Secure Warm Start for Distributed Optimal Power Flow via Privacy-Preserving Neural Networks

Rachel Harris, Daniel Turizo, and Daniel K. Molzahn
School of Electrical and Computer Engineering
Georgia Institute of Technology
{rharris94,djturizo,molzahn}@gatech.edu

***Abstract*—Distributed optimization algorithms can provide scalability and privacy advantages for coordinating controllable resources in future smart grids. However, distributed optimal power flow (OPF) problems may take many iterations to reach consensus over large networks. To reduce the number of iterations we propose computing a privacy-preserving warm start for distributed OPF, which is used to solve the OPF via the alternating direction method of multipliers (ADMM) algorithm. We train neural networks (NNs) to predict final consensus variable values, and evaluate the neural network outputs using secure multi-party computation (MPC) to preserve the privacy of input data. Since MPC significantly increases the computational overhead of evaluating the neural network, we design a specialized NN architecture and use a recently created library to perform the secure computations more efficiently. We further reduce the number of iterations by combining the privacy-preserving warm start with a bound tightening algorithm; this allows us to use a looser convergence tolerance while ensuring feasibility. Numerical results demonstrate that the combined warm start and bound tightening results in very few iterations to convergence, and reduces computation time by around an order of magnitude.**

***Index Terms*—Distributed OPF, ADMM, secure multi-party computation, privacy, warm start, bound tightening**

I. Introduction

Power systems operations are changing due to increasing deployments of distributed energy resources (DERs). Rather than dispatching a small number of generators at the transmission level, future operators may coordinate many controllable resources throughout both transmission and distribution networks. Distributed optimization may be useful for solving large-scale optimal power flow problems to find DER setpoints. In distributed algorithms for optimal power flow (OPF), local controllers iteratively solve subproblems over their respective regions of the network and exchange data until neighboring controllers reach consensus on boundary variables. Distributed algorithms can scale to large OPF problems since controllers solve smaller OPF subproblems over local regions. Additionally, distributed OPF has the potential to preserve sensitive local data, such as power consumption, since these data are not directly shared. However, distributed OPF algorithms may take many iterations to reach consensus, requiring unacceptably long computation times.

This paper focuses on rapidly and securely solving distributed AC OPF using the alternating direction method of multipliers (ADMM) algorithm. ADMM, a popular distributed optimization algorithm, has been applied to various power system optimization problems [1]. Like other first-order distributed optimization algorithms, a key challenge for ADMM is reducing the number of iterations to convergence [2]–[4]. While ADMM often reaches modest accuracy quickly, it may be slow to converge to the high accuracy that is necessary to ensure feasible OPF solutions [5], [6]. Most of the literature on reducing computation time for ADMM-based distributed AC OPF focuses on parameter tuning. While [2] performed extensive numerical simulations to find good parameters, other papers proposed residual balancing schemes [3] or assigning parameters based on coupling between variables [4]. Another approach used reinforcement learning to dynamically select penalty parameters [7]. The work in [8] proposed a two-level ADMM variant with convergence guarantees for AC OPF and some analysis on worst-case iteration complexity.

We focus on a different way to reduce distributed OPF computation time: improving the initialization. To solve distributed OPF, neighboring controllers must reach consensus on voltage phasors and power flows at the boundary between their respective regions of the network. We refer to the boundary variables as "primal consensus variables," and the dual variables for the consensus constraints as "dual consensus variables." We propose using specially tailored machine learning to predict the optimal primal and dual consensus variable values. Past work on learning to predict *centralized* OPF solutions generally predicted optimal values for all generator setpoints [9]–[13], to avoid repeatedly computing solutions with optimization algorithms during real-time operation. However, in the distributed OPF setting, the full problem is decomposed into smaller subproblems such that OPF over any region's subproblem is relatively fast. Given values for the primal and dual consensus variables, which are the inputs for each distributed OPF iteration, the optimal generator setpoints can thus be found relatively quickly by solving the region's OPF subproblem. By predicting the final consensus variable values,

Support from the National Science Foundation under grant number #2112533: NSF Artificial Intelligence Research Institute for Advances in Optimization (AI4OPT) and the Georgia Tech Strategic Energy Institute's Energy and National Security program.

we aim to reduce the number of iterations to convergence.

Previous literature proposed a decentralized approach in which controllers for each region train models to predict consensus variable values from local power demand information [14]. This approach is highly effective for settings in which the loads across the system are highly correlated such that local controllers have implicit information about loading conditions in other regions based on the loads in their own region. They can then make high-quality predictions based on local power demand information alone.

Our proposed approach generalizes to systems where the loads are not so highly correlated. In this setting, regions should exchange some data regarding their local power demand in order to predict warm-start values that reflect conditions across the connected system. However, one key advantage of distributed optimization over centralized optimization is the potential to preserve privacy of local data. To preserve privacy, we propose using secure multi-party computation (MPC) [15] with central coordinators that calculate the warm-start using privacy-preserving neural networks (PPNNs) [16]. By using PPNNs, we provide cryptographically rigorous guarantees that the warm-started ADMM will not reveal any more information than the traditional ADMM algorithm.

We also propose combining the secure warm start with the bound tightening algorithm we developed in [6] to significantly reduce the number of iterations, and thus total computation time, required to converge. Reducing iteration count decreases the amount of data exchanged between controllers, which may also have important implications for privacy and cybersecurity.

To summarize, the contributions of the paper are as follows:

- We present a method for computing a privacy-preserving warm start for distributed OPF. While this warm start uses system-wide loading conditions as input, we ensure that power demand data remains private to each local region by evaluating the neural network (NN) output using MPC.
- We mitigate the computational burden associated with MPC using 1) a specialized NN structure and 2) a new library, developed with our collaborators, for secure operations on floating-point numbers. We emulate realistic communication between the secure agents and illustrate how the warm start reduces time to convergence.
- We numerically demonstrate that combining the warm start with bound tightening techniques from [6] reduces the number of iterations and convergence time by approximately an order of magnitude and we discuss the associated data security benefits.

The remainder of the paper is organized as follows. Section II provides an overview of the use of ADMM to solve distributed OPF problems. Section III details the neural network architecture, and describes how coordinating agents use MPC techniques to jointly compute the PPNN output, without these coordinating agents or any other agent learning the private input data. Section IV proposes combining the warm start with bound tightening to significantly reduce the number of iterations to convergence. Section V presents numerical results on OPF test cases. Section VI summarizes the paper.

