case of ADMM that solves the consensus problem.

1) General Form: The ADMM algorithm solves:

$$\min_{x,z} \quad f(x) + g(z) \tag{2a}$$

$$\text{subject to:} \quad Ax + Bz = c, \tag{2b}$$

$$x \in \mathcal{X},\ z \in \mathcal{Z}, \tag{2c}$$

where x and z are length-n and m vectors of decision variables constrained to the convex closed sets $\mathcal{X}$ and $\mathcal{Z}$, and $A \in \mathbb{R}^{p \times n}$, $B \in \mathbb{R}^{p \times m}$, and $c \in \mathbb{R}^p$ are the consistency constraint parameters. The functions $f\colon \mathcal{X} \to \mathbb{R}$ and $g\colon \mathcal{Z} \to \mathbb{R}$ are convex functions. The formulation (2) has decision variables that can be divided into two sets of variables x and z with separable objective functions and linear consistency constraints.

The ADMM algorithm uses an augmented Lagrangian function to relax the consistency constraints (2b):

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}||Ax + Bz - c||_2^2, \tag{3}$$

where $y \in \mathbb{R}^p$ are dual variables and $\rho \in \mathbb{R}_{>0}$ is tuning parameter. The ADMM algorithm solves the augmented Lagrangian problem by alternatively solving for the x and z variables and then taking an ascending step to update the dual variables. Thus, the iterate $k+1$ solutions are:

$$x^{k+1} := \operatorname*{argmin}_{x \in \mathcal{X}} f(x) + (y^k)^T(Ax + Bz^k - c) + \frac{\rho}{2}||Ax + Bz^k - c||_2^2, \tag{4a}$$

$$z^{k+1} := \operatorname*{argmin}_{z \in \mathcal{Z}} g(z) + (y^k)^T(Ax^{k+1} + Bz - c) + \frac{\rho}{2}||Ax^{k+1} + Bz - c||_2^2, \tag{4b}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c). \tag{4c}$$

Defining the primal residual $r^k = Ax^k + Bz^k - c$ and the dual residual $s^k = \rho A^T B(z^{k+1} - z^k)$ at iteration k, the algorithm terminates when the norms (often l_2 or l_∞) of the primal and dual residuals are below a predefined tolerance.

2) Consensus Problem: Consensus problems are a special case of the problem in (2) that have multiple subproblems with separable objective functions and global shared variables. Let $\mathcal{A}$ be the set of agents solving the subproblems and $\mathcal{R}$ be the set of shared variables. The subset $\mathcal{R}_i \subset \mathcal{R}$ is the set of variables belonging to agent $i \in \mathcal{A}$ with $\bigcup_{i \in \mathcal{A}} \mathcal{R}_i = \mathcal{R}$. We group the decision variables belonging to agent $i \in \mathcal{A}$ in a length-n_i vector x_i where n_i is the number of variables belonging to agent i. Local convex constraints on x_i are denoted as $\mathcal{X}_i$ for each agent $i \in \mathcal{A}$. For any shared variables between agents, we introduce auxiliary variables $z \in \mathbb{R}^{|\mathcal{R}|}$ and constraints that enforce consistency of the shared variables. Let $z_i \in \mathbb{R}^{|\mathcal{R}_i|}$ be a subset of the auxiliary variables z that correspond to the shared variables belonging to agent i. The consensus optimization problem is:

$$\min_{x,z} \quad \sum_{i \in \mathcal{A}} f_i(x_i) \tag{5a}$$

$$\text{subject to:} \quad x_i - z_i = 0, \qquad \forall i \in \mathcal{A}, \tag{5b}$$

$$x_i \in \mathcal{X}_i, \qquad \forall i \in \mathcal{A}. \tag{5c}$$

Problem (5) is a special case of (2) with $f(x) = \sum_{i \in \mathcal{A}} f_i(x_i)$, $g(z) = 0$, $c = 0$, and $A = \text{diag}(A_1, A_2, \ldots, A_{|\mathcal{A}|})$, i.e., a block-diagonal matrix with blocks $A_i \in \{0, 1\}^{p_i \times n_i}$, where p_i and n_i are the number of agent i's shared variables and decision variables, respectively. The entries $[A_i]_{j_1, j_2} = 1$ if the variable $[x_i]_{j_2}$ is a shared variable that appears in the consistency constraints j_1 and $[A_i]_{j_1, j_2} = 0$ otherwise. $B \in \{-1, 0\}^{p \times |\mathcal{R}|}$ is defined such that $[B]_{j,r} = -1$ if the shared variable r appears in consistency constraints j and $[B]_{j,r} = 0$ otherwise. The ADMM algorithm then solves:

$$x_i^{k+1} := \operatorname*{argmin}_{x_i \in \mathcal{X}_i} f_i(x_i) + (y_i^k)^T x_i + \frac{\rho}{2}||x_i - z_i^k||_2^2, \qquad \forall i \in \mathcal{A}, \tag{6a}$$

$$[z^{k+1}]_r := \frac{1}{N_r} \sum_{i \in \mathcal{A}} [x_i^{k+1}]_r, \qquad \forall r \in \mathcal{R}, \tag{6b}$$

$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - z_i^{k+1}), \qquad \forall i \in \mathcal{A}, \tag{6c}$$

where N_r is the number of agents with shared variable r and $[z^{k+1}]_r$ is the r-th shared variable. We denote y_i as the dual variables associated with agent i.

To use ADMM algorithm (6) to solve OPF problems (1), consider the case when there are multiple agents defined by the set $\mathcal{A}$, each of which operates a partitioned subset of the buses $\mathcal{N}_i \subset \mathcal{N}, i \in \mathcal{A}$ along with the corresponding connected elements, i.e., generators $\mathcal{G}_j$, loads $\mathcal{L}_j$, and shunts $\mathcal{H}_j, \forall j \in \mathcal{N}_i$. While OPF problem's objective function (1a) is separable, the variables and constraints are coupled. To eliminate the coupling, we introduce auxiliary variables corresponding to the bus voltage phasors for each boundary bus and the power flows on each branch connecting two buses from different agents. Each agent has a copy of the coupled constraints and variables in addition to consistency constraints between the coupled variables and the corresponding auxiliary variables. The consensus OPF problem thus has the form of (5) with $\mathcal{X}_i = \Omega_i, \forall i \in \mathcal{A}$, i.e., the set of OPF constraints in (1), and can therefore be solved using the ADMM algorithm (6). However, the ADMM convergence guarantee is limited to convex problems, which is not the case for the OPF problem in (1). Nonetheless, empirical results show that the ADMM algorithm frequently converges to good solutions for non-convex OPF problems [13]. Further, other convex OPF formulations that use various linearization and relaxation methods can be solved using ADMM with convergence guarantees.

