Graph Neural Networks Based Detection of Stealth False Data Injection Attacks in Smart Grids

Osman Boyaci, Amarachi Umunnakwe, *Student Member, IEEE*, Abhijeet Sahu, *Student Member, IEEE*, Mohammad Rasoul Narimani, *Member, IEEE*, Muhammad Ismail, *Senior Member, IEEE*, Katherine R. Davis, *Senior Member, IEEE*, and Erchin Serpedin, *Fellow, IEEE*

***Abstract*—False data injection attacks (FDIAs) represent a major class of attacks that aim to break the integrity of measurements by injecting false data into the smart metering devices in power grids. To the best of authors' knowledge, no study has attempted to design a detector that automatically models the underlying graph topology and spatially correlated measurement data of the smart grids to better detect cyber attacks. The contributions of this article to detect and mitigate FDIAs are twofold. First, we present a generic, localized, and stealth (unobservable) attack generation methodology and publicly accessible datasets for researchers to develop and test their algorithms. Second, we propose a graph neural network (GNN) based, scalable and real-time detector of FDIAs that efficiently combines model-driven and data-driven approaches by incorporating the inherent physical connections of modern ac power grids and exploiting the spatial correlations of the measurement. It is experimentally verified by comparing the proposed GNN-based detector with the currently available FDIA detectors in the literature that our algorithm outperforms the best available solutions by 3.14%, 4.25%, and 4.41% in $F1$ score for standard IEEE testbeds with 14, 118, and 300 buses, respectively.**

***Index Terms*—False data injection attacks (FDIAs), graph neural networks (GNNs), machine learning (ML), power system security, smart grid.**

Nomenclature

$P_i + jQ_i$	Complex power injection at bus i.
$P_{ij} + jQ_{ij}$	Complex power flow between bus i and j.
V_i, θ_i	Voltage magnitude and phase angle of bus i.
θ_{ij}	$\theta_i - \theta_j$.
$G_{ij} + jB_{ij}$	ijth elements of bus admittance matrix.
$g_{ij} + jb_{ij}$	Series branch admittance between bus i–j.
$g_{si} + jb_{si}$	Shunt branch admittance at bus i.
Ω_i	Set of buses connected to bus i.
$\boldsymbol{z_o}, \boldsymbol{z_a} \in \mathbb{R}^m$	Original, attacked measurement vector.
$\hat{\boldsymbol{x}}, \check{\boldsymbol{x}} \in \mathbb{R}^n$	Original, attacked state vector.
$h(\boldsymbol{x})$	Nonlinear measurement function at $\boldsymbol{x}$.
$\boldsymbol{H} \in \mathbb{R}^{m\times n}$	Jacobian matrix.
$\boldsymbol{G} \in \mathbb{R}^{n\times n}$	Gain matrix.
$\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{m\times m}$	Error covariance, residual sensitivity matrix.
$\mathcal{T}$	Attacker's target area to perform FDIA.

I. Introduction

As a highly complex cyber-physical system, a smart grid consists of a physical power system infrastructure and a cyber-communication network. Physical measurement data are first acquired by the remote terminal units (RTUs) or phasor measurement units (PMUs) and are delivered to the supervisory control and data acquisition systems. Then, the communication network transfers the measurement data to the application level where are processed and evaluated by the power applications [1]. Thus, reliability of power system depends on the security of the cyber-physical pipeline [2].

Power system state estimation (PSSE) is a highly critical component of this pipeline since its outcome is directly fed into numerous energy management system (EMS) blocks such as load and price forecasting, contingency and reliability analysis, and economic dispatch processes [3], [4]. Thus, integrity and trustworthiness of the measurement data play a critical role in ensuring proper operation of smart grids [5]. By breaking this integrity, cyber-physical attacks target smart metering devices to harm the underlying physical systems.

False data injection attacks (FDIAs) represent a significant class of cyber threats that modify PSSE by maliciously altering the measurement data. In FDIAs, an attacker changes sensor data in such a way that a valid and misleading operating point converge in PSSE and the attack becomes unobservable [6]. Being unaware of the malicious data, the grid operator takes actions according to the false operating point of grid and consequently disrupts power system operation.

Traditional PSSE is performed using the weighted least squares estimation (WLSE) technique, and the presence of bad data is detected by employing the largest normalized residual test (LNRT) [4]. Stealth (unobservable) FDIA can easily bypass the bad data detection (BDD) systems. Therefore, FDIAs are one of the most critical attacks for today's smart power systems. FDIAs in power grids were first introduced a decade ago by [7], which showed that an attacker with enough knowledge of the

Manuscript received November 25, 2020; revised July 19, 2021; accepted August 19, 2021. Date of publication October 20, 2021; date of current version June 13, 2022. This work was supported by National Science Foundation under Grant 1808064. *(Corresponding author: Osman Boyaci.)*

Osman Boyaci, Amarachi Umunnakwe, Abhijeet Sahu, Katherine R. Davis, and Erchin Serpedin are with the Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: osman.boyaci@tamu.edu; amarachi@tamu.edu; abhijeet_ntpc@tamu.edu; katedavis@tamu.edu; eserpedin@tamu.edu).

Mohammad Rasoul Narimani is with the College of Engineering, Arkansas State University, Jonesboro, AR 72404 USA (e-mail: mnarimani@astate.edu).

Muhammad Ismail is with the Department of Computer Science, Tennessee Tech University, Cookeville, TN 38505 USA (e-mail: mismail@tntech.edu).

Digital Object Identifier 10.1109/JSYST.2021.3109082

grid topology can design an unobservable attack that satisfies the power flow equations and bypasses the BDD module. Influential reference [7] prompted an increased interest in detection of FDIAs [8]–[21].

Most of the works that deal with detection of FDIAs assume a linearized dc model [7]–[10], [12], [13], [15], [17], [20]. In the dc state estimation model, bus voltage magnitudes are assumed to be known as 1 p.u. and branch resistances and shunt elements are neglected. Hence, estimation of bus voltage angles is reduced to linear matrix operations, and in general it helps to analyze the grid at some extent. Although the linearized dc model is fast and simple, ignoring voltage magnitudes and reactive power components does not reflect the actual physical operation of the grid [4]. Therefore, the dc models cannot validate that the FDIAs being tested are stealthy because PSSE and BDD tools employing ac power flow modeling can easily detect these attacks without using extra detectors. In addition, only a few works exploit grid topology information into their detection model [11], [22], [23] together with graph signal processing (GSP) techniques to detect FDIAs. Although innovative and powerful, these methods manually design spectral filters, an operation which is not scalable since it requires manual and custom filter design steps. Scalability is an essential feature that has to be considered when designing detectors. Except a few highly scalable designs [24], [25], the majority of the proposed detectors for FDIAs are designed for small scale systems such as IEEE 14 [12], [13], [15], [16] or IEEE 30 [18], [20]. Therefore, extensibility issues may arise when deploying small-scale detectors at large-scale networks. Employing spatial-temporal correlations of the state variables and trust-based voting mechanisms, Chen *et al.* [8] define a consistency region and detection threshold to differentiate honest from malicious samples. Nevertheless, dc approximation and resolution of the time series data highly limit the applicability of the proposed design to realistic large-scale power grids.

Survey [26] classifies the FDIA detection algorithms into the following two categories: model-based methods [12]–[16] and data-driven methods [17]–[21]. In general, model-based algorithms require first to build a system model and estimate its parameters to detect FDIAs. Since there is no independent system to be trained, model-based methods do not need historical datasets; nevertheless, threshold finding, detection delays, and scalability aspects restrict applicability of model-based methods [26]. On the contrary, data-driven models do not interfere with the system and its parameters, yet they necessitate historical data and a training process in order to reduce the detection time and increase scalability.

Due to the superiority of machine learning (ML) methods along with the increasing volume of collected historical data samples, ML-based detectors have been proposed to identify FDIAs in smart grids. For example, decision tree (DTC) [21], support vector machine (SVM) [17], [18] multilayer perceptron (MLP) [18], recurrent neural network (RNN) [20], convolutional neural network (CNN) [19] models were proposed to detect FDIAs. Despite their effectiveness, ML-based methods may overfit and fail to detect FDIAs especially in situations when the ML architecture does not capture the underlying physical system generating the data [26]. To illustrate, CNNs are well-suited to image and video processing since locality of pixels is well modeled by the sliding kernels. Conversely, an RNN architecture might be more applicable to recurrent relations such as sequence to sequence language modeling and machine translation applications [27].

Undirected graphs can be used to capture the smart grid topology; buses and branches of the grid can be represented by nodes and edges of the undirected graph, respectively. The graph neural network (GNN) architecture, in particular, immensely benefits from this architectural matching promise [28], [29]. Besides, the prediction of the filter weights in GNNs instead of being performed manually (e.g., [11], [22], [23]) can be executed automatically via GSP techniques, which makes GNNs more attractive to smart grid applications. For example, in [30], GNNs are utilized for optimal power flow applications in power grids. Due to GNN's highly efficient modeling capability in non-Euclidean data structure, they are adopted in numerous areas such as social networks, physical systems, traffic networks, and molecule interaction networks [29]. Despite their potential, to the best of authors' knowledge, no study has explored GNNs to detect FDIAs.

In this article, we propose a GNN-based stealth FDIA detection model for smart power grids. To fully model the underlying complex ac power system and dynamism of the measurements data, we decided to use a hybrid model; while system topology is integrated into our model by the help of GNN graph adjacency matrix, historical measurement data are modeled by the GNN spatial layers. These features enable to take advantage of the benefits of both model-driven and data-driven approaches and hence better detect and mitigate FDIAs.

The contributions of this article are summarized as follows.

1) We properly model the inherent cyber system: due to the topology and distribution of the smart measurement devices, meter readings are correlated in the measurement space of the smart grid; hence, ignoring the location of the meter data and assuming independent and identical distribution of meter readings may not be accurate for a data-driven model. Therefore, we use GNN to match the cyber and physical layers of the grid.
2) We design a stealth FDIA attack methodology to test our detector: the main goal of any FDIA detector is to be able to detect stealth attacks since observable attacks can be easily detected by BDD systems. In other words, unproven random attacks do not require any extra detector other than traditional BDDs so the proposed detectors should be tested under stealth attacks to fully evaluate their performances. Therefore, we develop a stochastic gradient descent (SGD) based stealth FDIA detection algorithm to exploit the possible weak points of the grid and assess the performance in realistic conditions. It is experimentally verified that the designed attacks can easily bypass classical BDD algorithms; however, they are detected by the proposed GNN detectors.
3) We propose a scalable and real-time FDIA detector as an early warning/prediction system prior to the PSSE: since PSSE outcome is directly used by various EMS, the

integrity of the measurements should be preserved. Thus, a detector system indicating the false data injection to the measurements prior to the PSSE is crucial. In addition, custom methods developed for small case systems may not be applicable to larger cases; therefore, detection models should be efficiently extensible to larger networks. Moreover, depending on the system scale and topology, detection delays can be very critical for power grids, therefore, possible attacks should be detected as quickly as possible. Employing the standard test cases such as IEEE 14, 118, and 300 bus systems, it is demonstrated that the proposed method is linearly scalable both in parameter size and detection time.