II. Background: Distributed Optimal Power Flow

This section provides background on solving OPF in a distributed manner. We use the alternating direction method of multipliers (ADMM) algorithm [5] to solve distributed OPF in this paper, but our methods could also be applied to other distributed algorithms such as auxiliary problem principle [17] or alternating target cascading [18].

A. *Optimal Power Flow*

We first introduce notation and the AC OPF problem, which minimizes generation cost subject to engineering limits and the AC power flow equations. Let the sets of buses and lines be denoted $\mathcal{N}$ and $\mathcal{E}$, respectively. Each bus $i \in \mathcal{N}$ has active power demand p_i^d, reactive power demand q_i^d, active power generation p_i^g, reactive power generation q_i^g, voltage magnitude v_i, voltage angle θ_i, shunt conductance g_i^{sh}, and shunt susceptance b_i^{sh}. We denote the reference bus with $\mathcal{S}$. For each line $(i,j) \in \mathcal{E}$, we denote the series conductance as G_{ij}, the series susceptance as B_{ij}, the shunt susceptance as B_{ij}^{sh}, and the limit on apparent power flow as $\overline{S}_{ij}$. The active and reactive power flows from bus i to bus j are p_{ij} and q_{ij}, respectively. The angle difference between buses i and j is θ_{ij}. The total operating cost is the sum of the cost of active power generation at each bus i, denoted f_i. The AC OPF problem is

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

$$\text{s.t.} \quad \theta_i = 0 \text{ for } i \in \mathcal{S}, \tag{1b}$$

$$\forall i \in \mathcal{N},\ \forall (i,j)\in\mathcal{E}:$$

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

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

$$\begin{aligned} p_{ij} &= v_i^2 G_{ij} \\ &\quad - v_i v_j\left[G_{ij}\cos(\theta_{ij}) + B_{ij}\sin(\theta_{ij})\right], \end{aligned} \tag{1e}$$

$$\begin{aligned} q_{ij} &= -v_i^2 (B_{ij}^{sh} + B_{ij}) \\ &\quad - v_i v_j\left[G_{ij}\sin(\theta_{ij}) - B_{ij}\cos(\theta_{ij})\right], \end{aligned} \tag{1f}$$

$$\underline{P}_i^g \le p_i^g \le \overline{P}_i^g, \quad \underline{Q}_i^g \le q_i^g \le \overline{Q}_i^g, \tag{1g}$$

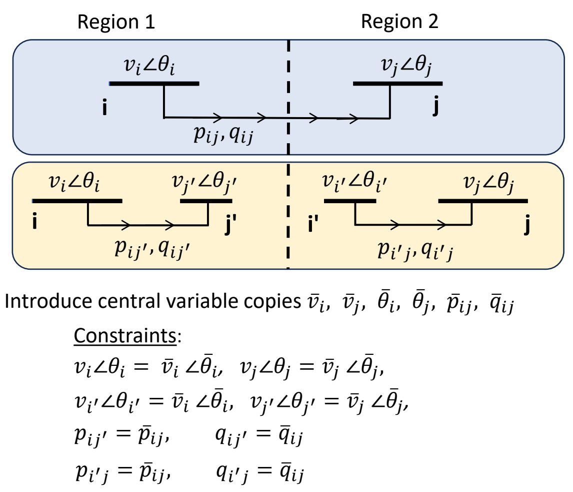
$$\underline{V}_i \le v_i \le \overline{V}_i, \tag{1h}$$

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

This problem minimizes generation cost (1a) subject to power balance and flow constraints (1c)–(1f) as well as limits on generation (1g), voltage magnitudes (1h), and line flows (1i).

B. *Solving Distributed OPF with ADMM*

To decompose the OPF problem, we divide the power network into multiple regions, each operated by a local controller. Controllers solve local OPF subproblems over their regions of the network. We must ensure that neighboring controllers reach consensus on values of primal consensus variables, which are the voltage phasors and power flows at



Introduce central variable copies $\bar{v}_i,\ \bar{v}_j,\ \bar{\theta}_i,\ \bar{\theta}_j,\ \bar{p}_{ij},\ \bar{q}_{ij}$

Constraints:

$$v_i\angle\theta_i = \bar{v}_i\,\angle\bar{\theta}_i,\ \ v_j\angle\theta_j = \bar{v}_j\,\angle\bar{\theta}_j,$$
$$v_{i'}\angle\theta_{i'} = \bar{v}_i\,\angle\bar{\theta}_i,\ \ v_{j'}\angle\theta_{j'} = \bar{v}_j\,\angle\bar{\theta}_j,$$
$$p_{ij'} = \bar{p}_{ij}, \qquad q_{ij'} = \bar{q}_{ij}$$
$$p_{i'j} = \bar{p}_{ij}, \qquad q_{i'j} = \bar{q}_{ij}$$

Fig. 1: Decomposition of power network

the boundary between their respective regions. To use ADMM to solve the distributed OPF problem over multiple regions, we introduce central copies of all primal consensus variables, and the consistency constraints require that each controller's local copy of primal consensus variables is equal to the central copies of these variables. The consistency constraints on power flows and voltage phasors are illustrated in Figure 1.

Let the set of regions across the network be $\mathcal{M}$. Region m contains a set of buses $\mathcal{N}_m$ and a set of lines $\mathcal{E}_m$. The values of primal consensus variables are contained in the vector $\boldsymbol{z}_m$. The central copies of primal consensus variables for region m are gathered into the vector $\bar{\boldsymbol{z}}_m$. If dual variables for consistency constraints are initialized to opposite values for each pair of primal consensus variables shared by neighboring controllers, then we can compute the central variable values in a distributed manner, without a central coordinator.