3) ADMM Convergence: Consider a problem in the form (2) (or (5)) solved using the ADMM algorithm in (4) (or (6)). If the objective functions f and g are closed, proper, and convex, and the Lagrange function L_0 in (3) has a saddle point, then, as $k \to \infty$, the primal residual

$r^k = Ax^k + Bz^k - c$ converges to zero ($r^k \to 0$), the dual residual $s^k = \rho A^T B(z^{k+1} - z^k)$ converges to zero ($s^k \to 0$), and the objective function converges to the optimal solution.

The ADMM convergence proof is stated in [11, Appendix A]. We will use this convergence proof in Section III to derive a sufficient condition to detect shared data manipulations. For this paper, we describe the main inequalities in the proof. We refer readers to [11] for a detailed derivation.

Let $p^k = f(x^k)+g(z^k)$ be the objective value at iteration k and p^* be the optimal objective value of (2). If (x^*, z^*, y^*) is a saddle point of the Lagrange function L_0, i.e., $L_0(x^*, z^*, y) \leq L_0(x^*, z^*, y^*) \leq L_0(x, z, y^*)$, $\forall (x, z, y) \in \mathcal{X} \times \mathcal{Z} \times \mathbb{R}^p$, then the Lagrange function value at the saddle point is equal to the optimal objective value, i.e., $L_0(x^*, z^*, y^*) = f(x^*)+g(z^*) = p^*$ since finding a saddle point is a sufficient optimality condition for convex problems [14]. Define V^k as

$$V^k := \frac{1}{\rho}||y^k - y^*||_2^2 + \rho||B(z^k - z^*)||_2^2. \quad (7)$$

The values of V^k are nonnegative and depend on the distance of the primal and dual variables from the optimal solutions. The convergence proof shows that V^k decreases at each iteration via deriving the following inequality:

$$\rho||r^{k+1}||_2^2 + \rho||B(z^{k+1} - z^k)||_2^2 \leq V^k - V^{k+1}. \quad (8)$$

Since V^k is a nonnegative, then V^k decreases in each iteration by an amount that depends on the norms of the primal residual and the change in the shared variables over two iterations. Summing the inequality (8) from $k = 0$ to ∞, we obtain

$$\rho \sum_{k=0}^{\infty} ||r^{k+1}||_2^2 + ||B(z^{k+1} - z^k)||_2^2 \leq V^0 - V^\infty \leq V^0, \quad (9)$$

where the last inequality is due to the nonnegativity of V^k. This implies $r^k \to 0$ and $B(z^{k+1} - z^k) \to 0$ as $k \to \infty$. Multiplying $B(z^{k+1} - z^k)$ by ρA^T, we obtain $s^k \to 0$ as $k \to \infty$. The remaining steps in the proof show that the objective function converges to the optimal solution.

III. Two Sufficient Conditions for Anomalous Shared Data

In this section, we provide two sufficient conditions to detect inconsistencies in the solutions shared by the agents when using the ADMM algorithm. The first condition uses the ADMM convergence proof to detect data manipulation, while the second condition exploits the fact that the agents should be solving the same subproblem at each iteration.

The main idea of the first condition is to check if the algorithm follows an expected convergence trajectory by assessing whether the values of V_k from (7) form a decreasing sequence. Thus, the first detection condition is:

Convergence Consistency (CC)**.** Let x^k, z^k, and y^k be the iterate k values of the ADMM algorithm (4) that solves a convex optimization problem in the form (2) and converged at iteration K. Let $V^k = \frac{1}{\rho}||y^k - y^K||_2^2 + \rho||B(z^k - z^K)||_2^2, k = 1, 2, \ldots, K$. If $V^k - V^{k-1} \geq \epsilon$ for any $k = 2, 3, \ldots, K$ and small $\epsilon \in \mathbb{R}_{>0}$, then there is data manipulation.

Here, we use ϵ to account for the tolerance in the ADMM stopping criteria as well as the solver's numerical tolerance when solving the subproblems. Looser stopping criteria and larger numerical tolerances used by the solver require larger values of ϵ since the condition uses y^K and z^K instead of the actual optimal solutions y^* and z^* to calculate V^k in (7).

Although the first condition is valid, using this condition to detect data manipulation has four drawbacks. This condition 1) requires knowledge of the optimal solution and thus can not be used before the algorithm converges to a solution, 2) requires access to all shared variables to evaluate the norms, 3) does not provide information about the agents that manipulated the data, and 4) only applies to convex problems. To overcome these drawbacks, we next propose another condition based on the optimality of the local subproblems.

The second condition uses the fact that the agents repeatedly solve the same subproblem with different parameters. We exploit the subproblem structure to derive a necessary condition for the subproblems' solutions that only depends on the shared variables, which are not private. When an agent violates this condition, the agent must not be solving the same subproblem, thus yielding a sufficient condition to detect data manipulation:

Solutions Consistency (SC)**.** Let x^k and z^k be the iterate k solutions of the ADMM algorithm (4). Let $\hat{z}^k = 2Bz^k - Bz^{k-1} - c, k \geq 1$. If $(x^{k+1} + A^{-1}\hat{z}^k)^T A^T A(x^{k+1} - x^k) \geq \epsilon$ for any $k \geq 1$ and small $\epsilon \in \mathbb{R}_{>0}$, then there is data manipulation.

We derive the SC condition using the local subproblems' first-order optimality conditions. Since the agents share optimal solutions at each iteration, we can validate solution optimality by cross evaluating their objective functions with the current and previous shared variables.

To derive this condition, let $\bar{f}(x) := f(x)+\delta_{\mathcal{X}}(x)$, the local subproblem's objective function with the indicator function $\delta_{\mathcal{X}}(x)$, where $\delta_{\mathcal{X}}(x) = 0$ if $x \in \mathcal{X}$ and $\delta_{\mathcal{X}}(x) = \infty$ otherwise. Let $\bar{L}_\rho(x, z, y) := \bar{f}(x) + y^T(Ax + Bz - c) + \frac{\rho}{2}||Ax + Bz - c||_2^2$, the augmented Lagrange function (4a) with the indicator function $\delta_{\mathcal{X}}(x)$. At iteration $k + 1$, we have $x^{k+1} := \operatorname{argmin}\ \bar{L}_\rho(x, z^k, y^k)$. Assume f is lower semi-continuous, but not necessarily convex, over the set $\mathcal{X}$ [15, Def. 1.5], and $\mathcal{X}$ satisfies the linear independence constraint qualification (LICQ) at x^{k+1} [16, Def. 12.4]. Since x^{k+1} minimizes $\bar{L}_\rho(x, z^k, y^k)$, then $0 \in \partial \bar{L}_\rho(x^{k+1}, z^k, y^k)$, the subgradient of the local subproblem objective function evaluated at x^{k+1} [15, Theorem 8.15]. Thus,

