Learning Joint Models of Prediction and Optimization

James Kotary[a,1], **Vincenzo Di Vito**[a,1], **Jacob Christopher**[a], **Pascal Van Hentenryck**[b] **and Ferdinando Fioretto**[a]

[a]University of Virginia
[b]Georgia Institute of Technology

Abstract. The Predict-Then-Optimize framework uses machine learning models to predict unknown parameters of an optimization problem from exogenous features before solving. This setting is common to many real-world decision processes, and recently it has been shown that decision quality can be substantially improved by solving and differentiating the optimization problem within an end-to-end training loop. However, this approach requires significant computational effort in addition to handcrafted, problem-specific rules for backpropagation through the optimization step, challenging its applicability to a broad class of optimization problems. This paper proposes an alternative method, in which optimal solutions are learned directly from the observable features by joint predictive models. The approach is generic, and based on an adaptation of the Learning-to-Optimize paradigm, from which a rich variety of existing techniques can be employed. Experimental evaluations show the ability of several Learning-to-Optimize methods to provide efficient and accurate solutions to an array of challenging Predict-Then-Optimize problems.

1 Introduction

The *Predict-Then-Optimize* (PtO) framework models decision-making processes as optimization problems whose parameters are only partially known while the remaining, unknown, parameters must be estimated by a machine learning (ML) model. The predicted parameters complete the specification of an optimization problem which is then solved to produce a final decision. The problem is posed as estimating the solution $\boldsymbol{x}^\star(\zeta) \in \mathcal{X} \subseteq \mathbb{R}^n$ of a *parametric* optimization problem:

$$\boldsymbol{x}^\star(\boldsymbol{\zeta}) = \arg\min_{\boldsymbol{x}} \; f(\boldsymbol{x}, \boldsymbol{\zeta}) \tag{1a}$$

$$\textit{such that: } \; \boldsymbol{g}(\boldsymbol{x}) \leq 0, \;\; \boldsymbol{h}(\boldsymbol{x}) = 0, \tag{1b}$$

given that parameters $\boldsymbol{\zeta} \in \mathcal{C} \subseteq \mathbb{R}^p$ are unknown, but that a correlated set of observable values $\boldsymbol{z} \in \mathcal{Z}$ are available. Here f is an objective function, and $\boldsymbol{g}$ and $\boldsymbol{h}$ define the set of the problem's inequality and equality constraints. The combined prediction and optimization model is evaluated on the basis of the optimality of its downstream decisions, with respect to f under its ground-truth problem parameters [13]. This setting is ubiquitous to many real-world applications confronting the task of decision-making under uncertainty, such as planning the shortest route in a city, determining optimal power generation schedules, or managing investment portfolios. For example, a vehicle routing system may aim to minimize a rider's total commute time by solving a shortest-path optimization model (1) given knowledge of the transit times $\boldsymbol{\zeta}$ over each individual city block. In absence of that knowledge, it may be estimated by prediction models based on exogenous data $\boldsymbol{z}$, such as weather and traffic conditions. In this context, more accurately predicted transit times $\hat{\boldsymbol{\zeta}}$ tend to produce routing plans $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$ with shorter commutes, with respect to the true city-block transit times $\boldsymbol{\zeta}$.

However, direct training of predictions from observable features to problem parameters can generalize poorly with respect to the ground-truth optimality achieved by a subsequent decision model [21, 16]. To address this challenge, *End-to-end Predict-Then-Optimize* (EPO) [13] has emerged as a transformative paradigm in data-driven decision making, where predictive models are trained to directly minimize loss functions defined on the downstream optimal solutions $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$.

On the other hand, EPO implementations present two key challenges: **(i)** They require backpropagation through the solution of the optimization problem (1) as a function of its parameters for end-to-end training. The required backpropagation rules are highly dependent on the form of the optimization model and are typically derived by hand analytically for limited classes of models [3, 1]. **(ii)** Furthermore, difficult decision models involving nonconvex or discrete optimization may not admit well-defined backpropagation rules.

To address these challenges, this paper outlines a framework for training Predict-Then-Optimize models by techniques adapted from a separate but related area of work that combines constrained optimization end-to-end with machine learning. This paradigm, called *Learn-to-Optimize* (LtO), learns a mapping between the parameters of an optimization problem and its corresponding optimal solutions using a deep neural network (DNN), as illustrated in Figure 1(c).

The resulting DNN mapping is then treated as an *optimization proxy* whose role is to repeatedly solve difficult, but related optimization problems in real time [30, 14]. Several LtO methods specialize in training proxies to solve difficult problem forms, especially those involving nonconvex optimization.

The methodology of this paper recognizes that existing LtO methods can provide an array of implementations for producing learned optimization proxies, which can handle hard optimization problem forms, have fast execution speeds, and are differentiable by construction. As such, they can be adapted to the Predict-Then-Optimize setting, offering an alternative to hard optimization solvers with handcrafted backpropagation rules. However, direct transfer of a pretrained optimization proxy into an EPO training loop leads to degradation of accuracy in the proxy solver, for which an end-to-end learning solution is proposed. The resulting *Learning to Optimize from Features* (LtOF) framework extends the Learn-to-Optimize problem setting to *encompass* that of Predict-Then-Optimize, by learning to construct optimal solutions directly from features, as illustrated in Figure 2(d).

