A Related Work

The following is an overview of related work in the Predict-Then-Optimize and Learning-to-Optimize settings.

A.1 Predict-Then-Optimize

While the idea is general and has broader applications, differentiation through the optimization of equation 1 is central to EPO approaches. Backpropagation of parametric quadratic programming problems was introduced by Amos & Kolter (2017), which implicitly differentiates the solution via its KKT conditions of optimality Boyd et al. (2004), proposes its use for defining general-purpose learnable layers in neural networks. Agrawal et al. (2019b) followed by proposing a differentiable cone programming solver, which uses implicit differentiation of problem-specific optimality conditions. That framework is leveraged by Agrawal et al. (2019a) to solve and differentiate general convex programs, by pairing it with a symbolic system for conversion of convex programs to canonical cone programs. Kotary et al. (2023) shows how to generate derivatives through a broad class of optimization problems, by implicitly differentiating the fixed-point conditions of a solution algorithm, which can be automated by leveraging automatic differentiation through a single solver step.

The above works focus on computing derivatives through optimization problems when they exist. For many practical problems with discrete structure, such as linear programs, the mapping defined by equation 1 is piecewise constant and cannot be differentiated. (Elmachtoub & Grigas, 2021) propose a surrogate loss function for equation 2 in cases where f is linear, which admits useful subgradients for stochastic gradient descent training. (Wilder et al., 2019) proposes backpropagation through linear programs by adding a smooth quadratic term to the objective and differentiating the resulting QP problem via Amos & Kolter (2017). (Berthet et al., 2020) also propose backpropagation through linear programs but by smoothing the mapping equation 1 through random noise perturbations to the objective function. (Pogančić et al., 2020) form approximate derivatives through linear optimization of discrete variables, by using a finite difference approximation between a pair of solutions with perturbed input parameters.

A.2 Learning to Optimize

Many approaches have been proposed to the concept of learning to solve optimization problems from their representations via problem parameters.

For continuous problems, learning of active constraints Misra et al. (2022) and learning warm-starts Sambharya et al. (2023) are two ways in which intermediate information can be learned to accelerate optimization solvers on a problem instance-specific basis. However, such methods are perpendicular to the idea of learning to optimize from features, since they do not produce solutions end-to-end from parameters, but rather intermediate information utilized by a hard solver. Such end-to-end learning methods include those adapted to LtoF in the paper, and are reviewed in the next section.

The comprehensive survey Bengio et al. (2021) focuses on machine learning methods aimed at boosting combinatorial solvers by predicted intermediate information. Related works involve learning heurstics for combinatorial solvers including branching rules Khalil et al. (2016) and cutting rules Deza & Khalil (2023) in conventional mixed-integer programming.

End-to-end learning for combinatorial optimization appeared as early as Vinyals et al. (2015), followed by Bello et al. (2017) which extended the idea to an unsupervised setting by training with reinforcement learning with policy gradient methods. This approach has been widely adapted to combinatorial problems such as vehicle routing Kool et al. (2018) and job scheduling Mao et al. (2019), and relies on softmax representations of permutations and subset selections to enforce feasibility. Learning combinatorial solutions via supervised penalty methods was proposed in Kotary et al. (2022), while it is shown in Kotary et al. (2021a) that supervision by precomputed target solutions can suffer from unlearnable patterns in the presence of symmetric ground-truth solutions or suboptimally computed labels.

B LEARNING TO OPTIMIZE METHODS

This section reviews in more those LtO methods which are adapted to solve PtO problems in Section 5 of this paper. Each description below assumes a DNN model $\mathcal{O}$ and its parameters ψ, which acts on parameters $\boldsymbol{c_i}$ specifying an instance of problem equation 1, to produce an estimate of the optimal solution $\hat{\boldsymbol{x}} \coloneqq \boldsymbol{F}_\omega(\boldsymbol{c})$, so that $\hat{\boldsymbol{x}} \approx \boldsymbol{x}^\star(\boldsymbol{c})$.

Lagrangian Dual Learning (LD). Fioretto et al. (2020) constructs the following modified Lagrangian as a loss function for training the predictions $\hat{\boldsymbol{x}} = \boldsymbol{F}_\omega(\boldsymbol{c})$

$$\mathcal{L}_{\mathbf{LD}}(\hat{\boldsymbol{x}}, \boldsymbol{c}) = \|\hat{\boldsymbol{x}} - \boldsymbol{x}^\star(\boldsymbol{c})\|_2^2 + \boldsymbol{\lambda}^T \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{c})\right]_+ + \boldsymbol{\mu}^T \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{c}). \tag{10}$$

At each iteration of LD training, the model $\boldsymbol{F}_\omega$ is trained to minimize the loss $\mathcal{L}_{\mathbf{LD}}$. Then, updates to the multiplier vectors λ and μ are calculated based on the average constraint violations incurred by the predictions $\hat{\boldsymbol{x}}$, mimicking a dual ascent method Boyd et al. (2011). In this way, the method minimizes a balance of constraint violations and proximity to the precomputed target optima $\boldsymbol{x}^\star(\boldsymbol{c})$.