$$\begin{aligned}
0 \quad &\in \partial \bar{L}_\rho(x^{k+1}, z^k, y^k), \\
&= \partial[\bar{f}(x^{k+1}) + (y^k)^T(Ax^{k+1} + Bz^k - c) \\
&\quad + \frac{\rho}{2}||Ax^{k+1} + Bz^k - c||_2^2], \\
&= \partial \bar{f}(x^{k+1}) + A^T(y^k) + \rho A^T(Ax^{k+1} + Bz^k - c), \\
&= \partial \bar{f}(x^{k+1}) + A^T(y^k) \\
&\quad + \rho A^T \left[\frac{1}{\rho}(y^{k+1} - y^k) - Bz^{k+1} + c + Bz^k - c\right], \\
&= \partial \bar{f}(x^{k+1}) + A^T(y^{k+1}) - \rho A^T(Bz^{k+1} - Bz^k),
\end{aligned}$$

where the fourth equality uses the dual update (4c) to substitute for $Ax^{k+1} = \frac{1}{\rho}(y^{k+1}-y^k)-Bz^{k+1}+c$. This implies that x^{k+1} also minimizes $\bar{f}(x) + (y^{k+1})^T Ax - \rho(Bz^{k+1} - Bz^k)^T Ax$. Thus, we have

$$\begin{aligned}\bar{f}(x^{k+1}) + (y^{k+1})^T Ax^{k+1} - \rho(Bz^{k+1} - Bz^k)^T Ax^{k+1} \leq \\ \bar{f}(\tilde{x}) + (y^{k+1})^T A\tilde{x} - \rho(Bz^{k+1} - Bz^k)^T A\tilde{x}_i, \qquad (10)\end{aligned}$$

for any $\tilde{x} \in \mathbb{R}^n$. Replacing k with $k-1$, we have

$$\begin{aligned}\bar{f}(x^k) + (y^k)^T Ax^k - \rho(Bz^k - Bz^{k-1})^T Ax^k \leq \\ \bar{f}(\tilde{x}) + (y^k)^T A\tilde{x} - \rho(Bz^k - Bz^{k-1})^T A\tilde{x}, \qquad (11)\end{aligned}$$

for any $\tilde{x} \in \mathbb{R}^n$. Letting $\tilde{x} = x^k$ in (10), $\tilde{x} = x^{k+1}$ in (11), and combining both inequalities, we obtain the following:

$$\begin{aligned}&\bar{f}(x^{k+1}) + (y^{k+1})^T Ax^{k+1} - \rho(Bz^{k+1} - Bz^k)^T Ax^{k+1} \\ &\quad + \bar{f}(x^k) + (y^k)^T Ax^k - \rho(Bz^k - Bz^{k-1})^T Ax^k \leq \\ &\quad \bar{f}(x^k) + (y^{k+1})^T Ax^k - \rho(Bz^{k+1} - Bz^k)^T Ax^k + \\ &\quad + \bar{f}(x^{k+1}) + (y^k)^T Ax^{k+1} - \rho(Bz^k - Bz^{k-1})^T Ax^{k+1}.\end{aligned}$$

The values of local objective functions with the local constraints $\bar{f}(x^k)$ and $\bar{f}(x^{k+1})$, which are private information, cancel out, and only the shared variables remain. Rearranging the terms and substituting for the dual variables, we obtain

$$\begin{aligned}&(y^{k+1})^T(Ax^{k+1}-Ax^k) - \rho(Bz^{k+1}-Bz^k)^T(Ax^{k+1}-Ax^k) \\ &\qquad - (y^k)^T(Ax^{k+1} - Ax^k) \\ &\qquad + \rho(Bz^k - Bz^{k-1})^T(Ax^{k+1} - Ax^k) \leq 0, \\ &(y^{k+1} - y^k)^T(Ax^{k+1} - Ax^k) \\ &\quad - \rho(Bz^{k+1} - 2Bz^k + Bz^{k-1})^T(Ax^{k+1} - Ax^k) \leq 0, \\ &\rho(Ax^{k+1} + Bz^{k+1} - c)^T(Ax^{k+1} - Ax^k) \\ &\quad - \rho(Bz^{k+1} - 2Bz^k + Bz^{k-1})^T(Ax^{k+1} - Ax^k) \leq 0, \\ &\rho(Ax^{k+1} - c)^T(Ax^{k+1} - Ax^k) \\ &\qquad - \rho(-2Bz^k + Bz^{k-1})^T(Ax^{k+1} - Ax^k) \leq 0, \\ &(Ax^{k+1} + 2Bz^k - Bz^{k-1} - c)^T(Ax^{k+1} - Ax^k) \leq 0. \qquad (12)\end{aligned}$$

We use the dual update (4c) in the third inequality to substitute $y^{k+1} - y^k = \rho(Ax^{k+1} + Bz^{k+1} - c)$. Factoring A and substituting $\hat{z}^k = 2Bz^k - Bz^{k-1} - c$ yields the SC condition.

For the consensus problem (5), we derive a similar condition by evaluating the consistency parameters A, B, and c as described in Section II-B2 and following the same derivation with the agents' subproblems. The condition thus becomes

$$(x_i^{k+1} - 2z_i^k + z_i^{k-1})^T(x_i^{k+1} - x_i^k) \leq 0, \qquad \forall i \in \mathcal{A}. \quad (13)$$

The only assumptions used to derive this condition are lower semi-continuity of the objective function f, satisfaction of LICQ, and agents' ability to solve their local subproblems to global optimality. The first two assumptions are typical for non-linear optimization (NLP) problems. The third assumption is hard to validate for non-convex problems. Nonetheless, empirical studies such as [17] show that local NLP solvers often find good solutions with small optimality gaps for many OPF instances. If an upper bound on the optimality gap is known, we can incorporate this bound in the value of ϵ when solving NLP problems to preclude false positives, i.e., avoid flagging an attack while there is no data manipulation.

Thus, this condition is sufficient for detecting data manipulation in non-convex problems with non-differentiable objectives, only requires the values of shared variables from three consecutive iterations, and can be checked at each iteration. Moreover, since we can evaluate this condition for each agent separately, the condition identifies the agents that manipulate the shared data. Unlike the CC condition where the value of ϵ depends on both the ADMM stopping criteria and the solver's numerical tolerances, the value of ϵ in the SC condition only depends on the solver's numerical tolerance since the derivation of this condition does not use the final solution of the ADMM algorithm. To the best of our knowledge, this is the first sufficient condition with these advantages.