Contributions. This paper makes the following novel contributions:

[1] Corresponding authors with equal contribution. Emails: jk4pn, eda8pc@virginia.edu

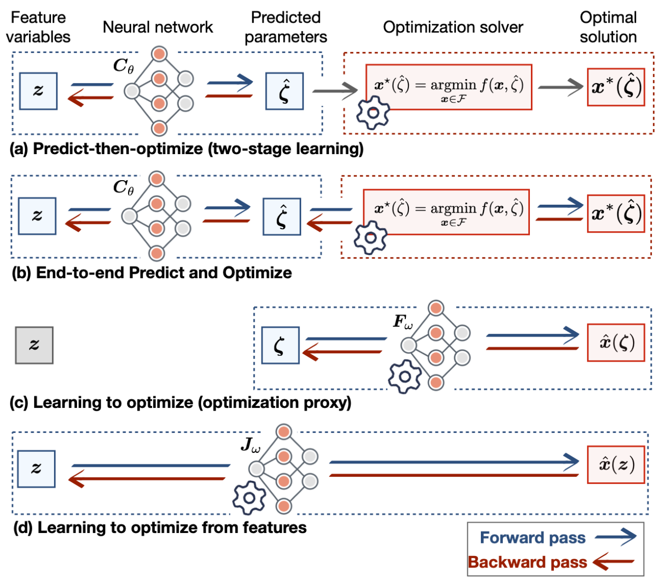
Feature variables
Neural network
Predicted parameters
Optimization solver
Optimal solution
z
C_θ
$\hat{\zeta}$
$x^\star(\hat{\zeta}) = \underset{x \in \mathcal{F}}{\operatorname{argmin}} f(x, \hat{\zeta})$
$x^\star(\hat{\zeta})$
(a) Predict-then-optimize (two-stage learning)
(b) End-to-end Predict and Optimize
F_ω
ζ
$\hat{x}(\zeta)$
(c) Learning to optimize (optimization proxy)
J_ω
$\hat{x}(z)$
(d) Learning to optimize from features
Forward pass
Backward pass

Figure 1: Illustration of Learning to Optimize from Features, in relation to other learning paradigms.

(1) It investigates the use of pretrained LtO proxy models as a means to approximate the decision-making component of an EPO pipeline, and demonstrates a distributional shift effect between prediction and optimization models that leads to loss of accuracy in EPO training. **(2)** It proposes Learning to Optimize from Features (LtOF), in which existing LtO methods are adapted to learn solutions to optimization problems directly from observable features, circumventing the distributional shift effect over the problem parameters. **(3)** The generic LtOF framework is evaluated by adapting several well-known LtO methods to solve Predict-then-Optimize problems with difficult optimization components. Besides outperforming conventional two-stage approaches, *the results show that difficult nonconvex optimization components can be incorporated into PtO pipelines naturally*, and illustrate how high decision quality can be reached in such cases even when gradient-based EPO alternatives fail.

2 Problem Setting and Background

In the Predict-then-Optimize (PtO) setting, three distributions of data are assumed. Observable features $\boldsymbol{z} \sim \mathcal{Z}$ are correlated with the *unknown* coefficients $\boldsymbol{\zeta} \sim \mathcal{C}$ of a parametric optimization problem (1), which in turn defines corresponding optimal solutions $\boldsymbol{x}^\star(\boldsymbol{\zeta}) \sim \mathcal{X}$. We aim to learn optimal solutions $\boldsymbol{x}^\star(\boldsymbol{\zeta})$ to problem (1) without knowing the objective coefficients $\boldsymbol{\zeta}$, but knowing instead the correlated features $\boldsymbol{z}$. More precisely, the goal is to learn a mapping $\hat{\boldsymbol{x}}_\theta : \mathcal{Z} \to \mathcal{X}$ from observable features $\boldsymbol{z}$ to feasible solutions of (1b), while aiming to optimize their objective value (1a) under the ground truth coefficients $\boldsymbol{\zeta}$. Assuming a joint distribution $(\boldsymbol{z}, \boldsymbol{\zeta}) \sim \Omega$, this can be expressed as

$$\underset{\theta}{\text{Minimize}} \ \mathbb{E}_{(\boldsymbol{z},\boldsymbol{\zeta})\sim\Omega} \left[f\left(\hat{\boldsymbol{x}}_\theta(\boldsymbol{z}), \boldsymbol{\zeta} \right) \right]. \tag{2}$$

A deterministic mapping $\boldsymbol{x}^\star : \mathcal{C} \to \mathcal{X}$ from problem coefficients to optimal solutions is defined by optimization problem (1), and can be implemented using any solution method which solves (1). As such, approaches to solving (2) are commonly based on framing $\hat{\boldsymbol{x}}_\theta$ as a composite prediction-and-optimization model $\boldsymbol{x}^\star \circ \boldsymbol{C}_\theta : \mathcal{Z} \to \mathcal{X}$, in which $\boldsymbol{C}_\theta : \mathcal{Z} \to \mathcal{C}$ is a neural network trained to estimate problem coefficients $\boldsymbol{\zeta}$ from the observable features $\boldsymbol{z}$. The prediction and optimization components $\boldsymbol{C}_\theta$ and $\boldsymbol{x}^\star$ are called, respectively, the *first* and *second* stage models.

Two main approaches are typically used to train the predictive component $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$, in order to realize the training goal (2):

■ **Two-stage Method.** A conventional approach to training the parameter prediction model $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$ is the *two-stage* method. It trains to predict the problem coefficients by MSE regression from their ground-truth values; i.e. with loss function $\ell(\hat{\boldsymbol{\zeta}}, \boldsymbol{\zeta}) = \|\hat{\boldsymbol{\zeta}} - \boldsymbol{\zeta}\|_2^2$, without accounting for the downstream optimization during training. This direct minimization of prediction errors is consistent with the goal (2) of optimizing the objective value $f(\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}), \boldsymbol{\zeta})$. However, by ignoring the effect of error propagation from predicted coefficients $\hat{\boldsymbol{\zeta}}$ to downstream optimal solutions $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$ in the second stage, this naive approach is known to result in suboptimal objective values $f(\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}), \boldsymbol{\zeta})$ [13], yielding poor performance on the PtO goal (2).

■ **End-to-End Predict-Then-Optimize.** Improving on the two-stage method, the End-to-end Predict-Then-Optimize (EPO) approach trains $\boldsymbol{C}_\theta$ directly to optimize the objective $f(\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}), \boldsymbol{\zeta})$ as a loss function by gradient descent. This is enabled by finding or approximating the derivatives through $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$. This corresponds to end-to-end training of the PtO goal (2), where $\hat{\boldsymbol{x}}_\theta = \boldsymbol{x}^\star \circ \boldsymbol{C}_\theta$:

$$\underset{\theta}{\text{Minimize}} \ \mathbb{E}_{(\boldsymbol{z},\boldsymbol{\zeta})\sim\Omega} \left[f\left(\boldsymbol{x}^\star(\boldsymbol{C}_\theta(\boldsymbol{z})), \boldsymbol{\zeta} \right) \right]. \tag{3}$$

EPO training consistently outperforms two-stage methods with respect to the goal (2), especially when the mapping $\boldsymbol{z} \to \boldsymbol{\zeta}$ is complex, due to the aforemention error propagation effect. See Figure 1 (a) and (b) for an illustrative comparison, where the constraint set is denoted with $\mathcal{F}$. An overview of related work on EPO is reported in Section 6.