To solve distributed OPF with ADMM, controllers iteratively solve local subproblems and share primal consensus variable values with their neighbors. The subproblem constraints are simply the OPF constraints over the controller's region. The objective for each controller's local subproblem is augmented with relaxed consistency constraints. At iteration k, controller m's OPF subproblem is as follows:

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

$$\text{s.t.}\quad \theta_i = 0 \text{ for } i\in\mathcal{S}, \tag{2b}$$

$\forall i\in\mathcal{N}_m, \forall(i,j)\in\mathcal{E}_m$:

$$p_{i,m}^{g,k} - p_i^d = \sum_{(i,j)\in\mathcal{E}_m} p_{ij,m}^k + g_i^{sh}(v_{i,m}^k)^2, \tag{2c}$$

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

$$p_{ij,m}^k = (v_{i,m}^k)^2 G_{ij} - v_{i,m}^k v_j^k\left[G_{ij}\cos(\theta_{ij}^k) + B_{ij}\sin(\theta_{ij,m}^k)\right], \tag{2e}$$

$$q_{ij,m}^k = -(v_{i,m}^k)^2(B_{ij}^{sh} + B_{ij}) - v_{i,m}^k v_j^k\left[G_{ij}\sin(\theta_{ij,m}^k) - B_{ij}\cos(\theta_{ij,m}^k)\right], \tag{2f}$$

$$\underline{P}_i^g \le p_{i,m}^{g,k} \le \overline{P}_i^g, \quad \underline{Q}_i^g \le q_{i,m}^{g,k} \le \overline{Q}_i^g, \tag{2g}$$

$$\underline{V}_i \le v_{i,m}^k \le \overline{V}_i, \tag{2h}$$

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

The ADMM penalty parameter α is user-selected. After solving local subproblems, controllers share primal consensus variables $\boldsymbol{z}_m$ with their neighbors, and update central variables as $\bar{\boldsymbol{z}}_{m,n}^k = \frac{1}{2}(\boldsymbol{z}_{m,n}^k + \boldsymbol{z}_{n,m}^k)$, where for neighboring controllers (m,n), $\boldsymbol{z}_{m,n}$ are the primal consensus variable values computed by controller m and $\boldsymbol{z}_{n,m}$ are the values computed by controller n.

Controller m then updates the dual consensus variables $\boldsymbol{y}_m$:

$$\boldsymbol{y}_m^k = \boldsymbol{y}_m^{k-1} + \alpha(\boldsymbol{z}_m^k - \bar{\boldsymbol{z}}_m^k). \tag{3}$$

Thus, each ADMM iteration consists of solving local subproblems (2), updating central values for primal consensus variables, and updating dual consensus variables (3). The algorithm converges when the controllers reach consensus within some convergence tolerance ϵ. Let the vector of primal residuals $\boldsymbol{r}^k$ contain the difference between local and central copies of all boundary variable values:

$$\boldsymbol{r}^k = \begin{bmatrix}\boldsymbol{z}_1^T - \bar{\boldsymbol{z}}_1^T & \boldsymbol{z}_2^T - \bar{\boldsymbol{z}}_2^T & \dots & \boldsymbol{z}_M^T - \bar{\boldsymbol{z}}_M^T\end{bmatrix}^T. \tag{4}$$

We terminate the ADMM algorithm when $||\boldsymbol{r}^k||_\infty \le \epsilon$.

III. Privacy-Preserving Neural Networks for Secure Warm Start

In this section, we propose training a neural network (NN) to predict the final values for distributed OPF primal and dual consensus variables, which are then used to warm-start the distributed OPF algorithm. The input data are power demands at loads across the system. To preserve privacy, so that these power demands are not revealed to a central coordinator, we use secure multi-party computation (MPC) to evaluate the NN output [15], [16], [19]. To emphasize that we compute the NN output securely, we refer to the NN as a *privacy-preserving* neural network (PPNN). Since the central coordinating agents use MPC techniques to jointly compute the PPNN output, we have cryptographic guarantees that neither of the coordinating agents, nor any local agent, can discern the private input data. We first present the PPNN architecture and describe the inputs, outputs, and training process. Next, we explain the communication setup used to compute the warm start in real-time with MPC techniques. We discuss the impacts of the communication speed between coordinating agents on the PPNN evaluation time. Finally, we discuss the trust assumptions and privacy guarantees for our method.

A. Neural Network Architecture

We now describe the PPNN architecture, inputs, and outputs. First, we note that the MPC techniques, which keep the input information secure, also add computational overhead [16], [20], [21]. Therefore, we must ensure that the time required to securely evaluate the PPNN is sufficiently small such that the warm-start still reduces total distributed OPF computation time. To reduce the amount of secure computation required, we design NNs with the following architecture. For the first n layers, the NN is divided into m regions, each processing inputs independently. For these n layers, the neurons are connected only within their respective regions. At layer $n+1$, the outputs from each region's n-th layer are connected. For layers $n+1$ through N, the NN is fully connected and computations to evaluate the output are performed securely. The output is denoted $\boldsymbol{\zeta}$ and contains the predictions for consensus variable values. Each region receives the elements of $\boldsymbol{\zeta}$ corresponding to primal boundary variables and dual variables for coupling constraints in their region, denoted $\boldsymbol{\zeta}_m$. Regions then use these values to initialize the distributed OPF for faster convergence.

To formalize the architecture of the NN, let $\boldsymbol{x}_m$ denote the vector of active and reactive power demands at loads across region m, $\boldsymbol{x}_m = \begin{bmatrix}(\boldsymbol{p}_m^d)^T & (\boldsymbol{q}_m^d)^T\end{bmatrix}^T$. This vector $\boldsymbol{x}_m$ is the input to region m's separate NN. Using the same notation as in Section II-B, we denote the primal and dual consensus variables that form the input to controller m's local OPF subproblem as $\bar{\boldsymbol{z}}_m$ and $\boldsymbol{y}_m$, respectively. Let $\boldsymbol{\zeta}_m = \begin{bmatrix}(\bar{\boldsymbol{z}}_m)^T & (\boldsymbol{y}_m)^T\end{bmatrix}^T$. The full output vector from the fully connected PPNN layers is $\boldsymbol{\zeta} = \begin{bmatrix}(\boldsymbol{\zeta}_1)^T & (\boldsymbol{\zeta}_2)^T & \dots & (\boldsymbol{\zeta}_M)^T\end{bmatrix}^T$, where $M = |\mathcal{M}|$ is the number of regions.

Let $g_{m,i}$ denote the output of layer i of region m's network. Then, let $\boldsymbol{y}$ contain the concatenated outputs of each region's network at layer n. This vector $\boldsymbol{y}$ is the input to the privacy-preserving fully connected layers. We denote the output of layer j of the fully connected PPNN as $\boldsymbol{h}_j$, and the final output of the PPNN is $\boldsymbol{\zeta}$. The NN weights at layer i for region m are given as $\boldsymbol{W}_{m,i}$, and the PPNN weights for the fully connected network at layer j are $\boldsymbol{W}_j$. Similarly, the bias vector at layer i for region m is $\boldsymbol{b}_{m,i}$ and the bias vector for the fully connected layer j is $\boldsymbol{b}_j$. The output vector $\boldsymbol{\zeta}$, containing the primal and dual consensus variable predictions, is computed from the input vectors $\boldsymbol{x}_m$, which contain the power demands for each region m, as follows:

$$\begin{aligned}
&\boldsymbol{g}_{m,1} = \phi(\boldsymbol{W}_{m,1}\boldsymbol{x}_m + \boldsymbol{b}_{m,1}), \quad \forall m \in \mathcal{M}, \\
&\boldsymbol{g}_{m,i} = \phi(\boldsymbol{W}_{m,i}\boldsymbol{g}_{m,i-1} + \boldsymbol{b}_{m,i}), \forall m \in \mathcal{M},\ \forall i \in \{2,3,...,n\}, \\
&\boldsymbol{y} = \begin{bmatrix}\boldsymbol{g}_{1,n}^T & \boldsymbol{g}_{2,n}^T & \cdots & \boldsymbol{g}_{|\mathcal{M}|,n}^T\end{bmatrix}^T, \\
&\boldsymbol{h}_{n+1} = \phi(\boldsymbol{W}_{n+1}\boldsymbol{y} + \boldsymbol{b}_{n+1}), \\
&\boldsymbol{h}_j = \phi(\boldsymbol{W}_j\boldsymbol{h}_{j-1} + \boldsymbol{b}_j), \qquad \forall j \in \{n+2,...,N-1\}, \\
&\boldsymbol{\zeta} = \phi(\boldsymbol{W}_N\boldsymbol{h}_{N-1} + \boldsymbol{b}_N).
\end{aligned} \tag{5}$$

The function ϕ can be any activation function. However, some commonly used activation functions, such as the sigmoid, are very expensive to evaluate via MPC [19]. We therefore use the rectified linear unit (ReLU) function $\phi(x) = \max(0,x)$, which is fast to compute via MPC. The layout of the NN layers is shown visually in Figure 2.

To create the PPNN training dataset, we generate synthetic data by randomly perturbing the loads across the network from probability distributions around forecasted values. For each set of perturbed loads, we solve the distributed OPF using ADMM as described in Section II-B. Once the distributed algorithm converges, the final primal boundary variable values $\boldsymbol{z}$ and the dual variables for coupling constraints $\boldsymbol{y}$ are saved. The loss function minimized during NN training is the mean-squared-error (MSE) between predicted and actual consensus variable values: $MSE = \sum_i ||\boldsymbol{\zeta}_i - \boldsymbol{\zeta}_i^*||_2^2$, where, for input sample i, $\boldsymbol{\zeta}_i$ contains the PPNN's predicted output and $\boldsymbol{\zeta}_i^*$ contains the true optimal consensus variable values.

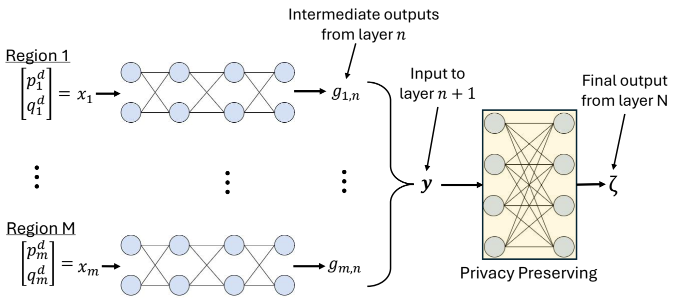

Intermediate outputs from layer n
Region 1
$\begin{bmatrix}p_1^d \\ q_1^d\end{bmatrix} = x_1$
$g_{1,n}$
Input to layer $n+1$
Final output from layer N
$\boldsymbol{y}$
ζ
Region M
$\begin{bmatrix}p_m^d \\ q_m^d\end{bmatrix} = x_m$
$g_{m,n}$
Privacy Preserving

Fig. 2: Setup for the distributed warm-start. Regions input local demands into local NNs and pass their outputs to central coordinators who compute the final output from fully connected layers using cryptographic privacy-preserving techniques.

B. Communication Setup and Warm Start Process

We now discuss the communication and computing setup for evaluating the PPNN output to obtain the warm start. Recall that the network is decomposed into some number of regions, each of which is operated by a local controller. In addition to these controllers, we also designate two coordinating agents, which we refer to as "Alice" and "Bob".[1] Coordinating agents Alice and Bob use MPC techniques to jointly compute the warm start, while the input data remains secret such that neither the coordinating agents nor any other agent obtain the private input data. Again, this input data consists of active and reactive power demands throughout the network.

The coordinating agents Alice and Bob could be any two local controllers, or they could be separate servers located elsewhere. In either case, the agents Alice and Bob must have communication links to the controllers and to each other, and must have access to the fully connected layers of the PPNN. The process to compute the warm start securely is as follows:

1) Local controllers gather power demand data from their region, which forms the input to their local NN layers.
2) Controllers compute the output of their local neural network layers, referred to as the "intermediate output."
3) Controllers send this intermediate output to the coordinating agents Alice and Bob securely, without revealing the true values to either Alice or Bob. To share data for MPC to Alice and Bob, without revealing that data to either agent, controllers use secret sharing [22]. The secret sharing scheme securely distributes the input information between Alice and Bob such that Alice and

[1]This is standard naming convention in the cryptography community.

Bob can jointly perform computations on the input, but neither agent can recover the original input information.[2]

4) Alice and Bob perform MPC, communicating with each other to securely compute the final output ζ, again without either agent learning the intermediate output values which form the input to the PPNN fully connected layers. The vector ζ consists of the primal and dual consensus variables for each region.
5) Alice and Bob send the elements of ζ corresponding to each region's variables to that region's controller, which the controller uses to initialize their primal and dual consensus variables. After initialization, controllers run ADMM to find the OPF solution.

In Figure 3, we show how the controllers use secret sharing to send intermediate outputs to coordinating agents Alice and Bob. Then, Alice and Bob use secure two-party computation techniques to jointly compute the PPNN output.

As some researchers have noted, it may be possible for eavesdroppers to infer some sensitive data, such as power demands, from the data communicated between controllers during ADMM iterations [23]–[26]. While we guarantee protecting the power demand data during the process of computing the warm start with MPC, we do not address the question of data inference during the ADMM run in this paper. Future work will explore modifications of the ADMM algorithm using methods such as those presented in [23], [25] to further guarantee privacy during the ADMM solution process.

