

Supplementary Material

A ANALYTICAL FORMULATION OF \mathcal{D} , UNIQUENESS OF THE OPLC SOLUTION, AND UNBOUNDED VARIABLE

Set \mathcal{D} is of problem dependent. For example, in DC-OPF problems, \mathcal{D} represents the interested load input domain which is set by the system operator, e.g., feasible load within [100%, 130%] of the default load. For others applications, \mathcal{D} represents region of *possible* feasible problem inputs. Calculating the analytical representation of the feasible region of θ is known as projection of a polyhedral set to lower dimension subspace. That is, \mathcal{D} can be analytically obtained by projecting the following set

$$\mathcal{P} = \{(\theta, \mathbf{x}) | \mathbf{A}_\theta \theta \leq \mathbf{b}_\theta, \text{ and } (1), (2) \text{ hold}\}$$

onto the subspace of θ , which is still a convex polytope. The goal can be achieved using the Fourier–Motzkin elimination technique. Nevertheless, in our design, we do not need to access the full analytical formulation of \mathcal{D} . Instead, we introduce a set of auxiliary variable $\tilde{\mathbf{x}}$ associated with each θ . That is, the constraint $\theta \in \mathcal{D}$ is indeed represented as $\{\mathbf{A}_\theta \theta \leq \mathbf{b}_\theta, g_j(\tilde{\mathbf{x}}, \theta) \leq e_j, \forall j \in \mathcal{E}\}$.

A.1 UNIQUENESS OF THE OPLC SOLUTION

We would like to further discuss the assumption of the uniqueness of the OPLC solution. First, many OPLC are unique given their objective functions are strictly convex. Such a condition holds for DC-OPF problems in power systems (Pan et al., 2019) and model-predictive control problems in control systems (Bemporad et al., 2000). As proved in (Pan et al., 2020a), if the optimal solution is unique, the input-solution mapping is continuous while the DNN function is also continuous, which forms the underlying reason why DNN can be applied to learning such a mapping from the Universal Approximation Theorem of DNN for continuous functions.

We would like to further discuss the situation if the optimal solution is not unique, which is an open problem and the challenge of the existing end-to-end DNN design.

Given a OPLC that admits multiple optimal solutions for the input, there indeed does not exist an injective mapping between input to solution, i.e., there exist multiple input-solution mappings. Consider the DNN training in this case, if the ground-truth training data are from different input-solution mappings, the DNN could present unsatisfactory performance as solutions to closely related instances may exhibit large differences and the learning task can become inherently more difficult (Kotary et al., 2021a; Huang & Chen, 2021; Pan et al., 2023). Nevertheless, our approach is still applicable to such a scenario as the first obtained DNN-FG after determining the sufficient DNN size can still guarantee universal feasibility. As introduced in Sec. 4.1 and Sec. 4.2, deriving the calibration rate and determining the sufficient DNN size is only related to the OPLC constraints. These steps only require obtaining one of the continuous feasible mappings but not optimality. Towards the Adversarial-Sample Aware algorithm, it is straightforward to adopt the approaches in (Kotary et al., 2021a; Huang & Chen, 2021; Pan et al., 2023) by improving the training data quality, applying the unsupervised learning idea, or learning the high-dimensional input+initial point to optimal solution mapping, which we leave for future work. Finally, the simulations on non-convex optimization (can have non-unique optimum) in Appendix N show that the ASA algorithm can still work well, showing the robustness of the design.

A.2 UNBOUNDED DECISION VARIABLES

There are two approaches to handle the unbounded variables: 1) setting \underline{x}_i or \bar{x}_i to be some arbitrarily small/large numbers. 2) only includes the bounded constraints into (4)-(5) and (6), e.g., for the variables 1) without lower bound, the DNN output is $\hat{x}_i = -\sigma(\bar{x}_i - (\mathbf{W}_o \mathbf{h}_{N_{\text{hid}}} + \mathbf{b}_o)_i) + \bar{x}_i$; 2) without upper bound $\hat{x}_i = \tilde{h}_i$; 3) without both upper and lower bound, $\hat{x}_i = (\mathbf{W}_o \mathbf{h}_{N_{\text{hid}}} + \mathbf{b}_o)_i$.

B HANDLING EQUALITY AND NON-LINEAR CONSTRAINTS

We remark that for general OPLC and other constrained optimizations, we can always removing the equality constraints explicitly/implicitly. Given $N + p$ variables and p (linear) equality con-

straints, we can remove these equalities and representing p variables by the remaining N variables using the equality constraints, e.g., applying the coefficient matrix inversion as discussed in Appendix L without losing optimality. We thus focus on OPLC with inequality constraints only. The similar predict-and-reconstruct idea is proposed in (Pan et al., 2019; Donti et al., 2021). In addition, we note that the proposed *preventive leaning* framework is also applicable to non-linear inequality constraints, e.g., AC-OPF problems with several thousand buses, but with additional computational challenges in solving the related programs corresponding to the required steps. We leave the application to optimization with non-linear constraints and non-convex objective with large DNN size for future study.

In this work, we consider the variation of the RHS of the linear inequality constraints. It is also interesting to study the varying a_j, b_j, e_j . We believe our approach is still applicable to such a case while may have additional computational challenges as the problem turn to be non-linearly constrained. Nevertheless, it is also great interest to study problems whose parameters are not varying. For example, in DC-OPF, a_j, b_j, e_j are determined by power network topology, which will not change significantly over a long time scale, e.g., months to years. Hence, it is reasonable and practical to study OPLC with varying inputs only.

C MIXED-INTEGER REFORMULATION OF BI-LEVEL LINEAR PROGRAMS

Consider the following the linear constrained bi-level min-max problem:

$$\min_{\theta} \max_x c^T x \quad (13)$$

$$\text{s.t. } \mathbf{A}x \leq \mathbf{b} + \mathbf{F}\theta, \quad (14)$$

$$\theta \in \mathcal{D}, \quad (15)$$

where $\mathbf{A} \in \mathcal{R}^{p \times N}$, $\mathbf{b} \in \mathcal{R}^p$, $\mathbf{F} \in \mathcal{R}^{p \times M}$.

The above linear constrained bi-level program can be reformulated by replacing the inner maximization problem by its sufficient and necessary KKT conditions (Boyd & Vandenberghe, 2004). We present the reformulated program in the following:

$$\min_{\theta, x, y} c^T x \quad (16)$$

$$\text{s.t. } \mathbf{A}x \leq \mathbf{b} + \mathbf{F}\theta, \quad (17)$$

$$\mathbf{A}^T y = c, \quad (18)$$

$$y_i \geq 0, i = 1, \dots, p, \quad (19)$$

$$y_i(a_i^T x - b_i - f_i^T \theta) = 0, i = 1, \dots, p, \quad (20)$$

$$\theta \in \mathcal{D}. \quad (21)$$

Here a_i and f_i denote the i -th row of matrix \mathbf{A} and \mathbf{F} respectively. We remark that the non-linear *Complementary Slackness* condition in (20) can be reformulated to be mixed-integer linear using the Fortuny-Amat McCarl linearization (Fortuny-Amat & McCarl, 1981):

$$y_i \leq (1 - r_i)C, \quad a_i^T x - b_i - f_i^T \theta \geq -r_i C. \quad (22)$$

Here the non-linear complementary slackness conditions are reformulated with the binary variable r_i and the large non-binding constant C for each $i = 1, \dots, p$. Therefore, problem (16)-(21) can be reformulated to be the mixed-integer linear program (MILP).

We remark that if $\nu^{f*} = 0$, implying that the system is too binding, e.g., for DC-OPF problem, some line/generator must always be at its capacity upper bound. Such a restrictive condition seldom happen in practice for the power system safety operation. Under such a scenario, one can consider a smaller input region \mathcal{D} such that the input is not so extreme and there could always exist an interior for the input region.

D MINIMUM SUPPORTING CALIBRATION REGION

We remark that the obtained uniform calibration rate on each constraints forms the *outer bound* of the minimum supporting calibration region defined as follows:

Definition 1 The minimum supporting calibration region is defined as the set of calibration rate $\{\eta_j\}_{j \in \mathcal{E}}$ and for each $\theta \in \mathcal{D}$, there exist an \mathbf{x} such that (2) and (3) hold. Meanwhile, there exist a $\theta \in \mathcal{D}$ and there does not exist an \mathbf{x} such that (2) and (3) hold under $\{\eta_j + \delta_j\}_{j \in \mathcal{E}}$ for any $\delta_j \geq 0$ and at least one $\delta_j > 0$.