Challenges in End-to-End Predict-Then-Optimize

Despite their advantages over the two-stage, EPO methods are known to face two key challenges: **(1) Differentiability**: the need for handcrafted backpropagation rules through $\boldsymbol{x}^\star(\boldsymbol{\zeta})$, which are highly dependent on the form of problem (1), and rely on the assumption of derivatives $\frac{\partial \boldsymbol{x}^\star}{\partial \boldsymbol{\zeta}}$ which may not exist, and require that the mapping (1) is unique, producing a well-defined function; **(2) Efficiency**: the need to solve the optimization (1) to produce $\boldsymbol{x}^\star(\boldsymbol{\zeta})$ for each sample, in deployment and at each iteration of training. The results of Section 5 demonstrate a *further* potential pitfall of EPO training: even when differentiable, nonconvexity of (1) can cause its gradients to provide unhelpful descent directions for EPO training.

This paper is motivated by a need to address these disadvantages. To do so, it recognizes a body of work on training DNNs as *learned optimization proxies* which have fast execution, are automatically differentiable by design, and specialize in learning mappings $\boldsymbol{\zeta} \to \boldsymbol{x}^\star(\boldsymbol{\zeta})$ of hard optimization problems. While the next section discusses why the direct application of learned proxies as differentiable optimization solvers in an EPO approach tends to fail, Section 4 presents a successful adaptation of the approach, in which predictive models are trained to solve the PtO training goal (2) *directly*.

3 EPO with Optimization Proxies

The Learning-to-Optimize problem setting encompasses a variety of distinct methodologies with the common goal of learning to solve optimization problems. This section characterizes the LtO setting, before proceeding to describe an adaptation of LtO methods to the Predict-Then-Optimize setting.

■ **Learning to Optimize.** The idea of training DNN models to emulate optimization solvers is referred to as *Learn-to-Optimize (LtO)*

[16]. Here the goal is to learn a mapping $\mathbf{F}_\omega : \mathcal{C} \to \mathcal{X}$ from the parameters $\boldsymbol{\zeta}$ of an optimization problem (1) to its corresponding optimal solution $\boldsymbol{x}^\star(\boldsymbol{\zeta})$(see Figure 1 (c)). The resulting *proxy* optimization model has as its learnable component a DNN denoted $\hat{\boldsymbol{F}}_\omega$, which may be augmented with further operations $\boldsymbol{\mathcal{S}}$ such as constraint corrections or unrolled solver steps, so that $\mathbf{F}_\omega = \boldsymbol{\mathcal{S}} \circ \hat{\mathbf{F}}_\omega$. While training such a lightweight model to emulate optimization solvers is in general difficult, it is made tractable by restricting the task over a *limited distribution* of problem parameters $\boldsymbol{\zeta} \sim \mathcal{C}$. A variety of LtO methods have been proposed, many specializing in learning to solve problems of a specific form. Some are based on supervised learning, where precomputed solutions $\boldsymbol{x}^\star(\boldsymbol{\zeta})$ are required as target data in addition to parameters $\boldsymbol{\zeta}$ for each sample. Others are *self-supervised*, requiring only knowledge of the problem form (1) along with instances of the parameters $\boldsymbol{\zeta}$ for supervision in training. LtO methods employ special learning objectives to train the proxy model $\boldsymbol{F}_\omega$:

$$\underset{\omega}{\text{Minimize}} \; \mathbb{E}_{\boldsymbol{\zeta}\sim\mathcal{C}} \left[\ell^{\text{LtO}} \Big(\boldsymbol{F}_\omega(\boldsymbol{\zeta}), \boldsymbol{\zeta} \Big) \right], \tag{4}$$

where ℓ^{LtO} represents a loss that is specific to the LtO method employed. A primary challenge in LtO is ensuring the satisfaction of constraints $\boldsymbol{g}(\hat{\boldsymbol{x}}) \leq 0$ and $\boldsymbol{h}(\hat{\boldsymbol{x}}) = 0$ by the solutions $\hat{\boldsymbol{x}}$ of the proxy model $\boldsymbol{F}_\omega$. This can be achieved, exactly or approximately, by a variety of methods, for example iteratively retraining Equation (4) while applying dual optimization steps to a Lagrangian loss function [14, 24], or designing $\boldsymbol{\mathcal{S}}$ to restore feasibility [12], as reviewed in Section 5.4. In cases where small constraint violations remain in the solutions $\hat{\mathbf{x}}$ at inference time, they can be removed by post-processing with efficient projection or correction methods as deemed suitable for the particular application [16].

EPO with Pretrained Optimization Proxies

Viewed from the Predict-then-Optimize lens, learned optimization proxies have two beneficial features by design: **(1)** they enable very fast solving times compared to conventional solvers, and **(2)** are differentiable by virtue of being trained end-to-end. Thus, a natural question is whether it is possible to use a pre-trained optimization proxy to substitute the differentiable optimization component of an EPO pipeline. Such an approach modifies the EPO objective (2) as:

$$\underset{\theta}{\text{Minimize}} \; \mathbb{E}_{(\boldsymbol{z},\boldsymbol{\zeta})\sim\Omega} \left[f\Big(\overbrace{\boldsymbol{F}_\omega\big(\underbrace{\boldsymbol{C}_\theta(\boldsymbol{z})}_{\hat{\boldsymbol{\zeta}}}\big)}^{\hat{\boldsymbol{x}}}, \boldsymbol{\zeta} \Big) \right], \tag{5}$$

where the solver output $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$ of problem (2) is replaced by the prediction $\hat{\boldsymbol{x}}$ obtained by LtO model $\boldsymbol{F}_\omega$ on input $\hat{\boldsymbol{\zeta}}$ (gray highlights a pretrained model, with frozen weights ω).