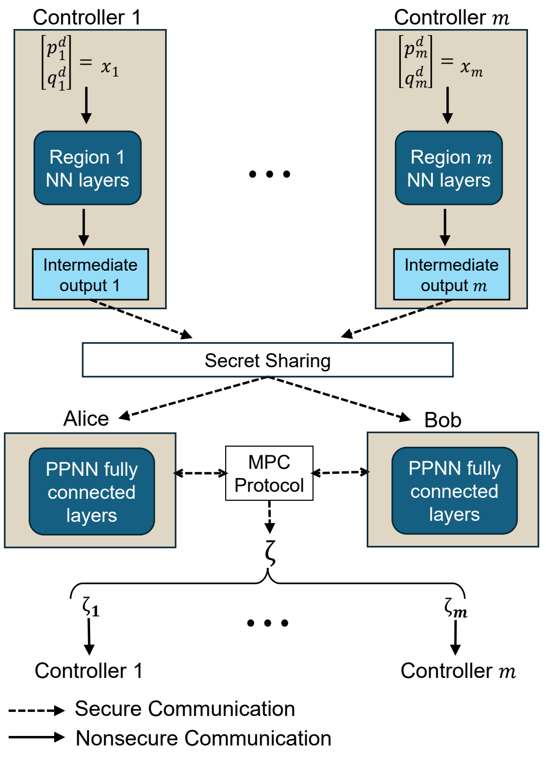


Fig. 3: Showing how the controllers and coordinating agents communicate to securely compute the warm start without revealing sensitive local power demand data.

C. Communication Time and MPC

MPC involves multiple rounds of communication between Alice and Bob involving "oblivious transfers" of data [16], [21]. Therefore, the computation time required to evaluate the PPNN depends heavily on the communication speed. If possible, the coordinators Alice and Bob should communicate via a local area network (LAN), as long as their computations are owned by independent organizations such that Alice and Bob do not collude in trying to recover private data. However, a LAN requires a nearby physical connection which could be difficult to achieve with independent servers owned by different organizations. Another option is communicating via a wide area network (WAN), which avoids a nearby physical connection but is slower than a LAN. Section V compares the performance of the program for both LAN and WAN networks.

D. Trust Assumptions and Privacy Guarantees

We now add a note about our trust assumptions and privacy guarantees, both during the offline training phase and during real-time operation. We assume that the data used for training is not cryptographically protected. Each region's controller stores and optimizes its own neural network weights, while some coordinating agent holds the privacy-preserving neural network weights during the training process. During training, controllers pass their intermediate outputs to the coordinating agent for feed-forward passes through the privacy-preserving neural network layers. Then, the coordinating agent passes back information about the loss function sensitivities to the controllers' intermediate outputs during back-propagation. Thus, the neural network model and power demand input data are not explicitly exchanged between controllers during training. However, the neural network model and power demand data are not cryptographically protected, so it is possible some information could be inferred during training. To ensure that no real historical power demand data is revealed, we generate synthetic power demand data for the training process. Federated learning techniques could also be used to preserve data privacy if real historical data were used for training [27].

We also make a trust assumption about the coordinators, Alice and Bob, who jointly compute the PPNN output. We assume that they perform the MPC processes correctly and do not collude with each other. This is a common assumption in the MPC literature known as *semi-honest parties* [28].

During real-time operation, the controllers do not trust either Alice or Bob with their actual power demand data. Since all data is passed to Alice and Bob through secret sharing and Alice and Bob compute the PPNN output via secure two-party computation, the input power demand data is not revealed to either agent or to any other controller.

[2]It may seem that the regional NN layer already produces obfuscated output, but we have no guarantees that the input could not be inferred from this output. Hence, secret sharing is necessary to guarantee data privacy.

IV. Combining Warm Start with Bound Tightening

We next propose combining the warm start described in this paper with the bound tightening algorithm from [6] to significantly reduce the number of iterations required to converge. Reducing the number of iterations in which data is communicated between regions may have important implications for privacy and cybersecurity. Distributed OPF has the potential to keep power demand data, cost data, and other sensitive information private, since controllers need only share information about boundary variable values with neighboring regions. However, several researchers have explored the possibility that an eavesdropper could infer sensitive data from the information exchanged between controllers [23]–[26]. For the task of inferring sensitive data, accuracy increases and computational effort decreases as the eavesdropper observes more iterations of the distributed algorithm [25]. In addition, each iteration of communication between controllers provides an opportunity for false data injection attacks [26], [29].

In [6], we investigated the impact of convergence tolerance on distributed OPF solution quality. If the convergence tolerance is ϵ, the distributed OPF algorithm converges when the primal residual norm falls below this tolerance, or $||\boldsymbol{r}^k|| \leq \epsilon$, where $\boldsymbol{r}^k$ denotes the vector of primal residuals at iteration k. Recall that the primal residuals are the mismatch between neighboring regions' copies of primal consensus variable values. If the convergence tolerance is large enough, and there are non-negligible mismatches between boundary variables shared by neighboring regions, then the distributed OPF dispatch may violate engineering limits when applied to the power grid. Reference [6] presented a bound tightening algorithm which ensures, given a certain convergence tolerance, that the network operating point corresponding to the distributed OPF solution will not violate constraints. Increasing the convergence tolerance reduces the distributed OPF solve time.

In general, the bound tightening algorithm allows for an increase in the convergence tolerance to reduce the number of iterations required to converge, trading off potential suboptimality of the solution for feasibility guarantees and computational speed. In [6], we found that the increase in OPF cost after bound tightening was relatively small.

To combine bound tightening with warm start, we run the bound tightening algorithm for the largest convergence tolerance ϵ for which the bounds can be tightened feasibly. We save the tightened bounds on voltage magnitudes, reactive power generation, and line power flows. Next, we train a PPNN on the test case with bounds tightened for convergence tolerance ϵ. This process is illustrated in Figure 4.

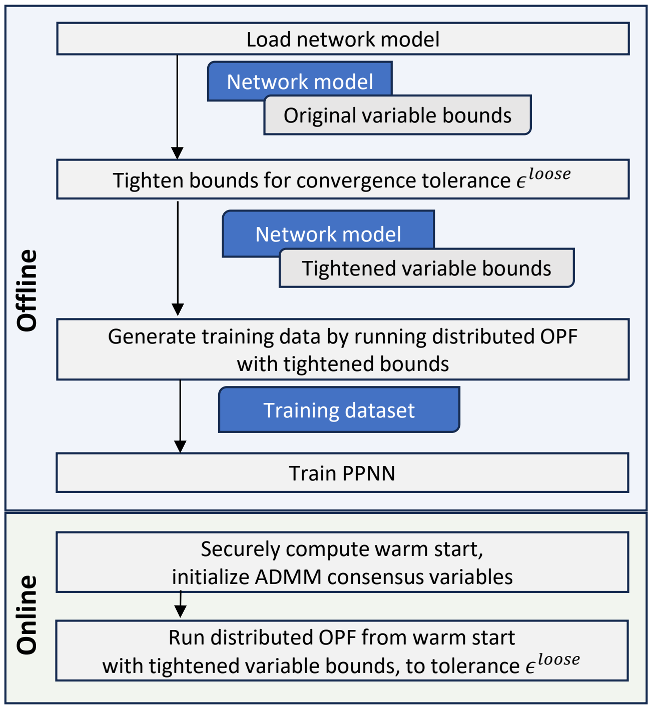

Offline
Load network model
Network model
Original variable bounds
Tighten bounds for convergence tolerance ϵ^{loose}
Network model
Tightened variable bounds
Generate training data by running distributed OPF with tightened bounds
Training dataset
Train PPNN
Online
Securely compute warm start, initialize ADMM consensus variables
Run distributed OPF from warm start with tightened variable bounds, to tolerance ϵ^{loose}

Fig. 4: Flowchart showing steps for combining bound tightening and privacy-preserving warm start for distributed OPF.