The minimum supporting calibration region describes the set of maximum calibration rate such that 1) the input parameter region is maintained, and 2) any further calibration on the constraints will lead some input to be infeasible. We remark that such minimum supporting calibration region is not unique. See the following example and the approach to obtain (one of) such minimum supporting calibration region.

We first provide a toy example to demonstrate the non-uniqueness of the minimum supporting calibration region defined in Def. 1. Consider the following modified network flow problem:

$$\min \quad x_1^2 + x_2^2 + x_3^2 \quad (23)$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 90, \quad (24)$$

$$0 \leq x_2 \leq 90, \quad (25)$$

$$x_3 \leq 70, \quad (26)$$

$$x_1 + x_2 \leq 90, \quad (27)$$

$$x_2 + x_3 \leq 90, \quad (28)$$

$$x_1 + x_2 + x_3 = l. \quad (29)$$

Here l is the input load within $[0, 100]$ and x_1 , x_2 , and x_3 can be seen as the network flow on the edges. Similar to the analysis in Sec. 4.1, the constraints (26)-(28) can be calibrated by at most 37.5% uniformly. However, such a calibration region is not the minimum one while forms the outer bound of it. Denote the calibration rate on (26)-(28) as (x, y, z) , it is easy to see that any combination such that $7x + 9y = 6$ and $z = 8/9 - y$ is the minimum supporting one.

We further provide the follow procedures to determine (one of) the minimum supporting region.

- Step 1. Solve (4)-(5) to obtain the uniform maximum calibration rate Δ . Let $k = 1$.
- Step 2. For constraint g_k , solve

$$\min_{\theta \in \mathcal{D}} \max_{\mathbf{x}} \frac{\hat{e}_k - g_k(\theta, \mathbf{x})}{|e_j|} \quad (30)$$

$$\text{s.t.} \quad (2)$$

$$g_j(\theta, \mathbf{x}) \leq \hat{e}_j, \quad \forall j \in \mathcal{E}, \quad (31)$$

where $\hat{e}_k = e_k \times (1_{e_k \geq 0}(1 - \Delta) + 1_{e_k < 0}(1 + \Delta))$. Denote the optimal value of (30)-(31) as δ_k , which represent the maximum additional individual calibration rate of constraint g_k considering all other constraints' calibrations.

- Update \hat{e}_k to be $e_k \times (1_{e_k \geq 0}(1 - \Delta - \delta_k) + 1_{e_k < 0}(1 + \Delta + \delta_k))$ and proceed to the next iteration $k + 1$. Go to Step 2.

We remark that after each update of \hat{e}_k , the next g_{k+1} is studied on a tighter region described by $\{\hat{e}_j, j = 1, \dots, k\}$. After solving the programs for each g_k , one can easily see that the calibration region $\{\Delta + \delta_j\}_{j \in \mathcal{E}}$ is the minimum supporting calibration region.

In this work, we consider the uniform calibration rate Δ for further analysis. We remark that the uniform calibration method may introduce the asymmetry on the calibration rate as large limit would have large calibration rate. An alternative approach is to set the individual calibration rate η_j for each constraint while maintain the supported input region as discussed above. However, the choice of such individual calibration rates is not unique due to the non-uniqueness of the minimum supporting calibration region. We leave the analysis of such individual constraints calibration for future study.

E DETAILS OF APPLYING *Danskin's Theorem* TO THE BI-LEVEL PROBLEM TO DETERMINE THE SUFFICIENT DNN SIZE

We provide the details of applying *Danskin's Theorem* to solve the bi-level mined-integer non-linear problem (9)-(10).

To solve such bi-level optimization problem, we optimize the upper-level variables (\mathbf{W}, \mathbf{b}) by gradient descent. This would simply involve repeatedly computing the gradient w.r.t. (\mathbf{W}, \mathbf{b}) for the object function, and taking a step in this negative direction. That is, we want to repeat the update

$$\mathbf{W} := \mathbf{W} - \alpha \cdot \nabla_{\mathbf{W}}(\max_{\theta} \nu^f(\mathbf{W}, \mathbf{b}, \theta)), \quad (32)$$

$$\mathbf{b} := \mathbf{b} - \alpha \cdot \nabla_{\mathbf{b}}(\max_{\theta} \nu^f(\mathbf{W}, \mathbf{b}, \theta)). \quad (33)$$

Here $\max_{\theta} \nu^f(\mathbf{W}, \mathbf{b}, \theta)$ denotes the maximum violation among the calibrated inequality constraints within the entire inputs domain \mathcal{D} , given the specific value of DNN parameters (\mathbf{W}, \mathbf{b}) . Note that the inner function itself contains a maximization problem. We apply the *Danskin's Theorem* to compute the gradient of the inner term. It states that the gradient of the inner function involving the maximization term is simply given by the gradient of the function evaluated at this maximum. In other words, to compute the (sub)gradient of a function containing a $\max(\cdot)$ term, we need to simply: 1) find the maximum, and 2) compute the normal gradient evaluated at this point (Dong et al., 2020b; Danskin, 2012). Hence, the relevant gradient is given by

$$\nabla_{\mathbf{W}}(\max_{\theta} \nu^f(\mathbf{W}, \mathbf{b}, \theta)) = \nabla_{\mathbf{W}} \nu^f(\mathbf{W}, \mathbf{b}, \theta^*), \quad (34)$$

$$\nabla_{\mathbf{b}}(\max_{\theta} \nu^f(\mathbf{W}, \mathbf{b}, \theta)) = \nabla_{\mathbf{b}} \nu^f(\mathbf{W}, \mathbf{b}, \theta^*), \quad (35)$$

where

$$\theta^* = \arg \max_{\theta} \nu^f(\mathbf{W}, \mathbf{b}, \theta). \quad (36)$$

Here the optimal θ^* depends on the choice of DNN parameters (\mathbf{W}, \mathbf{b}) . Therefore, at each iterative update of (\mathbf{W}, \mathbf{b}) , we need to solve the inner maximization problem once. Note that the optimal θ^* may not be unique. However, the gradient of $\nu^f(\mathbf{W}, \mathbf{b}, \theta^*)$ w.r.t. (\mathbf{W}, \mathbf{b}) can still be obtained given a specific θ^* , which is (one of the) gradient that optimizes the deep neural network. We remark that such approach is indeed widely adopted in existing literature (Dong et al., 2020b; Danskin, 2012). In addition, though the involved program is a mixed-integer linear problem, we observe that the solver can indeed provide its optimum efficiently, e.g., <20 mins for Case300 in DC-OPF problem in simulation. Nevertheless, we remark that finding a (sub-optimal) feasible solution for the inner maximization problem can be easily obtained by a heuristic trial of some particular θ , e.g., the worst-case input at the previous round as the initial point and the associate integer values in the DNN constraints (7)-(8), which are fixed given the specification of DNN parameters. Such a solution can still be utilized for the further steps to calculate the sub-gradient of the DNN. One can see the analogy between it and DNN training with stochastic gradient decent method.

In addition, note that to obtain the upper bound ρ , we do not need to access any feasible point of the inner maximization problem. The upper bound is provided by the relaxation in the branch-and-bound algorithm, e.g., relax (some) integer variables to continuous. This can be efficiently obtained by the solvers, e.g., APOPT or Gurobi. Such an upper bound is applied to verify whether universal feasibility guarantee is obtained and whether the DNN size is sufficient.

E.1 DETERMINING THE VALUES OF $h_i^{\max,k}/h_i^{\min,k}$

$h_i^{\max,k}/h_i^{\min,k}$ are constants and fixed during solving the (inner) MILP in optimization (9)-(10) (Tjeng et al., 2018). These numbers represent the maximum/minimum bounds on the values of the neuron outputs, which should be large/small enough numbers to let the DNN constraints not be binding in the reformulation (7)-(8). In our design, we follow the technique in (Venzke et al., 2020) to obtain such (tighter) upper/lower bounds for each updated (\mathbf{W}, \mathbf{b}) . In particular, we minimize and maximize the output of each neuron subject to the linear relaxation of the binary variables (to be continuous within 0 and 1) in the DNN constraints with parameters (\mathbf{W}, \mathbf{b}) in (7)-(8) and entire input region \mathcal{D} . Such upper/lower bounds can be efficiently obtained by solving the LPs after relaxation, which guarantees that the neuron output will not exceed the corresponding values. We note that for different DNN parameters (\mathbf{W}, \mathbf{b}) , $h_i^{\max,k}/h_i^{\min,k}$ could take different values that can always be efficiently obtained from the LPs after linear relaxation.