However, a critical challenge in LtO lies in the inherent limitation that ML models act as reliable optimization proxies *only within the distribution of inputs they are trained on*. This challenges the implementation of the idea of using pretrained LtOs as components of an end-to-end Predict-Then-Optimize model as the weights θ update during training, leading to continuously evolving inputs $\boldsymbol{C}_\theta(\boldsymbol{z})$ to the pretrained optimizer $\boldsymbol{F}_\omega$. Thus, to ensure robust performance, $\boldsymbol{F}_\omega$ must generalize well across virtually any input during training. However, due to the dynamic nature of θ, there is an inevitable *distribution shift* in the inputs to $\boldsymbol{F}_\omega$, destabilizing the EPO training.

Figures 2 and 3 illustrate this issue. The former highlights how the input distribution to a pretrained proxy drifts during EPO training, impacting both output and backpropagation. The latter quantifies this behavior, exemplified on a simple two-dimensional problem (described in Appendix A), showing rapid increase in proxy regret as $\hat{\boldsymbol{\zeta}}$ diverges from the initial training distribution $\boldsymbol{\zeta} \sim \mathcal{C}$ (in black). [2]

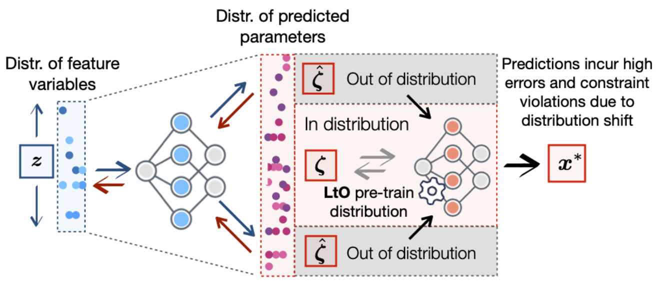
Distr. of predicted parameters
Distr. of feature variables
z
$\hat{\zeta}$
Out of distribution
In distribution
ζ
LtO pre-train distribution
$\hat{\zeta}$
Out of distribution
Predictions incur high errors and constraint violations due to distribution shift
x^*

Figure 2: Effect of shifting inputs received by the LtO proxy: a mismatch from its initial training distribution leads to inaccurate solutions when employed in PtO training.

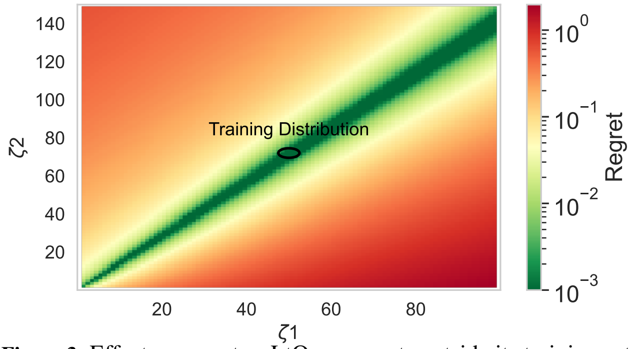

Figure 3: Effect on regret as LtO proxy acts outside its training set.

The experimental results of Table 2 reinforce these observations. While each proxy solver performs well within its training distribution, their effectiveness deteriorates sharply when utilized as in equation 5. A step toward resolving this distribution shift issue allows the weights of $\boldsymbol{F}_\omega$ to adapt to its changing inputs, by *jointly* training the prediction and optimization models:

$$\underset{\theta,\omega}{\text{Minimize}} \; \mathbb{E}_{(\boldsymbol{z},\boldsymbol{\zeta})\sim\Omega} \left[f\Big(\overbrace{\boldsymbol{F}_\omega\big(\underbrace{\boldsymbol{C}_\theta(\boldsymbol{z})}_{\hat{\boldsymbol{\zeta}}}\big)}^{\hat{\boldsymbol{x}}}, \boldsymbol{\zeta} \Big) \right]. \tag{6}$$

The predictive model $\boldsymbol{C}_\theta$ is then effectively absorbed into the predictive component of $\boldsymbol{F}_\omega$, resulting in a *joint* prediction and optimization proxy model $\boldsymbol{J}_\phi = \boldsymbol{F}_\omega \circ \boldsymbol{C}_\theta$, where $\phi = (\omega, \theta)$. Given the requirement for feasible solutions, the training objective (6) must be replaced with an LtO procedure that enforces the constraints on its outputs. This leads to the joint training framework presented next.

4 Learning to Optimize from Features

The distribution shift effect described above arises due to the disconnect in training between the first-stage prediction network $\boldsymbol{C}_\theta : \mathcal{Z} \to \mathcal{C}$ and the second-stage optimization proxy $\boldsymbol{F}_\omega : \mathcal{C} \to \mathcal{X}$. However, the Predict-Then-Optimize setting (see Section 2) ultimately only requires the combined model to produce a candidate optimal solution $\hat{\boldsymbol{x}} \in \mathcal{X}$ given an observation of features $\boldsymbol{z} \in \mathcal{Z}$. Thus, the intermediate prediction $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$ in Equation (6) is, in principle, not needed. This motivates the choice to learn direct mappings from features to optimal solutions of the second-stage decision problem. The joint

[2] The Appendix can be found online at https://arxiv.org/pdf/2311.13087.

model $\boldsymbol{J}_\phi : \mathcal{Z} \to \mathcal{X}$ is trained by LtO procedures, employing

$$\underset{\phi}{\text{Minimize}} \;\; \mathbb{E}_{(\boldsymbol{z},\boldsymbol{\zeta})\sim\Omega}\left[\ell^{\text{LtO}}\Big(\boldsymbol{J}_\phi(\boldsymbol{z}), \boldsymbol{\zeta}\Big)\right]. \tag{7}$$

This method can be seen as a generalization of the Learn-to-Optimize framework, to the Predict-then-Optimize setting. The key difference from the typical LtO setting is that problem parameters $\boldsymbol{\zeta} \in \mathcal{C}$ are not known as inputs to the model, but the correlated features $\boldsymbol{z} \in \mathcal{Z}$ are known instead. Therefore, estimated optimal solutions now take the form $\hat{\boldsymbol{x}} = \boldsymbol{J}_\phi(\boldsymbol{z})$ rather than $\hat{\boldsymbol{x}} = \boldsymbol{F}_\omega(\boldsymbol{\zeta})$. Notably, this causes the self-supervised LtO methods to become *supervised*, since the ground-truth parameters $\boldsymbol{\zeta} \in \mathcal{C}$ now act only as target data while the feature variable $\boldsymbol{z}$ takes the role of input data.