V. Numerical Results

This section presents numerical results demonstrating how the secure warm start can significantly decrease computation time. We include the time required to compute the PPNN predictions, evaluating the final layer using MPC. We provide an overview of the test cases and implementation details, present results that show how the warm start reduces computation time given two different MPC libraries, and then illustrate the impact of combining bound tightening with the warm start.

A. Test Cases and Implementation Details

Our test cases are the case118 and case200 synthetic transmission test cases from the PGLib-OPF archive [30], and case69 and case141 from MATPOWER [31], which represent balanced distribution networks. Note that we add controllable distributed energy resources to case69 and case141, which we model as providing power at no cost.[3] We divide case69 and case118 into three regions, case141 into four regions, and case200 into five regions for distributed OPF.

We train the PPNNs for each test case as described in Section III-A. Let the length of the input vector for each region consisting of active and reactive power demands across the region, $\boldsymbol{x}_m$, be denoted N_m^d. Each region's neural network contains seven hidden layers, each with $2N_m^d$ neurons. Then, the output of each region's NN is length N_m^d, so that the input to the fully connected PPNN is MN_m, where again M is the number of regions in the network. The PPNN contains just one layer with $2N^s$ neurons, where N^s is the number of consensus variables across the network.

We emulate realistic communication between controllers and the agents which securely compute the output of the final privacy-preserving NN layer. We present results for both local and wide area network communication with representative latency and data throughput. We use the Linux NetEm network emulator [32] to simulate the performance of local

[3]The modified MATPOWER case69 and case141 test cases are available online at https://github.com/rjuly7/test_case_modifications.

TABLE I: Network Emulator Parameters

	Delay	**Rate**
Local Area Network (LAN)	0.4 ms	3 Gbit/s
Wide Area Network (WAN)	5 ms	1 Gbit/s

and wide area networks. The parameters we use are listed in Table I. Here, *delay* indicates a constant delay before sending each packet, measured in milliseconds, and *rate* denotes the specified data transfer rate in Gigabits per second.

To obtain the results presented in the following sections, we generate synthetic data by randomly perturbing power demands at loads. For each load, we randomly generate a value $r_i \in [-0.5, 0.5]$ for each load i. Then, we assign the active and reactive power demands at load i as $p_i^d = (p_i^d)^{nom}(1+r_i)$ and $q_i^d = (q_i^d)^{nom}(1+r_i)$, where $(p_i^d)^{nom}$, $(q_i^d)^{nom}$ are the nominal active and reactive power demands, respectively, at load i. That is, we select random perturbations from a uniform distribution such that power demands may take on any value between 50% and 150% of their nominal values. We maintain a constant power factor by selecting the same perturbation for both active and reactive power demands at each load. In contrast to the work in [14], which generated synthetic data such that loads across the network were highly correlated, we select random perturbations independently for each load. We solve the distributed OPF problems with the ADMM algorithm as described in Section II-B. The penalty parameter is $\alpha = 1000$, and we terminate each run when the ℓ_∞-norm of the primal residuals is below the convergence tolerance ϵ. We ran all experiments on a laptop with an eight-core 2.3 GHz processor and 16.0 GB RAM.

B. MPC Libraries and Computation Time after Warm Start

The previous state-of-the-art open-source library for secure floating-point computation is SecFloat [21]. However, using SecFloat to evaluate the PPNN-based warm start is too slow to achieve significant speedups relative to a flat start. We therefore instead use our recently developed library NumSec for secure floating-point computations [33]. In this section, we present results for computation time, including the PPNN evaluation time, using both the SecFloat library [21] and the new NumSec library [33].

To evaluate how much the warm start reduces computation time, we run distributed OPF with randomly perturbed loads 400 times with a flat start, 400 times with the warm start computed under LAN emulation, and 400 times with the warm start computed under WAN emulation. For this section, we use convergence tolerance $\epsilon = 10^{-4}$. After each run, we record the number of iterations required to converge and the total computation time, which includes the time required to evaluate the PPNN securely. We present results for both the prior state-of-the-art open-source library for secure operations on floating-point numbers, SecFloat, and our newly developed library, NumSec [33].

Table II summarizes these results, showing the mean number of iterations and the mean computation times when using the SecFloat library and our new NumSec library with both LAN and WAN network emulation for the warm start. Observe that the warm start significantly reduces the number of iterations, by a factor between 4.3 and 27.1. When computing the warm start with the SecFloat library, although the number of iterations is reduced, computation time may not improve. Due to the time required to evaluate the PPNN, the total computation time may be slower for the warm start when using the SecFloat library, if coordinating agents communicate over WAN. The table also shows the percent reduction in computation time that results from the warm start.

When using the new NumSec library for secure computations, using the warm start is significantly faster than the flat start for both LAN and WAN. To achieve the greatest reduction in computation time, we should compute the warm start with secure agents communicating via LAN. In practice, even if controllers are physically scattered throughout the network and communicate via WAN, we could choose to locate the secure coordinating agents Alice and Bob close together so they can communicate over LAN to enable fast evaluation of the PPNN.

C. Combining Bound Tightening and Warm Start

Next, we run the bound tightening algorithm from [6] on each of the three test cases. We select the largest convergence tolerance for which the tightened bounds are feasible for OPF. We denote this loosened convergence tolerance as ϵ^{loose}.[4] To evaluate the performance of the combined warm start and bound tightening, we run distributed OPF with randomly perturbed loads again on the bound-tightened test cases, 400 times with flat start, 400 times with warm start computed over LAN, and 400 times with warm start computed over WAN. Each run is terminated when the ℓ_∞-norm of the shared boundary variable mismatches falls below the convergence tolerance ϵ^{loose}. We present the mean number of iterations to convergence and the mean computation time in Table III for each test case across the 400 runs. We also show the percent reduction in computation time from using the warm start and bound tightening, compared to the flat start without bound tightening.