IV. Data Manipulation with Embedded Detection Models

As we will show in Section VI with results for three attacker strategies presented in [6], the two conditions in Section III can detect various types of data manipulations. However, a sophisticated attacker with knowledge of the detection conditions could incorporate these conditions into their attack model to compute attacks that avoid detection. To analyze these sophisticated attacks, this section first reviews three data manipulation models from [6] that steer the ADMM algorithm to malicious operating points. This section then describes how an attacker could incorporate the conditions from Section III and the NN detection model from [6] to remain undetected.

A. Data Manipulation Strategies

We build on the attack model proposed in [6], which presents three data manipulation strategies that drive the distributed algorithm's results to a malicious operating point. We denote the attacker as agent a throughout this section.

1) Simple Attacker Model: In this model, the attacker simply shares the target values of the shared variables at each iteration. This model requires finding a target point that is feasible for neighboring agents before the attack starts. The simple attack is easily detected since the attacker sends the same values in each iteration.

2) Feedback Attacker Model: The feedback attack model is a generalization of the simple attack model that is harder to detect. Instead of repeatedly sending the exact target values, the attacker uses a proportional–integral–derivative (PID) feedback to compute the shared variables. Similar to the simple attack, the attacker determines the desired target values of the shared variables in advance. Let $\hat{x}$ be the attacker's target values, x_a be the shared variables that the attacker actually sends, and z be the shared variables as defined in (6). The attacker computes the shared variables at iteration $k+1$ as:

$$x_a^{k+1} = \hat{x} + K_p e_k + K_i \sum_{n=0}^{k} e_n + K_d(e_k - e_{k-1}), \qquad (14)$$

where K_p, K_i, and K_d are the PID feedback tuning parameters and $e_k = \hat{x} - z^k$ is the deviation of the neighboring agents' shared variables from the attacker's target values. Choosing $K_p = K_i = K_d = 0$ yields the simple attack model.

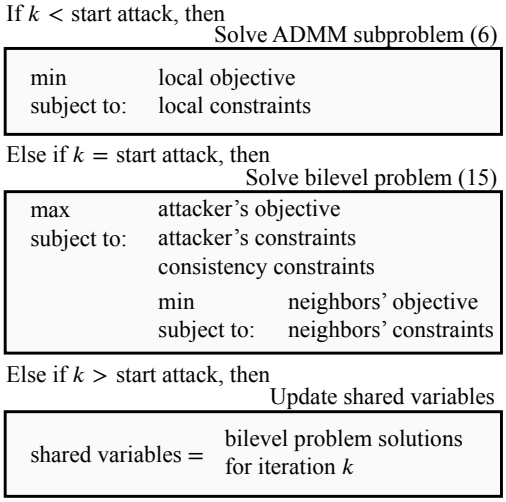

Figure 1. The attacker's logic when using the bilevel model. The attacker selects a starting iteration. Before starting the attack, the attacker solves normal ADMM subproblems (4). After starting the attack, the attacker shares manipulated data obtained by solving the bilevel problem (15).

3) Bilevel Attacker Model: This model involves solving a bilevel optimization that includes the neighboring agents' subproblems in the lower level and the attacker's objective and constraints in the upper level. As Fig. 1 illustrates, the attacker computes the shared data at a selected start iteration s and the next two iterations by solving (15):

$$\min_{x,y,z} \quad F(x_a^{s+2}) \tag{15a}$$

subject to:

$$x_i^k = \underset{x_i \in \Omega_i}{\operatorname{argmin}} f_i(x_i) + (y_i^{k-1})^T x_i + \frac{\rho}{2}||x_i - z_i^{k-1}||, \quad \forall i \in \mathcal{A}_a, k \in \{s, s+1, s+2\}, \tag{15b}$$

$$[z^k]_r = \frac{1}{N_r} \sum_{i \in \mathcal{A}} [x_i^k]_r, \forall r \in \mathcal{R}_i, i \in \mathcal{A}_a, k \in \{s, s+1\}, \tag{15c}$$

$$y_i^k = y_i^{k-1} + \rho(x_i^k - z_i^k), \quad \forall i \in \mathcal{A}_a, k \in \{s, s+1\}, \tag{15d}$$

$$x_a^{s+2} \in \Omega_a, \tag{15e}$$

$$x_a^{s+2} = x_i^{s+2}, \quad \forall i \in \mathcal{A}_a, \tag{15f}$$

where $F\colon \Omega_a \to \mathbb{R}$ is the attacker's objective function describing how they would like to manipulate the solution of the distributed optimization algorithm (e.g., increasing local generation in the attacker's region). The set Ω_i denotes the OPF constraints for agent i as defined in (1). The set $\mathcal{A}_a$ consists of the attacker's neighboring agents.

The lower-level problems in (15b) are the local subproblems of the neighboring agents for three consecutive iterations. The first iteration $k = s$ can be calculated in advance since the values of z^{s-1} and y^{s-1} are known, but we keep this iteration in the formulation to illustrate the need to find the corresponding x_i^s, $\forall i \in \mathcal{A}_a$. Constraints (15c)–(15d) are the ADMM update rules for iterations s and $s+1$. Constraints (15e) and (15f) ensure feasibility of the attacker's solution and reaching consensus on the shared variables at iteration $s+2$.

The complexity of the bilevel problem (15) depends on the power flow model used in the lower-level subproblems (15b) and the form of the upper-level objective (15a). With a linear power flow approximation and a linear objective function F in (15a), both the lower- and upper-level problems are linear programs. We can then solve the bilevel problem via reformulating the lower-level using the Karush–Kuhn–Tucker (KKT) conditions parameterized with the upper-level variables y and z. Then, we can use a standard big-M method [18] or a type-1 special ordered set (SOS1) method [19] to reformulate the KKT conditions as mixed-integer linear programming (MILP) constraints. In this paper, we use a DC power flow approximation for the sake of tractability, but other power flow models could be considered in future work.

B. Embedding Detection Models

This section extends the three data manipulation strategies we previously described to incorporate the SC detection condition in Section III and an NN detection model. We do not consider the CC condition described in Section III because of its dependence on the final solution and, as we will show in Section VI, we found that the SC condition is more accurate.