The remainder of this article is divided into the following five sections. Section II is devoted to preliminaries such as PSSE, FDIAs, and BDD mechanisms in smart grids. While Section III explains the proposed detection method and its mathematical modeling, Section IV describes the experimental results. Finally, Section V concludes this article.

II. Power System Preliminaries

A. Power System State Estimation

PSSE module aims to estimate the system state $\boldsymbol{x}$ (V_i, θ_i at each bus) in the steady state by using the complex power measurements $\boldsymbol{z}$ collected by noisy RTUs or PMUs via

$$\hat{\boldsymbol{x}} = \min_{\boldsymbol{x}} (\boldsymbol{z} - h(\boldsymbol{x}))^T \boldsymbol{R}^{-1} (\boldsymbol{z} - h(\boldsymbol{x})) \tag{1}$$

where $\boldsymbol{R}$ denotes the error covariance matrix of measurements and $\boldsymbol{z}$ consists of active and reactive power injections at buses (P_i, Q_i) and active and reactive power flows on branches (P_{ij}, Q_{ij}). In polar form, these can be expressed as [4]

$$\begin{aligned} P_i &= \sum_{j\in\Omega_i} V_i V_j (G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) = P_{Gi} - P_{Li} \\ Q_i &= \sum_{j\in\Omega_i} V_i V_j (G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) = Q_{Gi} - Q_{Li} \\ P_{ij} &= V_i^2(g_{si} + g_{ij}) - V_i V_j (g_{ij}\cos\theta_{ij} + b_{ij}\sin\theta_{ij}) \\ Q_{ij} &= -V_i^2(b_{si} + b_{ij}) - V_i V_j (g_{ij}\sin\theta_{ij} - b_{ij}\cos\theta_{ij}). \end{aligned} \tag{2}$$

Since (2) are nonlinear and nonconvex, (1) is carried out via iterative WLSE [31].

B. False Data Injection Attacks

The goal of FDIA is to find a new measurement vector $\boldsymbol{z_a}$ in the measurement space of the grid such that PSSE converges to another point in the state space of variables. Formally

$$\boldsymbol{z_o} = h(\hat{\boldsymbol{x}}),\ \boldsymbol{z_a} = \boldsymbol{a} + \boldsymbol{z_o} = h(\check{\boldsymbol{x}}) \tag{3}$$

where $\boldsymbol{a}$ represents the attack vector, $\hat{\boldsymbol{x}}$ and $\check{\boldsymbol{x}}$ denote the estimated (original) state vector and false data injected state vector, and $\boldsymbol{z_o}$ and $\boldsymbol{z_a}$ stand for the original and attacked measurements, respectively.

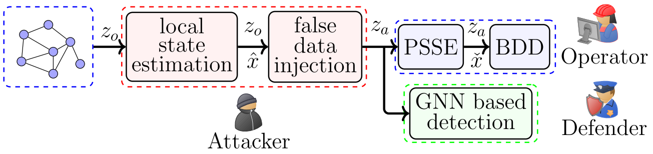

Fig. 1. Architectural overview and signal flow graph of the proposed design. While blue boxes represent smart grids and their operations run by operator, red and green boxes denote functional blocks of attacker and defender, respectively. Note that operator gets attacked measurements $\boldsymbol{z_a}$ instead of original ones $\boldsymbol{z_o}$ due to the FDIA. Defender, on the contrary, tries to detect possible attacks by using $\boldsymbol{z_a}$.

C. Bad Data Detection

Traditional power systems use the LNRT to detect bad samples using the following equations [4]:

$$\begin{aligned} &\boldsymbol{r} = \boldsymbol{z} - h(\hat{\boldsymbol{x}}),\ \boldsymbol{G} = \boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H} \\ &\boldsymbol{S} = \boldsymbol{I} - \boldsymbol{H}(\boldsymbol{G}^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1}),\ \boldsymbol{r}_i^N = \frac{|\boldsymbol{z}_i - h(\hat{\boldsymbol{x}})_i|}{\boldsymbol{R}_{ii}\boldsymbol{S}_{ii}}. \end{aligned} \tag{4}$$

After estimating the current state vector using (1), residues $\boldsymbol{r}$ are calculated as the difference between observed ($\boldsymbol{z}$) and calculated ($h(\hat{\boldsymbol{x}})$) measurements. Then, using the Jacobian matrix $\boldsymbol{H}$ and the diagonal error covariance matrix of the measurements $\boldsymbol{R}$, gain matrix $\boldsymbol{G}$ is computed. Sensitivity of the residues for each measurement represented by the residual sensitivity matrix $\boldsymbol{S}$ are computed right after $\boldsymbol{G}$. Finally, residues are normalized by dividing each one of them with the product of corresponding diagonal elements of $\boldsymbol{R}$ and $\boldsymbol{S}$, and normalized residue vector $\boldsymbol{r}^N$ is obtained. Since $\boldsymbol{r}^N$ is assumed to have a standard normal distribution, a large $\boldsymbol{r}_i^N$ can be classified as bad data, if $\boldsymbol{r}_i^N$ exceeds a predetermined threshold τ_{bdd} specified by the grid operator according to the desired level of sensitivity [4]. If an attacker wants to be considered stealthy, the maximum normalized residual value $\max(\boldsymbol{r}^N)$ should be less than the threshold τ_{bdd}.

III. GNN-Based Detection of FDIA

A. FDIA Scenario

The main architecture and signal flow of the proposed design is illustrated in Fig. 1.

First, active and reactive power injections P_i, Q_i at buses and active and reactive power flows P_{ij}, Q_{ij} on branches are read by RTUs. Next, as a man in the middle, an attacker attempts to inject false data to the original measurements $z_o = [P_i, P_{ij}, Q_i, Q_{ij}]$ before the grid operator receives them. Then, using $\boldsymbol{z_a}$, the operator estimates the state variables and runs the BDD block to indicate a possible attack. In parallel, the defender runs the GNN-based detector when it receives the measurements and, hence, predicts the probability of attack to warn the operator. In order not to raise suspicion from the operator, the attacker needs to design a stealth $\boldsymbol{z_a}$ that can bypass the BDD mechanism incorporated in (4). At the same time, the attack strength should be strong enough to cause intended consequences or damages to the grid. In this regard, she/he initially estimates the state variables of grid in the target area $\mathcal{T}$, where security of the meters

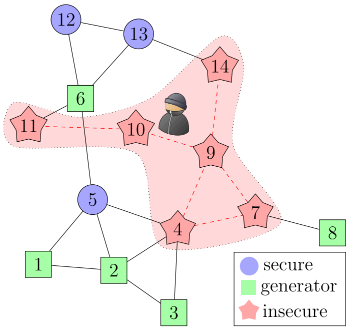

Fig. 2. Visualization of an example IEEE 14 bus system where an attacker enters the system from bus 10 (entry point) and affects the 1° neighbors: bus 9 and 11, and 2° neighbors: bus 4, 7, 14 depicted with red stars. Besides, it is assumed that she/he can change the power flows measurements depicted with red dashed lines in the target area $\mathcal{T}$ represented by the red surface. Note that since bus 6 is a generator node illustrated by a green square she/he skips it. Moreover, she avoids changing the line between bus 6 and 11 in order to not violate the KCL equation at 6.

is compromised. Then, she/he searches a set of measurements $\boldsymbol{z_a}$ in the measurement space that serves the intended aim.

As indicated by [1], [7], FDIAs require that an adversary know the parameters and topology of the targeted portion of the system and is able to tamper the measurement data before the operator uses them in PSSE. Since accessing information and hardware all over the grid is neither easy nor realistic, we use a realistic "local" attack model to test our system. Due to the lack of open source, ac power flow based stealth FDIA generation algorithms to fully test the detection system, we propose a generic, localized ac stealth FDIA generation method using the SGD algorithm. Herein scenario, the attacker focuses on a target area of the grid where the measurements she/he wants to inject the false data are located. To specify this area, it is assumed that she/he found an entry point p in the cyber layer and can manipulate the measurements up to the r-neighbor of p. Since generation buses and zero-injection buses would be too risky to change, she/he skips those buses even if they are in their active target region [32]–[35]. Moreover, she/he avoids to attack the power flow measurements if this alternation leads to violate the KCL at the bus that the line is connected to [36].

An example IEEE 14 case system is demonstrated in Fig. 2, where an attacker's entry point p is bus 10 and his radius r is 2, so she/he can change the measurements of buses 10 (entry point), 9, and 11 (1° neighbors), and 4, 7, 14 (2° neighbors) designated with red stars. It is presumed that she/he can alter the measurements of the power flows in the active area represented by red dashed lines. Note that she/he skips bus 6 since it is a generator node designated by a green square. In addition, she/he also skips the line between 6 and 11 since, for this scenario, it is the only attackable meter connected to bus 6 and any change in this line violates the KCL equation at 6. All the other measurements outside the target region $\mathcal{T}$ represented by red surface are presumed to be still secure to the attacker.

