# Comparing Machine Learning and Optimization Approaches for the $N - k$ Interdiction Problem

Alejandro D. Owen Aquino,[1] Alyssa Kody,[2] and Daniel K. Molzahn[1]

***Abstract*—In order to achieve reliable electric service, engineers must design power grids such a small number of line failures will not result in significant amounts of load shedding. To identify problematic failure combinations, the $N-k$ interdiction problem seeks the set of $k$ failed lines (out of the $N$ total lines in the system) that result in the largest amount of load shedding. This problem is formulated as a bilevel optimization where the upper-level selects the worst-case set line failures and the lower-level redispatches the system to minimize the load shedding. This problem becomes challenging due to the discrete variables associated with the failure selection in the upper-level problem and nonconvexities from the AC power flow equations in the lower-level problem. We compare the performance of (1) a state-of-the-art optimization approach based on a network flow relaxation of the power flow equations, (2) a newly developed reinforcement learning approach based on a deep-Q-network algorithm, and (3) a newly developed algorithm based on a neural network formulated as a mixed-integer linear program. We evaluate the effectiveness of each approach in finding the contingencies that produce the largest shedding. The results show the proposed approaches produce competitive results in identifying worst-case contingencies.**

***Index Terms*—Bilevel Optimization, Interdiction, Neural Networks, $N - k$.**

## I. Introduction

Identifying the sets of line failures that result in large amounts of load shedding is of key importance to power grid operators, with applications including protection against natural disasters and cyberattacks as well as scheduling day-to-day component maintenance. The optimization problem used to identify critical sets of line failures is typically formulated as an adversarial bilevel problem called the $N-k$ interdiction problem. Modeling the attacker, the upper level of this problem disconnects up to $k$ lines out of the $N$ lines in the system in order to maximize the total load shed. The lower level representing the defender optimally redispatches the system to maximize the load delivery given the disconnected lines.

When using an AC power flow model to accurately represent the network, this problem is a nonconvex bilevel mixed-integer nonlinear program that is extremely challenging to solve. Nevertheless, there have been several approaches for addressing this problem. While some approaches focus on preserving the nonlinearity of the problem [1]–[4], most formulations rely on linearizing the lower-level problem [5]–[10]. In essence, solving this problem involves a tradeoff between accuracy and computational tractability.

The existing approaches in [1]–[10] primarily apply traditional optimization tools to the $N - k$ interdiction problem. Emerging machine learning (ML) techniques provide new avenues for addressing several problems in power systems. The use of reinforcement learning for power systems operation and control, for example, has attracted substantial attention recently [11]–[14]. Other works such as [15], [16] apply neural networks (NN) formulated as mixed-integer linear programs (MILP) to the optimal power flow (OPF) problem. In the area of power system reliability specifically, works such as [17] use Q-learning to identify the worst-impacted zones of power grids during extreme events, and others such as [18]–[20] use NNs to rank the severity of contingencies.

To build on this existing work, we propose and benchmark two new ML approaches for identifying worse-case failures in power systems: 1) a reinforcement learning (RL) approach based on a deep Q-network (DQN) algorithm [21], and 2) a NN formulated as a MILP [22]–[24]. Both of these approaches predict the worst-case failure combinations by using NNs to generalize from the load shedding observed in sampled failure scenarios. We compare the performance of these new ML approaches to the state-of-the-art linearized bilevel optimization approach in [10]. The results show the competitiveness of the ML approaches versus the state-of-the-art approach in [10].

The rest of the paper is organized as follows. First, we formulate the $N-k$ interdiction problem in Section II. Second, we present the two new ML approaches proposed in this paper as well as the state-of-the-art linearized bilevel problem which provides a baseline for comparison in Section III. Third, we report the results of those approaches when applied to several test cases in Section IV. Lastly, we conclude the paper and discuss directions for future research in Section V.

## II. The $N - k$ Interdiction Problem

This section describes the bilevel optimization formulation of the $N - k$ interdiction problem. Modeling the attacker, the upper-level problem disrupts the system by choosing binary variables representing line failures. Taking these line failures as inputs, the lower level is an AC maximum load delivery (AC-MLD) problem that mitigates the disruption by redispatching the system. With a nonconvex feasible region due to the AC power flow equations, the lower-level problem by itself is NP-Hard [25], compounding the challenges associated with the bilevel and discrete nature of the overall problem. Thus, approximation, relaxation, and/or decomposition approaches are needed for tractability, as described in Section III.

We adapt the lower-level formulation from the AC-MLD problem in [26]. Let $\mathcal{N}$ be the set of buses, $\mathcal{G}$ the set of generators, and $\mathcal{E}$ the set of lines in the system. Each bus $k \in \mathcal{N}$ has a complex voltage phasor $V_k$, a shunt admittance

[1]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. {aaquino30, molzahn}@gatech.edu
[2]Energy Systems Division, Argonne National Laboratory, Argonne, IL, USA. akody@anl.gov

$Y_k^S$, a complex power demand $S_k^D$ (with non-negative real power), and at most one generator with a complex power output of $S_{g_k}^G$, where $g_k$ denotes the generator at bus $k$. Each line $(i,j) \in \mathcal{E}$ has a series admittance $Y_{ij}$ and a shunt admittance $Y_{ij}^c$ as well as a (possibly complex) turns ratio $T_{ij}$ in order to permit models of transformers. Each line also has a maximum apparent power flow limit denoted as $S_{ij}^u$.

Binary variables $x_{ij}$, $z_k^V$, and $z_g^G$ denote the statuses of the lines $(i,j) \in \mathcal{E}$, the buses $k \in \mathcal{N}$, and the generators $g \in \mathcal{G}$, respectively. For each of these binary variables, a value of 1 indicates that the component is connected to the system and a value of 0 indicates that the component is disconnected. For each bus $k \in \mathcal{N}$, the continuous variables $z_k^D$ and $z_k^S$ denote the fractions of the satisfied load demand and the shunt admittance connected to the system, respectively, considering a constant power factor. Using these variables, we model the disruption to the system by considering a weighted sum of the satisfied load demand and the connection statuses of the buses, generators, and shunts:

$$\sum_{k \in \mathcal{N}} \left( \Re(S_k^D) z_k^D + C^S z_k^S + C^V z_k^V \right) + \sum_{g \in \mathcal{G}} C^G z_k^G, \quad (1)$$

where $C^V$, $C^S$, and $C^G$ are weighting parameters and $\Re(\,\cdot\,)$ takes the real part of a complex number. When chosen as in [26], these weighting parameters result in the AC-MLD problem attempting to keep the components at their pre-contingency statuses while treating load shedding as the primary control mechanism. This formulation thus results in (1) essentially modeling the total load served in the system while providing enough flexibility in the generator, bus, and shunt statuses to prevent infeasibility.