F PROOF OF PROPOSITION 1

Proof: Consider the DNN with N_{hid} hidden layers each having N_{neu} neurons and parameters $(\mathbf{W}^f, \mathbf{b}^f)$ and $\rho \leq \Delta$. Since ρ is an upper bound on the optimal value of the bi-level problem (9)-(10), we have

$$(g_j(\boldsymbol{\theta}, \hat{\mathbf{x}}) - \hat{e}_j)/|e_j| \leq \rho, \forall \boldsymbol{\theta} \in \mathcal{D}, \forall j \in \mathcal{E}. \quad (37)$$

Therefore, we have for any $\boldsymbol{\theta} \in \mathcal{D}$ and $j \in \mathcal{E}$

$$\begin{cases} g_j(\boldsymbol{\theta}, \hat{\mathbf{x}}) - e_j(1 - \Delta) \leq \rho \cdot e_j, & \text{if } e_j \geq 0; \\ g_j(\boldsymbol{\theta}, \hat{\mathbf{x}}) - e_j(1 + \Delta) \leq -\rho \cdot e_j, & \text{otherwise,} \end{cases} \quad (38)$$

which is equivalent to

$$\begin{cases} g_j(\boldsymbol{\theta}, \hat{\mathbf{x}}) \leq e_j + (\rho - \Delta) \cdot e_j, & \text{if } e_j \geq 0; \\ g_j(\boldsymbol{\theta}, \hat{\mathbf{x}}) \leq e_j + (\Delta - \rho) \cdot e_j, & \text{otherwise.} \end{cases} \quad (39)$$

Since $\rho \leq \Delta$, we have

$$g_j(\boldsymbol{\theta}, \hat{\mathbf{x}}) \leq e_j, \forall \boldsymbol{\theta} \in \mathcal{D}, \forall j \in \mathcal{E}. \quad (40)$$

This completes the proof of Proposition 1.

G UNIVERSAL APPROXIMATION CAPABILITY OF DNN

We highlight the *Universal Approximation* of DNNs for approximate the input-solution for the OPLC in the following proposition.

Proposition 5 (Hornik, 1991) *Assume the target function to learn is continuous, there always exists a DNN whose output function can approach the target function arbitrarily well, i.e.,*

$$\max_{\boldsymbol{\theta} \in \mathcal{D}} \|h(\boldsymbol{\theta}) - \hat{h}(\boldsymbol{\theta})\| < \varepsilon,$$

hold for any ε arbitrarily small (distance from h to \hat{h} can be infinitely small). Here $h(\boldsymbol{\theta})$ and $\hat{h}(\boldsymbol{\theta})$ represent the target mapping to be approximated and the DNN function respectively.

Furthermore, given the fixed depth N_{hid} of the DNN, the learning ability of the DNN is increasing monotonically w.r.t. the width of the DNN. That is, consider two DNN width N_{neu}^1 and N_{neu}^2 such that $N_{\text{neu}}^1 > N_{\text{neu}}^2$, we have

$$\min_{h \in \mathcal{C}^{N_{\text{neu}}^1}} \max_{\boldsymbol{\theta} \in \mathcal{D}} \|h(\boldsymbol{\theta}) - \hat{h}(\boldsymbol{\theta})\| \leq \min_{h \in \mathcal{C}^{N_{\text{neu}}^2}} \max_{\boldsymbol{\theta} \in \mathcal{D}} \|h(\boldsymbol{\theta}) - \hat{h}(\boldsymbol{\theta})\|,$$

where $\mathcal{C}^{N_{\text{neu}}^1}$ and $\mathcal{C}^{N_{\text{neu}}^2}$ denote the class of all functions generated by a N_{hid} depth neural network with width N_{neu}^1 and N_{neu}^2 respectively.

Proposition 5 provides as the strong observation and theoretical basis for designing the iterative approach to determine the sufficient DNN size in guaranteeing universal feasibility.

H MINIMAL SUFFICIENT DNN SIZE

We remark that the obtained sufficient DNN size by doubling the DNN width may be substantial, introducing additional training time to train the DNN model and higher computational time when applied to solve OPLC. One can also determine the corresponding minimum sufficient DNN size by a simple and efficient binary search between

- the obtained sufficient DNN size N_{neu}^* and the pre-obtained DNN size $N_{\text{neu}}^*/2$ (before doubling the DNN width) which fails to achieve universal feasibility, if the initial tested DNN can not guarantee universal feasibility;

- the initial tested DNN size and some small DNN, e.g., zero width DNN, if the initial tested DNN size is sufficient in guaranteeing universal feasibility.

Such a minimum sufficient DNN size denotes the minimum width required for a given DNN structure with depth N_{hid} to achieve universal feasibility within the entire input domain. We use \hat{N}_{neu} to denote the determined minimum sufficient DNN size and propose the following proposition.

Proposition 6 *Consider the DNN with N_{hid} hidden layers each having \hat{N}_{neu} neurons, any DNN with depth N_{hid} and a smaller width than \hat{N}_{neu} can not guarantee universal feasibility for all input $\theta \in \mathcal{D}$. Meanwhile, any DNN with depth N_{hid} and at least \hat{N}_{neu} width can always achieve universal feasibility.*

I DESCRIPTION OF ADVERSARIAL-SAMPLE AWARE ALGORITHM

We outline the Adversarial-Sample Aware algorithm in Algorithm 2. As seen, the *Adversarial Sample-Aware* algorithm pre-trains the DNN model (line 3) and starts the iteration (line 5). Each iteration verifies whether the worst-case prediction error is within the room (maximum calibration rate) (lines 6 - 13). If the maximum constraints violation of the adversarial input exceeds the determined calibrated rate, the *Adversarial-Sample Aware* algorithm incorporates a set of adversarial samples into the existing training set and updates the DNN parameters (line 14 - 19), expecting the constraints violation of the adversaries are eliminated.

We expect that after a few training epochs, the post-trained DNN can restore feasibility at the identified adversarial sample θ^t and the points around it in \mathcal{S}^t . This is inspired by the observation that after adding the previously identified training pairs \mathcal{S}^t into the training set, the DNN training loss is dominated by the approximation errors and the penalties at the samples in \mathcal{S}^t . Though the training loss may not be optimized to 0, e.g., still has violations w.r.t. the calibrated constraints limits, the DNN solution is expected to satisfy the original inequality constraints after such preventive training procedure. Therefore, the post-trained DNN is capable of preserving feasibility and good accuracy at these input regions. Simulation results in Sec. 6.2.1 show the effectiveness of the propose algorithm. We provide the following proposition to state the guarantee of the algorithm in ensuring universal feasibility.

J PROOF OF PROPOSITION 3

Proof idea: Here we consider the post-trained DNN with N_{hid} hidden layers each having N_{neu}^* neurons. Given current iteration i , for $\forall j \leq i$, suppose it can always maintain feasibility at the correspondingly constructed neighborhoods around the identified worst-case input, i.e., $\hat{\mathcal{D}}^j$, by training on \mathcal{T}^{i+1} that combines \mathcal{T}^0 and all the auxiliary subset \mathcal{S}^j around the identified adversarial input θ^j , $\forall j \leq i$. Therefore, when the number of iterations is large enough, the union of the feasible regions $\hat{\mathcal{D}}^{i>C} = \hat{\mathcal{D}}^1 \cup \hat{\mathcal{D}}^2 \cup \dots \hat{\mathcal{D}}^i$ can cover the entire input domain \mathcal{D} . That is, the post-trained DNN can ensure feasibility for each small region $\hat{\mathcal{D}}^i$ within the input domain \mathcal{D} , and hence universal feasibility is guaranteed. Such observation is similar to the topic of minimum covering ball problem of the compact set in real analysis.

