End-to-End Feasible Optimization Proxies for Large-Scale Economic Dispatch

Wenbo Chen, *Student Member, IEEE*, Mathieu Tanneau, Pascal Van Hentenryck, *Member, IEEE*

***Abstract*—The paper proposes a novel End-to-End Learning and Repair (E2ELR) architecture for training optimization proxies for economic dispatch problems. E2ELR combines deep neural networks with closed-form, differentiable repair layers, thereby integrating learning and feasibility in an end-to-end fashion. E2ELR is also trained with self-supervised learning, removing the need for labeled data and the solving of numerous optimization problems offline. E2ELR is evaluated on industry-size power grids with tens of thousands of buses using an economic dispatch that co-optimizes energy and reserves. The results demonstrate that the self-supervised E2ELR achieves state-of-the-art performance, with optimality gaps that outperform other baselines by at least an order of magnitude.**

***Index Terms*—Economic Dispatch, Deep Learning, Optimization Proxies**

NOMENCLATURE

A. Sets and indices

$i \in \mathcal{N}$	buses
$e \in \mathcal{E}$	branches
$g \in \mathcal{G}$	generators

B. Variables

p_g	Energy dispatch of generator g
r_g	Reserve dispatch of generator g
ξ_e	Thermal limit violation on branch e

C. Parameters

d_i	Active power demand at bus i
c_g	Production cost function of generator g
$\underline{f}_e, \bar{f}_e$	Lower and upper thermal limits on branch e
M_{th}	Thermal violation penalty cost
$\bar{p}_g$	Maximum output of generator g
$\bar{r}_g$	Maximum reserve of generator g
R	Minimum reserve requirement
Φ	PTDF matrix
$\mathbf{0}$	Vector of all zeros
$\mathbf{e}$	Vector of all ones

I. INTRODUCTION

The optimal power flow (OPF) is a fundamental problem in power systems operations. Its linear approximation, the DC-OPF model, underlies most electricity markets, especially in the US. For instance, MISO uses a security-constrained economic dispatch (SCED) in their real-time markets, which has the DC-OPF model at its core [1].

In recent years, there has been a surge of interest, both from the power systems and Machine-Learning (ML) communities, in developing optimization proxies for OPF, i.e., ML models that approximate the input-output mapping of OPF problems. The main idea is that, once trained, these proxies can be used to generate high-quality solutions orders of magnitude faster than traditional optimization solvers. This capability allows to evaluate a large number of scenarios fast, thereby enabling real-time risk assessment.

There are however three major obstacles to the deployment of optimization proxies. First, ML models are not guaranteed to satisfy the physical and engineering constraints of the model. This causes obvious issues if, for instance, the goal is to use a proxy to evaluate whether the system is able to operate in a safe state. Several approaches have been proposed in the past to address the feasibility of ML predictions, each of which having some advantages and limitations. Second, most results in ML for power systems has considered systems with up to 300 buses, far from the size of actual systems that contain thousands of buses. Moreover, the resulting techniques have not been proven to scale or be accurate enough on large-scale networks. Third, power grids are not static: the generator commitments and the grid topology evolve over time, typically on an hourly basis [2]. Therefore, it is important to be able to (re)train models fast, typically within a few hours at most. This obviously creates a computational bottleneck for approaches that rely on the offline solving of numerous optimization problems to generate training data.

This paper addresses the above challenges by proposing an *End-to-End Learning and Repair* (E2ELR) architecture for a (MISO-inspired) SCED formulation where the prediction and feasibility restoration are integrated in a single ML pipeline and trained jointly. The E2ELR-SCED architecture guarantees feasibility, i.e., it always outputs a feasible solution to the SCED. The E2ELR-SCED architecture achieves these results through *closed-form repair layers*. Moreover, through the use of *self-supervised learning*, the E2ELR-SCED architecture is scalable, both for training and inference, and produces near-optimal solutions to systems with tens of thousands of buses. It also avoids the costly data-generation process of supervised-learning methods. The contributions of the paper can be summarized as follows:

- The paper proposes an E2ELR architecture that produces near-optimal feasible solutions for the SCED formulation that co-optimizes energy and reserve dispatches specified in MISO business manuals.
- The E2ELR architecture is trained end-to-end, leveraging closed-form repair layers. This contrasts to the traditional

approaches where feasibility is restored at inference time, which induces significant losses in accuracy.
- The E2ELR architecture was implemented using both supervised and self-supervised learning. Experimental results show that self-supervised learning outperforms supervised learning in accuracy and overall training time.
- The E2ELR architecture produces State-Of-The-Art results for systems with up to 30,000 buses.

The rest of the paper is organized as follows. Section II surveys the relevant literature. Section III presents an overview of the E2ELR architecture and contrast it with existing approaches. Section IV presents the problem formulation and the E2ELR architecture in detail. Section V presents supervised and self-supervised training. Section VI describes the experiment setting, and Section VII reports numerical results. Section VIII concludes the paper and discusses future research directions.

II. Related works

a) Optimization Proxies for OPF: The majority of the existing literature on OPF proxies employs Supervised Learning (SL) techniques. Each data point (x, y) consists of an OPF instance data (x) and its corresponding solution (y). The training data is obtained by solving a large number –usually tens of thousands– of OPF instances offline. The SL paradigm has successfully been applied both in the linear DCOPF [2]–[8] and nonlinear, non-convex ACOPF [9]–[20] settings. In almost all the above references, the generator commitments and grid topology are assumed to be fixed, with electricity demand being the only source of variability. Therefore, these OPF proxies must be re-trained regularly to capture the hourly changes in commitments and topology that occur in real-life operations [2]. In a SL setting, this comes at a high computational cost because of the need to re-generate training data. Recent works consider active sampling techniques to reduce this burden [21], [22].

Self-Supervised Learning (SSL) has emerged as an alternative to SL that does not require labeled data [23]–[25]. Namely, training OPF proxies in a self-supervised fashion does *not* require the solving of any OPF instance offline, thereby removing the need for (costly) data generation. In [23], the authors train proxies for ACOPF where the training loss consists of the objective value of the predicted solution, plus a penalty term for constraint violations. A similar approach is used in [24] in conjunction with Generative Adversarial Networks (GANs). More recently, Park et al [25] jointly train a primal and dual network by mimicking an Augmented Lagrangian algorithm. Predicting Lagrange multipliers allows for dynamically adjusting the constraint violation penalty terms in the loss function. Current results suggest that SSL-based proxies can match the accuracy of SL-based proxies.

b) Ensuring Feasibility: One major limitation of ML-based OPF proxies is that, in general, the predicted OPF solution violates physical and engineering constraints that govern power flows and ensure safe operations. To alleviate this issue, [6], [10] use a restricted OPF formulation to generate training data, wherein the OPF feasible region is artificially shrunk to ensure that training data consists of interior solutions. In [6], this strategy is combined with a verification step (see also [26]) to ensure the trained models have sufficient capacity to reach a universal approximation. Nevertheless, this requires solving bilevel optimization problems, which is very cumbersome: [6] reports training times in excess of week for a 300-bus system. In addition, it may not be possible to shrink the feasible region in general, e.g., when the lower and upper bounds are the same.

In the context of DCOPF, [2], [3], [7], [8] exploit the fact that an optimal solution can be quickly recovered from an (optimal) active set of constraints. A combined classification-then-regression architecture is proposed in [2], wherein a classification step identifies a subset of variables to be fixed to their lower or upper bound, thus reducing the dimension of the regression task. In [3], [8], the authors predict a full active set, and recover a solution by solving a system of linear equations. Similarly, [7] combine decision trees and active set-based affine policies. Importantly, active set-based approaches may yield infeasible solutions when active constraints are incorrectly classified [8]. Furthermore, correctly identifying an optimal active set becomes harder as problem size increases.

A number of prior work have investigated physics-informed models (e.g., [2], [10]–[12], [14], [15], [23], [24]). This approach augments the training loss function with a term that penalizes constraint violations, and is efficient at reducing – but not eliminating – constraint violations. To better balance feasibility and [11], [12] dynamically adjust the penalty coefficient using ideas from Lagrangian duality. In a similar fashion, [25] uses a primal and a dual networks: the latter predicts optimal Lagrange multipliers, which inform the loss function used to train the former.

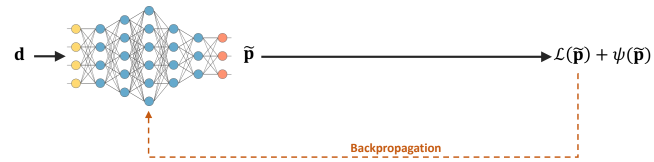
Although physics-informed models generally exhibit lower constraint violations, they still do not produce feasible solutions. Therefore, several works combine an (inexact) OPF proxy with a repair step, that feasibility restoration step. A projection step is used in [4], [11], wherein the (infeasible) predicted solution is projected onto the feasible set of OPF. In DCOPF, this projection is a convex (typically linear or quadratic) problem, whereas the load flow model used in [11] for ACOPF is non-convex. Instead of a projection step, [10] uses an AC power flow solver to recover voltage angles and reactive power dispatch from predicted voltage magnitudes and generator dispatches. This is typically (much) faster than a load flow. However, while the resulting solution satisfies the power flow equations (assuming the solver converges), it may not satisfy all engineering constraints such as the thermal limits of the lines. Finally, [27] uses techniques from state estimation to restore feasibility for ACOPF problems. This approach has not been applied in the context of OPF proxies.

The development of implicit differentiable layers [28] makes it possible to embed feasibility restoration inside the proxy architecture itself, thereby removing the need for post-processing [13], [29], [30]. This allows to train models in an *end-to-end* fashion, i.e., the predicted solution is guaranteed to satisfy constraints. For instance, [29] implement the aforementioned projection step as an implicit layer. However, because they require solving an optimization problem, these implicit layers incur a very high computational cost, both during training and testing. Equality constraints can also be handled implicitly via so-called constraint completion [13], [30]. Namely, a set of independent variables is identified, and dependent variables are recovered by solving the corresponding system of equations, thereby satisfying equality constraints by design. Note that constraint completion requires the set of independent variables to be the same across all instances, which may not hold in general. For instance, changes in generator commitments and/or grid topology may introduce dependencies between previously-independent variables. The difference between [13] and [30] lies in the treatment of inequality constraints. On the one hand, [13] replace a costly implicit layer with cheaper gradient unrolling, which unfortunately does not guarantee feasibility. On the other hand, [30] use gauge functions to define a one-to-one mapping between the unit hypercube, which is easy to enforce with sigmoid activations, and the set of feasible solutions, thereby guaranteeing feasibility. Nevertheless, the latter approach is valid only under restrictive assumptions: all constraints are convex, the feasible set is bounded, and a strictly feasible point is available for each instance.