To find a stealth attack vector in $\mathcal{T}$, the attacker tries to minimize the objective function

$$\min_{\check{\boldsymbol{x}}} \; \lambda_z||h(\check{\boldsymbol{x}})_i – h(\hat{\boldsymbol{x}})_i||_2 - \lambda_x||\check{\boldsymbol{x}}_j – \hat{\boldsymbol{x}}_j|| \quad \forall i \in \mathcal{T}_z \quad \forall j \in \mathcal{T}_x$$

$$\text{s.t. } h(\check{\boldsymbol{x}})_k = h(\hat{\boldsymbol{x}})_k,\; \check{\boldsymbol{x}}_l = \hat{\boldsymbol{x}}_l \quad \forall k \notin T_z \quad \forall l \notin \mathcal{T}_x$$
$$\tau_m^{\min} < ||\check{\boldsymbol{x}}|| < \tau_m^{\max},\; \tau_a^{\min} < \angle(\check{\boldsymbol{x}}) < \tau_a^{\max} \tag{5}$$

where $\hat{\boldsymbol{x}}$ denotes the honest state vector, $\check{\boldsymbol{x}}$ stands for false data injected state vector, λ_z and λ_x are weighting factors associated with loss terms, $\mathcal{T}_z$ and $\mathcal{T}_x$ denote the targeted measurements and state variables, $\tau_m^{\min}$ and $\tau_m^{\max}$ denote the minimum and maximum values of the magnitude of $\check{\boldsymbol{x}}$, and $\tau_m^{\min}$ and $\tau_m^{\max}$ represent minimum and maximum values of the angle of $\check{\boldsymbol{x}}$, respectively. In essence, she/he searches a vector $\check{\boldsymbol{x}}$ in the state space of the grid $\mathfrak{X}$ by only targeting some $\boldsymbol{x} \in \mathcal{T}_x$ so that the corresponding measurements $\boldsymbol{z_a} = h(\check{\boldsymbol{x}})$ resemble the original measurements $\boldsymbol{z_o}$ in the measurement space of the grid $\mathfrak{Z}$ restricted by $\mathcal{T}_z$. Note that the objective function in (5) consists of two competing losses. While the first part $||h(\check{\boldsymbol{x}})_i - h(\hat{\boldsymbol{x}})_i||_2$ aims to minimize the measurement differences in $\mathcal{T}_z$, the second part $||\check{\boldsymbol{x}}_j - \hat{\boldsymbol{x}}_j||$ maximizes the attack power injected into the state variables in $\mathcal{T}_x$. The tradeoff between these objectives is directly related to detection risk and attack power since deviation from the original state variables increases the probability of being detected. Consequently, an attacker can increase the attack power at the expense of higher risk of being detected.

The attacker aims to maximize the assault power by minimizing the detection risk. To do that, she/he first defines a free complex variable $\check{\boldsymbol{x}}_j \in \mathfrak{X}$ in the vicinity of original estimated values by probing them with a small Gaussian noise. Then, by the help of SGD algorithm, she/he calculates the gradient of the state variables with respect to the joint loss defined in (5) and updates them iteratively at each step until there is no improvement in the loss. Recall that she/he only updates a state variable if it is in the active insecure area. Eventually, she/he decides whether to inject this obtained false data to the related measurements in the cyber layer of the grid, according to the final loss value obtained during the iterations. In a sense, this individual latent vector search can be interpreted as "training" in the ML terminology [37]; however, it is very specific to the corresponding time slot and should be repeated for each case in order to minimize the detection risks. Note that this generic algorithm can be tailored according to the modeled electric grid and capabilities of the attacker.

B. GNN Modeling of Smart Grids

Smart power grids can be modeled by a connected, undirected, weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ that consists of a finite set of vertices $\mathcal{V}$ with $|\mathcal{V}| = n$, a finite set of edges $\mathcal{E}$, and a weighted adjacency matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ [38]. Buses are represented by vertices $\mathcal{V}$, branches and transformers are represented by edges $\mathcal{E}$ and line admittances are represented by $\boldsymbol{W}$ in this mapping. If the buses i and j are connected, the corresponding weight of the edge $e = (i, j)$ connecting vertices i and j is assigned to W_{ij}. A signal or a function $f : \mathcal{V} \to \mathbb{R}$ in $\mathcal{G}$ can be represented by a vector $\boldsymbol{f} \in \mathbb{R}^n$, where ith component of the vector $\boldsymbol{f}$ corresponds to scalar value at the vertex $i \in \mathcal{V}$.

A fundamental operator defined in spectral graph theory [38] is the graph Laplacian operator $L \in \mathbb{R}^{n \times n}$. Its normalized definition is represented as $L = I_n - D^{-1/2}WD^{-1/2}$ where I_n

is the identity matrix, and $D \in \mathbb{R}^{n\times n}$ is the diagonal degree matrix with $D_{ii} = \sum_j W_{ij}$. Since L is a real symmetric positive semidefinite matrix, all eigenvalues λ_i of it are real valued and nonnegative, and it has a complete set of orthonormal eigenvectors $\boldsymbol{u_i}$ [38]. Thus, L can be diagonalized as $L = U\lambda U^T$ where $U = [\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-1}] \in \mathbb{R}^{n\times n}$ represent the n orthonormal eigenvectors, and $\lambda = \text{diag}([\lambda_0, \lambda_1, \ldots, \lambda_{n-1}]) \in \mathbb{R}^{n\times n}$ denotes the diagonal matrix of n eigenvalues $0 = \lambda_0 < \lambda_1 < \cdots < \lambda_{n-1} < 2$ due to the normalization [38]. In fact, vectors $\boldsymbol{u}_i$ form the graph Fourier basis and λ_i values represent frequencies in the graph spectral domain [38].

The Fourier transform and its inverse can be defined in the graph spectral domain analogously to the classical Fourier transform. Namely, the graph Fourier transform (GFT) and inverse graph Fourier transformation (IGFT) are defined as $\tilde{s} = U^T s$ and $s = U\tilde{s}$ where s and $\tilde{s}$ denote vertex and spectral domain signals, respectively.

Unlike classical signal processing, a meaningful translation operator does not exist in the vertex domain [39]. Therefore, to apply a convolution operation to graph signals, they are first transformed into the spectral domain using GFT, then convolved (Hadamard product) in the spectral domain and finally the result transformed back to the vertex domain using IGFT [39]. Formally, $x *_{\mathcal{G}} y = U((U^T x) \odot (U^T y))$.

Similarly, a graph signal $x \in \mathbb{R}^n$ is filtered by a kernel g_θ

$$y = g_\theta *_{\mathcal{G}} x = g_\theta(U\lambda U^T)x = U g_\theta(\lambda) U^T x \in \mathbb{R}^n \qquad (6)$$

where $g_\theta(\lambda) = \text{diag}(\theta)$ is a nonparametric kernel, and $\theta \in \mathbb{R}^n$ is a vector of Fourier coefficients [38]. To put it differently, g_θ filters the signal x in the spectral domain by multiplying its spectral components with the free θ coefficients in a similar way with the classical signal processing in the Fourier domain. Eventually, the filtered signal is transformed back to the vertex domain by IGFT [38]. Nevertheless, those nonparametric filters are not spatially localized, and hence, computational complexity of (6) is $\mathcal{O}(n^2)$ due to the matrix multiplication with U. To thwart this problem, Defferrard *et al.* [39] proposed to parameterize $g_\theta(L)$ as a Cheybyshev polynomial function, which can be computed recursively from L.

The K order Chebyshev polynomial of the first kind $T_k(x)$ is computed recursively as follows [40]:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \qquad (7)$$

where $T_0(x) = 1$ and $T_1(x) = x$. Therefore, a filter g_θ can be approximated by Chebyshev polynomials, T_k, up to order $K - 1$ and a signal x can be filtered by g_θ

$$y = g_\theta *_{\mathcal{G}} x = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x \qquad (8)$$

where the parameter $\theta \in \mathbb{R}^K$ is a vector of Chebyshev coefficients, and $T_k(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\tilde{L}$ given by $\tilde{L} = 2\,L/\lambda_{\text{max}} - I_n$. Finally, filtered signal y can be calculated by the help of (7) and (8) as

$$y = \sum_{k=0}^{K-1} \theta_k \bar{x}_k \qquad (9)$$

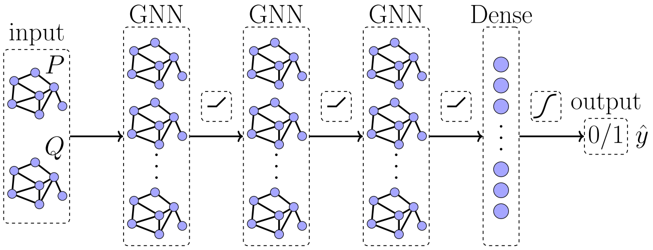

Fig. 3. Architecture of the proposed GNN-based detector.

where $\bar{x}_0 = x$, $\bar{x}_1 = \tilde{L}x$, and $\bar{x}_k$ is computed recursively

$$\bar{x}_k = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}. \qquad (10)$$

Note that convolution in (9) is K-localized, and its computational complexity is reduced to $\mathcal{O}(K|\mathcal{E}|)$. Thus, it can efficiently be utilized in the intermediate layers of the GNN to model the non-Euclidean measurement data of power grids. For detailed analysis, please refer to [38] and [39].

C. Detection of Attacks Using GNN

The architecture of the proposed GNN-based detector is depicted in Fig. 3. It contains one input layer to represent bus power injection measurements, L hidden Chebyshev graph convolution layers to extract spatial features and one output dense layer to predict the probability of the input sample being attacked. In this layered structure, X^0 denotes two channel input tensor $[P_i, Q_i] \in \mathbb{R}^{n\times 2}$, X^l represents the output tensor of hidden layer $l \in \mathbb{R}^{n\times c_l}$, $y \in \mathbb{R}$ designates the scalar output of the neural network (NN), $1 \leq l \leq L$, and c_l stands for the number of channels in layer l. Particularly, a GNN hidden layer l takes $X^{l-1} \in \mathbb{R}^{n\times c_{l-1}}$ as input and produces $X^l \in \mathbb{R}^{n\times c_l}$ as output. Different from the hidden graph layers, dense layer outputs y in classical feed-forward neural networks by feeding with the inputs $X^L \in \mathbb{R}^{n\times c_L}$.

In this multilayer architecture, each Chebyshev layer l for $1 \leq l \leq L$ transforms its input X^{l-1} by first applying graph convolution operation using (9) and (10), then adding a bias term and finally employing a nonlinear rectified linear unit function (ReLU) defined as $\text{ReLU}(x) = \max(0, x)$ to generate X^l. Namely

$$X^l = \text{ReLU}(\theta^l *_{\mathcal{G}} X^{l-1} + b^l) \qquad (11)$$

where $\theta^l \in \mathbb{R}^{K\times c_{l-1}\times c_l}$ denotes free Chebyshev coefficients and $b^l \in \mathbb{R}^{c_l}$ represents bias term of the layer l. Recall that each Chebyshev layer gets extra scaled Laplacian $\tilde{L}$ values. In a similar fashion, output of the dense layer is computed by $y = \sigma(W^L X^L + b^L)$, where $W^L \in \mathbb{R}^{n\times c_L}$ denotes the weights of each feature, $b^L \in \mathbb{R}$ represents the bias term and σ designates the nonlinear sigmoid operation: $\sigma(x) = 1/(1 + e^{-x})$.