Such a condition generally requires the DNN to preserve feasibility within some small regions by especially including the input-solution information during training, which may not be hard to satisfy. This can be understood from the observation that the worst-case violation in the smaller inner domain can be reduced significantly by training on the broader outer input domain (Venzke et al., 2020; Nellikkath & Chatzivasileiadis, 2022) as the adversarial inputs are always element-wise at the boundary of the entire input domain \mathcal{D} , which echoes our simulation findings in Sec. 6. Therefore, the post-trained DNN is expected to perform good feasibility guarantee in all small regions $\hat{\mathcal{D}}^j$, $\forall j \leq i$ after the preventive training procedure on \mathcal{T}^{i+1} , the training set on the entire domain \mathcal{D} . We remark that after gradually including these subsets \mathcal{S}^i into the existing training set, the loss function is determined by the joint loss among all samples in these regions. After the training process, the post-obtained DNN is hence expected to maintain feasibility at the points in the training set and the regions around them.

Algorithm 2: Adversarial-Sample Aware Algorithm

```

1: Input:  $\Delta$ , Initial training set  $\mathcal{T}^0$ , Training epochs  $T$ , Number of iterations  $I$ 
2: Output: DNN model with parameters  $(\mathbf{W}^*, \mathbf{b}^*)$ 
3: Pre-train the DNN model on  $\mathcal{T}^0$  using loss function (11) for  $T$  epochs
4: Save DNN parameters as  $(\mathbf{W}^0, \mathbf{b}^0)$ 
5: for  $i = 0$  to  $I$  do
6:   Find the maximum violation of  $(\mathbf{W}^i, \mathbf{b}^i)$  by solving:
       
$$\boldsymbol{\theta}^i = \arg \max_{\boldsymbol{\theta} \in \mathcal{D}} \nu^f \text{ s.t. } (7) - (8), 1 \leq i \leq N_{\text{hid}}, 1 \leq k \leq N_{\text{neu}}, (10).$$

7:   Set  $\gamma = \nu^f(\boldsymbol{\theta}^i)$ 
8:   if  $\gamma \leq \Delta$  then
9:     Set  $\mathbf{W}^* = \mathbf{W}^i, \mathbf{b}^* = \mathbf{b}^i$ ; Break
10:  else
11:    Construct  $\mathcal{S}^i$  by randomly sampling around  $\boldsymbol{\theta}^i$ 
12:    Set  $\mathcal{T}^{i+1} = \mathcal{T}^i \cup \mathcal{S}^i$  and  $(\mathbf{W}_0^i, \mathbf{b}_0^i) = (\mathbf{W}^i, \mathbf{b}^i)$ 
13:  end if
14:  for  $t = 0$  to  $T$  do
15:    Train the DNN on  $\mathcal{T}^{i+1}$  using loss function (11) and update DNN parameters to
       $(\mathbf{W}_{t+1}^i, \mathbf{b}_{t+1}^i)$ 
16:    Feed each  $\boldsymbol{\theta} \in \mathcal{S}^i$  in the DNN model to obtain predicted solution  $\hat{\mathbf{x}}$ 
17:    if Each  $\hat{\mathbf{x}}$  is feasible w.r.t. (1)- (2) then
18:      Break
19:    end if
20:  end for
21:  Set  $(\mathbf{W}^{i+1}, \mathbf{b}^{i+1}) = (\mathbf{W}_{t+1}^i, \mathbf{b}_{t+1}^i)$ 
22: end for

```

K RUN-TIME COMPLEXITY OF THE FRAMEWORK

The computational complexity of the framework consists of the complexity of using DNN to predict the solutions, which is $\mathcal{O}(N_{\text{hid}}N_{\text{neu}}^2)$ (Pan et al., 2020b). Recall that N_{hid} denote the number of hidden layers in DNN (depth), and N_{neu} denotes the number of neurons at each layer (width). In practice, we set N_{hid} to be 3 and observe that the DNN with width N_{neu} of $\mathcal{O}(N)$ can achieve satisfactory optimality performance with universal feasibility guarantee. Therefore, the complexity of using DNN to predict the N variables is $\mathcal{O}(N^2)$.

Note that the number of decision variables to be optimized is N . After taking $\mathcal{O}(|\mathcal{E}|M)$ operations to calculate the value of $\mathbf{b}_j^T \boldsymbol{\theta}$ for each $j \in \mathcal{E}$, the computational complexity of interior-point methods for solving such programs is $\mathcal{O}(N^4)$, measured by the number of elementary operations assuming that it takes a fixed time to execute each operation (Ye & Tse, 1989). Therefore, the traditional method for solve the OPLC has a computational complexity of $\mathcal{O}(N^4 + |\mathcal{E}|M)$.

We remark that the computational complexity of the proposed framework is lower than that of traditional algorithms.

L IMPLEMENTATIONS OF DEEPOPf+

Recall that the DC-OPF formulation is given as

$$\min_{\mathbf{P}_G, \Phi} \sum_{i \in \mathcal{G}} c_i(P_{Gi}) \quad (41)$$

$$\text{s.t. } \mathbf{P}_G^{\min} \leq \mathbf{P}_G \leq \mathbf{P}_G^{\max}, \quad (42)$$

$$\mathbf{M} \cdot \Phi = \mathbf{P}_G - \mathbf{P}_D, \quad (43)$$

$$-\mathbf{P}_{\text{line}}^{\max} \leq \mathbf{B}_{\text{line}} \cdot \Phi \leq \mathbf{P}_{\text{line}}^{\max}. \quad (44)$$

We first reduce the number of decision variables (without losing optimality) by adopting the predict-and-reconstruct framework (Pan et al., 2019). Specifically, it leverages that the admittance matrix (after removing the entries corresponding to the slack bus) $\tilde{\mathbf{M}}$ is of full rank $B - 1$, where $B = |\mathcal{B}|$ and is the size of the set of buses. Thus, given each \mathbf{P}_D , once the non-slack generations $\{P_{Gi}\}_{i \in \mathcal{G} \setminus n_0}$ (n_0 denotes the slack bus index) are determined, the slack generation and the bus phase angles of all buses can be uniquely reconstructed:

$$P_G^{\text{slack}} = \sum_{i \in \mathcal{B}} P_{Di} - \sum_{i \in \mathcal{G} \setminus n_0} P_{Gi}, \quad (45)$$

$$\tilde{\Phi} = \tilde{\mathbf{M}}^{-1} (\tilde{\mathbf{P}}_G - \tilde{\mathbf{P}}_D), \quad (46)$$

where n_0 and P_G^{slack} denote the slack bus index and slack bus generation respectively. $\tilde{\mathbf{P}}_G$ and $\tilde{\mathbf{P}}_D$ are the $(B - 1)$ -dimensional generation and load vectors for all buses except the slack bus. Consequently, the line flow capacity constraints in (44) can be reformulated as

$$-\mathbf{P}_{\text{line}}^{\text{max}} \leq \tilde{\mathbf{B}}_{\text{line}} \tilde{\mathbf{M}}^{-1} (\tilde{\mathbf{P}}_G - \tilde{\mathbf{P}}_D) \leq \mathbf{P}_{\text{line}}^{\text{max}}, \quad (47)$$

where $\tilde{\mathbf{B}}_{\text{line}}$ is the line admittance matrix after removing the column of slack bus.⁷ Therefore, the reformulated DC-OPF problem takes the form of

$$\min_{\tilde{\mathbf{P}}_G} \sum_{i \in \mathcal{G} \setminus n_0} c_i (P_{Gi}) + c_{n_0} \left(\sum_{i \in \mathcal{B}} P_{Di} - \sum_{i \in \mathcal{G} \setminus n_0} P_{Gi} \right) \quad (48)$$

s.t. (47),

$$P_{Gi}^{\min} \leq P_{Gi} \leq P_{Gi}^{\max}, \forall i \in \mathcal{G} \setminus n_0, \quad (49)$$

$$P_{\text{slack}}^{\min} \leq \sum_{i \in \mathcal{B}} P_{Di} - \sum_{i \in \mathcal{G} \setminus n_0} P_{Gi} \leq P_{\text{slack}}^{\max}. \quad (50)$$

Therefore, we can solve DC-OPF by employing DNNs to depict the mapping between \mathbf{P}_D and $\tilde{\mathbf{P}}_G$.

L.1 REMOVING NON-CRITICAL INEQUALITY CONSTRAINTS

L.1.1 REMOVING NON-CRITICAL BRANCH LIMITS.