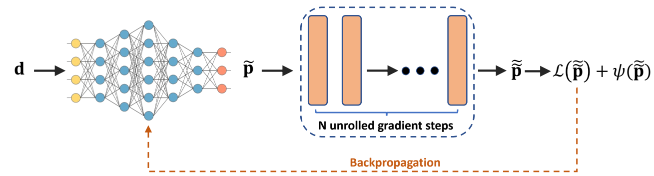
c) Scalability Challenges: There exists a significant gap between the scale of actual power grids, and those used in most academic studies: the former are typically two orders of magnitude larger than the latter. On the one hand, actual power grids comprise thousands to tens of thousands of buses [31], [32]. On the other hand, most academic studies only consider small, artificial power grids with no more than 300 buses. Among the aforementioned works, [8] considers a synthetic NYISO grid with 1814 buses, and only [2], [12], [20] report results on systems with more than 6,000 buses. This discrepancy makes it difficult to extrapolate most existing findings to scenarios encountered in the industry. Indeed, actual power grids exhibit complex behaviors not necessarily captured by small-scale cases [31].

III. Overview of the Proposed Approach

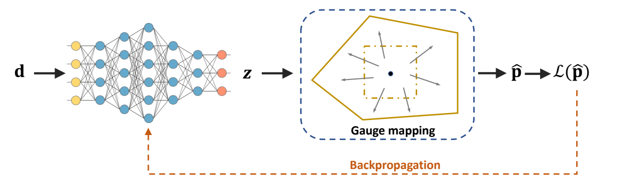
The paper addresses the shortcomings in current literature by combining learning and feasibility restoration in a single E2ELR architecture. Figure 1 illustrates the proposed architecture (Figure 1d), alongside existing architectures from previous works. In contrast to previous works, the proposed E2ELR uses specialized, closed-form repair layers that allow the architecture to scale to industry-size systems. E2ELR is also trained with self-supervised learning, alleviating the need for labeled data and the offline solving of numerous optimization problems. As a result, even for the largest systems considered, the self-supervised E2ELR is trained from scratch in under an hour, and achieves state-of-the-art performance, outperforming other baselines by an order of magnitude. Note that E2ELR also bridges the gap between academic DCOPF formulations and those used in the industry, by including reserve requirements in the ED formulation. To the best of the authors' knowledge, this is the first work to explicitly consider –and offer feasibility guarantees for– reserve requirements in the context of optimization proxies. Moreover, the repair layers are able to accommodate variations in operating parameters such as min/max limits and commitment status of generators, a key aspect of real-life systems overlooked in existing literature.



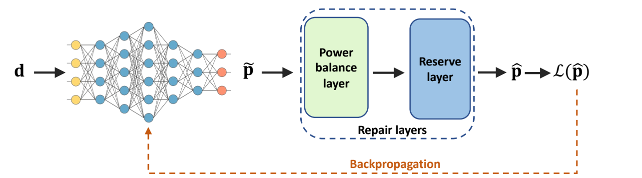
(a) The vanilla DNN architecture without feasibility restoration.



(b) The DC3 architecture with unrolled gradient [13]. The output $\tilde{\tilde{\mathbf{p}}}$ may violate inequality constraints.



(c) The LOOP-LC architecture [30]. The DNN outputs a latent vector $\mathbf{z} \in [0, 1]^n$, which is mapped onto a feasible $\hat{\mathbf{p}}$ via a gauge mapping.



(d) The proposed end-to-end feasible architecture. The output $\hat{\mathbf{p}}$ satisfies all hard constraints.

Fig. 1. Optimization proxy architectures for DCOPF

IV. End-to-end feasible proxies for DCOPF

This section presents the Economic Dispatch (ED) formulation considered in the paper, and introduces new repair layers for power balance and reserve requirement constraints. The repair layers are computationally efficient and differentiable: they can be implemented in standard machine-learning libraries, enabling end-to-end feasible optimization proxies.

a) Problem Formulation: The paper considers an ED formulation with reserve requirements. It is modeled as a linear program of the form

$$\begin{aligned}
\min_{\mathbf{p},\mathbf{r},\xi_{\text{th}}} \quad & c(\mathbf{p}) + M_{\text{th}} \|\xi_{\text{th}}\|_1 && \text{(1a)}\\
\text{s.t.} \quad & \mathbf{e}^\top \mathbf{p} = \mathbf{e}^\top \mathbf{d}, && \text{(1b)}\\
& \mathbf{e}^\top \mathbf{r} \geq R, && \text{(1c)}\\
& \mathbf{p} + \mathbf{r} \leq \bar{\mathbf{p}}, && \text{(1d)}\\
& \mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}, && \text{(1e)}\\
& \mathbf{0} \leq \mathbf{r} \leq \bar{\mathbf{r}}, && \text{(1f)}\\
& \underline{\mathbf{f}} - \xi_{\text{th}} \leq \Phi(\mathbf{p} - \mathbf{d}) \leq \bar{\mathbf{f}} + \xi_{\text{th}}, && \text{(1g)}\\
& \xi_{\text{th}} \geq \mathbf{0}. && \text{(1h)}
\end{aligned}$$

Constraints (1b) and (1c) are the global power balance and minimum reserve requirement constraints, respectively. Constraints (1d) ensure that each generator reserves can be deployed without violating their maximum capacities. Constraints (1e) and (1f) enforce minimum and maximum limits on each generator energy and reserve dispatch. Without loss of generality, the paper assumes that each bus has exactly one generator, each generator minimum output is zero, and $\bar{r}_g \leq \bar{p}_g, \forall g$. Constraints (1g) express the thermal constraints on each branch using a Power Transfer Distribution Factor (PTDF) representation. In this paper, the thermal constraints are soft constraints, i.e., they can be violated but doing so incurs a (high) cost. This is modeled via artificial slack variables ξ_{th} which are penalized in the objective. Treating thermal constraints as soft is in line with economic dispatch formulations used by system operators to clear electricity markets in the US [33], [34]. The PTDF-based formulation is also the state-of-the-art approach used in industry [32], [33]. In typical operations, only a small number of these constraints are active at the optimum. Therefore, efficient implementations add thermal constraints (1g) lazily.

The hard constraints in Problem (1) are the bounds on energy and reserve dispatch, the maximum output, the power balance (1b) and the reserve requirements (1c). Note that bounds on individual variables can easily be enforced in a DNN architecture e.g., via clamping or sigmoid activation. However, it is not trivial to *simultaneously* satisfy variable bounds, power balance and reserve requirements. To address this issue, the rest of this section introduces new, computationally efficient repair layers.

b) The Power Balance Repair Layer: The proposed power balance repair layer takes as input an initial dispatch vector $\mathbf{p}$, which is assumed to satisfy the min/max generation bounds (1e), and outputs a dispatch vector $\tilde{\mathbf{p}}$ that satisfies constraints (1e) and (1b). Formally, let $D = \mathbf{e}^\top \mathbf{d}$, and denote by $\mathcal{H}$ and $\mathcal{S}_D$ the following hypercube and hypersimplex

$$\mathcal{H} = \{\mathbf{p} \in \mathbb{R}^n \mid \mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}\}, \tag{2}$$
$$\mathcal{S}_D = \left\{\mathbf{p} \in \mathbb{R}^n \;\middle|\; \mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}},\ \mathbf{e}^\top \mathbf{p} = D\right\}. \tag{3}$$

Note that $\mathcal{H}$ is the feasible set of constraints (1e), while $\mathcal{S}_D$ is the feasible set of constraints (1b) and (1e). The proposed power balance repair layer, denoted by $\mathcal{P}$, is given by

$$\mathcal{P}(\mathbf{p}) \quad = \begin{cases} (1-\eta^\uparrow)\mathbf{p} + \eta^\uparrow \bar{\mathbf{p}} & \text{if } \mathbf{e}^\top \mathbf{p} < D \\ (1-\eta^\downarrow)\mathbf{p} + \eta^\downarrow \mathbf{0} & \text{if } \mathbf{e}^\top \mathbf{p} \geq D \end{cases} \tag{4}$$

where $\mathbf{p} \in \mathcal{H}$, and $\eta^\uparrow$, $\eta^\downarrow$ are defined as follows:

$$\eta^\uparrow = \frac{\mathbf{e}^\top \mathbf{d} - \mathbf{e}^\top \mathbf{p}}{\mathbf{e}^\top \bar{\mathbf{p}} - \mathbf{e}^\top \mathbf{p}}, \qquad \eta^\downarrow = \frac{\mathbf{e}^\top \mathbf{p} - \mathbf{e}^\top \mathbf{d}}{\mathbf{e}^\top \mathbf{p} - \mathbf{e}^\top \mathbf{0}}. \tag{5}$$

Theorem 1 below shows that $\mathcal{P}$ is well-defined.

Theorem 1. *Assume that $0 < \mathbf{e}^\top \mathbf{d} = D < \mathbf{e}^\top \bar{\mathbf{p}}$ and $\mathbf{p} \in \mathcal{H}$. Then, $\mathcal{P}(\mathbf{p}) \in \mathcal{S}_D$.*

Proof. Let $\tilde{\mathbf{p}} = \mathcal{P}(\mathbf{p})$, and assume $\mathbf{e}^\top \mathbf{p} < D$; the case $\mathbf{e}^\top \mathbf{p} \geq D$ is treated similary. It follows that $\eta^\uparrow \in [0, 1]$, i.e., $\tilde{\mathbf{p}}$ is a convex combination of $\mathbf{p}$ and $\bar{\mathbf{p}}$. Thus, $\tilde{\mathbf{p}} \in \mathcal{H}$. Then,

$$\begin{aligned} \mathbf{e}^\top \tilde{\mathbf{p}} &= (1-\eta^\uparrow)\mathbf{e}^\top \mathbf{p} + \eta^\uparrow \mathbf{e}^\top \bar{\mathbf{p}} \\ &= \eta^\uparrow (\mathbf{e}^\top \bar{\mathbf{p}} - \mathbf{e}^\top \mathbf{p}) + \mathbf{e}^\top \mathbf{p} \\ &= \frac{D - \mathbf{e}^\top \mathbf{p}}{\mathbf{e}^\top \bar{\mathbf{p}} - \mathbf{e}^\top \mathbf{p}} (\mathbf{e}^\top \bar{\mathbf{p}} - \mathbf{e}^\top \mathbf{p}) + \mathbf{e}^\top \mathbf{p} \\ &= D - \mathbf{e}^\top \mathbf{p} + \mathbf{e}^\top \mathbf{p} = D. \end{aligned}$$

Thus, $\tilde{\mathbf{p}}$ satisfies the power balance and $\tilde{\mathbf{p}} \in \mathcal{S}_D$. $\square$

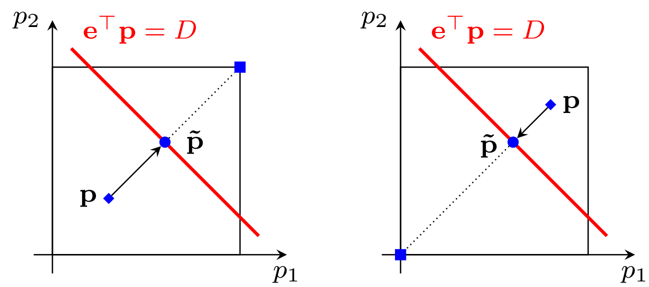


Fig. 2. Illustration of the power balance layer with input $\mathbf{p}$ and output $\tilde{\mathbf{p}}$. Left: $\mathbf{e}^\top \mathbf{p} < D$ (energy shortage) and generators' dispatches are increased. Right: $\mathbf{e}^\top \mathbf{p} > D$ (energy surplus) and generators' dispatches are decreased.