With $x_{ij}$ as decision variables, the upper-level problem maximizes the disruption to the system by minimizing (1) subject to a cardinality constraint on the number of disconnected lines:

$$\sum_{(i,j) \in \mathcal{E}} x_{ij} \leq N - \kappa, \quad (2)$$

where $\kappa$ is a specified parameter indicating the maximum number of line failures and $N$ is the total number of lines.

The lower-level problem redispatches the system to minimize the disruption by maximizing (1) subject to both the AC power flow equations and operational limits. The AC power flow equations ensure power balance at each bus $k \in \mathcal{N}$:

$$S_{g_k}^G - z_k^D S_k^D - z_k^S \left( Y_k^S \right)^* |V_k|^2 = \sum_{(i,j) \in \mathcal{E}_k} x_{ij} S_{ij}, \quad (3)$$

where $\mathcal{E}_k$ is the set of lines connected to bus $k$ and $(\,\cdot\,)^*$ denotes the complex conjugate. Constraints modeling the power flowing on each line $(i,j) \in \mathcal{E}$, $S_{ij}$, are:

$$M(1 - x_{ij}) \leq S_{ij} - (Y_{ij} - Y_{ij}^c)^* \frac{|V_i|^2}{|T_{ij}|^2} + Y_{ij}^* \frac{V_i V_j^*}{T_{ij}} \leq (1 - x_{ij}) M, \quad (4a)$$

$$M(1 - x_{ij}) \leq S_{ij} - (Y_{ij} - Y_{ji}^c)^* |V_j|^2 + Y_{ij}^* \frac{V_j V_i^*}{T_{ij}} \leq (1 - x_{ij}) M, \quad (4b)$$

where $M$ is a "Big-M" constant used in modeling the line status and complex inequalities are interpreted as separate constraints on the real and imaginary parts. Notice that (4) is non-binding when $x_{ij} = 0$ and is effectively an equality constraint modeling Ohm's law when $x_{ij} = 1$. For each line $(i,j) \in \mathcal{E}$, bus $k \in \mathcal{N}$, and generator $g \in \mathcal{G}$, limits on voltage magnitudes, generator outputs, and line flows are:

$$|S_{ij}| \leq S_{ij}^u x_{ij}, \quad (5a)$$

$$z_k^V v_k^l \leq |V_k| \leq z_k^V v_k^u, \quad (5b)$$

$$z_g^G S_g^{Gl} \leq S_g^G \leq z_g^G S_g^{Gu}, \quad (5c)$$

where (5a) is enforced for the apparent power flowing into both terminals of the line, $v_k^l$ and $v_k^u$ are lower and upper bounds on the voltage magnitudes, and $S_k^{Gl}$ and $S_k^{Gu}$ are lower and upper bounds on the generator's complex power outputs.

Using these definitions, the $N - k$ interdiction problem is:

$$\min_{x_{ij} \in \mathcal{E}} \quad \zeta \quad (6a)$$

$$\text{subject to } (\forall (i,j) \in \mathcal{E}):$$

$$(2), \; x_{ij} \in \{1, 0\}, \quad (6b)$$

$$\zeta = \max_{z_k^V, z_g^G, z_k^D, z_k^S} (1) \quad (6c)$$

$$\text{subject to } (\forall k \in \mathcal{N}, \; \forall g \in \mathcal{G}, \; \forall (i,j) \in \mathcal{E}):$$

$$(3)\text{–}(5), z_k^V, z_g^G, \in \{1, 0\}, \; z_k^D, z_k^S \in (1, 0). \quad (6d)$$

Due to the challenges inherent to this formulation, many solution approaches simplify the problem to yield a structure suited for a single-level reformulation. The network flow model used in [10] linearly relaxes the power flow equations in the lower-level problem. Conversely, the two new approaches presented in the next section of this paper train NNs with sampled solutions to the nonlinear lower-level problem (6c)–(6d) to preserve some of the nonlinear behavior.

## III. Solution Approaches

This section presents three approaches for solving the $N - k$ interdiction problem (6): a state-of-the-art bilevel programming approach that uses a network flow relaxation of the power flow equations [10], a new RL approach, and a new approach that uses a NN reformulated as a MILP.

### *A. Simplified Bilevel Formulation*

As a benchmark, we consider the recently proposed approach in [10] that solves the $N - k$ interdiction problem using a network flow relaxation of the power flow equations [27] in the lower-level problem. While the upper-level problem is similar to that of model (6), the lower-level problem is much simpler in that it (i) is linear, (ii) cannot remove other network components, and (iii) focuses only on minimizing the load shed. The formulation in [10] is:

$$\min_{x_{ij} \in \mathcal{E}} \zeta \quad (7a)$$

$$\text{subject to } (\forall (i,j) \in \mathcal{E}):$$

$$(2), \; x_{ij} \in \{1, 0\}, \quad (7b)$$

$$\zeta = \max_{z^D} \sum_{k \in \mathcal{N}} P_k^D z_k^D \quad (7c)$$

$$\text{subject to } (\forall k \in \mathcal{N}, \; \forall g \in \mathcal{G}, \; \forall (i,j) \in \mathcal{E}):$$

$$P_{g_k}^G - P_k^D z_k^D = \sum_{(i,j)\in\mathcal{E}_k} P_{ij}, \tag{7d}$$

$$0 \le P_g^G \le P_g^{Gu}, \tag{7e}$$

$$-P_{ij}^u x_{ij} \le P_{ij} \le P_{ij}^u x_{ij}, \tag{7f}$$

$$z_k^D \in (0,1). \tag{7g}$$

The lower-level objective (7c) maximizes the active power demand $P_k^D$ served at each bus $k \in \mathcal{N}$ where the continuous variable $z_k^D$ denotes the fraction of the satisfied load demand connected to the system. Equation (7d) enforces power balance at each bus $k \in \mathcal{N}$ where $P_{g_k}^G$ denotes the active power generated at that bus and $P_{ij}$ denotes the active power flow in each line $(i,j) \in \mathcal{E}_k$. Operational constraints (7e) and (7f) enforce limits on generators $g \in \mathcal{G}$ and thermal limits on lines $(i,j) \in \mathcal{E}$, respectively, where $P_g^{Gu}$ is the upper bound on the generator's active power output and $P_{ij}^u$ is the upper bound on the line's active power flow. Notice that there are no line flow equations in this problem, i.e., no equivalent of (4a) and (4b), and thus this problem is a relaxation of model (6) [27].