Figure 5 shows boxplots illustrating the distribution of computation times for the case141 and case200 test cases. We present the distribution of computation times for distributed OPF with vanilla ADMM, i.e., a flat start and no bound tightening. We also show the distribution of computation times for distributed OPF after applying bound tightening and warm start, where the warm start is computed with either LAN or WAN emulation. Note that for vanilla ADMM, some sets of loading conditions take a very long time to solve (more than twice the median solution time). However, when we initialize with the warm start, the solve times are much more consistent and significantly reduced compared to ADMM with flat start and no bound tightening.

For both initializations (flat start and warm start), using a looser convergence tolerance after bound tightening decreases

[4]For case69 and case141, $\epsilon^{loose} = 1.3 \times 10^{-3}$. For case118, $\epsilon^{loose} = 10^{-2}$. For case200, $\epsilon^{loose} = 1.9 \times 10^{-3}$. The tolerance ϵ^{loose} is in per unit for voltage magnitudes and power flows, and is in radians for voltage angles.

TABLE II: Comparison of Iterations and Computation Times for Flat Start vs. Warm Start

Network	**Number of Iterations**		**Computation Time (s)**				
			Flat Start	Warm Start, LAN (% Reduction)		Warm Start, WAN (% Reduction)	
	Flat Start	Warm Start		SecFloat	NumSec	SecFloat	NumSec
case69	300.3	7.0	30.7	4.3	1.7 (94.5%)	16.9	4.3 (86.0%)
case118	88.2	15.5	15.4	9.5	4.4 (71.4%)	26.2	7.4 (51.9%)
case141	103.4	3.2	13.0	3.6	1.1 (91.4%)	16.7	3.8 (70.8%)
case200	591.7	16.7	91.8	42.3	13.4 (85.4%)	97.4	21.1 (77.0%)

TABLE III: Iterations and Computation Time After Bound Tightening, with NumSec Library for Warm Start

Network	**Number of Iterations**		**Computation Time (s)**		
	Flat Start	Warm Start	Flat Start	Warm Start, LAN (% Reduction)	Warm Start, WAN (% Reduction)
case69	101.3	1.0	10.10	0.86 (97.1%)	3.43 (88.8%)
case118	24.4	1.36	4.03	1.87 (87.9%)	4.85 (68.5%)
case141	55.2	1.0	6.31	0.77 (94.1%)	3.45 (73.5%)
case200	244.0	1.37	36.56	10.95 (88.1%)	18.61 (79.7%)

the total number of iterations and computation time. For case69 and case141, the PPNN warm start predictions are accurate enough that the distributed OPF converges after only one iteration of local subproblem solves. Note that no communication between controllers is necessary during this iteration; each controller simply solves its local subproblem, using the initial consensus variable values from the warm start. For case118, 64.6% of the distributed OPF runs converge after one iteration, with 35.4% of the distributed OPF runs requiring two iterations (i.e., one round of ADMM communications). For case200, 67.0% of the distributed OPF runs converge after one iteration, with 31.3% of the distributed OPF runs requiring two iterations (i.e., one round of ADMM communications), and 1.6% of the distributed OPF runs requiring three iterations (i.e., two rounds of ADMM communications). When the distributed OPF converges after one iteration of local subproblem solves, no data needs to be communicated between neighboring regions. As discussed in Section IV, reducing the amount of data communicated between controllers improves privacy and cybersecurity. With less communicated data to observe, an eavesdropper's ability to infer sensitive data from the consensus variable data shared after each iteration is limited [23]–[26]. In addition, malicious attackers have less opportunity to interfere with or manipulate communication.

We note that bound tightening and training neural networks for warm start requires offline computation time. We chose to solve OPF centrally to create the training data, since solving distributed OPF with ADMM before applying any warm start would take longer. For each test case, bound tightening took less than one hour, generating training data took less than one hour, and training the neural networks took between four and twelve hours. Since we account for a range of loading conditions when performing bound tightening and learning the warm start, the offline computation need only be performed once, after which these methods can be used to accelerate distributed OPF many times during daily operation.

VI. Conclusion

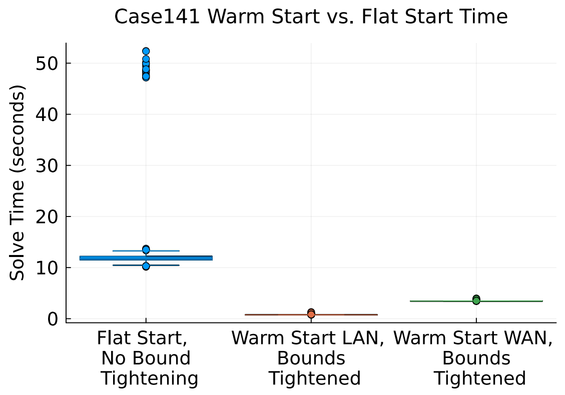
This paper has explored a secure warm start for distributed OPF solved with ADMM applicable to settings where loads across the network are not necessarily correlated. We use MPC to evaluate the output of privacy-preserving neural networks (PPNNs), which are trained to predict final primal and dual consensus variable values. Since evaluating PPNNs requires additional computational overhead to keep their operations secure, slowing computation times, we develop a scheme in which local regions pass input data to independent NNs, which are evaluated locally. The regions then pass their NN outputs to two coordinating agents, which use a single-layer PPNN to compute the final warm-start values without either agent learning the input data. We also use a recently developed library to perform the secure multi-party computations efficiently. The warm start can significantly reduce computation time without revealing the local demand data to the coordinating agents.

Next, we propose combining the warm start with a bound tightening technique that enables using a looser convergence tolerance. When the consensus variables are initialized with PPNN warm start predictions and we use a larger convergence tolerance for a bound-tightened test case, the number of iterations to convergence is reduced by an order of magnitude. In many cases, the distributed OPF converges in only one iteration, so that controllers need not share any consensus variable data with their neighbors. Eliminating the repeated communication between controllers, which is typically needed to solve distributed OPF, reduces the attack surface for false data injection attacks and prevents eavesdroppers from inferring sensitive local data.