Note that feasible predictions are not modified: if $\mathbf{p} \in \mathcal{S}_D$, then $\mathcal{P}(\mathbf{p}) = \mathbf{p}$. The edge cases not covered by Theorem 1 are handled as follows. When $D \leq 0$ (resp. $D \geq \mathbf{e}^\top \bar{\mathbf{p}}$), each generator is set to its lower (resp. upper) bound; this can be achieved by clamping $\eta^\uparrow$ (resp. $\eta^\downarrow$) to $[0, 1]$. If these inequalities are strict, Problem (1) is trivially infeasible, and $\mathcal{P}(\mathbf{p})$ is the solution that minimizes power balance violations.

The power balance layer is illustrated in Figure 2, for a two-generator system. The layer has an intuitive interpretation as a proportional response mechanism. Indeed, if the initial dispatch $\mathbf{p}$ has an energy shortage, i.e., $\mathbf{e}^\top \mathbf{p} < D$, the output of each generator is increased by a fraction $\eta^\uparrow$ of its upwards headroom. Likewise, if the initial dispatch has an energy surplus, i.e., $\mathbf{e}^\top \mathbf{p} > D$, the output of each generator is decreased by a fraction $\eta^\downarrow$ of it downwards headroom.

Note that $\mathcal{S}_D$ is defined by the combination of bound constraints and one equality constraint. Other works such as [13], [30] handle the latter via equality completion. While this approach satisfies the equality constraint by design, the recovered solution is not guaranteed to satisfy min/max bounds, and may fail to do so in general. In contrast, under the only assumption that $\mathbf{p} \in \mathcal{H}$, the proposed layer (4) jointly enforces *both* constraints, thus alleviating the need for the gradient unrolling of [13] or gauge mapping of [30]. Finally, the proposed layer generalizes to hypersimplices of the form

$$\left\{x \in \mathbb{R}^n \;\middle|\; l \leq x \leq u, a^T x = b\right\}, \tag{6}$$

where $a \in \mathbb{R}^n, b \in \mathbb{R}$ and $l \leq u \in \mathbb{R}^n$ are finite bounds.

c) The Reserve Repair Layer: This section presents the proposed reserve feasibility layer, which ensures feasibility with respect to constraints (1c), (1d), and (1f). The approach first builds a compact representation of these constraints by projecting out the reserve variables $\mathbf{r}$. This makes it possible to consider only the $\mathbf{p}$ variables, which in turn enables a computationally efficient and interpretable feasibility restoration. Let $\mathbf{p} \in \mathcal{S}_D$ be fixed, and consider the problem of maximizing total reserves, which reads

$$\max_{\mathbf{r}} \quad \mathbf{e}^\top \mathbf{r} \tag{7a}$$
$$\text{s.t.} \quad \mathbf{r} \leq \bar{\mathbf{p}} - \mathbf{p}, \tag{7b}$$
$$0 \leq \mathbf{r} \leq \bar{\mathbf{r}}. \tag{7c}$$

Since $\mathbf{p}$ is fixed, constraints (7b)–(7c) reduce to simple variable bounds on the $\mathbf{r}$ variables. It then immediately follows that the optimal solution to Problem (7) is given by

$$r_g^* = \min\{\bar{r}_g, \bar{p}_g - p_g\}, \ \forall g. \tag{8}$$

This observation is used to project out the reserve variables as stated in Lemma 1 below.

Lemma 1. *Let $\mathbf{p} \in \mathcal{S}_D$. There exists reserves $\mathbf{r}$ such that $(\mathbf{p}, \mathbf{r})$ is feasible for Problem (1) if and only if*

$$\sum_g \min\{\bar{r}_g, \bar{p}_g - p_g\} \geq R. \tag{9}$$

Proof. The proof follows from the fact that $(\mathbf{p}, \mathbf{r})$ is feasible for Problem (1) if and only if Problem (7) has an objective value not smaller than R. Substituting the optimal solution given in Eq. (8), this last statement is exactly equivalent to $\sum_g \min\{\bar{r}_g, \bar{p}_g - p_g\} \geq R$. $\square$

The proposed reserve repair layer builds on the power balance repair layer of Section IV-0b, and on the compact formulation of Eq. (9). Namely, it takes as input $\mathbf{p} \in \mathcal{S}_D$, and outputs $\mathcal{R}(\mathbf{p}) \in \mathcal{S}_D$ that satisfies Eq. (9). Given $\mathcal{R}(\mathbf{p})$, reserve variables can be recovered in $O(|\mathcal{G}|)$ time using Eq. (8).

The reserve repair layer is presented in Algorithm 1. First, a tentative reserve allocation is computed using Eq. 8, and the corresponding reserve shortage Δ_R is computed. Then, generators are split into two groups $G^\uparrow$ and $\mathcal{G}^\downarrow$. Generators in $\mathcal{G}^\uparrow$ are those for which constraint (1f) is active: their dispatch can be increased without having to reduce their reserves. Generators in $\mathcal{G}^\downarrow$ are those for which constraint (1d) is active: one must reduce their energy dispatch to increase their reserves. Then, the algorithm computes the maximum possible increase ($\Delta^\uparrow$) and decrease ($\Delta^\downarrow$) in energy dispatch for the two groups. Finally, each generator energy dispatch is increased (resp. decreased) proportionately to its increase (resp. decrease) potential so as to meet total reserve requirements. The total increase in energy dispatch is equal to the total decrease, so that power balance is always maintained.

The reserve feasibility recovery is illustrated in Figure 3 for a two-generator system. While it is easy to verify that $\mathcal{R}(\mathbf{p}) \in \mathcal{S}_D$, it is less clear whether $\mathcal{R}(\mathbf{p})$ satisfies Eq. (9). Theorem 2 provides the theoretical guarantee that either $\mathcal{R}(\mathbf{p})$ satisfies Eq. (9), or no feasible solution to Problem (1) exists. Because of space limitations, the proof is given in the appendix.

Theorem 2. *Let $\mathbf{p} \in \mathcal{S}_D$. Then, $\mathcal{R}(\mathbf{p}) \in \mathcal{S}_D$. Furthermore, $\mathcal{R}(\mathbf{p})$ satisfies Eq. (9) if and only if Problem (1) is feasible.*

Theorem 2 also provides a fast proof of (in)feasibility for Problem (1): it suffices to evaluate $\mathcal{R}(\mathbf{p})$ for any $\mathbf{p} \in \mathcal{S}_D$ and

Algorithm 1 Reserve Repair Layer

Require: Initial prediction $\mathbf{p} \in \mathcal{S}_D$, maximum limits $\bar{\mathbf{p}}, \bar{\mathbf{r}}$, reserve requirement R

$\Delta_R \leftarrow R - \sum_g \min\{\bar{r}_g, \bar{p}_g - p_g\}$

$\mathcal{G}^\uparrow \leftarrow \{g \mid p_g \leq \bar{p}_g - \bar{r}_g\}$

$\mathcal{G}^\downarrow \leftarrow \{g \mid p_g > \bar{p}_g - \bar{r}_g\}$

$\Delta^\uparrow \leftarrow \sum_{g \in \mathcal{G}^\uparrow} (\bar{p}_g - \bar{r}_g) - p_g$

$\Delta^\downarrow \leftarrow \sum_{g \in \mathcal{G}^\downarrow} p_g - (\bar{p}_g - \bar{r}_g)$

$\Delta \leftarrow \max(0, \min(\Delta_R, \Delta^\uparrow, \Delta^\downarrow))$

$\alpha^\uparrow \leftarrow \Delta/\Delta^\uparrow$, $\alpha^\downarrow \leftarrow \Delta/\Delta^\downarrow$,

Energy dispatch adjustment

$$\tilde{p}_g = \begin{cases} (1-\alpha^\uparrow)p_g + \alpha^\uparrow(\bar{p}_g - \bar{r}_g) & \forall g \in \mathcal{G}^\uparrow \\ (1-\alpha^\downarrow)p_g + \alpha^\downarrow(\bar{p}_g - \bar{r}_g) & \forall g \in \mathcal{G}^\downarrow \end{cases}$$

return $\mathcal{R}(\mathbf{p}) = \tilde{\mathbf{p}}$

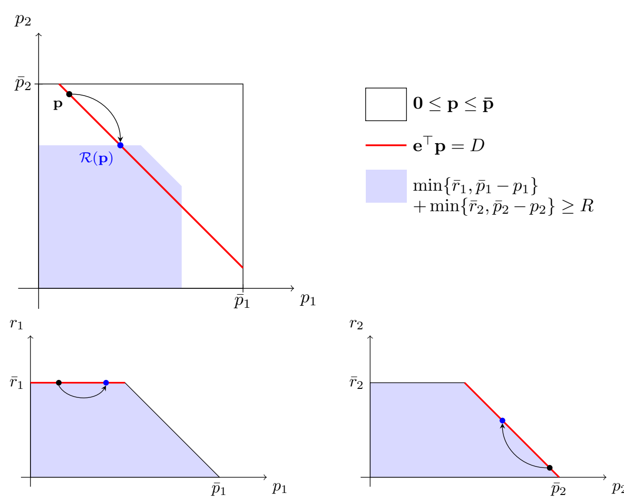


Fig. 3. Illustration of the reserve feasibility layer for $\bar{\mathbf{p}}$=$(1,1)$, $\bar{\mathbf{r}}$=$(0.5,0.5)$, D=1.1, R=0.8 and the initial prediction $\mathbf{p}$=$(0.15,0.95)$. The recovered feasible dispatch is $\tilde{\mathbf{p}}$=$(0.4,0.7)$. Top: effect of the layer in the (p_1,p_2) space. Bottom: effect of the layer on each generator (individually). The active constraint is shown in red. Generator 1 is in $\mathcal{G}^\uparrow$ and generator 2 is in $\mathcal{G}^\downarrow$.

check Eq. (9). This can have applications beyond optimization proxies, e.g., to quickly evaluate a large number of scenarios for potential reserve violations.

d) End-to-end Feasible Training: The repair layers are combined with a Deep Neural Network (DNN) architecture to provide an end-to-end feasible ML model, i.e., a differentiable architecture that is guaranteed to output a feasible solution to Problem (1) (if and only if one exists). The resulting architecture is illustrated in Figure 4. The proxy takes as input the vector of nodal demand $\mathbf{d}$. The DNN architecture consists of fully-connected layers with ReLU activation, and a final layer with sigmoid activations to enforce bound constraints on $\mathbf{p}$. Namely, the last layer outputs $\mathbf{z} \in [0,1]^n$, and $\tilde{\mathbf{p}} = \mathbf{z} \cdot \bar{\mathbf{p}}$ satisfies constraints (1e). Then, this initial prediction $\tilde{\mathbf{p}}$ is fed to the repair layers that restore the feasibility of the power balance and reserve requirements. The final prediction $\hat{\mathbf{p}}$ is feasible for Problem (1).