1) Embedding Detection Conditions: To bypass the SC detection condition described in Section III, the attacker could impose additional constraints on their shared variables to ensure satisfying (13). For the simple and feedback attack models, a sophisticated attacker could project the shared variables computed by the attacks onto the feasible region of the detection condition (13) by solving the following problem:

$$\min_{x} \quad ||x_a - \hat{x}_i||_2^2 \tag{16a}$$

subject to:

$$(x_a - 2z_i^k + z_i^{k-1})^T (x_a - x_a^k) \leq 0, \tag{16b}$$

$$x_a^{k+1} = \hat{x} + K_p e_k + K_i \sum_{n=0}^{k} e_n + K_d(e_k - e_{k-1}), \tag{16c}$$

where x_a is the attacker's shared variable vector, z_i is the vector of shared variables as defined in (6), $\hat{x}$ is the vector of target shared variable values, and $e_n = \hat{x}_i^n - z_i^n, n \in \{1, 2, \ldots, k\}$, are error terms. The objective function (16a) finds the closest shared variables to the target solution $\hat{x}$. Constraint (16b) ensures that the SC condition will not detect the attack, and (16c) is the output of the feedback attack.

Letting $K_p = K_i = K_d = 0$, we obtain the simple attack model with embedded detection conditions. The simple attack in this case is a projection of the attacker's target values onto the closest value that cannot be detected by the SC condition.

Due to the constraint (16b), problem (16) is a non-convex quadratic optimization problem that the attacker can solve using an NLP solver to find a local solution. Any feasible solution to (16) satisfies the inequality (13) and bypasses the SC condition. Thus, locally optimal solutions are sufficient to drive the ADMM algorithm to the attacker's target values while avoiding detection. However, there is no guarantee that the solutions of (16) are necessarily the best possible attack.

For the bilevel attack model, the attacker adds constraints to (15) that ensure satisfaction of the detection condition (13). The additional constraints are

$$(x_a^k - 2z_a^k + z_a^{k-1})^T (x_a^k - x_a^{k-1}) \leq 0 \tag{17}$$

for iterations $k \in \{s, s+1, s+2\}$, where x_a^s, x_a^{s+1}, and x_a^{s+2} are the attacker's shared variables; z_a^s, z_a^{s+1}, and z_a^{s+2} are the shared variables as defined in (6); and z_a^{s-1} are the shared variables from the prior iteration. As a non-convex quadratic inequality, enforcing (17) in the upper level increases the attacker's computational challenges when solving the bilevel problem (15). Nonetheless, commercial solvers like Gurobi are applicable to this type of quadratically constrained problem when appropriately formulated.

2) Embedding Neural Network Detection Model: Our previous work in [6] presents an NN detection model that shows promising results in detecting attacks. To avoid detection by the NN model in [6], this section presents a data manipulation attack that embeds the NN in the bilevel problem by adding constraints ensuring that the NN indicates no attack has occurred. For the simple and feedback attacks, embedding NN detection models into the attack strategies is more challenging, as we will discuss at the end of this section.

Consider an NN with a set $\mathcal{U}$ of layers indexed from 1 to $|\mathcal{U}|$, each with a set $\mathcal{V}_u, u \in \mathcal{U}$, of neurons indexed from 1 to $|\mathcal{V}_u|$. The inputs of the NN are $O_0 \in \mathbb{R}^{|\mathcal{V}_0|}$, where $\mathcal{V}_0$ is the set of the inputs indexed from 1 to $|\mathcal{V}_0|$. The output consists of a single neuron as $O_{|\mathcal{U}|} \in \mathbb{R}$. The NN layers' outputs are:

$$O_u = G_u(W_u O_{u-1} + b_u), \qquad \forall u \in \mathcal{U}, \tag{18}$$

where $G_u, u \in \mathcal{U}$, are the non-linear activation functions of the hidden and output layers, and $W_u \in \mathbb{R}^{|\mathcal{V}_u| \times |\mathcal{V}_{u-1}|}$ and $b_u \in \mathbb{R}^{|\mathcal{V}_u|}$ are the weight and bias parameters.

In [6], we trained an NN to detect shared data manipulation using the l_2-norm of the primal residuals $||r^k||_2$ from the last 50 iterations as inputs. When the NN outputs $O^{|\mathcal{U}|} < 0$, the NN flags an attack. We chose $G_u(x) = ReLU(x) = \max\{0, x\}$ (see Fig. 2) as the activation functions for the hidden layers and the identity for the output layer, i.e., $G_{|\mathcal{U}|}(x) = x$.

Building on the work in [6], we realized that using inputs to the NN based on the SC condition (13) evaluated for multiple iterations yields a better detection accuracy compared to the primal residuals. Specifically, the NN's input at iteration s is:

$$[O_0]_v = (x_a^k - 2z_a^{k-1} + z_a^{k-2})^T (x_a^k - x_a^{k-1}), \quad k = s - |\mathcal{V}_0| + v, \forall v \in \mathcal{V}_0. \tag{19}$$

To formulate this NN embedding, we use the method proposed in [20] that models an NN with activation functions $ReLU(x) = \max\{0, x\}$ as MILP constraints. We introduce binary variables $\phi_u \in \{0,1\}^{|V_u|}, \forall u \in \mathcal{U}$, to indicate the operation region of the $ReLU$ function as shown in Fig. 2. We also introduce positive real variables $I_u^+ \in \mathbb{R}_{\geq 0}^{|\mathcal{V}_u|}$ and $I_u^- \in \mathbb{R}_{\geq 0}^{|\mathcal{V}_u|}, \forall u \in \mathcal{U}$, to model the inputs of the hidden layer u, with constraints that permit only one of a particular neuron's variables to take a nonzero value. For each layer, the inputs of the activation functions are $I_u^+ - I_u^- = W_u O_{u-1} + b_u$ and the outputs are $O_u = ReLU(I_u^+ - I_u^-) = I_u^+$.

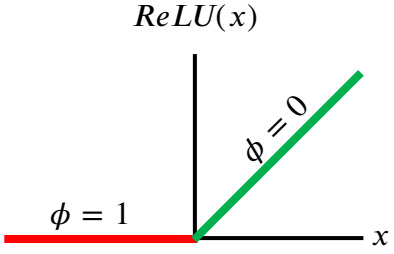

Figure 2. $ReLU(x) = \max\{0, x\}$ with binary variable $\phi \in \{0, 1\}$, where $\phi = 1$ indicates the inactive red region and $\phi = 0$ the active green region.