IV. Experimental Results

A. Data Generation

Generating reliable data is the first step in building a successful defense mechanism since all the future blocks depend on it. Since it is not possible to find publicly available power grid

Algorithm 1: Data Generation.

```
   Input  : normalized scaler $S$                    // $\mu = 0,\ \sigma = 1$
   Output: $Z_n, X_n$ for each test system $n$
1  $N \leftarrow [14,\ 118,\ 300]$                    // IEEE bus systems
2  $T \leftarrow [1$ to $9600]$                       // timestep index
3  $k,\ \sigma_s \leftarrow 0.1,\ 0.03$               // scaling coefficients
4  $\sigma_n \leftarrow 0.01$                         // noise coefficient
5  Function Generate($sg,\ t$):
6      foreach $bus \in sg.genbus \cup sg.loadbus$ do
7          $bus.scale \leftarrow \mathcal{N}(1 + k \times S_t,\ \sigma_s)$
8      $z_o = sg.PF()$                                // run AC power flow
9      $z_o \leftarrow \mathcal{N}(z_o,\ z_o \times \sigma_n)$,      // 1% additive noise
10     $\hat{x} \leftarrow sg.PSSE(z_o)$              // estimate state
11     return $z_o,\ \hat{x}$
12 Function Main:
13     foreach $n \in N$ do
14         $Z_n,\ X_n \leftarrow [\,],\ [\,]$         // empty vectors
15         $sg \leftarrow$ SG($n$)                    // smart grid obj.
16         foreach $t \in T$ do
17             $z,\ x \leftarrow$ Generate($sg,\ t$)
18             $Z_n[t],\ X_n[t] \leftarrow z,\ x$     // append
19         $Z_n.save(),\ X_n.save()$
```

data due to privacy issues, synthetic data are generated using Pandapower [41] for several test cases including IEEE 14, 118, and 300. Data generation steps are summarized in Algorithm 1. To make the data as realistic as possible, we first downloaded ERCOT's 15 min interval backcasted actual load profiles [42].

Next, we arbitrarily selected the "BUSHILF_SCENT" profile, which corresponds to south-central Texas having a high load factor. Then, we normalized the time series data to zero mean and unit variance "scaler" vector $\boldsymbol{S}$ so that it can be easily adapted to each test system. Having obtained $\boldsymbol{S}$, we run the main function of the Algorithm 1 where a smart grid object sg is created for each test system having n bus and Generate function is called for each timesteps t. In generate function, the scaling parameters of generator and load buses are assigned to a sample drawn from a normal distribution with $1 + 0.1 \times \boldsymbol{S_t}$ mean and 0.03^2 variance, where $\boldsymbol{S_t}$ denotes the value of $\boldsymbol{S}$ at time-step t. Due to the properties of normal distribution, the scaling operation provides practically more than $\pm 20\%$ dynamic range on average with respect to the static case. We limit the scaling range between 0.7 and 1.3 for the convergence of power flow solutions. As a next step, ac power flow solutions are calculated, and the measurements considered to have 1% noise are read. Finally, PSSE is conducted, and estimated state variables are returned along with original meter values to the Main function. In Fig. 4, the scaling process formulated with line 7 in Algorithm 1 is demonstrated for one week period (7 × 96 samples). Please note that $\boldsymbol{S}$ depicted in Fig. 4(b) is just a normalized version of the load profile given in Fig. 4(a). Next, the load and generation values of buses are multiplied with a value sampled from a distribution $\mathcal{N}(1 + k \times \boldsymbol{S_t},\ \sigma_s)$, which has $1 + k \times \boldsymbol{S_t}$ mean and σ_s standard deviation at time t. Namely, they follow the patterns in $\boldsymbol{S}$ by deviating around their static values defined in their test systems.

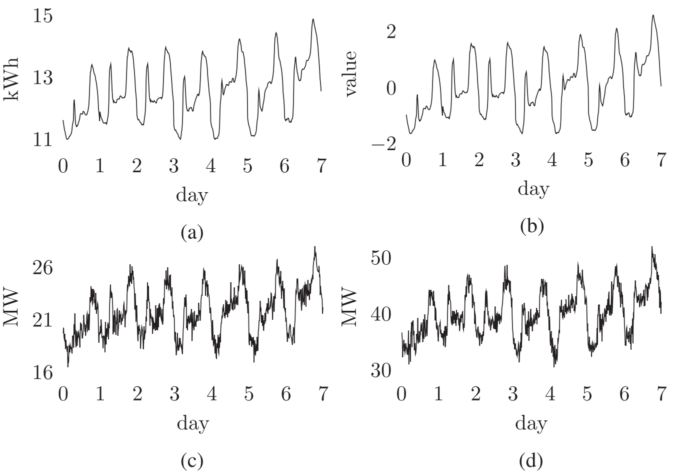

Fig. 4. Example scaling process for bus 2 in the IEEE-14 bus test system. First, load profile of south-central Texas region [see (a)] is normalized and the scalar $\boldsymbol{S}$ is obtained [see (b)]. Then, load [see (c)] and generation [see (d)] values of buses are multiplied with a scalar value sampled from the distribution $\mathcal{N}(1 + k \times \boldsymbol{S_t},\ \sigma_s)$ having $1 + k \times \boldsymbol{S_t}$ mean and σ_s standard deviation at time t. For this example, the static values of load and generation at bus 2 in the IEEE-14 bus system are 21.7 MW and 40 MW, respectively. Please note that while S has relatively smooth transition between the time-steps, load, and generation values have some spikes due to deviation of multiplier around $\boldsymbol{S_t}$, which increases the variety of samples in the dataset. (a) Load profile of south-central Texas. (b) Scaler $\boldsymbol{S}$. (c) Load of bus 2 in IEEE-14. (d) Generation of bus 2 in IEEE-14.

B. Attack Generation

After generating honest data samples, we focus on malicious data samples in this section, where the attack generation steps are summarized in Algorithm 2. The algorithm gets original measurements matrix $\boldsymbol{Z_n} \in \mathbb{R}^{T \times m}$ and estimated state variable matrix $\boldsymbol{X_n} \in \mathbb{R}^{T \times n}$ and produces their attacked version as well as corresponding sample vector $Y_n \in \mathbb{R}^T$, where 0 and 1 in $\boldsymbol{Y_n}$ represent honest and malicious samples, respectively. As can be seen from Algorithm 2, main function simply creates the smart grid and attacker objects, fetches the current sample, and calls generate function for each system having n buses at each time-step t.

Generate function, in contrast, simulates a "smart" intruder capable of entering the cyber layer of the grid, computing an unobservable attack vector and deciding to insert the false data into the measurement devices according to the "quality" of the attack. In this regard, since it is not realistic to assume that an attacker can inject false data at every time step due to practical reasons, generate function first models the attack frequency by a r.v. $f \sim \mathcal{N}(0, 1)$ where $f > \tau_{\text{freq}}$ means the attacker has successfully entered the system. To attack roughly 15% of total time-steps on average, τ_{freq} is selected as 1. Second, it models the target area of the attacker $\mathcal{T}$ similar to the red area given with Fig. 2 by help of a r.v. $p \sim \mathcal{U}(1, n)$ and a predefined attack radius r. To this end, it calls a breadth first search method of the attacker object to model the target area defined by a set of measurements captured by the attacker denoted by $\mathcal{T}_z$ and a set of state variables $\mathcal{T}$ intended to inject the false data. In fact, all the measurements and state variables located up to r-distance neighbor of the bus p are assumed to be in $\mathcal{T}_z$ and $\mathcal{T}_x$ except the generator buses and zero-injection buses. Then, it calls the

Algorithm 2: Attack Generation.

```
Input  : $\boldsymbol{Z_n},\ \boldsymbol{X_n}$ for each test system $n$
Output: $\boldsymbol{Z_n},\ \boldsymbol{X_n},\ \boldsymbol{Y_n}$ for each test system $n$
$N \leftarrow [14,\ 118,\ 300]$                          // IEEE bus systems
$T \leftarrow [1$ to $9600]$                             // timestep index
$\sigma_n \leftarrow 0.005$                              // initial disturbance
$\lambda_z,\ \lambda_x \leftarrow 1,\ 1$                 // loss weights
$\eta,\ E \leftarrow 0.001, 1000$                        // learning rate and epochs
$\tau_{freq},\ \tau_{loss} \leftarrow 1,\ 0.1$           // attackers thresholds
$R_{min} \leftarrow \{14:2,\ 118:3,\ 300:6\}$            // min radius
$R_{max} \leftarrow \{14:3,\ 118:4,\ 300:8\}$            // max radius
Function attacker.attack($\boldsymbol{z_o},\ \hat{\boldsymbol{x}},\ \mathcal{T}_z,\ \mathcal{T}_x$):
    trainable $\boldsymbol{V}: 0.9 < \boldsymbol{V} < 1.1$
    trainable $\boldsymbol{\theta}: -\pi < \boldsymbol{\theta} < +\pi$
    $\boldsymbol{V},\ \boldsymbol{\theta} \leftarrow \mathrm{abs}(\hat{\boldsymbol{x}}),\ \mathrm{angle}(\hat{\boldsymbol{x}})$
    foreach $j \in \mathcal{T}_x$ do
        $\boldsymbol{V_j} \leftarrow \boldsymbol{V_j} + \mathcal{N}(0,\ \sigma_n^2)$
        $\boldsymbol{\theta_j} \leftarrow \boldsymbol{\theta_j} + \mathcal{N}(0,\ \sigma_n^2)$
    foreach $epoch \in E$ do
        $\check{\boldsymbol{x}} \leftarrow \boldsymbol{V}e^{j\boldsymbol{\theta}}$          // complex state vars.
        $\boldsymbol{z_a} \leftarrow h(\check{\boldsymbol{x}})$                             // real measurements
        $L_z \leftarrow \sum_i ||\boldsymbol{z_a}_i - \boldsymbol{z_o}_i||_2,\ \forall i \notin \mathcal{T}_z$
        $L_x \leftarrow \sum_j ||\check{\boldsymbol{x}}_j - \hat{\boldsymbol{x}}_j||,\ \forall j \notin \mathcal{T}_x$
        $L \leftarrow \lambda_z L_z - \lambda_x L_x$
        foreach $j \in \mathcal{T}_x$ do
            $\boldsymbol{V_j} \leftarrow \boldsymbol{V_j} - \eta \frac{\partial L}{\partial \mathbf{V_j}}$
            $\boldsymbol{\theta_j} \leftarrow \boldsymbol{\theta_j} - \eta \frac{\partial L}{\partial \boldsymbol{\theta_j}}$
    $\check{\boldsymbol{x}} \leftarrow \boldsymbol{V}e^{j\boldsymbol{\theta}}$              // complex state vars.
    $\boldsymbol{z_a} \leftarrow h(\check{\boldsymbol{x}})$                                 // real measurements
    return $\boldsymbol{z_a},\ L$
Function Generate($attacker,\ \boldsymbol{z_o},\ \hat{\boldsymbol{x}}$):
    $y,\ \boldsymbol{z} \leftarrow 0,\ \boldsymbol{z_o}$                    // no attack yet
    $f \sim \mathcal{N}(0,\ 1)$                                              // attack frequency
    if $f > \tau_{freq}$ then
        $p \sim \mathcal{U}(1, n)$                                           // entry point
        $r \leftarrow \mathcal{U}(R_{min}[n],\ R_{max}[n])$                  // attack radius
        /* determine attack surface by BFS                                  */
        $\mathcal{T}_z,\ \mathcal{T}_x \leftarrow attacker.BFS(p, r)$
         $\boldsymbol{z_a},\ loss \leftarrow attacker.attack(\boldsymbol{z_o},\ \hat{\boldsymbol{x}},\ \mathcal{T}_z,\ \mathcal{T}_x)$
        if $loss < \tau_{loss}$ then
            $y,\ \boldsymbol{z} \leftarrow 1,\ z_a$                         // attack injected
    $\check{\boldsymbol{x}} \leftarrow sg.PSSE(\boldsymbol{z})$
    return $\boldsymbol{z},\ \check{\boldsymbol{x}},\ y$
Function Main:
    foreach $n \in N$ do
        $\boldsymbol{Y_n} \leftarrow [\ ]$                                   // empty label vector
        $sg \leftarrow \mathrm{SG}(n)$                                       // smart grid obj.
        $attacker \leftarrow Attacker(n)$                                    // attacker obj.
        foreach $t \in T$ do
            $\boldsymbol{Z_n}[\boldsymbol{t}],\ \boldsymbol{X_n}[\boldsymbol{t}],\ \boldsymbol{Y_n}[\boldsymbol{t}] \leftarrow$
             Generate($attacker,\ \boldsymbol{Z_n}[\boldsymbol{t}],\ \boldsymbol{X_n}[\boldsymbol{t}]$)
        $X_n.save(),\ Z_n.save(),\ Y_n.save()$
```