We refer to this approach as *Learning to Optimize from Features (LtOF)*. Figure 1 illustrates the key distinctions of LtOF relative to the other learning paradigms studied in this work. Figures (1c) and (1d) distinguish LtO from LtoF by a change in model's input space, from $\boldsymbol{\zeta} \in \mathcal{C}$ to $\boldsymbol{z} \in \mathcal{Z}$. This brings the framework into the same problem setting as that of the two-stage and end-to-end PtO approaches, shown in Figures (1a) and (1b). The key difference from the PtO approaches is that they produce an estimated optimal solution $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$ by using a true optimization solver, but applied to an imperfect parametric prediction $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$. In contrast, LtOF directly estimates optimal solution $\hat{\boldsymbol{x}}(\boldsymbol{z}) = \boldsymbol{J}_\phi(\boldsymbol{z})$ from features $\boldsymbol{z}$, circumventing the need to represent an estimate of $\boldsymbol{\zeta}$.

Advantages of LtOF Section 5 demonstrates various LtOF implementations which can greatly outperform two-stage methods in terms of the optimality of their learned solutions, and are competitive with EPO training based on exact differentiation through $\boldsymbol{x}^\star(\boldsymbol{\zeta})$. In contrast to EPO, this is achieved *without* access to exact optimization solvers, nor models of their derivatives. Two advantages of LtOF over EPO training, demonstrated in Section 5, are emphasized next.

4.1 Efficiency Benefits

Because the primary goal of the Learn-to-Optimize methodology is to achieve *faster solving times* than are possible with conventional optimization solvers, the LtOF approach broadly inherits this advantage. As learned neural network mappings, LtOF models of joint prediction and optimization can have order-of-magnitude lower runtimes than other PtO methods which require to solve the full optimization problem (1) at inference time (see Table 1). This enables the design of *real-time* PtO models within the LtOF framework.

4.2 Modeling Benefits

While EPO approaches require the use of problem-specific backpropagation rules, the LtOF framework instead requires an existing LtO implementation which can learn to solve the PtO problem's second-stage optimization component. Section 5.2 shows how several LtOF implementations can succeed where EPO training fails, on a problem with a nonconvex oscillating objective term (see Figure 5). This result is significant but intuitive, since derivatives through a nonconvex function often do not correspond to useful descent directions for minimization. By contrast, the LtOF approach learns to construct solutions to the nonconvex problem directly from features, without relying on their derivatives, by drawing from existing LtO methods [14, 24, 12] that reliably learn to solve both convex and nonconvex optimization.

5 Experimental Results

The LtOF approach is evaluated against *two-stage* and *EPO* baselines, on three Predict-Then-Optimize tasks, each with a distinct second stage optimization component $\boldsymbol{x}^\star : \mathcal{C} \to \mathcal{X}$, as in equation 1. These include a convex quadratic program (QP), a nonconvex QP variant, and a nonconvex program with sinusoidal constraints, to showcase the flexibility of LtOF over various problem forms.

Performance Criteria. Each PtO method considered in this section is evaluated on the basis of its downstream decisions $\hat{\boldsymbol{x}}$, which are required to be *feasible* to the problem constraints (1b). Subject to feasibility, the object is to minimize the expected ground-truth objective $f(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})$ as per (2). This is equivalent to minimizing expected *regret*, defined as the magnitude of suboptimality of a solution $\hat{\boldsymbol{x}}$ to problem (1) with respect to the ground-truth parameters:

$$\textit{regret}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) = f(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) - f(\boldsymbol{x}^\star(\boldsymbol{\zeta}), \boldsymbol{\zeta}). \tag{8}$$

LtOF methods. Three different LtOF implementations are evaluated on each PtO task, based on distinct Learn-to-Optimize methods, reviewed in detail in Section 5.4:

- *Lagrangian Dual Learning (***LD***)* [14], which augments a regression loss with penalty terms, updated to mimic a Lagrangian Dual ascent method to encourage the satisfaction of the problem's constraints.
- *Self-supervised Primal Dual Learning (***PDL***)* [24], which uses an augmented Lagrangian function to perform joint self-supervised training of primal and dual networks for solution estimation.
- *Deep Constraint Completion and Correction (***DC3***)* [12], which relies on a completion technique to enforce constraint satisfaction, while maximizing the empirical objective function in self-supervised training.

While several other Learn-to-Optimize methods have been proposed in the literature, the above-described collection represents diverse subset which is used to demonstrate the potential of adapting the LtO methodology as a whole to the PtO setting.

Feature generation. End-to-End Predict-Then-Optimize methods integrate learning and optimization to minimize the propagation of prediction errors–specifically, from feature mappings $\boldsymbol{z} \to \boldsymbol{\zeta}$ to the resulting decisions $\boldsymbol{x}^\star(\boldsymbol{\zeta})$ (regret). It's crucial to recognize that *even methods with high error propagation* can yield low regret *if the prediction errors are low*. To account for this, EPO studies often employ synthetically generated feature mappings to control prediction task difficulty [21]. Accordingly, for each experiment, we generate feature datasets $(\boldsymbol{z}_1, \ldots \boldsymbol{z}_N) \in \mathcal{Z}$ from ground-truth parameter sets $(\boldsymbol{\zeta}_1, \ldots \boldsymbol{\zeta}_N) \in \mathcal{C}$ using random mappings of increasing complexity. A feedforward neural network, $\boldsymbol{G}^k$, initialized uniformly at random with k layers, serves as the feature generator $\boldsymbol{z} = \boldsymbol{G}^k(\zeta)$. Evaluation is then carried out for each PtO task on feature datasets generated with $k \in \{1, 2, 4, 8\}$, keeping target parameters $\boldsymbol{\zeta}$ constant.