We propose the following program for each branch i to remove the non-critical branch limits given the entire load and generation space:

$$\max_{\tilde{\mathbf{P}}_G, \mathbf{P}_D} \nu_i - 1 \quad (51)$$

s.t. (49),

$$\mathbf{P}_D \in \mathcal{D}, \quad (52)$$

$$\nu = |\tilde{\mathbf{X}} (\tilde{\mathbf{P}}_G - \tilde{\mathbf{P}}_D)|. \quad (53)$$

Here we assume the load domain $\mathcal{D} = \{\mathbf{P}_D | \mathbf{A}_d \mathbf{P}_D \leq \mathbf{b}_d, \exists \tilde{\mathbf{P}}_G : (47) - (50) \text{ hold}\}$ is restricted to a convex polytope described by matrix \mathbf{A}_d and vector \mathbf{b}_d and the corresponding constraints. (49) enforces the feasibility of non-slack generations. (53) represents the normalized power flow level at each branch, where $\tilde{\mathbf{X}}$ is obtained from (47) by dividing each row of matrix $\tilde{\mathbf{B}}_{\text{line}} \tilde{\mathbf{M}}^{-1}$ with the value of corresponding line capacity and $\nu \in \mathcal{R}^{|\mathcal{E}|}$.

We remark that problem (51)-(53) can be reformulated as two linear programmings to perform the inference of the absolute sign of power flows in (53):

$$\max_{\tilde{\mathbf{P}}_G, \mathbf{P}_D} / \min_{\tilde{\mathbf{P}}_G, \mathbf{P}_D} \tilde{\nu}_i \quad (54)$$

s.t. (49), (52),

$$\tilde{\nu} = \tilde{\mathbf{X}} (\tilde{\mathbf{P}}_G - \tilde{\mathbf{P}}_D). \quad (55)$$

⁷The matrix $\tilde{\mathbf{B}}_{\text{line}} \tilde{\mathbf{M}}^{-1}$ is well-known as ‘‘Power Transfer Distribution Factors’’ (PTDF) matrix (Chatzivasileiadis, 2018).

If the optimal value of the above maximization (respectively minimization) problem is smaller or equal (respectively greater or equal) than 1 (respectively -1), then the optimal value of (54)-(55) is non-positive for some branch i . Therefore, such non-critical inequality constraint does not affect the feasible solution space such that it is always respected given any input load \mathbf{P}_D and can be removed from the DC-OPF problem. By solving (54)-(55), we can derive the set \mathcal{E} of critical branch capacity constraints whose optimal objectives are positive.⁸

L.1.2 REMOVING NON-CRITICAL SLACK BUS GENERATION LIMITS.

We provide the formulation to identify the critical slack generation limits given the entire load and generation space and the possible violation degree w.r.t. the upper and lower bounds here.

$$\max_{\bar{\mathbf{P}}_G, \mathbf{P}_D} \nu_{\text{slack}}^u \quad (56)$$

$$\text{s.t.} \quad (45), (49), (52),$$

$$\nu_{\text{slack}}^u = \frac{P_G^{\text{slack}} - P_{\text{slack}}^{\text{max}}}{P_{\text{slack}}^{\text{max}} - P_{\text{slack}}^{\text{min}}}, \quad (57)$$

and

$$\max_{\bar{\mathbf{P}}_G, \mathbf{P}_D} \nu_{\text{slack}}^l \quad (58)$$

$$\text{s.t.} \quad (45), (49), (52),$$

$$\nu_{\text{slack}}^l = \frac{P_{\text{slack}}^{\text{min}} - P_G^{\text{slack}}}{P_{\text{slack}}^{\text{max}} - P_{\text{slack}}^{\text{min}}}, \quad (59)$$

respectively. Here (57) and (59) denote the (normalized) exceeding of slack bus generation exceeding its upper bound and lower bound, respectively. Therefore, if the optimal values of these proposed optimization problem is non-positive, such slack generation limit is non-critical and does not affect the load-solution feasible region.

We remark that problems (56)-(57), and (58)-(59) are indeed linear programs and can be efficiently solved by the state-of-the-art solvers such as CPLEX or Gurobi. We find that all three test cases could have both critical upper bound and lower bound limits, i.e., both (56)-(57) and (58)-(59) have positive optimal values.

L.2 MAXIMUM CONSTRAINTS CALIBRATION RATE

To determine the maximum constraints calibration rate while preserving the input region, we solve the following bi-level optimization program:

$$\min_{\mathbf{P}_D} \max_{\mathbf{P}_G} \nu^c \quad (60)$$

$$\text{s.t.} \quad (42), (45) - (47), (52),$$

$$|PF_{ij}| = \left| \frac{1}{r_{ij}} (\phi_i - \phi_j) \right|, \forall (i, j) \in \mathcal{E}, \quad (61)$$

$$P_{\text{slack}}^u = (P_{\text{slack}}^{\text{max}} - P_G^{\text{slack}}) / (P_{\text{slack}}^{\text{max}} - P_{\text{slack}}^{\text{min}}), \quad (62)$$

$$P_{\text{slack}}^l = (P_G^{\text{slack}} - P_{\text{slack}}^{\text{min}}) / (P_{\text{slack}}^{\text{max}} - P_{\text{slack}}^{\text{min}}), \quad (63)$$

$$\nu^c \leq \frac{P_{Tij}^{\text{max}} - |PF_{ij}|}{P_{Tij}^{\text{max}}}, \forall (i, j) \in \mathcal{E}, \quad (64)$$

$$\nu^c \leq P_{\text{slack}}^u, \quad (65)$$

$$\nu^c \leq P_{\text{slack}}^l, \quad (66)$$

where PF_{ij} denotes the power flow on branch $(i, j) \in \mathcal{E}$. P_{slack}^u and P_{slack}^l represent the relative upper and lower bounds redundancy on slack bus, respectively. Constraint (52) describes the convex

⁸For the critical branch constraints not in \mathcal{E} , it is possible to encounter such load input and generation solution profiles using the DNN scheme with the upper/lower bounds truncate ReLU functions in (6) at output layer under the worst-case scenarios with which the power flow on branch i exceeds its transmission limit.

polytope of P_D . Constraints (42) and (45)-(46) denote the feasibility of the corresponding P_G . Constraints (64)-(66) enforce ν^c as the maximum calibration rate. We employ the KKT-based approach in Sec. 4.1 to solve the above bi-level problem and obtain the calibration rate for DeepOPF+.

L.3 DNN LOSS FUNCTION IN DC-OPF PROBLEM

When adopting the *Adversarial-Sample Aware* algorithm, we design the loss function \mathcal{L} consisting of two parts to guide the training process.

Similar to (Pan et al., 2019), we first represent the feasible active power generation P_{Gi} that satisfies (42) as:

$$P_{Gi} = P_{Gi}^{\min} + \alpha_i \cdot (P_{Gi}^{\max} - P_{Gi}^{\min}), \alpha_i \in [0, 1], i \in \mathcal{G}. \quad (67)$$

Therefore, instead of predicting $\{P_{Gi}\}_{i \in \mathcal{G} \setminus n_0}$, we use DNNs to generate the corresponding scaling factors and reconstruct the $\{P_{Gi}\}_{i \in \mathcal{G} \setminus n_0}$ and remaining variables in implementation. The first term of the loss function is the sum of mean square error between the generated scaling factors $\hat{\alpha}_i$ and the reference ones α_i of the optimal solutions:

$$\mathcal{L}_{P_G} = \frac{1}{|\mathcal{G} - 1|} \sum_{i \in \mathcal{G} \setminus n_0} (\hat{\alpha}_i - \alpha_i)^2. \quad (68)$$

The second part consists of penalty terms (denoted as \mathcal{L}_{pen}) as the summation of the violations for line flow limits and slack bus generation:

$$\begin{aligned} \mathcal{L}_{pen}^{\text{line}} &= \frac{1}{|\mathcal{E}|} \sum_{k=1}^{|\mathcal{E}|} \max \left(\left(\mathbf{X} \left(\tilde{P}_G - \tilde{P}_D \right) \right)_k^2 - 1, 0 \right) \\ \mathcal{L}_{pen}^{\text{slack}} &= \frac{1}{|\mathcal{E}|} \max \left(\frac{P_G^{\text{slack}} - P_{\text{slack}}^{\max}}{P_{\text{slack}}^{\max} - P_{\text{slack}}^{\min}}, 0 \right) + \\ &\quad \frac{1}{|\mathcal{E}|} \max \left(\frac{P_{\text{slack}}^{\min} - P_G^{\text{slack}}}{P_{\text{slack}}^{\max} - P_{\text{slack}}^{\min}}, 0 \right) \end{aligned} \quad (69)$$