To solve model (7), the lower-level problem is replaced with its Karush–Kuhn–Tucker (KKT) optimality conditions [28], which are added as constraints to the upper-level problem [29]. This yields a single-level MILP that can be solved with commercial solvers. The work in [10], specifically, gives an explicit reformulation of model (7) that will be used for the rest of this paper for solving the bilevel network flow approach. Along with other techniques for improving the tractability of the formulation, the authors of [10] show that known bounds on the dual variables are key for achieving a tractable implementation for this problem.

Once the bilevel network flow formulation problem is solved, its maximizer represents the contingency that produced the maximum load shed. Later, however, we want to be able to find not only the worst contingency, but also multiple other contingencies that generate large amounts of load shedding. To do this, we can solve the problem in a loop that iteratively adds constraints which prevent the solver from finding previously identified failure combinations.

## B. Reinforcement Learning via a Deep Q-Network (DQN)

Another approach for finding sets of worst-case contingencies is to train a policy using RL to select a set of line failures that result in the largest amount of load shed. Here, we apply a Deep Q-Network algorithm [30]. This strategy replaces the bilevel attacker-defender construction with an agent-environment paradigm.

DQN uses a convolutional NN that learns how good an action is at a particular state with the objective of maximizing the total rewards received in the course of an episode. We next describe the components of our RL approach for finding sets of worst-case contingencies.

*1) Episode:* Each episode consists of $k$ steps equal to the maximum number of permitted component failures. At the beginning of an episode, all the lines are in service. During each step, the agent selects an action $\alpha$ (line to disconnect from the system) and the environment calculates the reward $\rho$ and the new state $X'$, as described next.

*2) State:* The state of the network is given by the status of each of the lines in the system. A line status of 1 represents an in-service line, while a line status of 0 represents a line that has been disconnected. All the line statuses are collected in a vector $X$ with the same length as the number of lines.

*3) Agent:* We constructed a NN, known as the *policy network* $\pi$, that acts as our agent. The agent's input is the state vector $X$ and its output is a vector $A$ containing total expected rewards associated with each available action taken at that state. The first layer of our policy network $\pi$ is a dense layer with the same size as the number of lines in the system. We use second, third, and fourth layers of the network that are are fully connected and each have 16 neurons modeled with rectified linear unit (ReLU) activation functions. The final output layer is a fully connected linear layer that returns the resulting vector $A$.

*4) The Q Values:* The agent's objective is to cause the largest load shed, not only as the result of one action, but in the entire episode. This needs to be reflected in the outputs of vector $A$, also known as the Q-values. Hence, a Q-value is the representation of the rewards the agent will receive by taking an action at a given state plus the (discounted) future rewards it will receive by taking the optimal future actions:

$$Q_\pi(X_t, \alpha_t) = \mathbb{E}(\rho_t + \gamma\rho_{t+1} + \gamma^2\rho_{t+2} + \ldots), \tag{8}$$

where $\mathbb{E}(\,\cdot\,)$ denotes the expectation. The parameter $\gamma$, known as the discount factor, reflects how much weight the agent assigns to expected future rewards.

*5) Action:* Usually, we want the agent to choose an action based on the output Q-values from a given input state:

$$\alpha = \text{argmax}(\pi(X)) = \text{argmax}(A). \tag{9}$$

The largest Q-value indicates to the agent that the best action is to switch the status of the line associated with that Q-value. However, we also want the agent to occasionally take random actions to accumulate new experiences. By performing random actions, the agent might find a better solution. The former is known as *exploitation*, while the latter is known as *exploration*. This balance of exploration vs. exploitation is achieved with an $\epsilon$-greedy strategy [31]. Following this method, the probability of exploration is given by a parameter $\epsilon$ which starts as 1 and decreases with each episode. The probability of exploitation, on the other hand, is given by $1-\epsilon$. The algorithm continues until it reaches a predetermined number of $\tau$ explorations.

*6) Reward:* After taking an action, the agent will receive a reward $\rho$ from the environment. Since the objective of the problem is to find the line failure combinations that result in the largest amount of load shed, we chose to reward the agent with the load shed caused by each action that it takes. For instance, if disconnecting a line results in an additional 2 MW of load shed during a particular step, the agent will receive 2 units of reward at that step. If an action taken by the agent increases the amount of load served, the agent can receive negative rewards.

*7) Environment:* The environment calculates the reward $\rho$ based on the $(X,\alpha)$ pair given by the agent. Since $\rho$ is equal to the load shed, we use the lower-level problem of model (6) as our environment, i.e., (6c)–(6d). The minimal load shed is given by the solution to this AC-MLD formulation.

The process of training the agent occurs between episodes. During this time, we calculate the mean-squared-error (MSE) losses between a particular set of Q-values and their target, which provides a more accurate estimate of these Q-values based on previous experiences. This loss is used to update the parameters of the policy network $\pi$ via stochastic gradient descent. The NN changes in response to this calculated loss depend on learning rate $\eta$. This hyperparameter decays over time, which gives the NN more adaptability at first and more gradual learning as the episodes proceed. While training the policy network, we also adopt two other common strategies: experience replay and a target network $\pi^t$. These strategies, proposed in [30], greatly improve algorithmic stability.

*8) Experience replay:* During each step, the algorithm records the most recent experience $(X, \alpha, \rho, X')$ in an "experience buffer" $\Omega$. This buffer has a capacity $10\tau$; when full, the newest experiences will replace the oldest ones. A batch of $\psi$ random experiences sampled from $\Omega$ are used to train the agent. Using these random experiences instead of training the agent using only the latest experience breaks harmful correlations and allows the agent to learn from an individual experience multiple times and recall rare occurrences.