Baselines. In our experiments, LtOF models use feedforward networks with k hidden layers. For comparison, we also evaluate two-stage and, where applicable, EPO models, using architectures with m hidden layers for each $m \in \{1, 2, 4, 8\}$. Additionally, for each task and each $m \in \{1, 2, 4, 8\}$, we evaluate EPO model with pretrained optimization proxies, which predictive model $\boldsymbol{C}_\theta$ is, or is not, pretrained by MSE regression, as the *two-stage* predictive model. Further training specifics are provided in Appendix B.

Comparison to LtO setting. It is natural to ask how solution quality varies when transitioning from LtO to LtOF in a PtO setting, where solutions are learned directly from features. To address this question, each PtO experiment includes results from its analogous Learning to Optimize setting, where a DNN $\mathbf{F}_\omega : \mathcal{C} \to \mathcal{X}$ learns a mapping from

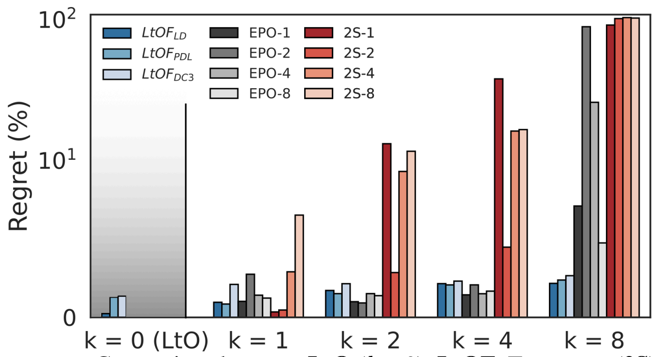

Figure 4: Comparison between **LtO** ($k=0$), **LtOF**, Two-stage (**2S**) and **EPO** ($k>1$) on the portfolio optimization. 2S(EPO)-m indicates that the prediction model of the respective PtO method is an m layer ReLU neural network. Plot y-axe is in semi log-scale.

	Method	**Portfolio**	**N/conv. QP**	**AC-OPF**
LtOF	LD it	**0.0003**	0.0000	**0.0004**
	LD fct	0.0000	0.0032	0.0516
	PDL it	**0.0003**	0.0000	0.0006
	PDL fct	0.0000	0.0032	0.0207
	DC3 it	0.0011	0.0001	-
	DC3 fct	0.0003	0.0000	-
PtO	PtO-1 et	0.0054	0.0122	2.5922
	PtO-2 et	0.0059	0.0104	2.5841
	PtO-4 et	0.0062	0.0123	2.5835
	PtO-8 et	0.0067	0.0133	2.5907

Table 1: Execution (*et*), inference (*it*), and feasibility correction (*fct*) times for **LtOF** and **PtO** (in seconds) for each sample. **Two-stage** methods execution times are comparable to PtO's ones.

the parameters $\boldsymbol{\zeta}$ of an optimization problem to its corresponding solution $\boldsymbol{x}^\star(\boldsymbol{\zeta})$. This is denoted $k=0$ (LtO), indicating the absence of any feature mapping. All figures report the regret obtained by LtO methods for reference, although they are not directly comparable to the Predict-then-Optimize setting.

Finally, all reported results are averages across 20 random seeds and we refer the reader to Appendix B for extensive additional details regarding experimental settings, architectures, data generation and hyperparamaters adopted.

5.1 Convex Quadratic Optimization

A well-known problem combining prediction and optimization is the Markowitz Portfolio Optimization [26]. This task has as its optimization component a convex Quadratic Program (QP):

$$\boldsymbol{x}^\star(\boldsymbol{\zeta}) = \arg\max_{\boldsymbol{x}\geq\mathbf{0}} \ \boldsymbol{\zeta}^T\boldsymbol{x} - \lambda\boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x}, \quad \text{s.t. } \mathbf{1}^T\boldsymbol{x} = 1 \tag{9}$$

in which parameters $\boldsymbol{\zeta} \in \mathbb{R}^D$ represent future asset prices, and decisions $\boldsymbol{x} \in \mathbb{R}^D$ represent their fractional allocations within a portfolio. The objective is to maximize a balance of risk, as measured by the quadratic form covariance matrix Σ, and total return $\boldsymbol{\zeta}^T\boldsymbol{x}$. Historical prices of $D = 50$ assets are obtained from the Nasdaq online database [23] and used to form price vectors $\boldsymbol{\zeta}_i, \ 1 \leq i \leq N$, with $N = 12,000$ individual samples collected from 2015-2019. In the outputs $\hat{\boldsymbol{x}}$ of each LtOF method, any feasibility violations are restored, at *low computational cost*, by first clipping $[\hat{\boldsymbol{x}}]_+$ to satisfy $\boldsymbol{x} \geq \mathbf{0}$, then dividing by its sum to satisfy $\mathbf{1}^T\boldsymbol{x} = 1$. The convex solver `cvxpy` [11] is used as the optimization component in each baseline method.

Results. Figure 4 shows the percentage regret due to LtOF variants based on *LD*, *PDL* and *DC3*. Two-stage and EPO models are evaluated for comparison, with predictive components given various numbers of layers. For feature complexity $k > 1$, each LtOF model outperforms the best two-stage model, increasingly with k and up to nearly *two orders of magnitude* when $k = 8$. The EPO model, trained using exact derivatives through (9) using the differentiable solver in `cvxpylayers` [1] is competitive with LtOF until $k = 4$, beyond which its best variant is outperformed by each LtOF variant. This shows the ability of LtOF models to reach high accuracy under complex feature mappings *without* access to optimization solvers *or* their derivatives, in training or inference, in contrast to EPO methods.

Table 1 presents LtOF inference times (*it*) and feasibility correction times (*fct*), compared with the per-sample execution times (*et*) for PtO methods. Run times for two-stage methods are closely aligned with those of EPO, and thus omitted. Notice how LtOF methods are at least an order of magnitude faster than the PtO baselines.