Here matrix \mathbf{X} is obtained from (47) by dividing each row of matrix $\tilde{\mathbf{B}}_{\text{line}} \tilde{\mathbf{M}}^{-1}$ with the value of corresponding line capacity. The first and second terms of $\mathcal{L}_{pen}^{\text{slack}}$ denote (normalized) the violations of upper bound and lower bound on slack generation, respectively. We remark that after the constraints calibration, the penalty loss is with respect to the adjusted limits. Note here the non-slack generations are always feasible as we predict the $(0, 1)$ scaling factors in (67). The total loss is a weighted sum of the two:

$$\mathcal{L} = w_1 \cdot \mathcal{L}_{P_G} + w_2 \cdot \mathcal{L}_{pen}, \quad (70)$$

where w_1 and w_2 are positive weighting factors representing the balance between prediction error and penalty. We apply the widely-used stochastic gradient descent (SGD) with momentum (Qian, 1999) method to update DNN's parameters (\mathbf{W}, \mathbf{b}) at each iteration.

L.4 RUN-TIME COMPLEXITY OF DEEPOPf+

According to Appendix K, the computational complexity of DeepOPF+ to predict the non-slack generations $\{P_{Gi}\}_{i \in \mathcal{G} \setminus n_0}$ is $\mathcal{O}(B^2)$. Reconstructing the phase angles Φ can be achieved by (46), which requires $\mathcal{O}(B^2)$ operations. Overall, the computational complexity of DeepOPF+ is $\mathcal{O}(B^2)$. For the traditional solver, the computational complexity of interior-point methods for solving DC-OPF is $\mathcal{O}(B^4)$, measured by the number of elementary operations. We remark that the computational complexity of DeepOPF+ is lower than that of traditional algorithms.

M DETAILS OF DEEPOPf+ DESIGN

We present the detailed result of each step in DeepOPF+ design in this appendix.

Table 2: Maximum calibration rates for IEEE Case30/118/300.

	Case30	Case118	Case300
Maximum calibration rate	7.0%	16.7%	21.6%

Table 3: Parameters for test cases.

Case	Number of buses	Number of generators	Number of load buses	Number of branches
Case30	30	6	20	41
Case118	118	19	99	186
Case300	300	69	199	411

* The number of load buses is calculated based on the default load on each bus. A bus is considered a load bus if its default active power consumption is non-zero.

Table 4: Parameters settings of DeepOPF+ for IEEE Case30/118/300

Test case	Variants	Calibration rate	Neurons per hidden layer
Case30	DeepOPF+-3	3.0%	60/30/15
	DeepOPF+-7	7.0%	32/16/8
Case118	DeepOPF+-3	3.0%	200/100/50
	DeepOPF+-7	7.0%	128/64/32
Case300	DeepOPF+-3	3.0%	360/180/90
	DeepOPF+-7	7.0%	256/128/64

Table 5: Preprocessing time to setup DeepOPF+ for IEEE Case30/118/300 in heavy-load regime

Test case	Variants	Determine Calibration rate	Determine DNN size	ASA algorithm	Total time
Case30	DeepOPF+-3	0.2 seconds	0.15 hours	0.83 hour	0.98 hour
	DeepOPF+-7	0.2 seconds	0.15 hours	0.73 hour	0.88 hour
Case118	DeepOPF+-3	20.9 seconds	5.47 hours	7.94 hour	13.42 hour
	DeepOPF+-7	20.9 seconds	5.47 hours	5.31 hour	10.79 hour
Case300	DeepOPF+-3	1185.7 seconds	178.46 hours	25.72 hour	204.51 hour
	DeepOPF+-7	1185.7 seconds	178.46 hours	10.52 hour	189.31 hour

Table 6: Preprocessing time to setup DeepOPF+ for IEEE Case30/118/300 in light-load regime

Test case	Variants	Determine Calibration rate	Determine DNN size	ASA algorithm	Total time
Case30	DeepOPF+-3	0.2 seconds	0.15 hours	0.81 hour	0.96 hour
	DeepOPF+-7	0.2 seconds	0.15 hours	0.72 hour	0.87 hour
Case118	DeepOPF+-3	20.9 seconds	5.47 hours	6.99 hours	12.47 hours
	DeepOPF+-7	20.9 seconds	5.47 hours	4.79 hours	10.27 hours
Case300	DeepOPF+-3	1185.7 seconds	178.46 hours	52.46 hours	231.25 hours
	DeepOPF+-7	1185.7 seconds	178.46 hours	15.82 hours	194.61 hours

First, for determining the maximum calibration rate, the obtained result is shown in Table 2, representing the room for DNN prediction error. We note that the off-the-shell solver returns exact solutions for the problem in (4)-(5).

Table 7: Average cost and runtime of SOTA DNN schemes in heavy-load regime.

Case	Scheme	Average cost (\$/hr)		Average running time (ms)	
		DNN scheme	Ref.	DNN scheme	Ref.
Case30	DNN-P	732.5	732.2	0.58	45.6
	DNN-D	732.4		0.63	
	DNN-W	732.2		53.02	
	DNN-G	732.5		1.78	
	DeepOPF+-3	732.4		0.50	
	DeepOPF+-7	732.9		0.49	
Case118	DNN-P	121074.7	120822.1	2.13	124.9
	DNN-D	121112.1		15.60	
	DNN-W	120822.1		55.33	
	DNN-G	121299.6		7.72	
	DeepOPF+-3	121051.3		0.56	
	DeepOPF+-7	121313.9		0.55	
Case300	DNN-P	926660.6	925955.0	3.33	83.5
	DNN-D	926590.1		57.92	
	DNN-W	925955.0		77.48	
	DNN-G	926512.3		31.55	
	DeepOPF+-3	926198.4		0.61	
	DeepOPF+-7	926500.4		0.60	

Table 8: Average cost and runtime of SOTA DNN schemes in light-load regime.

Case	Scheme	Average cost (\$/hr)		Average running time (ms)	
		DNN scheme	Ref.	DNN scheme	Ref.
Case30	DNN-P	619.8	619.7	0.50	42.4
	DNN-D	619.8		0.50	
	DNN-W	619.7		46.93	
	DNN-G	620.4		1.75	
	DeepOPF+-3	619.9		0.50	
	DeepOPF+-7	619.8		0.49	
Case118	DNN-P	101843.2	101673.0	1.71	115.4
	DNN-D	101873.6		5.02	
	DNN-W	101673.0		55.55	
	DNN-G	102983.3		4.37	
	DeepOPF+-3	101852.3		0.58	
	DeepOPF+-7	102049.3		0.57	
Case300	DNN-P	778342.4	777878.4	1.71	78.7
	DNN-D	778404.3		25.93	
	DNN-W	777878.4		75.77	
	DNN-G	780368.9		32.30	
	DeepOPF+-3	778070.6		0.60	
	DeepOPF+-7	778675.2		0.60	

Second, for determining the sufficient DNN size, we show the change of the difference between maximum relative constraints violation and calibration rate during iterative solving process via the *Danskin's Theorem* in Fig. 3. From Fig. 3, we observe that for all three test cases, the proposed approach succeeds in reaching a relative constraints violation no larger than the corresponding calibration rate Δ , i.e., $\rho \leq \Delta$, indicating that the verified DNNs, i.e., 32/16/8 neurons, 128/64/32 neurons and 256/128/64 neurons, for IEEE 30-/118/300-bus test cases respective, have enough size to guarantee feasibility within the given load input domain of [100%, 130%] of the default load. Note that we can directly construct DNNs to ensure universal feasibility for the three IEEE test cases. We further evaluate the performance of the DNN model obtained following the steps in Sec. 4.2.2 without using the *Adversarial-Sample Aware* algorithm. While ensuring universal feasibility, it suffers from an undesirable optimality loss, up to 2.31% and more than 130% prediction error.