*9) Target network:* During the DQN training process, we use a guess (the best estimate of the expected sum of all future rewards) to update a guess (the current Q-values). This procedure often results in instability if the same network is used to calculate both guesses since updating the NN changes both sides of the update equation. The target network $\pi^t$ helps solve this problem by stabilizing the learning. Here, $\pi^t$ is a copy of the policy network that is held fixed for $\Xi$ steps to serve as a stable estimate of the future rewards.

The size and structure of the NN as well as the parameters $\psi$, $\tau$, $\eta$, $\eta$ decay rate, $\epsilon$ decay rate, $\Xi$, and $\gamma$ are design choices that are chosen to balance the algorithm's speed and accuracy. The resulting approach is summarized as Algorithm 1, where $e$ denotes the total number of explorations taken and $\xi$ denotes the number of steps executed since the last $\pi^t$ update .

**Algorithm 1** $N-k$ Interdiction with DQN

```
1: Initialize: $\pi$, $\pi^t$, $\Omega$, $X$, $\epsilon$, $e$, $\xi$, $\eta$
2: while $e < \tau$ do
3:     for $i = 1, 2, \ldots, k$ do
4:         $r$ = random number between 0 and 1
5:         if $r \leq \epsilon$ then
6:             Explore: Choose random $\alpha$, get $\rho$ and $X'$
7:             $e = e + 1$
8:         else
9:             Exploit: Choose best $\alpha$, get $\rho$ and $X'$
10:        end if
11:        $\xi = \xi + 1$
12:        Decay $\epsilon$
13:        Save $(X, \alpha, \rho, X')$ to $\Omega$
14:        if $\xi \geq \Xi$ then
15:            $\pi^t = \pi$
16:            $\xi = 0$
17:        end if
18:        Sample $\psi$ samples from $\Omega$
19:        Calculate loss
20:        Train $\pi$
21:        Decay $\eta$
22:    end for
23: end while
24: Return $\pi$
```

**Algorithm 2** $N-k$ Interdiction with NN-MILP

```
1: Sample $\tau$ random failure combinations. Add to $\Omega$
2: Calculate the load shed of each random failure combination by solving AC-MLD (6c)–(6d)
3: for $i = 1, 2, \ldots, I$ do
4:     for $j = 1, 2, \ldots, J$ do
5:         Train $\omega$ with samples from $\Omega$.
6:     end for
7:     Build model (10)
8:     Exclude top 5% of samples from $\Omega$ from solution $X^*$
9:     for $c = 1, 2, \ldots C$ do
10:        Solve model (10). Get the solution $X^*$
11:        Exclude $X^*$ from being a solution again
12:        if $X^* \notin$ replay buffer then
13:            Add $X^*$ to $\Omega$
14:            Calculate the load shed of $X^*$ using AC-MLD
15:        end if
16:    end for
17:    Return $\omega$
18: end for
```

### *C. Neural Network Formulated with a Mixed-Integer Linear Program (NN-MILP)*

We next propose propose a "NN-MILP" approach for solving the $N-k$ interdiction problem. To summarize, this approach uses a feed-forward neural network $\omega$ that maps line failure combinations to the corresponding amount of load shed. The NN $\omega$ is first trained by sampling $\tau$ random failures and computing their resulting load sheds by solving the lower-level problem (6c)–(6d). Choosing a piecewise linear activation functions for $\omega$ permits reformulation of this NN as a system of MILP constraints [22]–[24]. By pairing these MILP constraints with a load-shed-maximizing objective, we obtain new failure combinations $X^*$ that the NN $\omega$ predicts will result in large amounts of load shed. As described in Algorithm 2, this process is repeated in a loop where $\omega$ learns from the new failure combinations sampled from the MILP.

The algorithm begins by sampling $\tau$ random failure combinations and computing their load sheds using the AC-MLD model (6c)–(6d). This information is stored in a replay buffer $\Omega$ and used for the initial training of the NN. ThisN-1emN $\omega$ consists of an input layer that takes in the status of each line in the system, two fully connected "hidden" layers with 10 neurons and saturating linear activation functions, and a final layer with a one node outputting the estimated load shed.

After the initial training, the MILP in model (10) is built based on the weights and biases of the NN. This method was proposed in [22]–[24] and previously applied in the context of power systems in [15], [16]. Here, $H$ denotes the total number of hidden layers, and $\mathcal{L}^{[m]}$ denotes the set of neurons in a particular layer $m \in (1, \ldots, H+1)$. The objective function (10a) maximizes the output of the single neuron of the last layer $O_1^{[H+2]}$ (i.e., the load shed) which is defined in (10l). The output of first layer $O^{[1]}$, defined in (10b), is the same as the state vector $X$. Each entry in $X$ contains a binary variable $x_{ij}$ with the status of each line $(i,j) \in \mathcal{E}$ in the system. The input of each hidden layer neuron $I_l^{[m]}$ is formulated in (10d) using a set of weights $W_{nl}^{[m]}$ and biases $b_{nl}^{[m]}$, where the superscript denotes the layer index $m$ and the subscript denotes the flow of information from a particular neuron $n$ in layer $m-1$ to a particular neuron $l$ in layer $m$. The output of each hidden layer neuron $O_l^{[m]}$ is formulated using equations (10e)–(10g). These equations use the binary variables $\phi_{l1}^{[m]}$, $\phi_{l2}^{[m]}$, and $\phi_{l3}^{[m]}$ for each neuron $l$ to formulate the linear saturation functions that bound the minimum and maximum outputs of the neurons to be between 0 and 1. Each of these binary variables denotes one section of the piecewise linear activation function whose logic is given in (10h)–(10j). Equations (10f) and (10h)–(10i) use "Big-M" constants $M_1$ and $M_2$, respectively, to model the activation functions.