Self-Supervised Primal-Dual Learning (PDL). Park & Van Hentenryck (2023) use an augmented Lagrangian loss

$$\mathcal{L}_{\mathbf{PDL}}(\hat{\boldsymbol{x}}, \boldsymbol{c}) = f(\hat{\boldsymbol{x}}, \boldsymbol{c}) + \hat{\boldsymbol{\lambda}}^T \boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{c}) + \hat{\boldsymbol{\mu}}^T \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{c}) + \frac{\rho}{2} \left(\sum_j \nu(g_j(\hat{\boldsymbol{x}})) + \sum_j \nu(h_j(\hat{\boldsymbol{x}})) \right), \tag{11}$$

where ν measures the constraint violation. At each iteration of PDL training, a separate estimate of the Lagrange multipliers is stored for each problem instance in training, and updated by an augmented Lagrangian method Boyd et al. (2011) after training $\hat{\boldsymbol{x}} = \boldsymbol{F}_\omega(\boldsymbol{c})$ to minimize equation 11. In addition to the primal network $\boldsymbol{F}_\omega$, an addition dual network $\mathcal{D}_{\boldsymbol{\zeta}}$ learns to store updates of the multipliers for each instance, and predict them as $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) = \mathcal{D}_{\boldsymbol{\zeta}}(\boldsymbol{c})$ to the next iteration. The method is self-supervised, requiring no precomputation of target solutions for training.

Deep Constraint Completion and Correction (DC3). Donti et al. (2021) use the loss function

$$\mathcal{L}_{\mathbf{DC3}}(\hat{\boldsymbol{x}}, \boldsymbol{c}) = f(\hat{\boldsymbol{x}}, \boldsymbol{c}) + \lambda \| \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{c})\right]_+ \|_2^2 + \mu \|\boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{c})\|_2^2 \tag{12}$$

which combines a problem's objective value with two additional terms which aggregate the total violations of its equality and inequality constraints. The scalar multipliers λ and μ are not adjusted during training. However, feasibility of predicted solutions is enforced by treating $\hat{\boldsymbol{x}} = \hat{\boldsymbol{F}}_\omega(\boldsymbol{c})$ as an estimate for only a subset of optimization variables. The remaining variables are completed by solving the underdetermined equality constraints $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$ as a system of equations. Inequality violations are corrected by gradient descent on the their aggregated values $\| \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{c})\right]_+ \|^2$. These completion and correction steps form the function $\boldsymbol{S}$, where $\boldsymbol{F}_\omega(\boldsymbol{c}) = \boldsymbol{S} \circ \hat{\boldsymbol{F}}_\omega(\boldsymbol{c})$.

C OPTIMIZATION PROBLEMS

Illustrative $2D$ example Used for illustration purposes, the $2D$ optimization problem used to produce the results of Figure 3 takes the form

$$\begin{aligned} \mathbf{x}^\star(\boldsymbol{\zeta}) = \arg\min_{\boldsymbol{x}} \;& \zeta_1 x_1^2 + \zeta_2 x_2^2 \\ \text{s.t.} \;& x_1 + 2x_2 = 0.5, \\ & 2x_1 - x_2 = 0.2, \\ & x_1 + x_2 = 0.3 \end{aligned}$$

and its optimization proxy model is learned using *PDL* training.