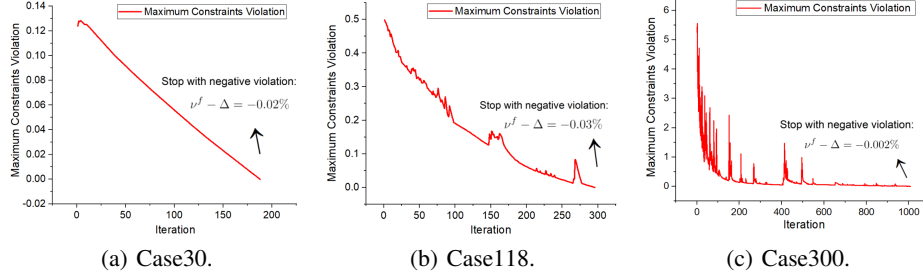


Figure 3: Maximum relative constraints violation compared with calibration rate ($\nu^f - \Delta$) at each iteration for IEEE Case30, Case118, and Case300 test case.

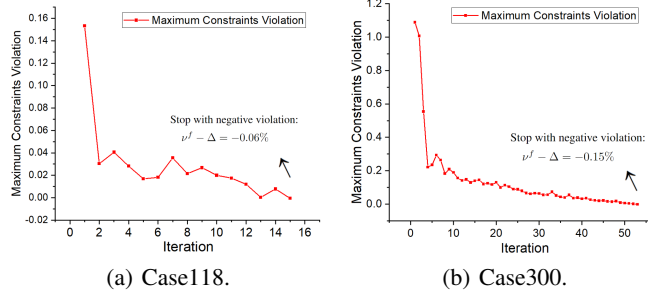


Figure 4: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case118 and IEEE Case300 in light-load regime with 7% calibration rate.

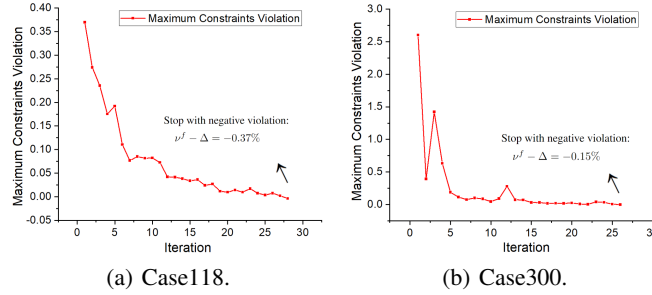


Figure 5: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case118 and IEEE Case300 in heavy-load regime with 7% calibration rate.

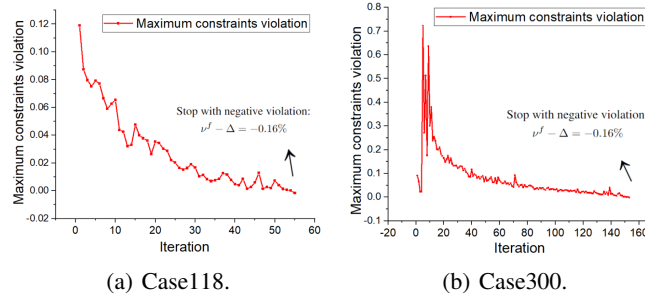


Figure 6: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case30/118/300 in light-load regime with 3% calibration rate.

Third, the DNN models trained with the *Adversarial-Sample Aware* algorithm achieve lower optimality loss (up to 0.19%) while preserving universal feasibility. The observation justifies the effectiveness of *Adversarial-Sample Aware* algorithm. We further present the relative violation ($\nu^f - \Delta$) on IEEE 30-/118/300-bus test cases at each iteration in both light-load and heavy-load regimes for

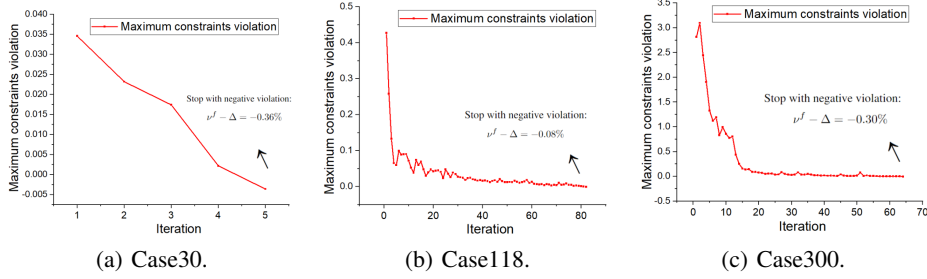


Figure 7: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for IEEE Case30/118/300 in heavy-load regime with 3% calibration rate.

illustration in Fig. 4 and Fig. 5 with a 7% calibration rate. The above observations show that the *Adversarial-Sample Aware* can efficiently achieve universal feasibility guarantee within both light-load and heavy-load regimes for IEEE 118-/300-bus test cases with at most 52 iterations. We remark that for Case30, the initial worst-case violation of the trained DNN with 7% calibration rate is less than zero (-9.28% and -2.93% in light-load and heavy-load regimes respectively) and hence without the need for adversarial training. The results under the 3% calibration rate are presented in Fig. 6 and Fig. 7, for which we observe that the ASA would take a longer number of iterations to achieve the universal feasibility guarantee due to the smaller room for prediction errors, e.g., at most 152 iterations. For Case30 under light-load regime with 3% calibration rate, its initial worst-case violation is less than zero (-7.53%) and hence without the need of ASA iterations.

Furthermore, we present the parameters of three IEEE test cases and the settings of two DeepOPF+ schemes in Table 3 and Table 4 respectively. The detailed runtime and cost and the time to configure the framework are listed in Table 6-Table 8 for each test case. Note that though a single DC-OPF may be efficiently solved by the existing solver, due to increasing uncertainty from renewable generation and flexible load, grid operators now need to solve DC-OPF problems under many scenarios in a short interval, e.g., 1000 scenarios in 1 minutes, to obtain a stochastically optimized solution, e.g., ~ 2 minutes for the iterative solvers to solve a large number of DC-OPF problems for Case118, resulting the fail of real-time operation. In contrast, the developed DNN scheme can return the solution with $\times 228$ speedups, i.e., less than 0.6 seconds in total. In addition, though our method takes additional training efforts, 1) it is conducted offline, once the DNN is configured, it can be continuously applied to many test instances such that the complexity is amortized, e.g., < 0.5 ms for DC-OPF problems if the system operator needs to solve DC-OPF per 5 minutes over 1000 scenarios over a year; 2) as illustrated, the obtained DNN outperforms the existing approaching in avoiding any post-processing and resulting in a lower real-time runtime complexity, showing its advantage; 3) our theoretical analysis shows that the design can always provide the corresponding useful upper/lower bounds in each step of the framework in polynomial time, which can still be utilized for constraints calibration and DNN performance analysis; 4) the process can be further accelerated by applying advanced computation parallel techniques. Finally, we remark that if an impractically large DNN size is required, it would introduce an additional computational challenge, which can require more configuration efforts of the approach and it can be a potential limitation. It is also an interesting direction for solving the constrained program w.r.t. the DNN parameters and determining the sufficient DNN size more efficiently. We would like to leave how to set up the DNNs more efficiently and accelerate the corresponding steps as future work, which is non-trivial and still an open problem in DNN scheme design.

N NON-CONVEX OPTIMIZATION EXAMPLE

We further consider solving a non-convex linearly constrained program with a non-convex objective function and linear constraints adapted from (Donti et al., 2021). We examine this task for illustration:

$$\min_{y \in \mathcal{R}^n} \frac{1}{2} y^T Q y + p^T \sin(y), \text{ s.t. } Ay = x, -h \leq Gy \leq h, \quad (71)$$

for constants problem parameter $Q \in \mathcal{R}^{n \times n}$, $p \in \mathcal{R}^n$, $A \in \mathcal{R}^{n_{\text{eq}} \times n}$, $G \in \mathcal{R}^{n_{\text{ineq}} \times n}$, $h \in \mathcal{R}^{n_{\text{ineq}}}$. Here $x \in \mathcal{R}^{n_{\text{eq}}}$ is the problem input and $y \in \mathcal{R}^n$ denotes the decision variable. n_{ineq} and n_{eq} are the number of inequality and equality constraints. Here we focus on the non-degenerate case such that $n_{\text{eq}} \leq n$. Therefore, the DNN task aims to learn the mapping between x to the optimal y . Similar to (Donti et al., 2021), Q is set to be a diagonal matrix whose diagonal entries are drawn i.i.d. from the uniform distribution on $[0, 1]$. The entries of A, G are drawn i.i.d. from the unit normal distribution. The problem input region of x is set to be $[-1, 1]$ for each dimension. To ensure the problem feasibility, we set $h_i = \sum_j |(GA^+)_{ij}|$, where A^+ is the Moore-Penrose pseudoinverse of A . The feasibility of the problem can be seen that the point $y = A^+x$ is feasible. However, such a point can be generally non-optimal with large optimality loss. In our simulation, we set $n = 50$, $n_{\text{eq}} = 25$, and $n_{\text{ineq}} = 25$. Therefore, the considered optimization has 50 variables, 25 equality constraints, and 100 inequality constraints.