$$\max_{X} \quad O_1^{[H+2]} \tag{10a}$$

subject to $(\forall l \in \mathcal{L}^{[m]},\ \forall m \in 2,\ldots,H+1)$ :

$$O^{[1]} = X \tag{10b}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in \mathcal{E} \tag{10c}$$

$$I_l^{[m]} = \sum_{n=1}^{L^{[m-1]}} O_n^{[m-1]} W_{nl}^{[m]} + b_{nl}^{[m]} \tag{10d}$$

$$-(1-\phi_{l1}^{[m]}) \le O_l^{[m]} \le 1 - \phi_{l1}^{[m]} \tag{10e}$$

$$I_l^{[m]} - M_1(1-\phi_{l2}^{m}) \le O_l^{m} \le I_l^{[m]} + M_1(1-\phi_{l2}^{m}) \tag{10f}$$

$$\phi_{l3}^{[m]} \le O_l^{[m]} \le 2 - \phi_{l3}^{[m]} \tag{10g}$$

$$-I_l^{[m]} \le M_2 \phi_{l1}^{[m]} \tag{10h}$$

$$I_l^{[m]} - 1 \le M_2 \phi_{l3}^{[m]} \tag{10i}$$

$$\phi_{l1}^{[m]} + \phi_{l2}^{[m]} + \phi_{l3}^{[m]} = 1 \tag{10j}$$

$$\phi_{l1}^{[m]}, \phi_{l2}^{[m]}, \phi_{l3}^{[m]} \in \{0,1\} \tag{10k}$$

$$O_1^{[H+2]} = \sum_{n=1}^{L^{[H+1]}} O_1^{[H+1]} W_{n1}^{[H+2]} + b_{n1}^{[H+2]} \tag{10l}$$

Each maximizer obtained from solving model (10) gives a new failure combination that will be stores in $\Omega$ used in the next round of NN training. Every time the MILP is built, new constraints are added to prevent the optimization problem from finding the same results from previous iterations, which helps the NN find and learn from different high-load-shed combinations. As with the DQN algorithm, the size and structure of the NN, the choice of activation functions, and the number of iterations at each each step are design choices that were selected based on empirical experiments. Fig. 1 shows the components of the NN that are modeled in this MILP.

## IV. Experiments and Results

This section benchmarks the approaches described in Section III for selected test cases from the PGLib-OPF

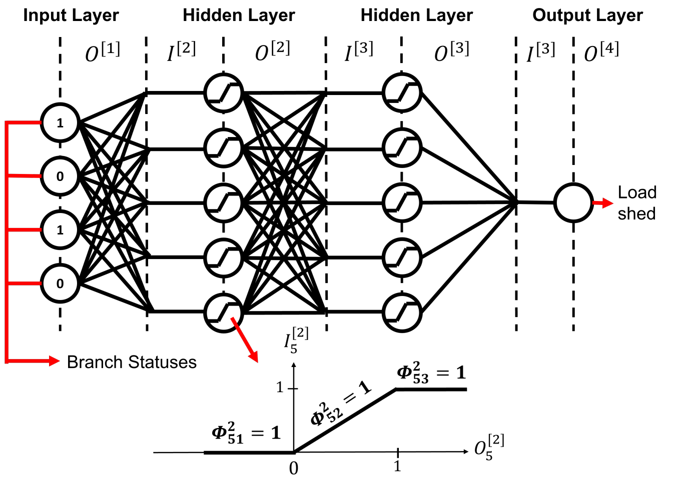

Fig. 1. Illustration of a NN with all the components that are modeled as a MILP in model (10).

archive [32]. MATLAB R2019b, YALMIP [33], and MATLAB's Machine Learning Toolbox were used for the implementation of the bilevel network flow formulation and the NN-MILP approaches on a computer with a quad-core 1.8 GHz processor and 16 GB of RAM. Julia v1.6.1 together with PowerModels.jl [34], PowerModelsRestoration.jl [35], and Flux.jl [36] were used for the implementation of the DQN algorithm and the AC-MLD formulation on Georgia Tech's PACE cluster, where each node had a quad-core 2.7 GHz processor and 9 GB of RAM. All non-linear programs (NLP) were solved using Ipopt [37], while all MILP problems were solved using Gurobi 9.0.

Since the DQN and NN-MILP approaches sample random contingencies (NN-MILP by direct sampling and DQN by exploring), we give both algorithms a similar sample complexity to ensure a fair comparison. We give the NN-MILP a sample size equal to 35% of the total sample space, i.e.:

$$\tau = 0.35\left(\frac{N!}{(k!(N-k)!)}\right). \tag{11}$$

Similarly, the DQN algorithm is set to stop training once it has explored $\tau$ unique combinations.

In the following subsections, we compare the bilevel network flow optimization problem, the DQN algorithm, and the NN-MILP algorithm with respect to 1) their ability to correctly determine the single failure combination that results in the largest load shed and 2) what percentage of the top-100 load shedding combinations they include in their own top 100. We also compare the neural networks $\pi$ and $\omega$ to see how their outputs correlate to one another and to the actual load shed given by the AC-MLD formulation.

### A. Finding the Worst Contingency

Table I summarizes the worst contingencies found by each of the three approaches considered in this paper. In the case where $k = 2$, the bilevel network flow formulation had the best results, finding the worst contingencies for all but two test cases (case73_ieee_rts and case118_ieee). As we increased $k$, however, this approach was less accurate in correctly identifying the worst contingencies. For $k = 3$, the bilevel approach correctly identified the worst-case failure combination for one of the four test cases considered, while for $k = 4$ it was not able to identify the worst-case failure combination for the considered test case.

Both machine learning approaches were overall less effective than the bilevel network flow formulation in finding the worst contingencies. Each approach was only able correctly determine the worst failure combinations in 25% of the experiments. Similar to the bilevel network flow approach, none of found the worst contingency with $k = 4$.

TABLE I
WORST FAILURE COMBINATIONS RESULTING FROM EACH APPROACH

| | k | Actual | Bilevel | DQN | NN-MILP |
|---|---|---|---|---|---|
| case14_ieee | k=2 | [1,2] | **[1,2]** | **[1,2]** | [1,6] |
| | k=3 | [1,2,3] | [1,2,4] | **[1,2,3]** | [1,2,8] |
| | k=4 | [3,4,7,10] | [2,3,4,5] | [1,2,3,10] | [1,2,3,10] |
| case24_ieee_rts | k=2 | [19,23] | **[19,23]** | [10,19] | [10,11] |
| | k=3 | [10,19,23] | [29,36,37] | [7,10,11] | **[10,19,23]** |
| case30_ieee | k=2 | [1,2] | **[1,2]** | **[1,2]** | **[1,2]** |
| | k=3 | [1,2,5] | [1,2,16] | [1,2,4] | [1,4,10] |
| case39_epri | k=2 | [5,46] | **[5,46]** | [14,20] | [14,20] |
| | k=3 | [5,20,46] | **[5,20,46]** | [20,33,37] | **[5,20,46]** |
| case57_ieee | k=2 | [8,22] | **[8,22]** | [41,50] | [41,49] |
| case73_ieee_rts | k=2 | [10,51] | [20,25] | [51,89] | [10,74] |
| case118_ieee | k=2 | [7,38] | [177,183] | [8,183] | [8,183] |

## *B. Identifying the Top-100 Worst Contingencies*

Another way to compare these different approaches is to see how many high-quality solutions they are able to identify. Even if an approach is able to find the absolute worst contingency, it is not clear that it will be also able to find other failure combinations that result in high load shedding. The opposite is also true: if one of these approaches is not able to find the worst contingency it might still be able to find several failure combinations that produce a similar amount of load shedding.