$$
\begin{aligned}
\text{Minimize}: \quad & \sum_{i\in\mathcal{N}} \text{cost}(p_i^g, \zeta_i) && \text{(2a)}\\
\text{s.t.} \quad & \boldsymbol{v}_i^{\min} \le v_i \le \boldsymbol{v}_i^{\max} \ \ \forall i \in \mathcal{N} && \text{(2b)}\\
& -\boldsymbol{\theta}_{ij}^{\Delta} \le \theta_i - \theta_j \le \boldsymbol{\theta}_{ij}^{\Delta} \ \ \forall (ij) \in \mathcal{E} && \text{(2}\bar{c}\text{)}\\
& \boldsymbol{p}_i^{g\,\min} \le p_i^g \le \boldsymbol{p}_i^{g\,\max} \ \ \forall i \in \mathcal{N} && \text{(2}\bar{d}\text{)}\\
& \boldsymbol{q}_i^{g\,\min} \le q_i^g \le \boldsymbol{q}_i^{g\,\max} \ \ \forall i \in \mathcal{N} && \text{(2e)}\\
& (p_{ij}^f)^2 + (q_{ij}^f)^2 \le \boldsymbol{S}_{ij}^{f\,\max} \ \ \forall (ij) \in \mathcal{E} && \text{(2}\bar{f}\text{)}\\
& p_{ij}^f = \boldsymbol{g}_{ij} v_i^2 - v_i v_j (\boldsymbol{b}_{ij} \sin(\theta_i - \theta_j) + \boldsymbol{g}_{ij} \cos(\theta_i - \theta_j)) \ \ \forall (ij) \in \mathcal{E} && \text{(2}\bar{g}\text{)}\\
& q_{ij}^f = -\boldsymbol{b}_{ij} v_i^2 - v_i v_j (\boldsymbol{g}_{ij} \sin(\theta_i - \theta_j) - \boldsymbol{b}_{ij} \cos(\theta_i - \theta_j)) \ \ \forall (ij) \in \mathcal{E} && \text{(2h)}\\
& p_i^g - \boldsymbol{p}_i^d = \textstyle\sum_{(ij)\in\mathcal{E}} p_{ij}^f \ \ \forall i \in \mathcal{N} && \text{(2}\bar{i}\text{)}\\
& q_i^g - \boldsymbol{q}_i^d = \textstyle\sum_{(ij)\in\mathcal{E}} q_{ij}^f \ \ \forall i \in \mathcal{N} && \text{(2j)}\\
\text{Output}: \quad & (p^g, v) \text{ – The system operational parameters}
\end{aligned}
$$

Figure 6: AC Optimal Power Flow (AC-OPF).

AC-Optimal Power Flow Problem. The OPF determines the least-cost generator dispatch that meets the load (demand) in a power network. The OPF is defined in terms of complex numbers, i.e., *powers* of the form $S = (p + jq)$, where p and q denote active and reactive powers and j the imaginary unit, *admittances* of the form $Y = (g + jb)$, where g and b denote the conductance and susceptance, and *voltages* of the form $V = (v\angle\theta)$, with magnitude v and phase angle θ. A power network is viewed as a graph $(\mathcal{N}, \mathcal{E})$ where the nodes $\mathcal{N}$ represent the set of *buses* and the edges $\mathcal{E}$ represent the set of *transmission lines*. The OPF constraints include physical and engineering constraints, which are captured in the AC-OPF formulation of Figure 6. The model uses p^g, and p^d to denote, respectively, the vectors of active power generation and load associated with each bus and p^f to describe the vector of active power flows associated with each transmission line. Similar notations are used to denote the vectors of reactive power q. Finally, the model uses v and θ to describe the vectors of voltage magnitude and angles associated with each bus. The OPF takes as inputs the loads $(\boldsymbol{p}^d, \boldsymbol{q}^d)$ and the admittance matrix $\boldsymbol{Y}$, with entries $\boldsymbol{g}_{ij}$ and $\boldsymbol{b}_{ij}$ for each line $(ij) \in \mathcal{E}$; It returns the active power vector p^g of the generators, as well the voltage magnitude v at the generator buses. The problem objective equation 2a captures the cost of the generator dispatch and is typically expressed as a quadratic function. Constraints equation 2b and equation 2$\bar{c}$ restrict the voltage magnitudes and the phase angle differences within their bounds. Constraints equation 2$\bar{d}$ and equation 2e enforce the generator active and reactive output limits. Constraints equation 2$\bar{f}$ enforce the line flow limits. Constraints equation 2$\bar{g}$ and equation 2h capture *Ohm's Law*. Finally, Constraint equation 2$\bar{i}$ and equation 2j capture *Kirchhoff's Current Law* enforcing flow conservation at each bus.

In the experiment, an energy market is simulated where producers adjust their prices based on bids and prevailing weather conditions. Similarly, consumers decide on additional energy purchases influenced by these variables. The Predict-then-optimize focus is to predict the cost implications for energy generators. Synthetic features $\boldsymbol{z}$ are used to create increasingly complex feature (simulating weather) to costs mappings and ground-truth costs $\boldsymbol{\zeta}$ are sampled uniformly from the interval $[50, 100]$. Recognizing that bids from both sides may not be reliable, the model uses its own forecasts for energy demand and supply to determine if new generators need to be activated. This approach enables the study of how market dynamics and forecasting inaccuracies affect generator costs.