We follow the procedures in the *preventive learning* framework to generate the DNN with universal feasibility guarantee and achieve strong optimality performance.

N.1 REFORMATTING THE PROBLEM WITH ONLY INEQUALITY CONSTRAINTS

We reformulate the non-convex optimization with only $n - n_{\text{eq}}$ independent variables of y_2 . Note that the equality constraints can be reformulated as

$$[A_1, A_2] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = x \quad (72)$$

Here $A_1 \in \mathcal{R}^{n_{\text{eq}} \times n_{\text{eq}}}$, $y_1 \in \mathcal{R}^{n_{\text{eq}}}$ and $A_2 \in \mathcal{R}^{n_{\text{eq}} \times (n - n_{\text{eq}})}$, $y_2 \in \mathcal{R}^{n - n_{\text{eq}}}$. Therefore, given x and y_2 , the corresponding y_1 can be uniquely recovered, i.e., $y_1 = A_1^{-1}(x - A_2 y_2)$. Based on the above reformulation, the inequality constraints are given as

$$[G_1, G_2] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \leq h, \quad -[G_1, G_2] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \leq h \quad (73)$$

and hence

$$G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2) y_2 \leq h, \quad G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2) y_2 \geq -h \quad (74)$$

The objective can be equivalent modified by replacing the terms w.r.t. y_1 to be y_2 from $y_1 = A_1^{-1}(x - A_2 y_2)$. This completes the pre-reformulation of the above non-convex optimization.

N.2 DETERMINE THE MAXIMUM ALLOWABLE CALIBRATION RATE

We first examine that all inequality constraints are critical, i.e., exist a y such that the constraint is binding. We then further determine the maximum calibration rate. From the description in Sec. 4.1, the program to determine the maximum calibration rate is given as

$$\begin{aligned} \min_{x \in [-1, 1]} \max_{y, \nu^c} \quad & \nu^c \\ \text{s.t.} \quad & (74) \end{aligned} \quad (75)$$

$$\nu^c \leq (h_i - (G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2) y_2)_i) / h_i, \quad i = 1, \dots, n_{\text{ineq}}, \quad (76)$$

$$\nu^c \leq (h_i + (G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2) y_2)_i) / h_i, \quad i = 1, \dots, n_{\text{ineq}}. \quad (77)$$

Note that given x , the inner problem is an LP and can be equivalently expressed by its sufficient and necessary KKT conditions. Following the MILP steps in Sec. 4.1, we solve the above program to determine the maximum allowable calibration rate, we observe that the Gurobi solver with the branch-and-bound provides its optimal solution with zero optimality gap within 42ms. The corresponding optimal $\nu^{c*} = 100\%$, implying we can set $h = 0$ such that problem is still feasible for each problem input $x \in [-1, 1]$.

N.3 DETERMINE THE SUFFICIENT DNN SIZE TO GUARANTEE UNIVERSAL FEASIBILITY

In our simulation, we consider a DNN with 3 hidden layers and each layer has 50 neurons. Following the steps in Sec. 4.2, we observe that such a DNN size is sufficient to guarantee universal feasibility. The corresponding program is given as

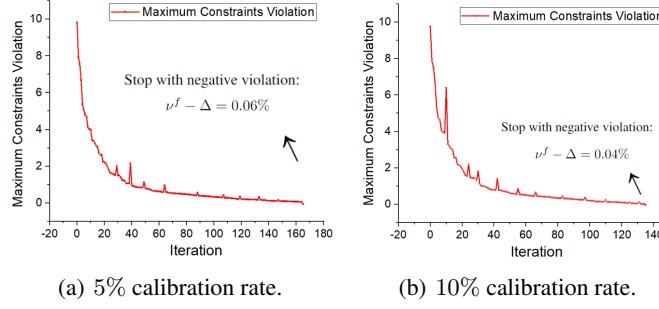


Figure 8: Worst-case violation of *Adversarial-Sample Aware* algorithm at each iteration for the non-convex optimization example with 5% and 10% calibration rate.

$$\min_{\mathbf{W}, \mathbf{b}} \max_{x \in [-1, 1]} \nu^f \quad (78)$$

$$\text{s.t. } (7) - (8), 1 \leq i \leq N_{\text{hid}}, 1 \leq k \leq N_{\text{neu}},$$

$$\nu^f = \max_{i=1, \dots, n_{\text{ineq}}} \left\{ \begin{array}{l} (G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2) \hat{y}_2)_i / h_i \\ -(G_1 A_1^{-1} x + (G_2 - G_1 A_1^{-1} A_2) \hat{y}_2)_i / h_i \end{array} \right\}. \quad (79)$$

Here \hat{y}_2 is the prediction of the DNN. We observe that the tested DNN size is sufficient to guarantee universal feasibility by achieve an upper bound of the relative violation of $\rho - \nu^f$ as -9.3% within ~ 6 minutes. It implies that the tested DNN size is sufficient to guarantee universal feasibility. Recall that the obtained DNN-FG achieves unsatisfactory optimality performance (71.38% optimality loss) as it only focuses on feasibility.

N.4 APPLICATION OF ADVERSARIAL-SAMPLE AWARE TRAINING ALGORITHM

We hence implement the proposed **ASA** training algorithm to further improve the optimality performance of the DNN with 5% and 10% calibration rates respectively. The time to obtain the corresponding (Pre-DNN-5, Pre-DNN-10) with 5% and 10% calibration rate are < 52 minutes and < 44 minutes respectively.

We compare our approach against the classical non-convex optimization solver IPOPT and the other DNN schemes DNN-P, DNN-D, DNN-W, and DNN-G. The number of training data is 15,000, and the number of test data is 3,000. The DNN size is set as 3 hidden layers and each layer has 50 neurons. The results are listed in Table 9, and the worst-case violation at each iteration in the ASA training algorithm are given in Fig. 8. Here the optimality *Loss* metric is calculated as the average of $(\text{DNN objective} - \text{Optimal objective}) / |\text{Optimal objective}|$. The negativity of *Scheme* and *Ref* simply means that the obtained DNN objective and Optimal objective of optimization (71) is negative.

Table 9: Simulation results of different DNN schemes for the non-convex optimization example.

Scheme	Average objective			Average running time (ms)			Feasibility rate (%)	Worst-case violation (%)
	Scheme	Ref.	Loss (%)	Scheme	Ref.	Speedup		
DNN-P	-5.44		0.40	1.36		85.7	39.8	68.3
DNN-D	-5.44		0.42	0.79		117.0	39.8	41.5
DNN-W	-5.47	-5.47	0	86.6	86.6	1.02	100	0
DNN-G	53.69		1076.0	1.00		87.0	100	0
Pre-DNN-5	-5.45		0.34	0.60		144.9	100	0
Pre-DNN-10	-5.43		0.67	0.60		145.3	100	0

* Feasibility rate and Worst-case violation are the results *before* post-processing. Feasibility rates (resp Worst-case violation) after post-processing are 100% (resp 0) for all DNN schemes. We hence report the results before post-processing to better show the advantage of our design. Speedup and Optimality loss are the results *after* post-processing of the final obtained feasible solutions.

* The *correction* step in DNN-D (with 10^{-4} rate) is faster compared with L_1 -projection in DNN-P, resulting in higher speedups.

We remark that our obtained DNN schemes (Pre-DNN-5, Pre-DNN-10) with 5% and 10% calibration rates outperform the existing DNN scheme in ensuring universal feasibility and maintaining minor optimality loss. The speedups of our scheme are also significantly larger than the other methods as post-processing steps to recover solution feasibility are avoided.