To test this, we obtained the list of the top-100 failure combinations given by each approach and compared them to the actual 100 worst contingencies from model (6) using brute force evaluation of all possibilities. If a particular failure combination in the list of one of these approaches was also found in the list of the actual worst-case contingencies, we say that that particular combination was correctly identified, even if that failure combination was not in the same rank. Table II summarizes these results.

While the bilevel network flow formulation generally performed the best in the task of finding the very worst contingencies, this did not always translate to finding many other high-quality solutions. This can be seen in case24_ieee_rts and case57_ieee with $k = 2$, where the bilevel approach found the worst contingencies but only found 15% and 11%, respectively, of the other combinations in the top 100. The bilevel approach's effectiveness in this particular task ranged from 4% to 88%, with an average of 51% and a standard deviation of 27%. Both machine learning approaches were more consistent in this regard. The DQN ranged from 25% to 87% of correctly identified combinations, with an average of 56% and a standard deviation of 17%. The NN-MILP had the best performance in this task, with a range of 21% to 91%, an average of 67%, and a standard deviation of 22%.

One last thing to consider is the case when there are several failure combinations that produce the same amount of load

TABLE II
PERCENTAGE OF CORRECT PREDICTIONS IN TOP-100 COMBINATIONS

| | k | Bilevel | DQN | NN-MILP |
|---|---|---|---|---|
| case14_ieee | k=2 | **60** | 52 | 50 |
| | k=3 | 65 | 62 | **69** |
| | k=4 | 71 | **76** | 62 |
| case24_ieee_rts | k=2 | 15 | **46** | 43 |
| | k=3 | 47 | 43 | **84** |
| case30_ieee | k=2 | 81 | 81 | **89** |
| | k=3 | 88 | 59 | **91** |
| case39_epri | k=2 | 52 | 57 | **72** |
| | k=3 | 70 | 46 | **90** |
| case57_ieee | k=2 | 11 | **87** | 85 |
| case73_ieee_rts | k=2 | 4 | **25** | 21 |
| case118_ieee | k=2 | 47 | 43 | **49** |
| Average | | 51 | 56 | **67** |
| Std. Dev. | | 27 | **17** | 22 |

shed that could place them in the top-100 contingencies. For example, if the 100th worst contingency produces the same or nearly the same amount of load shed as the 101st, but one of the algorithms only includes the latter in their top 100, it would look like the algorithm is less effective than it actually is. To avoid this issue and to see how close the top solutions from the different approaches are to the actual top contingencies, we compute running sums of the percent load shed of each top-100 list, the results of which are shown in Fig. 2. In each experiment, all the lines are measured against the blue line which represents the running sum of the true worst-case load sheds. The blue line is therefore the upper bound in each experiment. If another line closely follows the blue line it means that the corresponding approach was able to find contingencies that are very close to (or exactly) the ones that create the most load shed. If a line is separated considerably from the blue line, that particular approach failed to find many high quality solutions to the problem.

For example, that the worst performance for the DQN algorithm does not happen in case73_ieee_rts as Table II would suggest. The largest separation between the yellow DQN line and the blue line representing the actual worst contingencies actually happens in case30_ieee with $k = 3$. As explained above, this occurs because case73_ieee_rts has several failure combinations that produce the same amount of load shed. Conversely, this figure confirms that the bilevel network flow formulation performed very poorly for case24_ieee_rts and case57_ieee with $k = 2$ and for case73_ieee_rts. Similarly, we see that the worst performance for the NN-MILP algorithm does indeed happen for case73_ieee_rts.

## *C. Comparison of Neural Networks*

As explained in Section III, both the DQN and NN-MILP approaches use neural networks to learn the worst contingencies for a particular system, although they do so using very different techniques. While the NN-MILP approach tries to accurately map failure combinations to their load sheds, the DQN algorithm tries to figure out what consecutive set of lines to disconnect in order to obtain the most reward in an episode. Their resulting NNs ($\omega$ and $\pi$, respectively), however, are similar to one another.

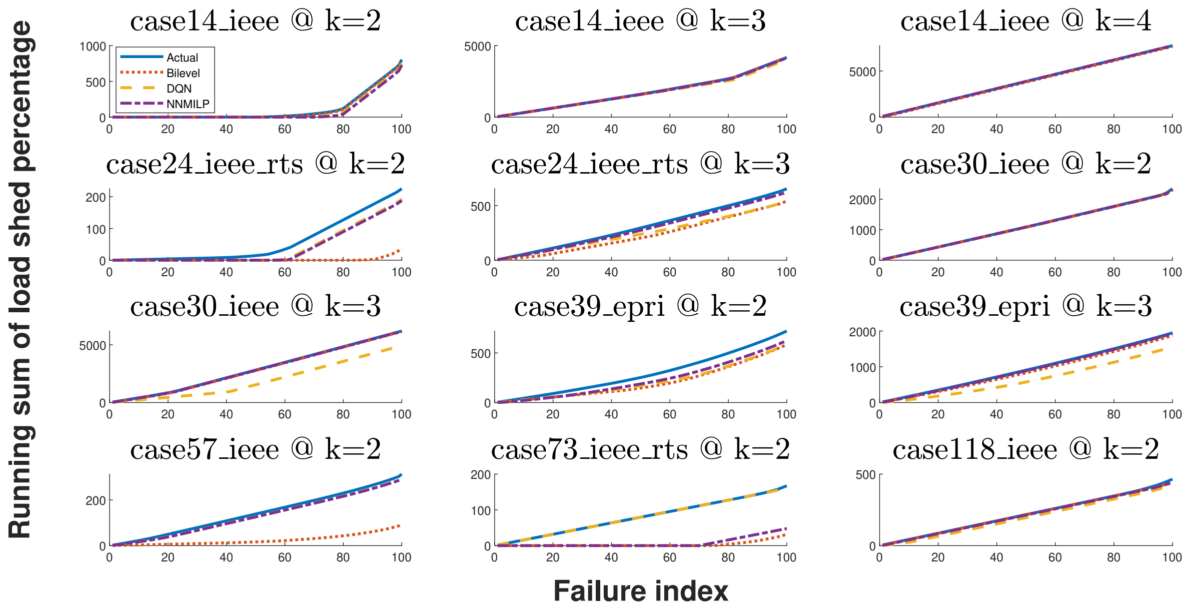

Fig. 2. Running sums of top-100 contingencies' load sheds by approach.