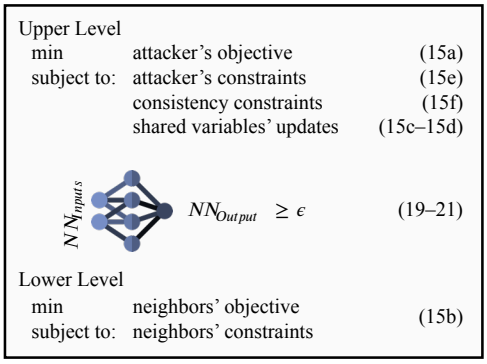

Figure 3. Conceptual illustration of the attacker's bilevel problem. The upper level minimizes the attacker's objective subject to the attacker's local constraints, consistency constraints, and embedded neural network. The lower-level problem consists of the neighboring agents' subproblems.

Using this notation, we formulate the NN as MILP at iteration k as follows. For each hidden layer $u \in \mathcal{U} \setminus \{|\mathcal{U}|\}$, we enforce the following constraints:

$$I_u^+ - I_u^- = W_u O_{u-1} + b_u, \tag{20a}$$
$$O_u = I_u^+, \tag{20b}$$
$$[\phi_u]_v = 1 \rightarrow [I_u^+]_v \leq 0, \qquad \forall v \in \mathcal{V}_u, \tag{20c}$$
$$[\phi_u]_v = 0 \rightarrow [I_u^-]_v \leq 0, \qquad \forall v \in \mathcal{V}_u, \tag{20d}$$
$$I_u^+ \geq 0,\ I_u^- \geq 0. \tag{20e}$$

Constraints (20c)–(20d) are disjunctive inequalities that ensure only one of same entry of I_u^+ and I_u^- has a nonzero value. These constraints can be reformulated as an MILP using Big-M or SOS1 methods. We define the inputs of the NN using (19) and the outputs as:

$$O^{|\mathcal{U}|} = W_{|\mathcal{U}|} O_{|\mathcal{U}|-1} + b_{|\mathcal{U}|}, \tag{21a}$$
$$O^{|\mathcal{U}|} \geq \epsilon. \tag{21b}$$

Constraint (21b) ensures that the NN does not detect the outputs of the bilevel problem. A small positive number ϵ in (21b) accounts for the feasibility tolerance of the solver. Fig. 3 illustrates the bilevel problem with an embedded NN.

Solving the bilevel problem with an embedded NN is hard and does not scale well with large systems. Moreover, the complexity of the problem increases as the number of the hidden layers increases. Methods for efficiently embedding NNs in optimization problems has been an increasingly studied topic in recent years. Other NN embedding models may produce stronger MILP formulation than (20) [21]. In future work, we plan to explore other NN embedding models with the bilevel optimization to scale to larger systems.

Embedding an NN to avoid detection with the simple and feedback attacks is more challenging than the bilevel optimization attack. The simple and feedback attacks compute the

shared variables for a single iteration. Solving the projection problem (16) with an embedded NN ensures the next iteration is undetected. However, the inputs of the NN detection model will be dominated by manipulated shared data after a few iterations. This makes finding the next iteration's value by solving (16) with the NN embedding (19)–(21) challenging because the attacker needs to consider many iterations after the next iteration to ensure finding feasible solutions. On the other hand, the bilevel attack avoids this problem because this attack ensures termination after two iterations. We will show in Section VI that embedding an NN with the simple and feedback attacks leads to infeasible solutions.

V. Adversarially Trained Neural Network

When augmented with (17)–(21), the outputs of the bilevel optimization (15) bypass both the detection condition (13) and the NN detection model presented in Section IV-B2. To better detect such sophisticated data manipulation attacks, we next develop an adversarial training framework. The framework uses the bilevel attack model to generate an adversarial training samples. We then iteratively train the NN on the adversarial samples as depicted in Fig. 4. We focus on the bilevel attack model because the simple and feedback attacks fail to find adversarial samples.

The framework's parameters are the numbers of initial training samples T_{train}, adversarial training iterations M_{train}, and samples for each adversarial training iteration $N_{samples}$. Initially, we generate a dataset with both attacked and unattacked samples. We then train an NN to detect the attacked samples. The trained NN detects all the attacked samples even with a small number of layers and neurons. However, the bilevel attack with an embedded NN, as described in Section IV-B2, can bypass this detection model.

Accordingly, the second stage of the framework iteratively trains the NN on the bilevel attack model's outputs. The goal of this framework is to train the NN until the bilevel problem with an embedded NN becomes infeasible or too computationally expensive to solve, rendering this attack strategy ineffective.

VI. Simulation Results

This section presents the computational results of the data manipulation strategies and detection methods when solving OPF problems using the ADMM algorithm. We first show the impacts of the attack strategies on the solutions of the ADMM algorithm. We next demonstrate the effectiveness of the detection methods and compare their performance. We then present the results of embedding the detection methods into the data manipulation strategies. Finally, we show results from the adversarial training of the NN framework to identify data manipulation with the embedded detection methods.

A. Simulation Setup

We solved OPF problems with the case3_lmbd, IEEE 14-bus, and IEEE 118-bus test cases from the PGLib-OPF library [22]. We decomposed the test systems into three areas and assume the attacker controls one of the areas. We set the

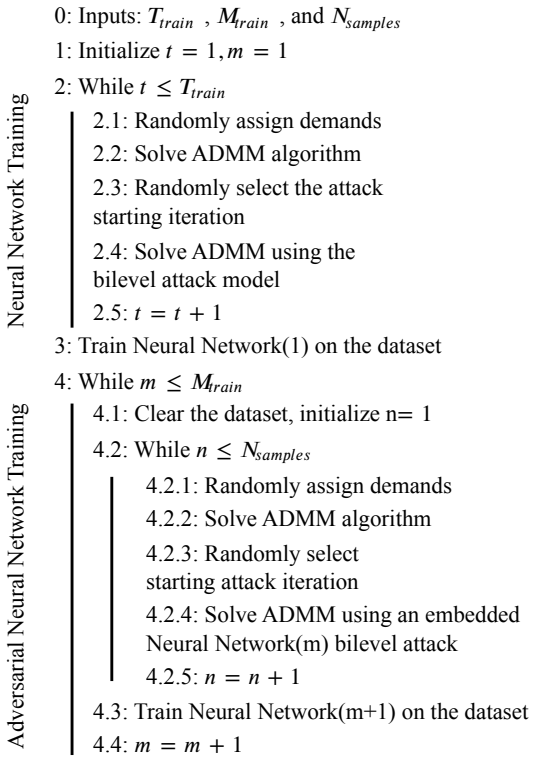

Figure 4. Adversarial training framework flowchart consisting of two stages: (1) initial training in steps 2 and 3, and (2) adversarial training in step 4.

tuning parameter $\rho = 100$ and terminate the ADMM algorithm when the l_∞-norm of the primal residuals is less than 10^{-2}.