Projection (Load Flow Model) Being an approximation, a LtO solution $\hat{\boldsymbol{p}}^g$ may not satisfy the original constraints. Feasibility can be restored by applying a load flow optimization. A simple load flow is shown in Figure 7. It is a least square minimization that finds a feasible solution minimizing the distance to the approximated one. The use of such a projection allows for detailed comparison between the various exact and approximate models. Observe that the load flow itself is a nonlinear nonconvex problem. How-

$$
\begin{aligned}
\text{Minimize}: \quad & \|p^g - \hat{\boldsymbol{p}}^g\|^2 + \|v - \hat{\boldsymbol{v}}\|^2 && \text{(3)}\\
\text{s.t.:} \quad & \text{Eqns. 2}\bar{b} - \text{2}\bar{j}\\
\text{Output}: \quad & (p^g, v)
\end{aligned}
$$

Figure 7: AC Load Flow.

	Method	$k = 0$ (LtO)	$k = 1$	$k = 2$	$k = 4$	$k = 8$
LtOF	LD Regret	0.0680	0.0673	**0.1016**	**0.4904**	**0.7470**
	LD Regret (*)	0.0009	0.0009	0.0013	0.0071	0.0195
	LD Violation (*)	0.0035	0.0017	0.0020	0.0037	0.0042
	PDL Regret	0.6305	0.7958	0.9603	0.8543	0.8304
	PDL Regret (*)	0.0210	0.0242	0.0260	0.0243	0.0242
	PDL Violation (*)	0.0001	0.0002	0.0000	0.0002	0.0002
	Two-Stage Regret (Best)	-	**0.0370**	0.3300	1.1380	2.4740
	EPO Proxy Regret (Best)	-	431.7664	389.0421	413.8941	404.7452

Table 3: Regret and Constraint Violations for AC-OPF Experiment. (*) denotes "Before Restoration".

	Method	$k = 0$ (LtO)	$k = 1$	$k = 2$	$k = 4$	$k = 8$
LtOF	LD Regret	**1.2785**	0.9640	1.7170	2.1540	**2.1700**
	LD Regret (*)	1.1243	1.0028	1.5739	2.0903	2.1386
	LD Violation (*)	0.0037	0.0023	0.0010	0.0091	0.0044
	PDL Regret	1.2870	0.8520	1.5150	2.0720	2.3830
	PDL Regret (*)	1.2954	0.9823	1.4123	1.9372	2.0435
	PDL Violation (*)	0.0018	0.0097	0.0001	0.0003	0.0003
	DC3 Regret	1.3580	2.1040	2.1490	2.3140	2.6600
	DC3 Regret (*)	1.2138	1.8656	2.0512	1.9584	2.3465
	DC3 Violation (*)	0.0000	0.0000	0.0000	0.0000	0.0000
	Two-Stage Regret (Best)	-	**0.3480**	2.8590	4.4790	91.3260
	EPO Regret (Best)	-	1.0234	**0.9220**	**1.4393**	4.7495
	EPO Proxy Regret (Best)	-	136.4341	154.3960	119.3082	114.6953

Table 4: Regret and Constraint Violations for Portfolio Experiment. (*) denotes "Before Restoration".

ever, when started with a good approximation it is typically much easier to solve than the AC-OPF **?**.

D Experimental Details

D.1 Portfolio Optimization Dataset

The stock return dataset is prepared exactly as prescribed in Sambharya et al. (2023). The return parameters and asset prices are $\zeta = \alpha(\hat{\zeta}_t + \epsilon_t)$ where $\hat{\zeta}$ is the realized return at time t, ϵ_t is a normal random variable, $\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon I)$, and $\alpha = 0.24$ is selected to minimize $\mathbb{E}\|\hat{\zeta}_t - \zeta\|_2^2$. For each problem instance, the asset prices ζ are sampled by circularly iterating over the five year interval. In the experiments, see Prob. 8, $\lambda = 2.0$.

The covariance matrix $\mathbf{\Sigma}$ is constructed from historical price data and set as $\mathbf{\Sigma} = F\mathbf{\Sigma}_F F^T + D$, where $F \in \mathbb{R}^{n,l}$ is the factor-loading matrix, $\mathbf{\Sigma} \in \mathbb{S}_+^l$ estimates the factor returns and $D \in \mathbb{S}_+^l$, also called the idiosyncratic risk, is a diagonal matrix which takes into account for additional variance for each asset.