Fig. 3 compares the outputs of each of the NNs when given the same failure combinations as their inputs. In this case, the inputs were $\tau$ randomly generated contingencies. Each random contingency was given to $\omega$ to predict the percent load shed for that combination and to $\pi$ to get the total rewards resulting from that episode. When plotting these predicted load sheds and rewards on their respective axes, we see the high correlation between them in most experiments. (Given their different units, we do not expect to see the same *values*; however, similar *patterns* in the outputs often appear in Fig. 3.) Furthermore, we see the high correlation between both approaches and the AC-MLD load shed shown in the blue line. This shows not only the ability of both NNs to predict the load shed of a given failure, but also how they learn and preserve some of the nonlinearity in the optimization model (6).

## V. Conclusion and Future Work.

Identifying small sets of component failures that result in substantial amounts of load shedding is a key problem in power systems engineering. The most natural bilevel optimization formulation of this problem is computationally intractable due to the nonconvexity of the AC power flow equations, and thus alternative approaches are required. This paper compares the performance of three approaches: 1) a state-of-the-art bilevel optimization approach that uses a "network flow" relaxation of the AC power flow equations to obtain a computationally tractable formulation, 2) a reinforcement learning approach that iteratively removes lines and receives a reward based on the amount of load shed, and 3) a neural network trained on a set of line failure scenarios that is formulated as a mixed-integer linear program.

The results of this comparison on several PGLib-OPF test cases show that these approaches are capable of identifying the worst-case or near-worst-case sets of line failures. While the bilevel network flow formulation achieved the best results in identifying the single worst failure in most of the tests, the two machine learning based approaches were better at identifying large sets of high-quality solutions. The NN-MILP stands out in particular as the best all around approach for the $N-k$ interdiction problem. Furthermore, we note that both neural networks resulting from the vastly different machine learning algorithms seem to share similar learning patterns that correlate with the predicted load shed.

These results raise several future research directions:

1) Extending the results to larger test cases to see if the patterns for small/medium test cases continue to hold.
2) Running the machine learning algorithms with smaller sample sizes to see how their performance is affected.
3) Evaluating possible benefits from using more complex neural network structures as well as blending model-based and data-driven approaches.

## References

[1] B. C. Dandurand, K. Kim, and S. Leyffer, "A Bilevel Approach for Identifying the Worst Contingencies for Nonconvex Alternating Current Power Systems," *SIAM J. Optimiz.*, vol. 31, no. 1, pp. 702–726, Jan. 2021.

[2] J. M. López-Lezama, J. Cortina-Gómez, and N. Muñoz-Galeano, "Assessment of the Electric Grid Interdiction Problem Using a Nonlinear Modeling Approach," *Electr. Power Syst. Res.*, vol. 144, no. 1, pp. 243–254, Mar. 2017.

[3] J. M. Arroyo and F. J. Fernández, "Application of a Genetic Algorithm to N-K Power System Security Assessment," *Int. J. of Electr. Power Energy Syst.*, vol. 49, pp. 114–121, Jul. 2013.

[4] F. Capitanescu and L. Wehenkel, "Computation of Worst Operation Scenarios Under Uncertainty for Static Security Management," *IEEE Trans. Power Syst.*, vol. 28, no. 2, pp. 1697–1705, May 2013.

[5] K. Sundar, S. Misra, R. Bent, and F. Pan, "Credible Interdiction for Transmission Systems," *IEEE Trans. Control Netw. Syst.*, vol. 8, no. 2, pp. 738–748, 2021.

[6] A. Motto, J. Arroyo, and F. Galiana, "A Mixed-Integer LP Procedure for the Analysis of Electric Grid Security Under Disruptive Threat," *IEEE Trans. Power Syst.*, vol. 20, no. 3, pp. 1357–1365, Aug. 2005.

[7] J. Arroyo and F. Galiana, "On the Solution of the Bilevel Programming Formulation of the Terrorist Threat Problem," *IEEE Trans. Power Syst.*, vol. 20, no. 2, pp. 789–797, May 2005.

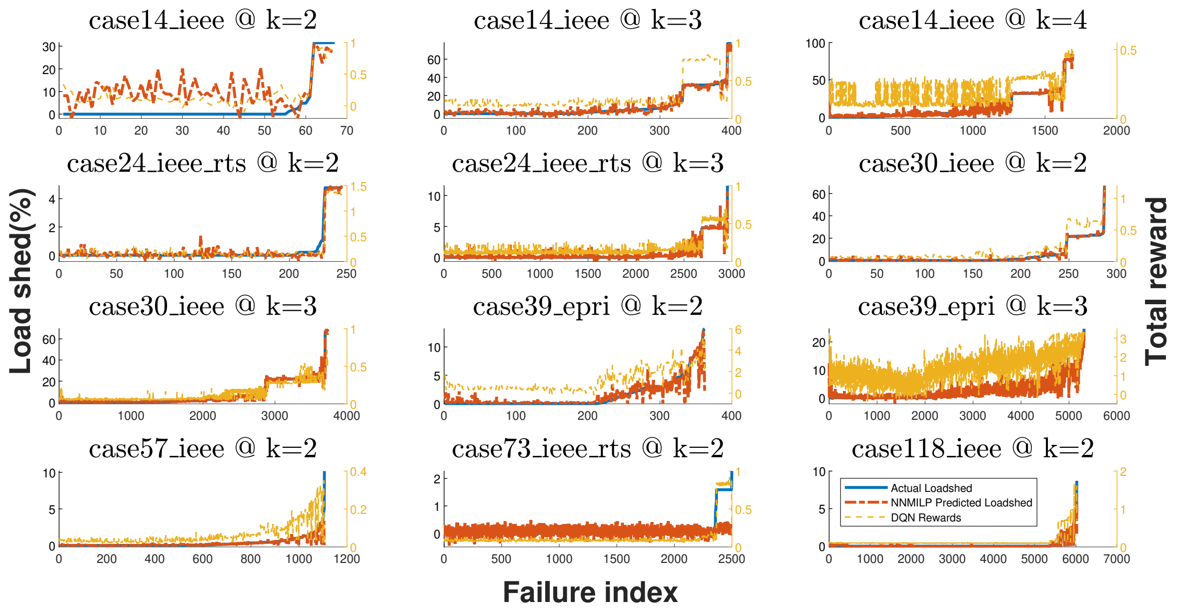

Fig. 3. Comparison between NN-MILP and DQN fully trained neural network outputs.