The power balance and reserve feasibility layers only require elementary arithmetic and logical operations, all of

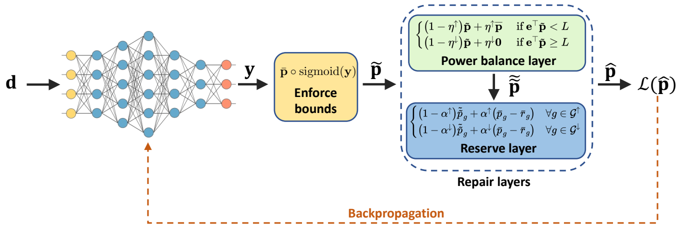


Fig. 4. The Proposed End-To-End Feasible Architecture.

which are supported by mainstream ML libraries like PyTorch and TensorFlow. Therefore, it can be implemented as a layer of a generic artificial neural network model trained with back-propagation. Indeed, these layers are differentiable almost everywhere with informative (sub)gradients. Finally, the proposed feasibility layers can be used as a stand-alone, post-processing step to restore feasibility of any dispatch vector that satisfies generation bounds. This can be used for instance to build fast heuristics with feasibility guarantees.

V. Training Methodology

This section describes the supervised learning (SL) and self-supervised learning (SSL) approaches for training optimization proxies. The difference between these two paradigms lies in the choice of loss function for training, not in the model architecture. Denote by $\mathbf{x}$ the input data of Problem (1), i.e.,

$$\mathbf{x} = (c, \mathbf{d}, R, \bar{\mathbf{p}}, \bar{\mathbf{r}}, \Phi, \underline{\mathbf{f}}, \overline{\mathbf{f}}, M_{\text{th}}),$$

and recall, from Section IV-0c, that it is sufficient to predict the (optimal) value of variables $\mathbf{p}$. Denote by f_θ the mapping of a DNN architecture with trainable parameters θ; given an input $\mathbf{x}$, $f_\theta(\mathbf{x})$ predicts a generator dispatch $\hat{\mathbf{p}}$.

Consider a dataset of N data points

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{p}^{(1)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{p}^{(N)}\right) \right\}, \tag{10}$$

where each data point corresponds to an instance of Problem (1) and its solution, i.e., $\mathbf{x}^{(i)}$ and $\mathbf{p}^{(i)}$ denote the input data and solution of instance $i \in \{1, ..., N\}$, respectively. The training of the DNN f_θ can be formalized as the optimization problem

$$\theta^* = \underset{\theta}{\text{arg-min}} \quad \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\hat{\mathbf{p}}^{(i)}, \mathbf{p}^{(i)}\right), \tag{11}$$

where $\hat{\mathbf{p}}^{(i)} = f_{\theta^*}(\mathbf{x}^{(i)})$ is the prediction for instance i, and $\mathcal{L}$ denotes the loss function. The rest of this section describes the choice of $\mathcal{L}$ for the SL and SSL settings.

a) Supervised Learning: The supervised learning loss $\mathcal{L}^{\text{SL}}$ has the form

$$\mathcal{L}^{\text{SL}}(\hat{\mathbf{p}}, \mathbf{p}) = \varphi^{\text{SL}}(\hat{\mathbf{p}}, \mathbf{p}) + \lambda\psi(\hat{\mathbf{p}}), \tag{12}$$

where $\varphi^{\text{SL}}(\hat{\mathbf{p}}, \mathbf{p})$ penalizes the distance between the predicted and the target (ground truth) solutions, and $\psi(\hat{\mathbf{p}})$ penalizes constraint violations. The paper uses the Mean Absolute Error (MAE) on energy dispatch, i.e.,

$$\varphi^{\text{SL}}(\hat{\mathbf{p}}, \mathbf{p}) = \frac{1}{|\mathcal{G}|} \|\hat{\mathbf{p}} - \mathbf{p}\|_1. \tag{13}$$

Note that other loss functions, e.g., Mean Squared Error (MSE), could be used instead. The term $\psi(\hat{\mathbf{p}})$ penalizes power balance violations and reserve shortages as follows:

$$\psi(\hat{\mathbf{p}}) = M_{\text{pb}}|\mathbf{e}^\top \mathbf{d} - \mathbf{e}^\top \hat{\mathbf{p}}| + M_{\text{r}}\xi_{\text{r}}(\hat{\mathbf{p}}), \tag{14}$$

where M_{pb} and M_{r} are penalty coefficients, and ξ_{r} denotes the reserve shortages, i.e.,

$$\xi_{\text{r}}(\hat{\mathbf{p}}) = \max\left\{0, R - \sum_g \min(\bar{r}_g, \bar{p}_g - \hat{p}_g)\right\}. \tag{15}$$

The penalty term ψ is set to zero for end-to-end feasible models. Finally, while thermal constraints are soft, preliminary experiments found that including thermal violations in the loss function yields more accurate models. Therefore, the final loss considered in the paper is

$$\mathcal{L}^{\text{SL}}(\hat{\mathbf{p}}, \mathbf{p}) = \varphi^{\text{SL}}(\hat{\mathbf{p}}, \mathbf{p}) + \lambda\psi(\hat{\mathbf{p}}) + \mu M_{\text{th}}\|\xi_{\text{th}}(\hat{\mathbf{p}})\|_1, \tag{16}$$

where $\xi_{\text{th}}(\hat{\mathbf{p}})$ denotes thermal violations (Eq. (1g)).

b) Self-Supervised Learning: SSL does not require labelled data and its training loss directly minimizes the original objective value and includes a term penalizing constraints violations. The loss function $\mathcal{L}^{\text{SSL}}$ has the form

$$\mathcal{L}^{\text{SSL}}(\hat{\mathbf{p}}, \mathbf{p}) = \varphi^{\text{SSL}}(\hat{\mathbf{p}}) + \lambda\psi(\hat{\mathbf{p}}), \tag{17}$$

where $\psi(\hat{\mathbf{p}})$ is the same as Eq. (14), and

$$\varphi^{\text{SSL}}(\hat{\mathbf{p}}) = c(\hat{\mathbf{p}}) + M_{\text{th}}\xi_{\text{th}}(\hat{\mathbf{p}}) \tag{18}$$

is the objective value of the predicted solution. Again, the constraint penalty term is dropped when training end-to-end feasible models. Because the SSL loss directly minimizes the (true) cost of the predicted solution, SSL does not require labeled data. This, in turn, eliminates the need to solve numerous instances offline. Nevertheless, care must be take to satisfy constraints to avoid spurious solutions. For instance, in the SCED setting, simply minimizing total generation cost yields the trivial solution $\mathbf{p} = \mathbf{0}$. This highlights the importance of ensuring feasibility, which is the core advantage of the proposed end-to-end feasible architecture.

VI. Experiment Settings

This section presents the numerical experiments used to assess E2ELR. The experiments are conducted on power grids with up to 30,000 buses and uses two variants of Problem 1 with and without reserve requirements. The section presents the data-generation methodology, the baseline ML architectures, performance metrics, and implementation details. Additional information is in Appendices B and C.

a) Data Generation: Instances of Problem (1) are obtained by perturbing reference test cases from the PGLib [35] library. Two categories of instances are generated: instances without any reserve requirements (ED), and instances with reserve requirements (ED-R). The instances are generated as follows. Denote by $\mathbf{d}^{\text{ref}}$ the nodal load vector from the reference PGLib case. ED instances are obtained by perturbing this reference load profile. Namely, for instance i, $\mathbf{d}^{(i)} = \gamma^{(i)} \times \eta^{(i)} \times \mathbf{d}^{\text{ref}}$, where $\gamma^{(i)} \in \mathbb{R}$ is a global scaling factor, $\eta \in \mathbb{R}^{|\mathcal{N}|}$ denotes load-level multiplicative white noise,

TABLE I
SELECTED TEST CASES FROM PGLIB

System	$\|\mathcal{N}\|$	$\|\mathcal{E}\|$	$\|\mathcal{G}\|$	$D_{\text{ref}}{}^{\dagger}$	α_{r}
`ieee300`	300	411	69	23.53	34.16%
`pegase1k`	1354	1991	260	73.06	19.82%
`rte6470`	6470	9005	761	96.59	14.25%
`pegase9k`	9241	16049	1445	312.35	4.70%
`pegase13k`	13659	20467	4092	381.43	1.32%
`goc30k`	30000	35393	3526	117.74	4.68%

†Total active power demand in reference PGLib case, in GW.

and the multiplications are element-wise. For the case at hand, γ is sampled from a uniform distribution $U[0.8, 1.2]$, and, for each load, η is sampled from a log-normal distribution with mean 1 and standard deviation 5%.

ED-R instances are identical to the ED instances, except that reserve requirements are set to a non-zero value. The PGLib library does not include reserve information, therefore, the paper assumes $\bar{r}_g = \alpha_{\text{r}}\bar{p}_g, \forall g \in \mathcal{G}$, where $\alpha_{\text{r}} = 5 \times \|\bar{\mathbf{p}}\|_\infty \times \|\bar{\mathbf{p}}\|_1^{-1}$. This ensures that the total reserve capacity is 5 times larger than the largest generator in the system. Then, the reserve requirements of each instance is sampled uniformly between 100% and 200% of the size of the largest generator, thereby mimicking contingency reserve requirements used in industry.