attack method of the attacker to compute and insert z_a if the method returns a loss value smaller than threshold τ_{loss}.

The attacker's assault method solves the nonlinear and nonconvex minimization (5) in the Tensorflow [43] library. As a first step, it defines a free trainable vector tuple to represent the new complex state variables $\check{\boldsymbol{x}}$, which constitutes the "fake" operating point at the end of attack: voltage magnitude $\boldsymbol{V}$ is limited to $0.9 < \boldsymbol{V} < 1.1$ p.u. and voltage angle $\boldsymbol{\theta}$ is limited to $-\pi < \boldsymbol{\theta} < \pi$. Next, it initializes the jth elements of this tuple in the vicinity of their original variables by adding a small Gaussian white noise $\mathcal{N}(0,\ \sigma_n^2)$ if $j \in \mathcal{T}_x$ to ignite the optimization. This small proximity could play a vital role because SGD may fail to reduce the objective function if the initial point is not balanced [37]. A $\check{\boldsymbol{x}}$ too close to $\hat{\boldsymbol{x}}$ might result to no update at all in optimization variables $\boldsymbol{V}$ and $\boldsymbol{\theta}$, whereas a $\check{\boldsymbol{x}}$ too distant to $\hat{\boldsymbol{x}}$ might get stuck in a secluded region of $\mathfrak{X}$ and produce a highly suspicious $\boldsymbol{z_a}$. Thus, $\sigma_n = 0.005$ is found to be accurate according to the minimization loss. Then, for each epoch, it obtains $\boldsymbol{z_a}$ using $h(\boldsymbol{x})$ and consequently calculates loss term L_z as a root mean squared error between $\boldsymbol{z_a}$ and $\boldsymbol{z_o}$, and L_x as a mean absolute error between $\check{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}$. Eventually, it calculates gradients of total loss $L = \lambda_z L_z - \lambda_x L_x$ with respect to optimization variables $\boldsymbol{V_j}$ and $\boldsymbol{\theta_j} \in \mathcal{T}_x$ and updates corresponding terms in the reverse direction of gradients by scaling the gradients with learning rate η before starting the next epoch. Lastly, it returns $\boldsymbol{z_a}$ and final loss L to Generate function and halts. Distributions of some important values of IEEE 300 test system are given in Fig. 5 after running Algorithm 2.

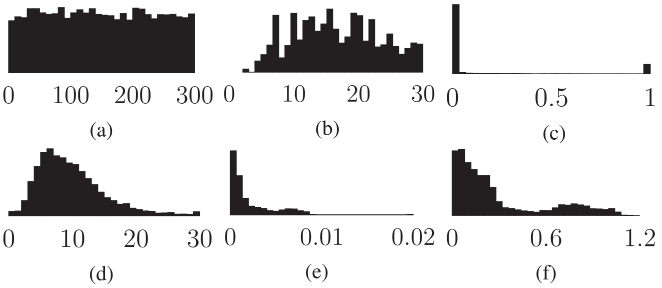

Fig. 5. Distributions of (a) attacker's entry point, (b) ratio of seized meters in percentage, (c) minimization loss values by solving (5), (d) maximum absolute difference of attack to measurements and maximum absolute difference of state variables due to attacks in terms of (e) voltage magnitude and (e) angle obtained by Algorithm 2. Roughly speaking, the attacker initiates the assault arbitrarily and uniformly from any node [see (a)] by capturing up to 30% of the available meters [see (b)] and succeeding 83.4% of the attempts [see (c)]. Adding maximum 30 MW or 30 MVAR of attack to measurement devices [see (d)] creates maximum 2% deviation in magnitudes [see (e)] and maximum of 1.2° in angles (f) of state variables. Attack frequency and attack power might be strengthened by increasing τ_{freq} and τ_{loss} parameters in Algorithm 2 at risk of high detection by operator.

C. Attack Detection

In order to immediately predict the attack probability in our models instead of waiting for PSSE result, we only use measurement values in our detectors. Moreover, since $P_i + jQ_i = \sum_{k\in\Omega_i} P_{ik} + jQ_{ik}$, node values can represent branch values as summation in their corresponding Ω_i and the proposed GNN-based detector accepts features in its nodes, we decide to use only P_i and Q_i as input to our models. PSSE and BDD modules, on the contrary, continue to receive every available measurement to operate as depicted in Fig. 1.

Having decided to input features $[P_i, Q_i]_n \in \mathbb{R}^{9600\times n\times 2}$ and output labels $\boldsymbol{Y_n} \in \mathbb{R}^{9600}$ for $n \in \{14, 118, 300\}$ bus test systems where 0 denotes honest and 1 denotes malicious samples of $\boldsymbol{Y_n}$, we partition the first 60% of the samples for training the proposed detectors, the next 20% for validating and tuning the hyperparameter of the models, and the last 20% for evaluating the performances of the detectors. Then, we standardize each split separately, with a zero mean and a standard deviation of one, to have a faster and more stable learning process [44].

As a next step, we implement the GNN-based FDIA detector having a multilayer Chebyshev graph convolution layer in its hidden layer and one dense layer on top of that as depicted in Fig. 3. We add a bias term and ReLU activation functions between graph convolutional layers and sigmoid activation functions at the last dense layer to increase the detector's nonlinear modeling ability [44]. As for weighted adjacency matrix $\boldsymbol{W}$, we use the magnitude of complex sparse Ybus matrix of the corresponding grid, which models the relation between nodes, determine the graph Laplacian L and scale it to obtain $\tilde{L}$.

All free unknown parameters defined in the model are computed by a supervised training using cross-entropy loss

$$L(\hat{y}, W_\theta) = \frac{-1}{N} \sum_{n=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \quad (12)$$

over the training set where N denotes the number of samples in the training set, W_θ represents all trainable parameters θ_l and b_l for $1 \leq l \leq L$ along with W^L and b^L in the model, and y_i and $\hat{y}_i$ stand for true and predicted class probability for sample i, respectively. Training samples are fed into the model as mini batches having 64 samples with 128 maximum number of epochs in addition to the early stopping where 16 epochs are tolerated without any improvement in the cross entropy loss of validation set. All the implementation was carried out in Python 3.8 using Pandapower [41], Sklearn [45], and Tensorflow [43] libraries on Intel i9-8950 HK CPU 2.90 GHz with NVIDIA GeForce RTX 2070 GPU.

To evaluate the performance of proposed model in the binary classification task, we use true positive rate or detection rate (DR) $\text{DR} = \text{TP}/(\text{TP} + \text{FN})$ as probability of attack detection, false positive rate, or false alarm (FA) rate $\text{FA} = \text{FP}/(\text{FP} + \text{TN})$ as probability of falsely alarming the system even though there is no attack, and the F-measure or $F1$ score $F_1 = 2 * \text{TP}/(2\text{TP} + \text{FP} + \text{FN})$ as the harmonic mean of the precision and sensitivity of classifier [44], where TP, FP, TN, and FN stand for true positives, false positives, true negatives, and false negatives, respectively.

Predicted class probabilities $\hat{y}$ obtained by GNN-based detector along with $\boldsymbol{r}^N$ values computed by the LNRT-based BDD system are given side-by-side for each test system in Fig. 6. Note that while it is almost impossible to separate honest and malicious samples by $\boldsymbol{r}^N$ due to intricate class distributions on the left side, the proposed GNN-based detector efficiently "filters" malicious samples in its hidden layers and provides easily separable $\hat{y}$ distributions. Please refer to Table I for detailed classification results.

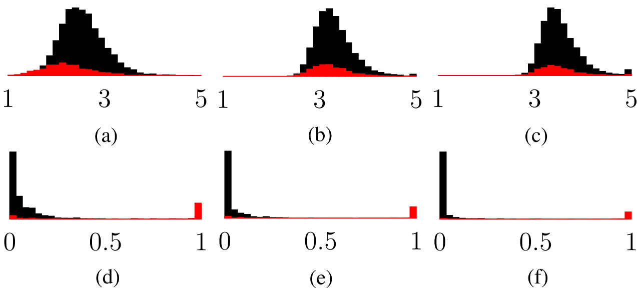

Fig. 6. Distributions of normalized residues $\boldsymbol{r}^N$ and predicted class probabilities $\hat{y}$ for each IEEE test system having 14, 118, and 300 buses, respectively, computed on test dataset where black and red bars denote honest and malicious samples, respectively. GNN-based detector transforms the class distributions so that they can be easily separated. In traditional BDD, in contrast, it is not possible to isolate the bad samples due to stealthy FDIA. (a) $\boldsymbol{r}^N$ for IEEE 14. (b) $\boldsymbol{r}^N$ for IEEE 118. (c) $\boldsymbol{r}^N$ for IEEE 300. (d) Å· for IEEE 14. (e) Å· for IEEE 118. (f) Å· for IEEE 300.