[8] J. Salmeron, K. Wood, and R. Baldick, "Analysis of Electric Grid Security Under Terrorist Threat," *IEEE Trans. Power Syst.*, vol. 19, no. 2, pp. 905–912, May 2004.

[9] R. E. Alvarez, "Interdicting Electrical Power Grids," Thesis, Monterey, California. Naval Postgraduate School Department of Operations Research, Mar. 2004. [Online]. Available: https://calhoun.nps.edu/handle/10945/1715

[10] E. S. Johnson and S. S. Dey, "A Scalable Lower Bound for the Worst-Case Relay Attack Problem on the Transmission Grid," May 2021, arXiv: 2105.02801.

[11] X. Chen, G. Qu, Y. Tang, S. Low, and N. Li, "Reinforcement Learning for Decision-Making and Control in Power Systems: Tutorial, Review, and Vision," Feb. 2021, arXiv: 2102.01168.

[12] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep Reinforcement Learning for Power System Applications: an Overview," *CSEE J. Power Energy Syst.*, vol. 6, no. 1, pp. 213–225, 2020.

[13] M. Glavic, "(Deep) Reinforcement Learning for Electric Power System Control and Related Problems: a Short Review and Perspectives," *Annu. Rev. Control*, vol. 48, pp. 22–35, Jan. 2019.

[14] D. Cao, W. Hu, J. Zhao, G. Zhang, B. Zhang, Z. Liu, Z. Chen, and F. Blaabjerg, "Reinforcement Learning and its Applications in Modern Power and Energy Systems: A Review," *J. Modern Power Syst. and Clean Energy*, vol. 8, no. 6, pp. 1029–1042, 2020.

[15] I. Murzakhanov, A. Venzke, G. S. Misyris, and S. Chatzivasileiadis, "Neural Networks for Encoding Dynamic Security-Constrained Optimal Power Flow," Oct. 2021, arXiv:2003.07939.

[16] A. Venzke, G. Qu, S. Low, and S. Chatzivasileiadis, "Learning Optimal Power Flow: Worst-Case Guarantees for Neural Networks," Nov. 2020, arXiv:2006.11029.

[17] S. Paul and F. Ding, "Identification of Worst Impact Zones for Power Grids During Extreme Weather Events Using Q-Learning," in *IEEE PES Innovative Smart Grid Technologies Conf. (ISGT)*, 2020, pp. 1–5.

[18] S. R, R. S. Kumar, and A. T. Mathew, "Online Static Security Assessment Module Using Artificial Neural Networks," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 4328–4335, Nov. 2013.

[19] B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici, "Anticipating Contingenciesin Power Grids Using Fast Neural Net Screening," May 2018, arXiv: 1805.02608.

[20] K. Shanti Swarup and G. Sudhakar, "Neural Network Approach to Contingency Screening and Ranking in Power Systems," *Neurocomputing*, vol. 70, no. 1, pp. 105–118, Dec. 2006.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[22] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating Robustness of Neural Networks with Mixed Integer Programming," Nov. 2017, arXiv:1711.07356.

[23] M. Fischetti and J. Jo, "Deep Neural Networks and Mixed Integer Linear Optimization," *Constraints*, vol. 23, no. 3, pp. 296–309, Jul. 2018.

[24] B. Grimstad and H. Andersson, "ReLU Networks as Surrogate Models in Mixed-Integer Linear Programs," *Computers & Chemical Engineering*, vol. 131, no. 1, p. 106580, Dec. 2019.

[25] D. Bienstock and A. Verma, "Strong NP-hardness of AC power flows feasibility," *Oper. Res. Lett.*, vol. 47, no. 6, pp. 494–501, Nov. 2019.

[26] C. Coffrin, R. Bent, B. Tasseff, K. Sundar, and S. Backhaus, "Relaxations of AC Maximal Load Delivery for Severe Contingency Analysis," *IEEE Trans. Power Syst.*, vol. 34, no. 2, pp. 1450–1458, Mar. 2019.

[27] C. Coffrin, H. Hijazi, and P. Van Hentenryck, "Network Flow and Copper Plate Relaxations for AC Transmission Systems," in *19th Power Syst. Comput. Conf. (PSCC)*, June 2016.

[28] H. W. Kuhn and A. W. Tucker, "Nonlinear Programming," *Proc. 2nd Berkeley Symp. Math. Stat. Probab.*, pp. 481–492, 1951.

[29] D. G. Luenberger and Y. Yinyu, *Linear and Nonlinear Programming*, 2nd ed. Wiley, 1989.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[31] R. S. Sutton, "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding," in *Advances in Neural Information Processing Systems (NIPS)*, D. Touretzky, M. C. Mozer, and M. Hasselmo, Eds., vol. 8. MIT Press, 1996.

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

[33] J. Lofberg, "YALMIP: A Toolbox for Modeling and Optimization in MATLAB," in *IEEE Int. Conf. Robotics Automat.*, Sep. 2004, pp. 284–289.

[34] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "Powermodels.jl: An Open-Source Framework for Exploring Power Flow Formulations," in *20th Power Systems Computation Conference (PSCC)*, June 2018.

[35] N. Rhodes, D. M. Fobes, C. Coffrin, and L. Roald, "PowerModelsRestoration.jl: An Open-Source Framework for Exploring Power Network Restoration Algorithms," *Electr. Power Syst. Res.*, vol. 190, no. 1, p. 106736, Jan. 2021.

[36] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, "Fashionable Modelling with Flux," Nov. 2018, arXiv: 1811.01457.

[37] A. Wächter and L. T. Biegler, "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, Mar. 2006.