Table I presents the systems used in the experiments. The table reports: the number of buses ($|\mathcal{N}|$), the number of branches ($|\mathcal{E}|$), the number of generators ($|\mathcal{G}|$), the total active power demand in the reference PGLib case (D_{ref}, in GW), and the value of α_{r} used to determine reserve capacities. The experiments consider test cases with up to 30,000 buses, significantly larger than almost all previous works. Large systems have a smaller value of α_{r} because they contain significantly more generators, whereas the size of the largest generator typically remains in the same order of magnitude. For every test case, $50,000$ instances are generated and solved using Gurobi. This dataset is then split into training, validation, and test sets which comprise 40000, 5000, and 5000 instances.

b) Baseline Models: The proposed end-to-end learning and repair model (E2ELR) is evaluated against three architectures. First, a naive, fully-connected DNN model without any feasibility layer (DNN). This model only includes a sigmoid activation layer to enforce generation bounds (constraint (1e)). Second, a fully-connected DNN model with the DC3 architecture [13] (DC3). This architecture uses a fixed-step unrolled gradient descent to minimize constraint violations; it is however not guaranteed to reach zero violations. Note that the DC3 architecture requires a significant amount of hypertuning to achieve decent results. Third, a fully-connected DNN model, combined with the LOOP-LC architecture from [30] (LOOP). The gauge mapping used in LOOP does not support the compact form of Eq. (9), therefore it is not included in the ED-R experiments. These baseline models are detailed in Appendix B.

c) Performance Metrics: The performance of each ML model is evaluated with respect to several metrics that measure both accuracy and computational efficiency. Given an instance $\mathbf{x}$ with optimal solution $\mathbf{p}^*$ and a predicted solution $\hat{\mathbf{p}}$, the optimality gap is defined as gap $= (\hat{Z} - Z^*) \times |Z^*|^{-1}$, where Z^* is the optimal value of the problem, and $\hat{Z}$ is the objective value of the prediction, plus a penalty for hard constraint violations, i.e.,

$$c(\hat{\mathbf{p}}) + M_{\text{th}}\|\xi_{\text{th}}(\hat{\mathbf{p}})\|_1 + M_{\text{pb}}|\mathbf{e}^\top(\hat{\mathbf{p}} - \mathbf{d})| + M_{\text{r}}\xi_{\text{r}}(\hat{\mathbf{p}}), \quad (19)$$

where $\xi_{\text{r}}(\hat{\mathbf{p}})$ is defined as in function (15). Penalizing hard constraint violations is necessary to ensure a fair comparison between models that output feasible solutions and those that do not. Because all considered models enforce constraints (1d)–(1f), they are not penalized in Eq. (19).

The paper uses realistic penalty prices, based on the values used by MISO in their operations [36], [37]. Namely, the thermal violation penalty price M_{th} is set to \$1500/MW. The power balance violation penalty M_{pb} is set to \$3500/MW, which corresponds to MISO's value of lost load (VOLL). Finally, the reserve shortage penalty M_{r} is set to \$1100/MW, which is MISO's reserve shortage price. The ability of optimization proxies to output feasible solution is measured via the proportion of feasible predictions, which is reported as a percentage over the test set. The paper uses an absolute tolerance of 10^{-4} p.u. to decide whether a constraint is violated; note that this is 100x larger than the default absolute tolerance of optimization solvers. The paper also reports the mean constraint violation of infeasible predictions.

The paper also evaluates the computational efficiency of each ML model and learning paradigm (SL and SSL), as well as that of the repair layers. Computational efficiency is measured by (i) the training time of ML models, including the data-generation time when applicable, and (ii) the inference time. Note that ML models evaluate *batches* of instances, therefore, inference times are reported per batch of 256 instances. The performance of the repair layers presented in Section IV is benchmarked against a standard euclidean projection solved with state-of-the-art optimization software.

Unless specified otherwise, average computing times are arithmetic means; other averages are shifted geometric means

$$\mu_s(x_1, ..., x_n) = \exp\left(\frac{1}{n}\sum_i \log(x_i + s)\right) - s.$$

The paper uses a shift s of 1% for optimality gaps, and 1p.u. for constraint violations.

d) Implementation Details: All optimization problems are formulated in Julia using JuMP [38], and solved with Gurobi 9.5 [39] with a single CPU thread and default parameter settings. All deep learning models are implemented using PyTorch [40] and trained using the Adam optimizer [41]. All models are hyperparameter tuned using a grid search, which is detailed in Appendix C. For each system, the best model is selected on the validation set and the performances on the test set are reported. Experiments are conducted on dual Intel Xeon 6226@2.7GHz machines running Linux, on the PACE Phoenix cluster [42]. The training of ML models is performed on Tesla V100-PCIE GPUs with 16GBs HBM2 RAM.

VII. NUMERICAL RESULTS

a) Optimality Gaps: Table II reports, for the ED and ED-R problems, the mean optimality gap of each ML model, under

TABLE II
MEAN OPTIMALITY GAP (%) ON TEST SET

		ED				ED-R		
Loss	System	DNN	E2ELR	DC3	LOOP	DNN	E2ELR	DC3
SL	ieee300	69.55	**1.42**	3.03	38.93	75.06	**1.52**	2.94
	pegase1k	48.77	**0.74**	2.80	32.53	47.84	**0.74**	2.97
	rte6470	55.13	**1.35**	3.68	50.21	70.57	**1.82**	3.49
	pegase9k	76.06	**0.38**	1.25	33.78	81.19	**0.38**	1.29
	pegase13k	71.14	**0.29**	1.79	52.94	76.32	**0.28**	1.81
	goc30k	194.13	**0.46**	2.75	36.49	136.25	**0.45**	2.35
SSL	ieee300	35.66	**0.74**	2.51	37.78	45.56	**0.78**	2.80
	pegase1k	62.07	**0.63**	2.57	32.20	64.69	**0.68**	2.61
	rte6470	40.73	**1.30**	2.82	50.20	55.16	**1.68**	3.04
	pegase9k	43.68	**0.32**	0.82	33.76	44.74	**0.29**	0.93
	pegase13k	57.58	**0.21**	0.84	52.93	61.28	**0.19**	0.91
	goc30k	108.91	**0.39**	0.72	36.73	93.91	**0.33**	0.71

All gaps are shifted geometric means with a shift of 1%.

TABLE III
POWER BALANCE CONSTRAINT VIOLATION STATISTICS

		ED				ED-R			
		DNN		DC3†		DNN		DC3†	
Loss	System	%feas	viol*	%feas	viol*	%feas	viol*	%feas	viol*
SL	ieee300	0.0%	0.498	100%	0.000	0.0%	0.701	100%	0.253
	pegase1k	0.0%	1.822	65.3%	0.044	0.0%	1.905	64.6%	0.080
	rte6470	0.0%	2.080	8.9%	0.029	0.0%	2.264	6.7%	0.010
	pegase9k	0.0%	6.249	29.1%	0.000	0.0%	5.914	28.9%	0.000
	pegase13k	0.0%	6.630	22.8%	0.001	0.0%	7.792	22.1%	0.001
	goc30k	0.0%	3.805	57.3%	0.029	0.0%	2.313	68.9%	0.005
SSL	ieee300	0.0%	0.576	99.9%	0.156	0.0%	0.732	99.9%	0.288
	pegase1k	0.0%	2.425	62.8%	0.028	0.0%	2.298	39.3%	0.054
	rte6470	0.0%	1.896	5.8%	0.108	0.0%	2.715	5.0%	0.012
	pegase9k	0.0%	7.224	26.6%	0.050	0.0%	7.003	24.7%	0.083
	pegase13k	0.0%	6.408	19.3%	0.011	0.0%	6.817	20.0%	0.007
	goc30k	0.0%	3.176	52.0%	0.014	0.0%	2.609	62.1%	0.016

†with 200 gradient steps. *geometric mean of non-zero violations, in p.u.

TABLE IV
SAMPLING AND TRAINING TIME COMPARISON (ED)

Loss	System	Sample	DNN	E2ELR	DC3	LOOP
SL	ieee300	0.2hr	7min	37min	121min	33min
	pegase1k	0.7hr	8min	14min	41min	19min
	rte6470	5.1hr	11min	30min	73min	18min
	pegase9k	12.7hr	15min	24min	123min	25min
	pegase13k	20.6hr	14min	19min	126min	19min
	goc30k	63.4hr	25min	20min	108min	127min
SSL	ieee300	–	15min	27min	102min	27min
	pegase1k	–	8min	15min	46min	14min
	rte6470	–	9min	17min	42min	15min
	pegase9k	–	18min	20min	100min	29min
	pegase13k	–	17min	18min	125min	15min
	goc30k	–	38min	45min	105min	60min

Sampling (training) times are for 1 CPU (1 GPU). Excludes hypertuning.

TABLE V
SAMPLING AND TRAINING TIME COMPARISON (ED-R)

Loss	System	Sample	DNN	E2ELR	DC3
SL	ieee300	0.2hr	12min	43min	115min
	pegase1k	0.8hr	14min	19min	53 min
	rte6470	4.6hr	14min	19min	71 min
	pegase9k	14.0hr	15min	22min	123min
	pegase13k	22.7hr	16min	27min	126min
	goc30k	65.9hr	32min	39min	129min
SSL	ieee300	–	21min	37min	131min
	pegase1k	–	6min	19min	67min
	rte6470	–	12min	21min	71min
	pegase9k	–	20min	24min	123min
	pegase13k	–	13min	22min	125min
	goc30k	–	52min	53min	128min

Sampling (training) times are for 1 CPU (1 GPU). Excludes hypertuning.

the SL and SSL learning modes. Bold entries denote the best-performing method. Recall that LOOP is not included in ED-R experiments. *E2ELR systematically outperforms all other baselines across all settings.* This stems from two reasons. First, DNN and DC3 exhibit violations of the power balance constraint (1b), which yields high penalties and therefore large optimality gaps. Statistics on power balance violations for DNN and DC3 are reported in Table III. Second, LOOP's poor performance, despite not violating any hard constraint, is because the non-convex gauge mapping used inside the model has an adverse impact on training. Indeed, after a few epochs of training, LOOP gets stuck in a local optimum.

E2ELR, when trained in a self-supervised mode, achieves the best performance. This is because SSL directly minimizes the true objective function, rather than the surrogate supervised loss. With the exception of `rte6470`, the performance of E2ELR improves as the size of the system increases, with the lowest optimality gaps achieved on `pegase13k`, which has the most generators. Note that `rte6470`is a real system from the French transmission grid: it is more congested than other test cases, and therefore harder to learn.

b) Computing Times: Tables IV and V report the sampling and training times for ED and ED-R, respectively. Each table reports the total time for data-generation, which corresponds to the total solving time of Gurobi on a single thread. There is no labeling time for self-supervised models. While training times for SL and SSL are comparable, for a given architecture, the latter does not incur any labeling time. The training time of DC3 is significantly higher than other baselines because of its unrolled gradient steps. These results demonstrate that ML models can be trained efficiently on large-scale systems. Indeed, *the self-supervised E2ELR needs less than an hour of total computing time to achieve optimality gaps under* 0.5% *for systems with thousands of buses.*

Tables VI and VII report, for ED and ED-R, respectively, the average solving time using Gurobi (GRB) and average inference times of ML methods. Recall that the Gurobi's solving times are for a single instance solved on a single CPU core, whereas the ML inference times are reported for a batch of 256 instances on a GPU. Also note that the number of gradient steps used by DC3 to recover feasibility is set to 200 for inference (compared to 50 for training).