TABLE I
COMPARISON OF DETECTOR PERFORMANCES (BEST IN BOLD, WORST IN ITALIC) IN TERMS OF DR, FA, AND F-MEASURE ($F1$) CLASSIFICATION METRICS FOR EACH IEEE TEST CASE SYSTEM WITH 14-, 118-, AND 300-BUS TEST SYSTEMS

	IEEE 14			**IEEE 118**			**IEEE 300**		
model	**DR**	**FA**	**F1**	**DR**	**FA**	**F1**	**DR**	**FA**	**F1**
BDD	**100.0**	*100.*	*27.35*	**100.0**	*100.*	*26.32*	**100.**	*100.*	*23.26*
DTC	*68.64*	29.0	67.91	75.63	22.4	77.06	74.19	9.69	78.79
SVC	75.49	**0.13**	85.97	*67.21*	11.2	74.53	64.84	11.6	71.08
MLP	79.42	2.95	87.17	79.65	8.86	83.89	60.83	7.51	70.54
RNN	78.38	3.31	86.33	74.73	3.05	83.87	64.49	9.73	71.08
CNN	79.30	3.25	87.00	86.33	4.88	89.82	92.11	3.33	93.26
GNN	83.97	2.43	**90.14**	90.61	**1.18**	**94.07**	96.51	**0.72**	**97.67**

D. *Model Scalability*

Model scalability in terms of total number of parameters and prediction time is examined herein subsection. We first assess the total number of free trainable parameters in the proposed models. While each K-localized Chebyshev layer l having c_l channels for $1 \leq l \leq L$ consists of $K \times c_{l-1} \times c_l$ Chebyshev coefficients and c_l bias terms, the final dense layer assumes $n \times c_L$ dense weights and a bias term. Thus, the total number of parameters in the model is given by

$$K \sum_{l=1}^{l=L} ((c_{l-1} + 1) \times c_l) + n \times c_L + 1. \quad (13)$$

It can be seen from (13) that except for the last dense layer, the number of parameters in a GNN is free from bus size n and it linearly and independently scales with the neighborhood order K, previous layer's filter size c_{l-1} and its own filter size c_l. Second, we measure and save the prediction delays of each system. To fairly analyze how prediction time t and total number of parameters p change with the increasing bus size n, we fix the other variables at $K = 3$, $L = 3$, and $c_l = 32$ for each layer l. Fig. 7 demonstrates that models are linearly scalable in terms of n.

E. *Visualization of How Information Spreads Through Layers*

In this section, to explain and visualize how the proposed GNN-based detector distinguishes a malicious sample from an honest one, we examine the output of the filters from each layer of a trained network. To this end, first we arbitrarily select a node from the center region of the grid, for instance, bus 68 of the IEEE 118 bus system. Second, we randomly choose an

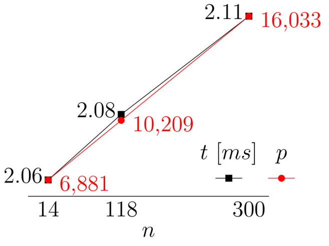

Fig. 7. Linear scalability of the proposed models in terms of prediction time t [ms] and total number of parameters p.

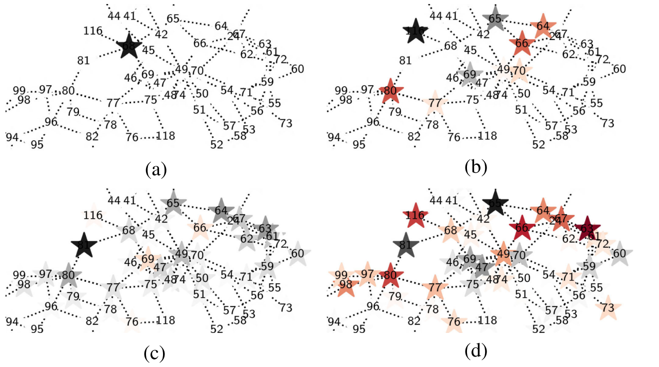

Fig. 8. Visualization of anomaly propagation through the layers of the proposed network. To better follow how the anomaly spreads, we plotted the signal differences at each layer. It can be clearly seen that each node sends information up to its K-neighbor in each filter and the message advances K nodes through each layer. Note that K is chosen 3 for this network in the training phase. For clarity, only affected area of the grid is depicted. (a) Input layer difference $\boldsymbol{\delta}^0$. (b) Layer 1 filter difference $\boldsymbol{\delta}^1$. (c) Layer 2 filter difference $\boldsymbol{\delta}^2$. (d) Layer 3 filter difference $\boldsymbol{\delta}^3$.

honest sample $\boldsymbol{s}$ from the training data set and create a malicious sample $\acute{\boldsymbol{s}}$ by adding a point-wise attack to the bus 68 with a magnitude one to easily follow the spreading of this anomalous information through the hidden layers of the network. To focus on the anomaly, we calculate the difference of the Chebyshev filter outputs $\boldsymbol{\delta}^l = \acute{\boldsymbol{s}}^l - \boldsymbol{s}^l$ at each layer l for $0 \leq l \leq 4$ including the input layer $l = 0$. Starting from $\boldsymbol{\delta}_0$, the example filter output differences from each of the Chebyshev layers are depicted in Fig. 8. It can be clearly seen from Fig. 8 that each node transmits this anomalous data to its K-neighbors, where K is chosen as 3 for this network, and the information advances in K-locality through each of the Chebyshev layers. The dense layer at the end of the model, in contrast, uses the anomalous features and decides its outputs by a sigmoid function. As expected, $\boldsymbol{s}^4 = 0$ and $\acute{\boldsymbol{s}}^4 = 1$ at the output of the model. In essence, K-localized Chebyshev filters of the proposed detector extract this spatial information through its GNN and dense layers to predict the probability of attacks for the input sample.

F. Comparison With Other Methods

To compare our GNN-based models with the available detectors, we also implement DTC [21], support vector (SVC) [17], MLP [18], RNN [20], and CNN [19] based FDIA detectors. Since we do not have access to the dataset of corresponding works, we train, validate, and test these models similar to our proposed detector using our dataset.

DTC is a member of the nonparametric and supervised ML algorithms family aiming to create a multitude of decision rules on the input features to predict the class labels [46]. SVC, in contrast, tries to predict the hyperplane fitting the target variable by maximizing the margin and keeping the error within a threshold [47]. Therefore, only the SVCs residing in the margin contribute to the decision boundary and determine the error tolerance of the fitted hyperplane. MLP is a feed-forward type NN consisting of one input layer, one or more hidden layers, and one output layer. It is trained by using a backpropagation algorithm, which iterates backward the errors from the output layer to the lower layers, and feed-forwards the weight updates from the input layer to the higher layers [48]. Different from MLP, RNN is an NN that utilizes an internal memory component to remember its previous outputs to be used as next inputs, which enables it to appropriately model sequential data [49]. CNN, on the contrary, is a regularized form of MLP where fully connected relations are replaced with shift invariant and weight shared convolution filters, which allows it to better model the spatial and temporal correlations of the data [50].

The DR, FA rate, and $F1$ score of each model for each test system are given as percentages in Table I. Clearly, the LNR-based BDD system falls behind every other model due to nonseparable class distributions of $\boldsymbol{r}^N$ values, as depicted in Fig. 6. It simply predicts each sample as malicious, and this results in 100% FA rate for each test system. Non-NN-based approaches such as DTC and SVM, in contrast, enhance the FA and perform better than BDD by an $F1$ score range between 67.91% and 85.97% due to their nonlinear modeling capabilities. The NN-based family surpasses the non-NN-based models in general, except the MLP where it achieves comparable results with SVC and DTC. The RNN-based detector yields 86.33%, 83.87%, and 71.08% $F1$ score for IEEE 14, 118, and 300 bus systems, respectively. Only CNN- and GNN-based detectors reach the 90% $F1$ range. Nevertheless, GNN outperforms CNN models by 3.14%, 4.25%, and 4.41% in $F1$ for IEEE test cases with 14, 118, and 300 buses, respectively.

Our experiments point out that architectural differences in the NN family play a vital role in terms of the detection performance. MLP-based detectors tend to overfit the training data and fail to generalize due to its fully connected relationship between its units. RNNs, in contrast, cannot achieve desired results since node values do not form a sequence type of data. The performance of CNN-based models comes after GNN due to their ability to model the temporal and spatial relations of the input data in the Euclidean space, where the locality of the input features can be represented by regular linear grids such as in image or video data. Nevertheless, the inherent graph structure of power grid measurements cannot be modeled in the Euclidean space except in trivial cases. As a matter of fact, graph data requires topology-aware models such as GNN to better reflect the adjacency relations of the measurement data.

G. Impact of Different Weights in the Attack Generation

To assess the impact of different weights in (5) on the detection performance of the proposed approach, we generate two extra

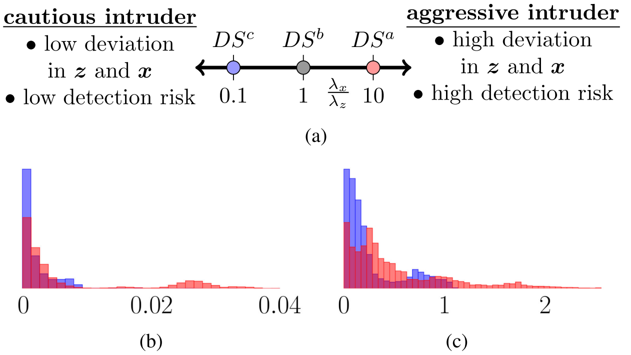

Fig. 9. Impact of attack power on the datasets and distributions of maximum absolute difference of state variables due to attacks in DS^c (blue) and DS^a (red) in terms of voltage magnitude [see (b)] and angle [see (c)] obtained by Algorithm 2. Note that while max $\Delta|V|$ and $\Delta|\theta|$ values can reach 0.01 p.u. and 1.1° in DS^c, they are spread up to 0.04 p.u. and 2.5° in DS^a, respectively. (a) Impact of attack power $\frac{\lambda_x}{\lambda_z}$ on the generated datasets. (b) $\max \Delta|V|$ [p.u.]. (c) $\max \Delta|\theta|$ [degree].