Our implementation in Julia uses the PowerModelsADA library [23] for solving OPF problems with the ADMM algorithm and the BilevelJuMP library [24] for solving the bilevel optimization problems. We trained the NN models using the Flux library [25] and used Gurobi to solve all the optimization problems. The simulations use an Intel Xeon CPU with 24 physical cores and 16GB memory.

B. Data Manipulation

We first present the results of the data manipulation attacks. The attacker controls one of the areas and tries to steer the output of the ADMM to a suboptimal solution that increases the attacker's local generation. The attacker uses three data manipulation strategies: (1) simple attack, (2) feedback attack, and (3) bilevel optimization attack. In both the simple and feedback attacks, the attacker finds the target value before starting the iterative process, while the bilevel attack finds the maximum achievable target values within the next two iterations after starting the attack.

We used 100 runs to generate the results. In each run, we randomly select the load demands between 50% and 150% of the base demands with a constant power factor. We also randomly select a starting iteration for the attack within the first hundred iterations. We use the *optimality gap*, defined as the relative change in the objective function with and without data manipulation, to measure the impacts of the attacks. Table I shows the average optimal objective function value of the DCOPF problem and the average optimality gap when

Table I
THE AVERAGE OPTIMAL SOLUTION OF THE TEST CASES OVER 100 RUNS AND THE OPTIMALITY GAP FROM THE THREE ATTACK STRATEGIES

Case	Optimal Solution	Simple Attack	Feedback Attack	Bilevel Attack
3-bus	4388.2	70.2%	70.2%	70.2%
14-bus	7628.3	24.4%	24.4%	29.2%
118-bus	125945.9	28.6%	28.6%	19.7%

Table II
ACCURACY OF THE DETECTION METHODS

case	Detection Method	No Attack[1]	Simple Attack[2]	Feedback Attack[2]	Bilevel Attack[2]
3-bus	CC	100%	73.7%	97.2%	100%
	SC	100%	98.5%	100%	100%
	NN	100%	100%	100%	100%
14-bus	CC	100%	0%	1.92%	100%
	SC	100%	98.4%	100%	100%
	NN	100%	100%	100%	100%
118-bus	CC	100%	80.9%	91.1%	94.2%
	SC	100%	98.5%	100%	94.2%
	NN	97.8%	99.2%	99.9%	94.2%

[1]True negative. [2]True positive.

using each attack strategy. The three attacks find suboptimal solutions that increase the attacker's local generation. The optimality gap increases up to 70% for the 3-bus system and around 20% for the 14- and 118-bus systems after the attacks.

C. Detection Methods

We present the results of detection methods with the CC and SC Conditions and a trained NN detection model. We checked the detection conditions for all iterations until convergence. For the NN detection, our previous work [6] shows that we can achieve high detection accuracy by increasing the number of hidden layers. The work in [6] uses 8 hidden layers with 50 inputs corresponding to the mismatches of the shared variables over the last 50 iterations. Here, we show that we can maintain high detection accuracy with fewer hidden layers while considering fewer prior iterations as inputs to the NN.

As described in (19), we use a NN with 4 hidden layers and inputs from the last 10 iterations of the ADMM algorithm before convergence. We train the NN by generating 4000 samples of the three attack strategies and normal ADMM solutions. We then test the results using 1000 samples to produce the statistics shown in Table II.

Both detection conditions avoid false negatives when the value of ϵ is appropriately selected. For the SC condition, we choose $\epsilon = 0.01$ to match the ADMM algorithm's tolerance, while for the CC condition, we choose $\epsilon = 0.1$ for the 3- and 14-bus test systems and $\epsilon = 5$ for the 118-bus system to avoid false negative, i.e., flagging an attack for unattacked samples.

The SC condition and the NN detection model successfully identify most of the attacks, with accuracy above 97% for most of the test cases and often achieving above 99% accuracy. The CC condition has the lowest detection accuracy in all the test cases and fails to detect the simple and feedback attacks on the 14-bus system. Although the NN models use 10 iterations as inputs, their performance is close to the SC condition, which uses all the shared data from the first to the last iteration as inputs. Moreover, we can enhance the accuracy of the NN models by increasing the number of hidden layers and inputs. However, there is no control over the false negative results of the NN outputs, which can be seen in the 118-bus test system.

Table III
SUCCESS RATE OF THE ATTACK STRATEGIES WITH EMBEDDED DETECTION METHODS

case	Detection Method	Feedback Attack	Bilevel Attack
3-bus	CC+SC	100%	100%
	NN	0%	72%
	NN+CC+SC	0%	44%
14-bus	CC+SC	100%	0%
	NN	0%	0%
	NN+CC+SC	0%	0%
118-bus	CC+SC	100%	0%
	NN	0%	0%
	NN+CC+SC	0%	0%

D. Embedding Detection Methods

The high detection accuracy demonstrated in Table II can be compromised if the attacker knows the detection models. To show this, we use the two attack strategies with the embedded detection model described in Section IV. We solve the models with 100 instances of the three test systems while varying the loads. We observe that the solver may take a very long time to find a solution due to the complexity of the problem. We set a maximum time limit of 100 seconds for the bilevel attack and 5 seconds for the feedback attack (since the attacker solves the bilevel problem once but repeatedly solves projection problems for each iteration).

The success rate of the two attacks is summarized in Table III. The results indicate that knowing the detection models is not enough to ensure a successful attack. For the 14- and 118-bus systems, the attack strategies fail to find solutions in all instances when embedding the trained NN model. Moreover, the feedback attack always bypasses the two detection conditions, while the bilevel attack fails with the 14- and 118-bus systems. On the other hand, the feedback attack fails to bypass the NN detection model for the reason we previously discussed at the end of Section IV.

E. Adversarial Training

To enhance the NN detection accuracy, we use the adversarial training framework described in Section V. We use the 3-bus system to demonstrate the effectiveness of the framework because the attack models fail to avoid detection with the other two test systems. We iteratively trained the NN on the outputs of the bilevel attack models with an embedded NN. At each iteration, we solve the bilevel problem with $N_{samples} = 100$ samples and collect the successful samples to retrain the NN. We use the bilevel attack model while only considering the NN model and then with both the NN and SC detection methods.

The results of the adversarial training are shown in Fig. 5. The NN successfully detects almost all (more than 99.9%) of the attacks with $M_{train} = 16$ iterations. We also noticed that the detection accuracy improved more quickly when using the SC condition alongside the NN. Thus, the detection conditions increase the efficiency of the training process by focusing on a subset of all the possible data manipulation scenarios to enhance the detection accuracy.