On systems with more than 6,000 buses, DC3 is typically 10–30 times slower than other baselines, again due to its unrolled gradient steps. In contrast, the DNN, E2ELR, and LOOP architectures all require in the order of 5–10 milliseconds to evaluate a batch of 256 instances. For the largest systems, this represents about 25,000 instances per second, on a single GPU. Solving the same volume of instances with Gurobi would require more than a day on a single CPU. Getting this time down to the order of seconds, thereby matching the speed of ML proxies, would require thousands of CPUs, which comes at high financial and environmental costs.

c) Benefits of End-to-End Training: Tables VIII and IX further demonstrate the benefits of training end-to-end feasible models: they report, for ED and ED-R problems, the optimality

TABLE VI
SOLVING AND INFERENCE TIME COMPARISON (ED)

Loss	System	DNN	E2ELR	DC3†	LOOP	GRB*
SL	ieee300	3.4ms	4.5ms	15.4ms	5.3ms	12.1ms
	pegase1k	4.1ms	5.3ms	18.3ms	5.9ms	51.5ms
	rte6470	5.1ms	6.6ms	35.3ms	7.1ms	364.4ms
	pegase9k	6.0ms	7.3ms	91.5ms	8.2ms	913.5ms
	pegase13k	7.3ms	8.3ms	523.6ms	13.9ms	1481.3ms
	goc30k	9.5ms	10.0ms	443.0ms	14.4ms	4566.9ms
SSL	ieee300	3.4ms	6.0ms	15.1ms	5.2ms	12.1ms
	pegase1k	4.0ms	5.3ms	18.4ms	5.8ms	51.5ms
	rte6470	5.9ms	6.5ms	36.7ms	9.5ms	364.4ms
	pegase9k	6.1ms	7.0ms	93.2ms	10.3ms	913.5ms
	pegase13k	7.1ms	8.2ms	561.2ms	12.8ms	1481.3ms
	goc30k	10.9ms	11.7ms	444.2ms	21.7ms	4566.9ms

†with 200 gradient steps. *solution time per instance (single thread). All ML inference times are for a batch of 256 instances.

TABLE VII
SOLVING AND INFERENCE TIME COMPARISON (ED-R)

Loss	System	DNN	E2ELR	DC3†	GRB*
SL	ieee300	3.9ms	6.5ms	16.5ms	12.6ms
	pegase1k	4.5ms	6.0ms	18.9ms	56.5ms
	rte6470	5.7ms	10.4ms	36.1ms	333.6ms
	pegase9k	6.3ms	7.7ms	91.6ms	1008.0ms
	pegase13k	8.3ms	10.7ms	531.2ms	1632.7ms
	goc30k	9.3ms	11.1ms	438.7ms	4745.7ms
SSL	ieee300	3.9ms	7.6ms	17.6ms	12.6ms
	pegase1k	4.4ms	5.9ms	19.1ms	56.5ms
	rte6470	6.4ms	10.5ms	37.3ms	333.6ms
	pegase9k	7.1ms	8.3ms	92.9ms	1008.0ms
	pegase13k	7.8ms	8.9ms	522.4ms	1632.7ms
	goc30k	10.2ms	12.4ms	435.8ms	4745.7ms

†with 200 gradient steps. *solution time per instance (single thread). All ML inference times are for a batch of 256 instances.

TABLE VIII
COMPARISON OF OPTIMALITY GAPS (%) WITH AND WITHOUT FEASIBILITY RESTORATION (ED)

			DNN			DC3		
Loss	System	E2ELR	–	RL	EP	–	RL	EP
SL	ieee300	**1.42**	69.55	36.33	36.37	3.03	3.03	3.03
	pegase1k	**0.74**	48.77	3.98	3.94	2.80	2.44	2.44
	rte6470	**1.35**	55.13	21.28	21.41	3.68	3.36	3.36
	pegase9k	**0.38**	76.06	34.61	34.65	1.25	1.24	1.24
	pegase13k	**0.29**	71.14	32.70	32.72	1.79	1.79	1.79
	goc30k	**0.46**	194.13	57.53	57.41	2.75	2.45	2.45
SSL	ieee300	**0.74**	35.66	3.82	3.73	2.51	2.51	2.51
	pegase1k	**0.63**	62.07	3.24	3.25	2.57	2.35	2.35
	rte6470	**1.30**	40.73	11.52	11.47	2.82	2.10	2.09
	pegase9k	**0.32**	43.68	3.20	3.22	0.82	0.64	0.64
	pegase13k	**0.21**	57.58	20.59	20.59	0.84	0.81	0.81
	goc30k	**0.39**	108.91	7.89	7.89	0.72	0.62	0.62

gaps achieved by DNN and DC3 *after applying a repair step at inference time*. Two repair mechanisms are compared: the proposed Repair Layers (RL) and a Euclidean Projection (EP). The tables also report the mean gap achieved by E2ELR as a reference baseline. The results can be summarized as follows. First, the additional feasibility restoration improves the quality of the initial prediction. This is especially true for DNN, which exhibited the largest constraint violations (see Table III): optimality gaps are improved by a factor 2–20, but remain very high nonetheless. Second, the two repair mechanisms yield similar optimality gaps. For DC3, there is virtually no

TABLE IX
COMPARISON OF OPTIMALITY GAPS (%) WITH AND WITHOUT FEASIBILITY RESTORATION (ED-R)

			DNN			DC3		
Loss	System	E2ELR	–	RL	EP	–	RL	EP
SL	ieee300	**1.52**	75.06	30.47	30.49	2.94	2.94	2.94
	pegase1k	**0.74**	47.84	2.52	2.50	2.97	2.34	2.34
	rte6470	**1.82**	70.57	30.20	29.90	3.49	3.32	3.29
	pegase9k	**0.38**	81.19	41.34	41.40	1.29	1.29	1.29
	pegase13k	**0.28**	76.32	30.00	30.02	1.81	1.81	1.81
	goc30k	**0.45**	136.25	53.34	53.41	2.35	2.31	2.31
SSL	ieee300	**0.78**	45.56	4.50	4.34	2.80	2.78	2.78
	pegase1k	**0.68**	64.69	4.56	4.44	2.61	1.87	1.87
	rte6470	**1.68**	55.16	9.76	9.43	3.04	2.75	2.70
	pegase9k	**0.29**	44.74	4.33	4.33	0.93	0.66	0.66
	pegase13k	**0.19**	61.28	21.35	21.32	0.91	0.89	0.89
	goc30k	**0.33**	93.91	10.00	9.98	0.71	0.64	0.64

TABLE X
COMPUTING TIME OF FEASIBILITY RESTORATION USING FEASIBILITY LAYERS (FL) AND EUCLIDEAN PROJECTION (EP).

Problem	System	RL	EP	Speedup
ED	ieee300	0.13μs	0.45ms	3439x
	pegase1k	0.55μs	1.41ms	2572x
	rte6470	1.40μs	3.75ms	2686x
	pegase9k	2.37μs	6.90ms	2911x
	pegase13k	6.42μs	20.71ms	3227x
	goc30k	5.67μs	17.87ms	3155x
ED-R	ieee300	1.06μs	1.00ms	939x
	pegase1k	4.58μs	3.42ms	748x
	rte6470	10.46μs	10.19ms	974x
	pegase9k	20.14μs	18.38ms	913x
	pegase13k	42.67μs	60.73ms	1423x
	goc30k	39.80μs	49.17ms	1236x

Median computing times as measured by BenchmarkTools

difference between RL and EP. Third, across all experiments, even after feasibility restoration, E2ELR remains the best-performing model, with optimality gaps 2–6x smaller than DC3. Table X compares the computing times of the feasibility restoration using either the repair layers (RL) or the euclidean projection (EP). The latter is solved as a quadratic program with Gurobi. All benchmarks are conducted in Julia on a single thread, using the BenchmarkTools utility [43], and median times are reported. The results of Table X show that evaluating the proposed repair layers is three orders of magnitude faster than solving the euclidean projection problem.

VIII. CONCLUSION

The paper proposed a new *End-to-End Learning and Repair* (E2ELR) architecture for training optimization proxies for economic dispatch problems. E2ELR combines deep learning with closed-form, differential repair layers, thereby integrating prediction and feasibility restoration in an end-to-end fashion. The E2ELR architecture can be trained trained with self-supervised learning, removing the need for labeled data and the solving of numerous optimization problems offline. The paper conducted extensive numerical experiments on the ecocomic dispatch of large-scale, industry-size power grids with tens of thousands of buses. It also presented the first study that considers reserve requirements in the context of optimization proxies, reducing the gap between between academic and industry formulations.

The results demonstrate that the combination of E2ELR and self-supervised learning achieves state-of-the-art performance, with optimality gaps that outperform other baselines by at least an order of magnitude. Future research will investigate security-constrained economic dispatch (SCED) formulations, and the extension of repair layers to the nonlinear, non-convex AC Optimal Power Flow problem.

Acknowledgments

This research is partly funded by NSF awards 2007095 and 2112533, and by ARPA-E PERFORM award AR0001136.