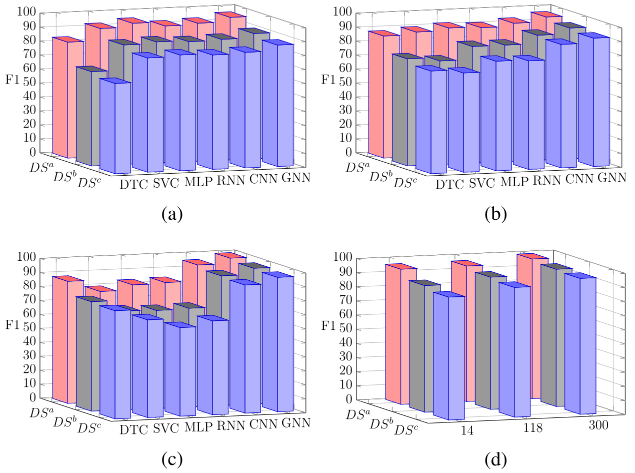

Fig. 10. Impact of attack power, detector model, and grid size on the $F1$ score of implemented detectors. For each model and test system, the performance of the detectors increase from "cautious" (DS^c) to "aggressive" (DS^a) attacks in (a)–(c), for IEEE 14-, 118-, and 300-bus test systems, respectively. In (d), effect of attack power and grid side is visualized for the proposed GNN-based detector. The performance of the detector for the proposed model increases with larger systems and more aggressive attacks. (a) Effect of attack power and detection model for $n = 14$. (b) Effect of attack power and detection model for $n = 118$. (c) Effect of attack power and detection model for $n = 300$. (d) Effect of attack power and number of buses for the GNN model.

datasets for each test system having 14-, 118-, and 300-bus test systems. In this connection, the Algorithm 2 is executed two more times for each test system with $\lambda_z = 10,\ \lambda_x = 1$ for dataset-c (DS^c) and $\lambda_z = 1,\ \lambda_x = 10$ for dataset-a DS^a as specified in line 4 of Algorithm 2 where DS^c and DS^a indicates *cautious* and *aggressive* intruder, respectively. In DS^c, the intruder becomes more cautious and avoids potential detection by decreasing $\frac{\lambda_x}{\lambda_z}$ ratio, which corresponds to the attack power. In contrast, the intruder becomes more aggressive and increases the attack power by increasing this ratio at the expense of high detection risk in DS^a. She/he keeps weighting factors *balanced* in previously generated dataset-b (DS^b) by assigning $\lambda_z = 1,\ \lambda_x = 1$. Fig. 9 illustrates the dataset distributions corresponds to DS^c and DS^a with blue and red colors, respectively. It can be seen from Fig. 9 that the ratio of weighting factors λ_x and λ_z defined in (5) directly affect the attack power on the state variables.

After obtaining DS^c and DS^a for each test system, we applied our detectors on DS^c and DS^a to compare model performances similar to the previous comparisons conducted on DS^b. Namely, we split and scale the datasets and train detector models on the training split, optimize the parameters on the validation split and evaluate the final results on the test split. Fig. 10 summarizes the classification results in terms of $F1$ ratios. As expected, detection performances increase from DS^c to DS^a for each model and test system due to comparably more separable class distributions between honest and malicious samples. However, the proposed GNN outperforms the best available solutions in the literature for 14-, 118-, and 300-bus test systems by 4.31%, 3.46%, 4.48% for the "cautious," 3.14%, 4.25%, 4.41% for the "balanced," and 3.19%, 3.43%, 3.88% for the "aggressive" intrusion, respectively. In addition, it is observed from our experiments that GNN performs better compared to other models in larger cases because when the number of nodes increases, the spatial correlation between adjacent measurements becomes more dominant compared to the global correlations between all measurements. Since GNN is specifically designed to exploit this spatial information of the data, it performs a better job for larger cases. In other words, the denser topology translates into more spatial correlation, which improves GNN's accuracy.

H. Tuning Model Hyperparameters

Traditional hyperparameter tuning algorithms such as random search and grid search try to find the optimal parameters by randomly or sequentially sampling the parameters from the hyperparameter space of the model and ignoring the results of previous trials. For example, even though any combination of a parameter set for a specific value of a parameter would fail to perform well regardless of the other parameters, these techniques might continue to run its trials with these specific values. Thereupon, selecting the optimal hyper parameters using random or grid search could be highly stagnant particularly for large hyperparameter spaces. On the contrary, the Bayesian optimization technique considers performances of the past trials to better explore the parameter space. By focusing on more "promising" regions of the parameters space in the light of its past experiences, it tries to select parameter combinations, which give better validation performance. As a consequence of this informed navigation and sampling, it reduces the search time and offers a better set of parameters, which can lead to better model performance [44].

All model hyperparameters are tuned using Bayesian optimization techniques, Sklearn [45] and Keras-tuner [51] Python libraries. Whereas the model fitting is performed on the training split of data, evaluation and optimal parameter selection are carried out on the validation split. After choosing the best hyperparameters in 200 trials for each model, performances of the models having optimal parameters are assessed on the test splits, and results are saved. Please refer to Table II for the hyperparameters, their space, and optimal values for each model and test system.

TABLE II
OPTIMIZED MODEL HYPERPARAMETERS

model	param	space	IEEE-14	IEEE-118	IEEE-300
BDD	threshold	{0.01, 0.02, …, 5.0}	1.05	2.37	2.62
DTC	criterion	{gini, entropy}	entropy	gini	gini
	depth	{8, 9,…, 64}	64	64	64
	features	{0.1, 0.2, …, 0.9}	0.3	0.4	0.5
	min. leaf	{1, 2, …, 8}	4	1	2
SVC	C	$10^{\{-6,-5,\ldots,2\}}$	10^2	10^2	10^1
	degree	{1, 2, …, 5}	2	2	-
	gamma	$10^{\{-6,-5,\ldots,2\}}$	10^{-1}	10^{-3}	10^{-3}
	kernel	{linear, poly, rbf}	poly	poly	rbf
MLP	layers	{1, 2, 3, 4}	4	3	3
	units	{8, 16, 32, 64}	16	16	64
	activation	{relu, elu, tanh}	elu	elu	elu
	optimizer	{adam, sgd, rmsprop}	rmsprop	adam	rmsprop
RNN	layers	{1, 2, 3, 4}	3	4	4
	units	{8, 16, 32, 64}	16	32	16
	activation	{relu, elu, tanh}	relu	relu	relu
	optimizer	{adam, sgd, rmsprop}	adam	adam	rmsprop
CNN	layers	{1, 2, 3, 4}	3	2	3
	units	{8, 16, 32, 64}	16	16	32
	K	{2, 3, 4, 5}	5	5	5
	activation	{relu, elu, tanh}	relu	relu	relu
	optimizer	{adam, sgd, rmsprop}	rmsprop	adam	adam
GNN	layers	{1, 2, 3, 4}	3	3	4
	units	{8, 16, 32, 64}	32	16	32
	K	{2, 3, 4, 5}	3	3	2
	activation	{relu, elu, tanh}	relu	relu	relu
	optimizer	{adam, sgd, rmsprop}	adam	adam	adam

V. CONCLUSION

In this article, we addressed the detection of stealth FDIA in modern ac power grids. To that end, we first developed a generic, locally applied, and stealth FDIA generation technique by solving a nonlinear nonconvex optimization problem using SGD algorithm and made available the labeled data to the research community. Second, we proposed a scalable and real-time detection mechanism for FDIAs by fusing the underlying graph topology of the power grid and spatially correlated measurement data in GNN layers. Finally, we tested our algorithms on standard test beds such as IEEE 14-, 118-, and 300-bus systems and demonstrated that the proposed GNN detector surpasses the currently available methods in literature by 3.14%, 4.25%, and 4.41% in $F1$ score, respectively.

REFERENCES

[1] K. R. Davis, K. L. Morrow, R. Bobba, and E. Heine, "Power flow cyber attacks and perturbation-based defense," in *Proc. IEEE 3rd Int. Conf. Smart Grid Commun.*, 2012, pp. 342–347.

[2] S. Sridhar, A. Hahn, and M. Govindarasu, "Cyber-physical system security for the electric power grid," *Proc. IEEE*, vol. 100, no. 1, pp. 210–224, Jan. 2012.

[3] G. B. Giannakis, V. Kekatos, N. Gatsis, S.-J. Kim, H. Zhu, and B. F. Wollenberg, "Monitoring and optimization for power grids: A signal processing perspective," *IEEE Signal Process. Mag.*, vol. 30, no. 5, pp. 107–128, Sep. 2013.

[4] A. Abur and A. Expósito, *Power System State Estimation: Theory and Implementation [Power Engineering (Willis)]*. Boca Raton, FL, USA: CRC Press, 2004.

[5] H. He and J. Yan, "Cyber-physical attacks and defences in the smart grid: A survey," *IET Cyber-Phys. Syst.: Theory Appl.*, vol. 1, no. 1, pp. 13–27, 2016.

[6] G. Liang, J. Zhao, F. Luo, S. R. Weller, and Z. Y. Dong, "A review of false data injection attacks against modern power systems," *IEEE Trans. Smart Grid*, vol. 8, no. 4, pp. 1630–1638, Jul. 2017.

[7] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, pp. 1–33, 2011.

[8] P.-Y. Chen, S. Yang, J. A. McCann, J. Lin, and X. Yang, "Detection of false data injection attacks in smart-grid systems," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 206–213, Feb. 2015.

[9] O. Kosut, L. Jia, R. J. Thomas, and L. Tong, "Malicious data attacks on the smart grid," *IEEE Trans. Smart Grid*, vol. 2, no. 4, pp. 645–658, Dec. 2011.

[10] Y. Huang, J. Tang, Y. Cheng, H. Li, K. A. Campbell, and Z. Han, "Real-time detection of false data injection in smart grid networks: An adaptive cusum method and analysis," *IEEE Syst. J.*, vol. 10, no. 2, pp. 532–543, Jun. 2016.

[11] E. Drayer and T. Routtenberg, "Detection of false data injection attacks in smart grids based on graph signal processing," *IEEE Syst. J.*, vol. 14, no. 2, pp. 1886–1896, Jun. 2020.