D.2 Hyperparameters

For all the experiments, the size of the mini-batch $\mathcal{B}$ of the training set is equal to 200. The optimizer used for the training of the optimization proxy's is Adam, and the learning rate is set to $1e-4$. The same optimizer and learning rate are adopted to train the Two-Stage, EPO (w/o) proxy's predictive model. For each optimization problem, an early stopping criteria based on the evaluation of the test-set precentage regret after restoring feasibility, is adopted to all the LtO(F) the proxies, and the predictive EPO (w/o) proxy. For each optimization problem, an early stopping criteria based on the evaluation of the mean squared error is adopted to all the Two-Stage predictive model.
For each optimization problem, the LtOF proxies are 2-layers ReLU neural networks with dropout

	Method	$k = 0$ (LtO)	$k = 1$	$k = 2$	$k = 4$	$k = 8$
LtOF	LD Regret	8.0757	**8.6826**	9.9279	**9.7879**	9.5473
	LD Regret (*)	8.1120	8.7416	9.9250	9.8211	9.5556
	LD Violation (*)	0.0753	0.0375	0.0148	0.0162	0.0195
	PDL Regret	**7.4936**	11.424	**7.2699**	10.7474	**7.6399**
	PDL Regret (*)	7.7985	11.429	7.2735	10.749	7.6394
	PDL Violation (*)	0.0047	0.0032	0.0028	0.0013	0.0015
	DC3 Regret	13.946	14.623	14.271	11.028	10.666
	DC3 Regret (*)	14.551	14.517	13.779	11.755	10.849
	DC3 Violation (*)	1.4196	0.8259	0.5158	0.5113	0.5192
	Two-Stage Regret (Best)	-	23.2417	36.1684	37.3995	38.2973
	EPO Proxy Regret (Best)	-	793.2369	812.7521	804.2640	789.5043

Table 5: Regret and Constraint Violations for Nonconvex QP Experiment. (*) denotes "Before Restoration".

equal to 0.1 and batch normalization. All the LtO proxies are $(k+1)$-layers ReLU neural networks with dropout equal to 0.1 and batch normalization, where k denotes the complexity of the feature mapping. For the LtOF, Two-Stage, EPO (w/o) Proxy algorithm, the feature size of the Convex Quadratic Optimization and Non Convex AC Optimal Power Flow $|z| = 30$, while for the Non Convex Quadratic Optimization $|z| = 50$. The hidden layer size of the feature generator model is equal to 50, and the hidden layer size of the LtO(F) proxies, and the 2Stage, EPO and EPO w/ proxy's predictive model is equal to 500.
A grid search method is adopted to tune the hyperparameters of each LtO(F) models. For each experiments, and for each LtO(F) methods, below is reported the list of the candidate hyperparameters for each k, with the chosen ones marked in bold. We refer to Fioretto et al. (2020), Park & Van Hentenryck (2023) and Donti et al. (2021) for a comprehensive description of the parameters of the LtO methods adopted in the proposed framework. In our result, two-stage methods report the *lowest regret* found in each experiment and each k across all hyperparameters adopted, providing a very strong baseline.

D.2.1 Convex Quadratic Optimization and Non Convex Quadratic Optimization

LD

- λ : **0.1**, 0.5, 1.0, 5.0, 10.0, 50.0.
- μ : 0.1, **0.5**, 1.0, 5.0, 10.0, 50.0.
- LD step size : 50, 100, **200**, 300, 500.
- LD updating epochs : 1.0, 0.1, 0.01, **0.001**, 0.0001.

PDL

- τ : 0.5, 0.6, 0.7, **0.8**, 0.9.
- ρ : 0.1, **0.5**, 1, 10.
- ρ_{max} : 1000 **5000**, 10000.
- α : 1, 1.5, 2.5, **5**, 10.

DC3

- $\lambda + \mu$: 0.1, 1.0, **10.0**, 50.0, 100.0.
- $\frac{\lambda}{\lambda+\mu}$: 0.1, 0.5, **0.75**, 1.
- t_{test} : 1, 2, **5**, 10, 100.
- t_{train} : 1, 2, **5**, 50, 100.

D.2.2 NON CONVEX AC-OPTIMAL POWER FLOW

LD

- λ : $\mathbf{0.1}$, 0.5, 1.0, 5.0, 10.0, 50.0.
- μ : $\mathbf{0.1}$, 0.5, 1.0, 5.0, 10.0, 50.0.
- LD step size : 50, 100, 200, $\mathbf{300}$, 500.
- LD updating epochs : $\mathbf{1.0}$, 0.1, 0.01, 0.001, 0.0001.

PDL

- τ : 0.5, 0.6, 0.7, $\mathbf{0.8}$, 0.9.
- ρ : 0.1, 0.5, $\mathbf{1}$, 10.
- ρ_{max} : 1000 $\mathbf{5000}$, 10000.
- α : 1, 1.5, 2.5, $\mathbf{5}$, 10.