References

[1] MISO, "Energy and operating reserve markets," 2022, Business Practices Manual Energy and Operating Reserve Markets.
[2] W. Chen, S. Park, M. Tanneau, and P. Van Hentenryck, "Learning optimization proxies for large-scale security-constrained economic dispatch," *Electric Power Systems Research*, vol. 213, p. 108566, 2022.
[3] Y. Ng, S. Misra, L. A. Roald, and S. Backhaus, "Statistical learning for dc optimal power flow," in *2018 Power Systems Computation Conference (PSCC)*, 2018, pp. 1–7.
[4] X. Pan, T. Zhao, M. Chen, and S. Zhang, "Deepopf: A deep neural network approach for security-constrained dc optimal power flow," *IEEE Transactions on Power Systems*, vol. 36, no. 3, pp. 1725–1735, 2020.
[5] R. Nellikkath and S. Chatzivasileiadis, "Physics-informed neural networks for minimising worst-case violations in dc optimal power flow," in *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2021, pp. 419–424.
[6] T. Zhao, X. Pan, M. Chen, and S. H. Low, "Ensuring DNN solution feasibility for optimization problems with convex constraints and its application to DC optimal power flow problems," *arXiv:2112.08091*, 2021.
[7] A. Stratigakos, S. Pineda, J. M. Morales, and G. Kariniotakis, "Interpretable Machine Learning for DC Optimal Power Flow with Feasibility Guarantees," Mar. 2023, working paper or preprint. [Online]. Available: https://hal.science/hal-04038380
[8] R. Ferrando, L. Pagnier, R. Mieth, Z. Liang, Y. Dvorkin, D. Bienstock, and M. Chertkov, "A Physics-Informed Machine Learning for Electricity Markets: A NYISO Case Study," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.00062
[9] N. Guha, Z. Wang, M. Wytock, and A. Majumdar, "Machine learning for ac optimal power flow," 2019.
[10] A. S. Zamzam and K. Baker, "Learning Optimal Solutions for Extremely Fast AC Optimal Power Flow," in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020, pp. 1–6.
[11] F. Fioretto, T. W. Mak, and P. Van Hentenryck, "Predicting AC Optimal Power Flows: Combining deep learning and lagrangian dual methods," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 630–637.
[12] M. Chatzos, T. W. Mak, and P. Van Hentenryck, "Spatial network decomposition for fast and scalable AC-OPF learning," *IEEE Transactions on Power Systems*, vol. 37, no. 4, pp. 2601–2612, 2021.
[13] P. Donti, D. Rolnick, and J. Z. Kolter, "DC3: A learning method for optimization with hard constraints," in *International Conference on Learning Representations*, 2021.
[14] R. Nellikkath and S. Chatzivasileiadis, "Physics-informed neural networks for ac optimal power flow," *Electric Power Systems Research*, vol. 212, p. 108412, 2022.
[15] X. Pan, M. Chen, T. Zhao, and S. H. Low, "Deepopf: A feasibility-optimized deep neural network approach for ac optimal power flow problems," *IEEE Systems Journal*, vol. 17, no. 1, pp. 673–683, 2023.
[16] X. Pan, W. Huang, M. Chen, and S. H. Low, "DeepOPF-AL: Augmented Learning for Solving AC-OPF Problems with Multiple Load-Solution Mappings," *arXiv:2206.03365*, 2022.
[17] M. Zhou, M. Chen, and S. H. Low, "DeepOPF-FT: One Deep Neural Network for Multiple AC-OPF Problems With Flexible Topology," *IEEE Transactions on Power Systems*, vol. 38, no. 1, pp. 964–967, 2022.
[18] S. Liu, C. Wu, and H. Zhu, "Topology-aware Graph Neural Networks for Learning Feasible and Adaptive AC-OPF Solutions," *IEEE Transactions on Power Systems*, pp. 1–11, 2022.
[19] T. Falconer and L. Mones, "Leveraging Power Grid Topology in Machine Learning Assisted Optimal Power Flow," *IEEE Transactions on Power Systems*, pp. 1–13, 2022.
[20] S. Park, W. Chen, T. W. Mak, and P. Van Hentenryck, "Compact Optimization Learning for AC Optimal Power Flow," *arXiv:2301.08840*, 2023.
[21] M. Klamkin, M. Tanneau, T. W. K. Mak, and P. Van Hentenryck, "Active bucketized learning for acopf optimization proxies," 2022.
[22] Z. Hu and H. Zhang, "Optimal power flow based on physical-model-integrated neural network with worth-learning data generation," 2023.
[23] W. Huang and M. Chen, "DeepOPF-NGT: Fast No Ground Truth Deep Learning-Based Approach for AC-OPF Problems," in *ICML 2021 Workshop Tackling Climate Change with Machine Learning*, 2021. [Online]. Available: https://www.climatechange.ai/papers/icml2021/18
[24] J. Wang and P. Srikantha, "Fast optimal power flow with guarantees via an unsupervised generative model," *IEEE Transactions on Power Systems*, 2022.
[25] S. Park and P. Van Hentenryck, "Self-supervised primal-dual learning for constrained optimization," *arXiv:2208.09046*, 2022.
[26] A. Venzke and S. Chatzivasileiadis, "Verification of neural network behaviour: Formal guarantees for power system applications," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 383–397, 2021.
[27] B. Taheri and D. K. Molzahn, "Restoring ac power flow feasibility from relaxed and approximated optimal power flow models," 2023.
[28] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in neural information processing systems*, vol. 32, 2019.
[29] M. Kim and H. Kim, "Projection-aware deep neural network for dc optimal power flow without constraint violations," in *2022 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2022, pp. 116–121.
[30] M. Li, S. Kolouri, and J. Mohammadi, "Learning to solve optimization problems with hard linear constraints," *arXiv:2208.10611*, 2022.
[31] C. Josz, S. Fliscounakis, J. Maeght, and P. Panciatici, "AC Power Flow Data in MATPOWER and QCQP Format: iTesla, RTE Snapshots, and PEGASE," 2016.
[32] J. Holzer, Y. Chen, Z. Wu, F. Pan, and A. Veeramany, "Fast Simultaneous Feasibility Test for Security Constrained Unit Commitment," 2022. [Online]. Available: https://dx.doi.org/10.36227/techrxiv.20280384.v1
[33] X. Ma, H. Song, M. Hong, J. Wan, Y. Chen, and E. Zak, "The security-constrained commitment and dispatch for midwest iso day-ahead co-optimized energy and ancillary service market," in *2009 IEEE Power & Energy Society General Meeting*, 2009, pp. 1–8.
[34] MISO, "Real-time energy and operating reserve market software formulations and business logic," 2022, Business Practices Manual Energy and Operating Reserve Markets Attachment D.
[35] S. Babaeinejadsarookolaee, A. Birchfield, R. D. Christie, C. Coffrin, C. DeMarco, R. Diao, M. Ferris, S. Fliscounakis, S. Greene, R. Huang, C. Josz, R. Korab, B. Lesieutre, J. Maeght, T. W. K. Mak, D. K. Molzahn, T. J. Overbye, P. Panciatici, B. Park, J. Snodgrass, A. Tbaileh, P. Van Hentenryck, and R. Zimmerman, "The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms," 2019.
[36] MISO, "Schedule 28 – Demand Curves for TOperating Reserves," 2023. [Online]. Available: https://www.misoenergy.org/legal/tariff/
[37] ——, "Schedule 28A – Demand Curves for Transmission Constraints," 2019. [Online]. Available: https://www.misoenergy.org/legal/tariff/
[38] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.
[39] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com
[40] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.
[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.
[42] PACE, *Partnership for an Advanced Computing Environment (PACE)*, 2017. [Online]. Available: http://www.pace.gatech.edu
[43] J. Chen and J. Revels, "Robust benchmarking in noisy environments," *arXiv e-prints*, Aug 2016.
[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.
[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

Appendix

A. Proofs

1) Proof of Theorem 1: Let $\tilde{\mathbf{p}} = \mathcal{P}(\mathbf{p})$, and assume $\mathbf{e}^\top \mathbf{p} < D$. It is immediate that $\eta^\uparrow \in [0,1]$, i.e., $\tilde{\mathbf{p}}$ is a convex combination of $\mathbf{p}$ and $\bar{\mathbf{p}}$. Thus, $\tilde{\mathbf{p}} \in \mathcal{H}$. Then,

$$\begin{aligned}\mathbf{e}^\top \tilde{\mathbf{p}} &= (1-\eta^\uparrow)\mathbf{e}^\top \mathbf{p} + \eta^\uparrow \mathbf{e}^\top \bar{\mathbf{p}} \\ &= \eta^\uparrow(\mathbf{e}^\top \bar{\mathbf{p}} - \mathbf{e}^\top \mathbf{p}) + \mathbf{e}^\top \mathbf{p} \\ &= \frac{D - \mathbf{e}^\top \mathbf{p}}{\mathbf{e}^\top \bar{\mathbf{p}} - \mathbf{e}^\top \mathbf{p}}(\mathbf{e}^\top \bar{\mathbf{p}} - \mathbf{e}^\top \mathbf{p}) + \mathbf{e}^\top \mathbf{p} \\ &= D - \mathbf{e}^\top \mathbf{p} + \mathbf{e}^\top \mathbf{p} = D.\end{aligned}$$

Thus, $\tilde{\mathbf{p}}$ satisfies the power balance and $\tilde{\mathbf{p}} \in \mathcal{S}_D$. Similarly, assume $\mathbf{e}^\top \mathbf{p} \geq D$. Then

$$\begin{aligned}\mathbf{e}^\top \tilde{\mathbf{p}} &= (1-\eta^\downarrow)\mathbf{e}^\top \mathbf{p} + \eta^\downarrow \mathbf{e}^\top \mathbf{0} \\ &= (1 - \frac{\mathbf{e}^\top \mathbf{p} - D}{\mathbf{e}^\top \mathbf{p}})\mathbf{e}^\top \mathbf{p} = D,\end{aligned}$$

which concludes the proof. □

2) Proof of Theorem 2: Let $\mathbf{p}$ be the initial prediction, and let $\tilde{\mathbf{p}} = \mathcal{R}(\mathbf{p})$. First, the bound constraints $\mathbf{0} \leq \mathbf{p} \leq \bar{\mathbf{p}}$ are satisfied because $\tilde{\mathbf{p}}$ is a convex combination of $\mathbf{p}$ and $\bar{\mathbf{p}} - \bar{\mathbf{r}}$, both of which satisfy bound constraints.

Second, we show that $\mathbf{e}^\top \mathbf{p} = D$. Indeed, we have

$$\sum_g \tilde{p}_g = \sum_{g \in G^\uparrow} \tilde{p}_g + \sum_{g \in G^\downarrow} \tilde{p}_g \tag{20}$$

This yields

$$\begin{aligned}\sum_{g \in G^\uparrow} \tilde{p}_g &= \sum_{g \in G^\uparrow} (1-\alpha^\uparrow)p_g + \alpha^\uparrow(\bar{p}_g - \bar{r}_g) \\ &= \sum_{g \in G^\uparrow} p_g + \alpha^\uparrow \sum_{g \in G^\uparrow} (\bar{p}_g - \bar{r}_g - p_g) \\ &= \sum_{g \in \uparrow} p_g + \alpha^\uparrow \Delta^\uparrow \\ &= \sum_{g \in G^\uparrow} p_g + \Delta \\ \sum_{g \in G^\downarrow} \tilde{p}_g &= \sum_{g \in G^\downarrow} (1-\alpha^\downarrow)p_g + \alpha^\downarrow(\bar{p}_g - \bar{r}_g) \\ &= \sum_{g \in G^\downarrow} p_g + \alpha^\downarrow \sum_{g \in G^\downarrow} (\bar{p}_g - \bar{r}_g - p_g) \\ &= \sum_{g \in G^\downarrow} p_g - \alpha^\downarrow \Delta^\downarrow \\ &= \sum_{g \in G^\downarrow} p_g - \Delta\end{aligned}$$

Summing the two terms yields $\mathbf{e}^\top \mathbf{p} + \Delta - \Delta = D$.

Finally, we show that Algorithm 1 provides a feasible point iff DCOPF is feasible. If $\tilde{\mathbf{p}}$ is reserve feasible, then DCOPF is trivially feasible.

Assume $\tilde{\mathbf{p}}$ is infeasible, i.e., $\sum_g \min(\bar{r}_g, \bar{p} - \tilde{p}_g) < R$. Note that if $\Delta = \Delta_R$ in Algorithm 1, then $\tilde{\mathbf{p}}$ is feasible, so we must have $\Delta^\uparrow < \Delta_R$ or $\Delta^\downarrow < \Delta_R$.