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

[13] M. N. Kurt, Y. Yılmaz, and X. Wang, "Real-time detection of hybrid and stealthy cyber-attacks in smart grid," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 2, pp. 498–513, Feb. 2019.

[14] M. G. Kallitsis, S. Bhattacharya, S. Stoev, and G. Michailidis, "Adaptive statistical detection of false data injection attacks in smart grids," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2016, pp. 826–830.

[15] M. N. Kurt, Y. Yılmaz, and X. Wang, "Distributed quickest detection of cyber-attacks in smart grid," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 8, pp. 2015–2030, Aug. 2018.

[16] J. Hao *et al.*, "An adaptive Markov strategy for defending smart grid false data injection from malicious attackers," *IEEE Trans. Smart Grid*, vol. 9, no. 4, pp. 2398–2408, Jul. 2018.

[17] M. Esmalifalak, L. Liu, N. Nguyen, R. Zheng, and Z. Han, "Detecting stealthy false data injection using machine learning in smart grid," *IEEE Syst. J.*, vol. 11, no. 3, pp. 1644–1652, Sep. 2017.

[18] E. M. Ferragut, J. Laska, M. M. Olama, and O. Ozmen, "Real-time cyber-physical false data attack detection in smart grids using neural networks," in *Proc. IEEE Int. Conf. Comput. Sci. Comput. Intell.*, 2017, pp. 1–6.

[19] D. Wang, X. Wang, Y. Zhang, and L. Jin, "Detection of power grid disturbances and cyber-attacks based on machine learning," *J. Inf. Secur. Appl.*, vol. 46, pp. 42–52, 2019.

[20] S. Binna, S. R. Kuppannagari, D. Engel, and V. K. Prasanna, "Subset level detection of false data injection attacks in smart grids," in *Proc. IEEE Conf. Technol. Sustainability*, 2018, pp. 1–7.

[21] K. Vimalkumar and N. Radhika, "A Big Data framework for intrusion detection in smart grids using apache spark," in *Proc. IEEE Int. Conf. Adv. Comput., Commun. Inform.*, 2017, pp. 198–204.

[22] E. Drayer and T. Routtenberg, "Detection of false data injection attacks in power systems with graph Fourier transform," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2018, pp. 890–894.

[23] R. Ramakrishna and A. Scaglione, "Detection of false data injection attack using graph signal processing for the power grid," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2019, pp. 1–5.

[24] R. Deng, G. Xiao, and R. Lu, "Defending against false data injection attacks on power system state estimation," *IEEE Trans. Ind. Inform.*, vol. 13, no. 1, pp. 198–207, Feb. 2017.

[25] L. Liu, M. Esmalifalak, Q. Ding, V. A. Emesih, and Z. Han, "Detecting false data injection attacks on power grid by sparse optimization," *IEEE Trans. Smart Grid*, vol. 5, no. 2, pp. 612–621, Mar. 2014.

[26] A. S. Musleh, G. Chen, and Z. Y. Dong, "A survey on the detection algorithms for false data injection attacks in smart grids," *IEEE Trans. Smart Grid*, vol. 11, no. 3, pp. 2218–2234, May 2020.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[28] J. Wang, C. Jiang, H. Zhang, Y. Ren, K.-C. Chen, and L. Hanzo, "Thirty years of machine learning: The road to pareto-optimal wireless networks," *IEEE Commun. Surv. Tut.*, vol. 22, no. 3, pp. 1472–1514, Jul.–Sep. 2020.

[29] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[30] D. Owerko, F. Gama, and A. Ribeiro, "Optimal power flow using graph neural networks," in *Proc. IEEE Int. Conf. Acoustics, Speech Signal Process.*, 2020, pp. 5930–5934.

[31] E. Handschin, F. C. Schweppe, J. Kohlas, and A. Fiechter, "Bad data analysis for power system state estimation," *IEEE Trans. Power App. Syst.*, vol. 94, no. 2, pp. 329–337, Mar. 1975.

[32] Q. Yang, J. Yang, W. Yu, D. An, N. Zhang, and W. Zhao, "On false data-injection attacks against power system state estimation: Modeling and countermeasures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 717–729, Mar. 2014.

[33] M. Esmalifalak, H. Nguyen, R. Zheng, and Z. Han, "Stealth false data injection using independent component analysis in smart grid," in *Proc. IEEE Int. Conf. Smart Grid Commun.*, 2011, pp. 244–248.

[34] M. A. Rahman and H. Mohsenian-Rad, "False data injection attacks with incomplete information against smart power grids," in *Proc. IEEE Global Commun. Conf.*, 2012, pp. 3153–3158.

[35] Z.-H. Yu and W.-L. Chin, "Blind false data injection attack using PCA approximation method in smart grid," *IEEE Trans. Smart Grid*, vol. 6, no. 3, pp. 1219–1226, May 2015.

[36] G. Hug and J. A. Giampapa, "Vulnerability assessment of ac state estimation with respect to false data injection cyber-attacks," *IEEE Trans. smart grid*, vol. 3, no. 3, pp. 1362–1370, Sep. 2012.

[37] P. Bojanowski, A. Joulin, D. Lopez-Pas, and A. Szlam, "Optimizing the latent space of generative networks," in *Proc. 35th Int. Conf. Mach. Learn., Proc. Mach. Learn. Res.*, 2018, vol. 80, pp. 600–609.

[38] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018.

[39] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, Barcelona, Spain, 2016, pp. 3844–3852.

[40] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*. Boca Raton, FL, USA: CRC Press, 2002.

[41] L. Thurner *et al.*, "Pandapower—An open source python tool for convenient modeling, analysis and optimization of electric power systems," *IEEE Trans. Power Syst.*, vol. 33, no. 6, pp. 510–6521, Nov. 2018.

[42] *The Electric Reliability Council of Texas (ERCOT). Backcasted (actual) load profiles—Historical*, May 10, 2020. [Online]. Available: http://www.ercot.com/mktinfo/loadprofile/alp/

[43] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.

[44] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.

[45] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[46] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 3, pp. 660–674, May/Jun. 1991.

[47] D. Lee and J. Lee, "Domain described support vector classifier for multi-classification problems," *Pattern Recognit.*, vol. 40, no. 1, pp. 41–51, 2007.

[48] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 683–697, Sep. 1992.

[49] P. Rodriguez, J. Wiles, and J. L. Elman, "A recurrent neural network that learns to count," *Connection Sci.*, vol. 11, no. 1, pp. 5–40, 1999.

[50] J. S. Denker *et al.*, "Neural network recognizer for hand-written zip code digits," in *Proc. Adv. Neural Inf. Process. Syst.*, 1989, pp. 323–331.

[51] T. O'Malley *et al.*, "Keras tuner," 2019. [Online]. Available: https://github.com/keras-team/keras-tuner

Osman Boyaci received the B.Sc. (Hons.) degree in electronics engineering and in computer engineering (double major) (Hons.) and the M.Sc. degree in computer engineering from Istanbul Technical University, Istanbul, Turkey, in 2013 and 2017. He is currently working toward the Ph.D. degree in computer engineering with Texas A&M University, College Station, TX, USA.

His research interests include machine learning, artificial intelligence, and cybersecurity.

Amarachi Umunnakwe (Student Member, IEEE) received the B.S. degree in electronic engineering from the University of Nigeria, Nsukka, Nigeria, in 2016, and the M.S. degree in electrical and computer engineering from the University of Utah, Salt Lake City, UT, USA, in 2020. She is currently working toward the Ph.D. degree in electrical and computer engineering with Texas A&M University, College Station, TX, USA.

Her research interests include cyber-physical resilience, situational awareness, and security of electric power systems using intelligent techniques.

Her research interests include cyber-physical resilience, situational awareness, and security of electric power systems using intelligent techniques.

Abhijeet Sahu received the B.S. degree in electronics and communications from the National Institute of Technology, Rourkela, India in 2011, and the M.S. degree in electrical and computer engineering in 2018 from Texas A&M University, College Station, TX, USA, where he is currently working toward the Ph.D. degree in cyberphysical resilient energy systems.

His research interests include network security, cyber-physical modeling for intrusion detection and response, and artificial intelligence for cyber-physical security in power systems.

Mohammad Rasoul Narimani (Member, IEEE) received the B.Sc. degree from Razi University, Kermanshah, Iran, in 2008, and the M.Sc. degree from the Shiraz University of Technology, Shiraz, Iran, in 2011, and the Ph.D. degree from the Missouri University of Science and Technology, Rolla, MO, USA, 2019, all in electrical engineering.

He is currently an Assistant Professor with the College of Engineering, Arkansas State University, Jonesboro, AR, USA. Before joining Arkansas State University, he was a Postdoc with Texas A&M University, College Station, TX, USA. His research focuses on the application of optimization techniques to electric power systems.

Muhammad Ismail (Senior Member, IEEE) received the B.Sc. (Hons.) and M.Sc. degrees in electrical engineering (electronics and communications) from Ain Shams University, Cairo, Egypt, in 2007 and 2009, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2013.

He is currently an Assistant Professor with the Department of Computer Science, Tennessee Technological University, Cookeville, TN, USA.

Dr. Ismail was a corecipient of the best paper awards in the IEEE International Conference on Communications (ICC) 2014, the IEEE Globecom 2014, the International Conference on Smart Grid and Renewable Energy 2015, the Green 2016, the Best Conference Paper Award from the IEEE Transactions on Green Communications and Networking at the IEEE ICC 2019, and IEEE IS 2020.

Katherine R. Davis (Senior Member, IEEE) received the B.S. degree in electrical engineering from the University of Texas at Austin (TAMU), Austin, TX, USA, in 2007, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2009 and 2011, respectively.

She is currently an Assistant Professor of Electrical and Computer Engineering, TAMU.

Erchin Serpedin (Fellow, IEEE) is currently a Professor with the Electrical and Computer Engineering Department, Texas A&M University in College Station, College Station, TX, USA. He is the author of four research monographs, one textbook, 17 book chapters, 170 journal papers, and 270 conference papers. His current research interests include signal processing, machine learning, artificial intelligence, cyber security, smart grids, and wireless communications.

Dr. Serpedin was an Associate Editor for more than 12 journals, including journals such as the IEEE Transactions on Information Theory, IEEE Transactions on Signal Processing, IEEE Transactions on Communications, IEEE Signal Processing Letters, IEEE Communications Letters, IEEE Transactions on Wireless Communications, IEEE Signal Processing Magazine, and *Signal Processing* (Elsevier), and as a Technical Chair for six major conferences.