5.2 Nonconvex QP Variant

As a step in difficulty beyond convex QPs, this experiment considers a generic QP problem augmented with an oscillating objective term, resulting in a *nonconvex* optimization problem:

$$\mathbf{x}^\star(\boldsymbol{\zeta}) = \arg\min_{\boldsymbol{x}} \ \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{\zeta}^T\sin(\boldsymbol{x}) \tag{10a}$$

$$\texttt{s.t.} \ \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \ \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}. \tag{10b}$$

A variant of this formulation was used to evaluate the LtO methods proposed both in [12] and in [24]. Following those works, $\mathbf{0} \preccurlyeq \boldsymbol{Q} \in \mathbb{R}^{n\times n}$, $\boldsymbol{A} \in \mathbb{R}^{n_{\text{eq}}\times n}$, $\boldsymbol{b} \in \mathbb{R}^{n_{\text{eq}}}$, $\boldsymbol{G} \in \mathbb{R}^{n_{\text{ineq}}\times n}$, $\boldsymbol{h} \in \mathbb{R}^{n_{\text{ineq}}}$ and each $\boldsymbol{\zeta_i}$ has elements drawn from a normal distribution. The EPO baseline for comparison differentiates (10) via the fixed-point conditions of a locally convergent Projected Gradient Descent method, implemented using the `fold-opt` library [17]; details on this EPO model are in Appendix B. Feasibility is restored by a projection onto the feasible set, efficiently calculated by as a *convex* QP. The problem dimensions are $n = 50$ $n_{\text{eq}} = 25$, and $n_{\text{ineq}} = 25$.

Results. Figure 5 (top) shows regret due to LtOF models based on *LD*, *PDL* and *DC3*, along with two baseline PtO methods. The best two-stage models perform poorly for all values of k, implying that the regret is particularly sensitive to prediction errors in the oscillating term. As alluded earlier, EPO training based on differentiation of (10) performs even worse. This is intuitive, since gradients of the objective in (10a) do not correspond to descent directions, due to its nonconvex sin term [8]. Thus, backpropagation through (10) guides the EPO model (3) to poor local minima in gradient descent training. The LtOF models achieve over $4\times$ times lower regret than the best baselines, suggesting strong potential for LtOF when predicting parameters of nonconvex objective functions. Additionally, the LtOF methods execute several times *faster* than both baselines, after restoring feasibility.

5.3 Nonconvex AC-Optimal Power Flow

Given a vector of marginal costs $\boldsymbol{\zeta}$ for each generator in an electrical grid, the AC-Optimal Power Flow problem optimizes the generation and dispatch of electrical power from generators to nodes with predefined demands. The objective is to minimize cost, while meeting demand exactly. The full optimization problem is specified in Appendix A, where a quadratic cost objective is minimized subject to

	Method	**Portfolio** $k=2$	**Portfolio** $k=4$	**Portfolio** $k=8$	**Nonconvex QP** $k=2$	**Nonconvex QP** $k=4$	**Nonconvex QP** $k=8$	**AC-OPF** $k=2$	**AC-OPF** $k=4$	**AC-OPF** $k=8$
LtOF	LD Regret	1.7170	2.1540	**2.1700**	9.9279	**9.7879**	9.5473	**0.0748**	**0.3762**	**0.7231**
	LD Regret (*)	1.5739	2.0903	2.1386	9.9250	9.8211	9.5556	0.0013	0.0071	0.0195
	LD Violation (*)	0.0010	0.0091	0.0044	0.0148	0.0162	0.0195	0.0020	0.0037	0.0042
	PDL Regret	1.5150	2.0720	2.3830	**7.2699**	10.747	**7.6399**	0.8714	0.8012	0.8373
	PDL Regret (*)	1.4123	1.9372	2.0435	7.2735	10.749	7.6394	0.0260	0.0243	0.0242
	PDL Violation (*)	0.0001	0.0003	0.0003	0.0028	0.0013	0.0015	0.0000	0.0002	0.0002
	DC3 Regret	2.1490	2.3140	2.6600	14.271	11.028	10.666	-	-	-
	DC3 Regret (*)	2.1490	2.3140	2.6600	13.779	11.755	10.849	-	-	-
	DC3 Violation (*)	0.0000	0.0000	0.0000	0.5158	0.5113	0.5192	-	-	-
EPO	EPO Regret (Best)	**0.9220**	**1.4393**	4.7495	103.978	117.651	130.379	-	-	-
	EPO w/ Proxy Regret (Best)	154.40	119.31	114.69	812.75	804.26	789.50	389.04	413.89	404.74
	EPO w/ Proxy and pretrained prediction Regret (Best)	14.783	23.052	22.158	214.53	198.02	241.93	52.344	55.195	60.052
	Two-Stage Regret (Best)	2.8590	4.4790	91.326	36.168	37.399	38.297	1.4090	1.5280	2.4740

Table 2: Percentage Regret and Constraint Violations for all experiments. (*) denotes "Before Restoration". Missing entries denote experiments not achievable with the associated method.

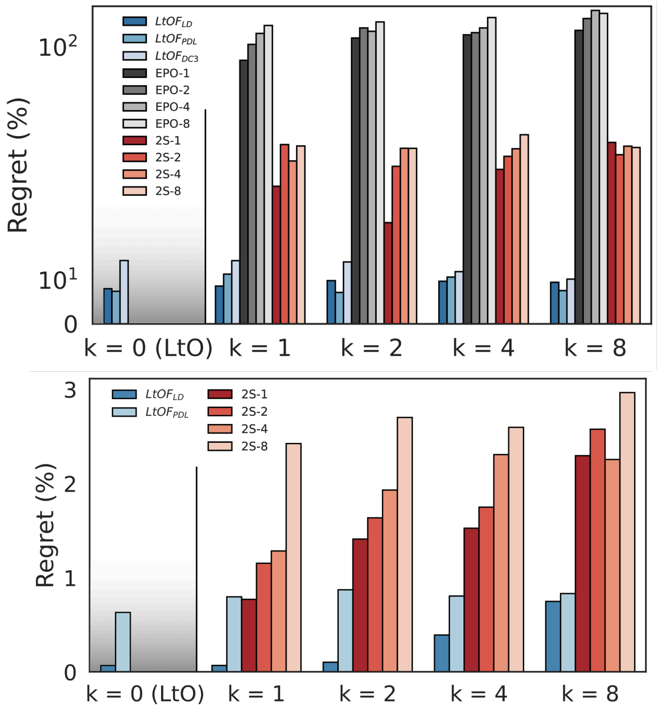

Figure 5: Comparison between LtO ($k = 0$), LtOF, and Two Stage Method (2S) on the nonconvex QP (**top**) and AC-OPF case (**bottom**). Top plot y-axe is in semi log-scale.