Assume the former holds, i.e., $\Delta^\uparrow < \Delta_R, \Delta_\downarrow$. In other words:

$$\begin{aligned}\sum_{g \in G^\uparrow} \bar{p}_g - \bar{r}_g - p_g &< R - \sum_{g \in G^\uparrow} \bar{r}_g - \sum_{g \in G^\downarrow} \bar{p}_g - p_g \\ \sum_{g \in G^\uparrow} (\bar{p}_g - \bar{r}_g - p_g + \bar{r}_g) &< R - \sum_{g \in G^\downarrow} \bar{p}_g - p_g \\ \sum_{g \in G^\uparrow} (\bar{p}_g - p_g) &< R - \sum_{g \in G^\downarrow} \bar{p}_g - p_g \\ \sum_{g \in G^\uparrow} \bar{p}_g + \sum_{g \in G^\downarrow} \bar{p}_g &< R - \sum_g p_g \\ \sum_g \bar{p}_g &< R - D\end{aligned}$$

Summing constraints (1d) and (1b), we obtain

$$\sum_g r_g \leq (\sum_g \bar{p}_g) - D < R$$

hence DCOPF is infeasible. The proof for $\Delta^\downarrow < \Delta_R$ is similar. □

B. Details of the baseline models

Both DC3 [13] and LOOP-LC [30] follow the steps of neural network prediction, inequality correction, and equality completion. First, the decision variables are divided into two groups: $|\mathcal{G}| - N_{eq}$ independent decision variables and N_{eq} dependent decision variables, where N_{eq} indicates the number of equality constraints. In the PTDF formulation of DC-OPF, the only equality constraint is the power balance constraint (1b), and thus $N_{eq} = 1$. Therefore, given the dispatches of the independent generator are predicted, the dispatch of the dependent generator can be recovered by

$$p_1 = D - \sum_{g \in \mathcal{G} \setminus 1} p_g. \tag{21}$$

DC3 and LOOP-LC differ in their inequality corrections.

a) DC3: Given the input load profile $\mathbf{l} \in \mathbb{R}^{|\mathcal{L}|}$, the neural network outputs $\mathbf{z} \in [0,1]^{|\mathcal{G}|-1}$. This is achieved by applying a sigmoid function to the final layer of the network. Then the capacity constraints (1d) are enforced by:

$$p_g = z_g * \bar{p}_g, \ \ \forall g \in \mathcal{G} \setminus 1.$$

In the inequality correction steps, DC3 minimizes the constraint violation by unrolling gradient descent with a fixed number of iterations T. Denote the constraint violation

$$g(\mathbf{p}) = \sum_{g \in \mathcal{G}} \max(p_g - \bar{p}_g, 0) + \max(R - \sum_{g \in \mathcal{G}} r_g, 0),$$

where $r_g = \min\{\bar{r}_g, \bar{p}_g - p_g\}$ and $p_1 = D - \sum_{g \in \mathcal{G} \setminus 1} p_g$. The dispatch is updated using:

$$\mathbf{p}^t = \mathbf{p}^{t-1} - \rho * \nabla_\mathbf{p} \|g(\mathbf{p}^{t-1})\|_2^2,$$

where $\mathbf{p}^0$ is the output of the neural network. In the experiment, the step size ρ is set as $1e-4$ and the total iteration T is set as 50 when training and 200 when testing. The longer testing T is suggested in [13] to mitigate the constraint violation of DC3 predictions.

b) LOOP-LC: Similar to DC3, the neural network maps the load profile $\mathbf{l} \in \mathbb{R}^{|\mathcal{L}|}$ to $\mathbf{z} \in [0,1]^{|\mathcal{G}|-1}$ by applying a sigmoid function at the end. In the inequality correction step, LOOP-LC uses gauge function mapping $\mathbf{z}$ in the l_∞ norm ball to the dispatches in the feasible region. The gauge mapping needs an interior point to shift the domain. The work in [30] proposes an interior point finder by solving an optimization, which could be computationally expensive. Instead, the experiments exploit the proposed feasibility restoration layers to find the interior point effectively. Specifically, the interior point finder consists of two steps. First, the optimal dispatches of the nominal case $\mathbf{p}^n$ are obtained by solving the instance with the nominal active power demand as the input, where the upper bounds $\bar{p}_g$ in constraints (1d) and (1e) are scaled with $\beta \in (0,1)$: $\mathbf{p} + \mathbf{r} \le \beta\bar{\mathbf{p}}$ and $\mathbf{0} \le \mathbf{p} \le \beta\bar{\mathbf{p}}$, where the β is set as 0.8 in the experiments. The scaling aims at providing a more interior point such that the gauge mapping is more smooth. However, the $\mathbf{p}^n$ may not be feasible when changing the input load profile $\mathbf{l}$. To obtain the feasible solution, the proposed feasibility layers are used to convert $\mathbf{p}^n$ to an interior point.

C. Hyperparameter Tuning

For each test case and method, the number of instances in the training and test minibatch are set to 64 and 256, respectively. For all deep learning models, a batch normalization layer [44] and a dropout layer [45] with a dropout rate 0.2 are appended after each dense layer except the last one. The number of layers l is selected from $\{3,4,5\}$ and the hidden dimension hd of the dense layers is selected from $\{128, 256\}$.

The penalty coefficients λ of the constraint violation in the loss function 16 and 17 are selected from $\{1, 0.1\}$ for self-supervised learning and selected from $\{1e-3, 1e-4, 1e-5\}$ for supervised learning. For the models with feasibility guarantees such as DNN-F and LOOP, the λ is set as 0 for self-supervised learning. μ is set to be equal to λ for supervised learning. For DC3 model, the unrolled iteration is set as 50 iterations in training and 200 iterations in testing. The gradient step size is set as $1e-4$, where a larger step size results in numerical issues.

The models are trained with Adam optimizer [41] with an initial learning rate is set as $1e{-}2$ and weight delay $1e{-}6$. The learning rate is decayed by 0.1 when the validation loss does not improve for consecutive 10 epochs and the training early stops if the validation loss does not decrease for consecutive 20 epochs. The maximum training time is set as 150 minutes.

TABLE XI
HYPERPARAMETERS OF THE BEST MODELS FOR DCOPF

Loss	System	Model	λ	l	hd
SL	ieee300	DNN	10^{-4}	3	256
	ieee300	DNN-F	10^{-4}	4	256
	ieee300	DC3	10^{-5}	4	256
	ieee300	LOOP	10^{-4}	4	128
	pegase1k	DNN	10^{-6}	3	128
	pegase1k	DNN-F	10^{-6}	4	256
	pegase1k	DC3	10^{-6}	4	128
	pegase1k	LOOP	10^{-5}	3	256
	rte6470	DNN	10^{-6}	3	256
	rte6470	DNN-F	10^{-6}	3	256
	rte6470	DC3	10^{-6}	3	256
	rte6470	LOOP	10^{-5}	4	128
	pegase9k	DNN	10^{-6}	4	256
	pegase9k	DNN-F	10^{-6}	4	256
	pegase9k	DC3	10^{-6}	4	256
	pegase9k	LOOP	10^{-5}	3	256
	pegase13k	DNN	10^{-6}	3	256
	pegase13k	DNN-F	10^{-6}	3	128
	pegase13k	DC3	10^{-6}	3	256
	pegase13k	LOOP	10^{-5}	3	128
	goc30k	DNN	10^{-5}	4	128
	goc30k	DNN-F	10^{-6}	3	128
	goc30k	DC3	10^{-6}	3	256
	goc30k	LOOP	10^{-5}	3	128
SSL	ieee300	DNN	10^{-1}	3	256
	ieee300	DNN-F	-	3	256
	ieee300	DC3	10^{-1}	3	256
	ieee300	LOOP	-	3	256
	pegase1k	DNN	10^{-1}	3	256
	pegase1k	DNN-F	-	4	256
	pegase1k	DC3	10^{-1}	4	128
	pegase1k	LOOP	-	3	256
	rte6470	DNN	10^{-0}	3	128
	rte6470	DNN-F	-	3	256
	rte6470	DC3	10^{-1}	4	256
	rte6470	LOOP	-	4	128
	pegase9k	DNN	10^{-1}	4	256
	pegase9k	DNN-F	-	3	256
	pegase9k	DC3	10^{-1}	4	256
	pegase9k	LOOP	-	3	128
	pegase13k	DNN	10^{-1}	3	256
	pegase13k	DNN-F	-	4	128
	pegase13k	DC3	10^{-1}	3	256
	pegase13k	LOOP	-	3	128
	goc30k	DNN	10^{-1}	4	256
	goc30k	DNN-F	-	4	256
	goc30k	DC3	10^{-1}	4	256
	goc30k	LOOP	-	4	128

TABLE XII
HYPERPARAMETERS OF THE BEST MODELS FOR DCOPF-R

Loss	System	Model	λ	l	hd
SL	ieee300	DNN	10^{-4}	3	128
	ieee300	DNN-F	10^{-4}	3	256
	ieee300	DC3	10^{-5}	4	256
	pegase1k	DNN	10^{-6}	3	256
	pegase1k	DNN-F	10^{-6}	3	128
	pegase1k	DC3	10^{-6}	3	256
	rte6470	DNN	10^{-4}	3	256
	rte6470	DNN-F	10^{-6}	3	256
	rte6470	DC3	10^{-6}	3	256
	pegase9k	DNN	10^{-5}	3	256
	pegase9k	DNN-F	10^{-6}	3	256
	pegase9k	DC3	10^{-6}	3	256
	pegase13k	DNN	10^{-5}	3	256
	pegase13k	DNN-F	10^{-6}	4	256
	pegase13k	DC3	10^{-6}	3	256
	goc30k	DNN	10^{-6}	3	128
	goc30k	DNN-F	10^{-6}	4	256
	goc30k	DC3	10^{-6}	3	256
SSL	ieee300	DNN	10^{-1}	3	256
	ieee300	DNN-F	-	3	256
	ieee300	DC3	10^{-1}	3	256
	pegase1k	DNN	10^{-1}	3	128
	pegase1k	DNN-F	-	3	256
	pegase1k	DC3	10^{-1}	4	128
	rte6470	DNN	10^{-0}	3	128
	rte6470	DNN-F	-	3	256
	rte6470	DC3	10^{-0}	4	256
	pegase9k	DNN	10^{-1}	4	256
	pegase9k	DNN-F	-	3	256
	pegase9k	DC3	10^{-1}	3	256
	pegase13k	DNN	10^{-1}	4	256
	pegase13k	DNN-F	-	3	128
	pegase13k	DC3	10^{-1}	3	256
	goc30k	DNN	10^{-1}	3	128
	goc30k	DNN-F	-	4	256
	goc30k	DC3	10^{-1}	3	256