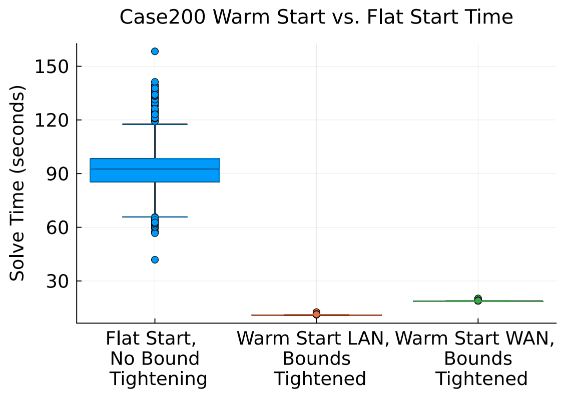
References

[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] S. Mhanna, A. C. Chapman, and G. Verbič, "Component-based dual decomposition methods for the OPF problem," *Sustainable Energy, Grids and Networks*, vol. 16, pp. 91–110, 2018.



(a) Distributions of computation times for distributed OPF on the case141 distribution network.



(b) Distributions of computation times for distributed OPF on the case200 transmission network.

Fig. 5: Boxplots show the distributions of computation times for distributed OPF on two illustrative test cases, in three cases: 1) flat start with no bound tightening, 2) warm start computed over LAN with bound tightening, and 3) warm start computed over WAN with bound tightening. Combining the warm start with bound tightening produce significant speedup.

[3] S. Mhanna, G. Verbič, and A. C. Chapman, "Adaptive ADMM for distributed AC optimal power flow," *IEEE Trans. Power Syst.*, vol. 34, no. 3, pp. 2025–2035, 2019.

[4] C. Mavromatis, M. Foti, and M. Vavalis, "Auto-tuned weighted-penalty parameter ADMM for distributed optimal power flow," *IEEE Trans. Power Syst.*, vol. 36, no. 2, pp. 970–978, 2021.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[6] R. Harris, M. Alkhraijah, and D. K. Molzahn, "Optimally managing the impacts of convergence tolerance for distributed optimal power flow," *IEEE Trans. Power Syst.*, vol. 40, no. 5, pp. 3901–3913, 2025.

[7] S. Zeng, A. Kody, Y. Kim, K. Kim, and D. K. Molzahn, "A reinforcement learning approach to parameter selection for distributed optimization in power systems," *Electric Power Systems Research*, vol. 212, p. 108546, 2022. Presented at the *22nd Power Systems Computation Conf. (PSCC)*.

[8] K. Sun and X. A. Sun, "A two-level ADMM algorithm for AC OPF with global convergence guarantees," *IEEE Trans. Power Syst.*, vol. 36, no. 6, pp. 5271–5281, 2021.

[9] A. S. Zamzam and K. Baker, "Learning optimal solutions for extremely fast AC optimal power flow," in *IEEE Int. Conf. Commun., Control, Comput. Tech. Smart Grids (SmartGridComm)*, 2020.

[10] R. Nellikkath and S. Chatzivasileiadis, "Physics-informed neural networks for AC optimal power flow," *Electric Power Systems Research*, vol. 212, p. 108412, 2022. Presented at the *22nd Power Systems Computation Conf. (PSCC)*.

[11] S. Park, W. Chen, T. W. Mak, and P. Van Hentenryck, "Compact optimization learning for AC optimal power flow," *IEEE Trans. Power Syst.*, vol. 39, no. 2, pp. 4350–4359, 2024.

[12] A. Venzke, G. Qu, S. Low, and S. Chatzivasileiadis, "Learning optimal power flow: Worst-case guarantees for neural networks," in *IEEE Int. Conf. Commun., Control, Comput. Tech. Smart Grids (SmartGridComm)*, 2020.

[13] S. Chevalier, I. Murzakhanov, and S. Chatzivasileiadis, "GPU-accelerated verification of machine learning models for power systems," in *57th Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2024.

[14] T. W. K. Mak, M. Chatzos, M. Tanneau, and P. V. Hentenryck, "Learning regionally decentralized AC optimal power flows with ADMM," *IEEE Trans. Smart Grid*, vol. 14, no. 6, pp. 4863–4876, 2023.

[15] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.

[16] L. L. Ng and S. M. Chow, "SoK: Cryptographic neural-network computation," in *IEEE Symp. Security Privacy (SP)*, pp. 497–514, May 2023.

[17] B. H. Kim and R. Baldick, "Coarse-grained distributed optimal power flow," *IEEE Trans. Power Syst.*, vol. 12, no. 2, pp. 932–939, 2002.

[18] 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, 2014.

[19] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *IEEE Symp. Security Privacy (SP)*, pp. 19–38, 2017.

[20] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Infor. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[21] D. Rathee, A. Bhattacharya, R. Sharma, D. Gupta, N. Chandran, and A. Rastogi, "SecFloat: Accurate floating-point meets secure 2-party computation," in *IEEE Symp. Security Privacy (SP)*, pp. 576–595, 2022.

[22] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure multiparty computation and secret sharing*. Cambridge University Press, 2015.

[23] V. Dvorkin, P. Van Hentenryck, J. Kazempour, and P. Pinson, "Differentially private distributed optimal power flow," in *59th IEEE Conference on Decision and Control (CDC)*, pp. 2092–2097, 2020.

[24] X. Huo and M. Liu, "On privacy preservation of distributed energy resource optimization in power distribution networks," *IEEE Trans. Control Netw. Syst.*, vol. 12, no. 1, pp. 228–240, 2025.

[25] M. Ryu and K. Kim, "A privacy-preserving distributed control of optimal power flow," *IEEE Trans. Power Syst.*, vol. 37, no. 3, pp. 2042–2051, 2022.

[26] T. Wu, C. Zhao, and Y.-J. A. Zhang, "Privacy-preserving distributed optimal power flow with partially homomorphic encryption," *IEEE Trans. Smart Grid*, vol. 12, no. 5, pp. 4506–4521, 2021.

[27] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Syst.*, vol. 216, p. 106775, 2021.

[28] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *23rd ACM Conf. Computer Commun. Security (CCS)*, p. 805–817, 2016.

[29] 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, 2022.

[30] IEEE PES PGLib-OPF Task Force, "The power grid library for benchmarking AC optimal power flow algorithms," 2019. arXiv:1908.02788.

[31] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, 2011.

[32] S. Hemminger, "Network Emulation with NetEm," 2005. https://www.rationali.st/blog/files/20151126-jittertrap/netem-shemminger.pdf.

[33] D. Turizo, L. K. L. Ng, X. Yongyu, V. Kolesnikov, S. Zonouz, and D. K. Molzahn, "NumSeC: Numerical secure computations via parallelism and look-up tables," 2025. Working Paper.