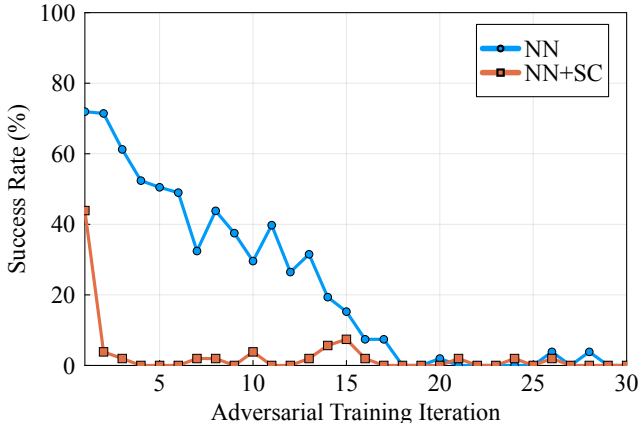

Figure 5. Success rate of the bilevel attack with embedded detection methods over the adversarial training iterations. The blue line indicates the result of the bilevel attack with an embedded neural network (NN) and the red line with an embedded neural network and solution consistency condition (NN+SC).

The results also indicate that there are a few instances where the bilevel attack finds a solution that is undetectable by the adversely trained NN. The framework does not guarantee that there is no possible bilevel attack solution. However, this result indicates that the possibility of finding one within 100 seconds is very low (i.e., less than 0.1% with the setup in this paper).

VII. Conclusion

Distributed algorithms have many advantages that make them appealing approaches for coordinating systems with multiple agents. Future power systems with thousands or millions of controllers may coordinate their operations by solving OPF problems using distributed algorithms. However, as we demonstrate in this paper, distributed algorithms may be vulnerable to shared data manipulation attacks that drive the algorithm to suboptimal solutions. Since the agents repeatedly solve the same subproblems, we exploit the subproblem structure to detect shared data manipulations by deriving two sufficient conditions. We also show that a sophisticated attacker with knowledge of the detection methods may avoid detection by embedding the detection conditions in their decision process. We then use an adversarially trained NN framework to enhance the detectability of data manipulation strategies even when the attacker knows the detection methods.

The proposed framework and results in this paper are based on embedding the detection methods into the attacker's data manipulation strategies. The embedding of the detection methods involves solving mixed-integer programs with non-convex quadratic constraints, which are computationally challenging. Accordingly, the results indicate that embedding the detection model into the attacker's problem does not scale well to large test systems. Our future work aims to develop embedding models that are more scalable through approximating or using stronger MILP models for the detection methods. Moreover, even the attacker's problem without the NN embedding is computationally difficult due to the need to ensure convergence of the distributed algorithm to the target solutions. Thus, our ongoing work aims to develop more scalable attack models that ensure convergence of the distributed algorithm.

Acknowledgment

The authors would like to thank Scott Moura for insightful discussions on convergence theory for ADMM algorithms.

References

[1] D. K. Molzahn, F. Dörfler, H. Sandberg, S. H. Low, S. Chakrabarti, R. Baldick, and J. Lavaei, "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] M. Alkhraijah, C. Menendez, and D. K. Molzahn, "Assessing the impacts of nonideal communications on distributed optimal power flow algorithms," *Electric Power Syst. Res.*, vol. 212, p. 108297, 2022, presented at *22nd Power Syst. Comput. Conf. (PSCC 2022)*.

[3] J. Duan, W. Zeng, and M. Chow, "Resilient distributed DC optimal power flow against data integrity attack," *IEEE Trans. Smart Grid*, vol. 9, no. 4, pp. 3543–3552, 2018.

[4] Z. Cheng and M. 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.

[5] Y. Yang, G. Raman, J. Peng, and Z. 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.

[6] M. Alkhraijah, R. Harris, S. Litchfield, D. Huggins, and D. K. Molzahn, "Analyzing malicious data injection attacks on distributed optimal power flow algorithms," in *54th North American Power Symp. (NAPS)*, 2022.

[7] 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 Conf. (TPEC)*, 2022.

[8] R. Harris and D. K. Molzahn, "Detecting and mitigating data integrity attacks on distributed algorithms for optimal power flow using machine learning," to appear in *57th Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2024.

[9] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, 2021.

[10] E. Munsing and S. Moura, "Cybersecurity in distributed and fully-decentralized optimization: Distortions, noise injection, and ADMM," arXiv:1805.11194, 2018.

[11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, 2010.

[12] D. K. Molzahn and I. A. Hiskens, "A survey of relaxations and approximations of the power flow equations," *Found. Trends Electric Energy Syst.*, vol. 4, no. 1-2, pp. 1–221, 2019.

[13] A. Kargarian, Y. Fu, and Z. Li, "Distributed security-constrained unit commitment for large-scale power systems," *IEEE Trans. Power Syst.*, vol. 30, no. 4, pp. 1925–1936, 2015.

[14] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[15] R. T. Rockafellar and R. J.-B. Wets, *Variational Analysis*. Springer Science & Business Media, 2009, vol. 317.

[16] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999.

[17] S. Gopinath and H. L. Hijazi, "Benchmarking large-scale ACOPF solutions and optimality bounds," in *IEEE Power & Energy Society General Meeting (PESGM)*, 2022.

[18] J. Fortuny-Amat and B. McCarl, "A representation and economic interpretation of a two-level programming problem," *J. Oper. Res. Soc.*, vol. 32, no. 9, pp. 783–792, 1981.

[19] J. P. Vielma and G. L. Nemhauser, "Modeling disjunctive constraints with a logarithmic number of binary variables and constraints," *Math. Prog.*, vol. 128, pp. 49–72, 2011.

[20] M. Cacciola, A. Frangioni, and A. Lodi, "Structured pruning of neural networks for constraints learning," arXiv:2307.07457, 2023.

[21] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. P. Vielma, "Strong mixed-integer programming formulations for trained neural networks," *Math. Prog.*, vol. 183, no. 1-2, pp. 3–39, 2020.

[22] IEEE PES Task Force on Benchmarks for Validation of Emerging Power System Algorithms, "The Power Grid Library for benchmarking AC optimal power flow algorithms," arXiv:1908.02788, Aug. 2019.

[23] M. Alkhraijah, R. Harris, C. Coffrin, and D. K. Molzahn, "PowerModelsADA: A framework for solving optimal power flow using distributed algorithms," to appear in *IEEE Trans. Power Syst.*, 2023.

[24] J. D. Garcia, G. Bodin, and A. Street, "BilevelJuMP.jl: Modeling and solving bilevel optimization in Julia," arXiv:2205.02307, 2022.

[25] M. Innes *et al.*, "Fashionable modelling with Flux," arXiv:1811.01457, 2018.