nonconvex physical and engineering power systems constraints. This experiment simulates an energy market situation in which generation costs are as-yet unknown to the power system planners, and must be estimated based on correlated data. The goal is to predict costs so as to minimize cost-regret over an example network with 54 generators, 99 demand loads, and 118 buses taken from the NESTA energy system test case archive [10]. Cost data are generated as perturbations around the test case's cost values as described in Appendix B. Feasibility is restored after LtOF by Newton's method on the constraint violations as specified in Appendix A.

Results. Figure 5 (bottom) presents regret percentages, comparing LtOF to a two-stage baseline. Note that no general EPO exists for handling such nonconvex decision components. It is found empirically that the Lagrangian Hessian is not positive-semidefinite, preventing even a differentiable convex QP appromixation at its optima, as leveraged in [31]. Further, the solving times reported in Table 1, due to Casadi solver [4], are prohibitively slow for EPO training. *DC3* is also omitted, following [24], due to its incompatibility with the LtO variant of this experiment. Notice how the *best* two-stage model is outperformed by the *LD* variant of LtOF for $k > 1$, and also by the *PDL* variant for $k > 2$. Notably, PDL appears robust to increases in the feature mapping complexity k. On the other hand, it is outperformed in each case by the LD variant. *Notice how, in the complex feature mapping regime $k > 1$, the best LtOF variant achieves up to an order of magnitude improvement in regret relative to the most competitive two-stage method.* Additionally, LtOF methods report orders-of-magnitude speed advantages in Table 1.

Table 2 collects an abridged set of accuracy results due to each LtOF implementation and PtO baseline, across all experimental tasks. In particular, average constraint violations and objective regret are shown in addition to regret after restoring feasibility. Infeasible results are shown in grey for purposes of comparing the regret loss due to restoration. Best results on each task are shown in bold. Additionally, regret achieved by the EPO framework with pretrained proxies (discussed in Section 3) are included. Average regrets between 10 and 1000 times higher than LtOF illustrate the effect of their distributional shifts on accuracy. Notice how, in the context of complex feature mappings $k \geq 2$, LtOF is competitive with EPO, while bringing substantial computational advantages, and consistently outperforms two-stage methods, often, beyond an order of magnitude in regret.

5.4 *Learning to Optimize Methods*

Finally, this section describes in more detail those LtO methods which were adapted to solve PtO problems by LtOF, in the above experiments. Each description below assumes a DNN model $\mathcal{F}$ and its weights ω, which acts on problem parameters $\boldsymbol{\zeta}$ specifying an instance of problem (1), to produce an estimate of the optimal solution $\hat{\boldsymbol{x}} := \boldsymbol{F}_\omega(\boldsymbol{\zeta})$, so that $\hat{\boldsymbol{x}} \approx \boldsymbol{x}^\star(\boldsymbol{\zeta})$.

Lagrangian Dual Learning (LD). Fioretto et al. [14] constructs the following modified Lagrangian as a loss function for training the predictions $\hat{\boldsymbol{x}} = \boldsymbol{F}_\omega(\boldsymbol{\zeta})$:

$$\mathcal{L}_{\textbf{LD}}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) = \|\hat{\boldsymbol{x}} - \boldsymbol{x}^\star(\boldsymbol{\zeta})\|_2^2 + \boldsymbol{\lambda}^T \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})\right]_+ + \boldsymbol{\mu}^T \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}). \quad (11)$$

At each iteration of LD training, the model $\boldsymbol{F}_\omega$ is trained to minimize the loss $\mathcal{L}_{\textbf{LD}}$. 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 [9]. In this way, the method minimizes a balance of constraint violations and proximity to the precomputed target optima $\boldsymbol{x}^\star(\boldsymbol{\zeta})$.

Self-Supervised Primal-Dual Learning (PDL). Park and Van Hentenryck [24] use an augmented Lagrangian loss function

$$\begin{aligned}\mathcal{L}_{\textbf{PDL}}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) = f(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \hat{\boldsymbol{\lambda}}^T \boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \hat{\boldsymbol{\mu}}^T \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \\ \frac{\rho}{2}\left(\sum_j \nu(g_j(\hat{\boldsymbol{x}})) + \sum_j \nu(h_j(\hat{\boldsymbol{x}}))\right),\end{aligned} \tag{12}$$

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, and updated by an augmented Lagrangian method [9] after training $\hat{\boldsymbol{x}} = \boldsymbol{F}_\omega(\boldsymbol{\zeta})$ to minimize equation 5.4. In addition to the primal network $\boldsymbol{F}_\omega$, a dual network $\mathcal{D}_{\boldsymbol{\zeta}}$ learns to store updates of the multipliers for each instance, and predict them as $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) = \mathcal{D}_{\boldsymbol{\zeta}}(\boldsymbol{\zeta})$ 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. [12] use the loss function

$$\mathcal{L}_{\textbf{DC3}}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) = f(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \lambda \| \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})\right]_+ \|_2^2 + \mu \|\boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})\|_2^2 \tag{13}$$

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 $\hat{\boldsymbol{x}} = \hat{\boldsymbol{F}}_\omega(\boldsymbol{\zeta})$ 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 $\| \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})\right]_+ \|^2$. These completion and correction steps form the function $\boldsymbol{S}$, where $\boldsymbol{F}_\omega(\boldsymbol{\zeta}) = \boldsymbol{S} \circ \hat{\boldsymbol{F}}_\omega(\boldsymbol{\zeta})$.

6 Related Work

This section gives an overview of related work in the Predict-Then-Optimize setting. While the idea is general and has broader applications, differentiation through the optimization of (1) is central to EPO approaches. Backpropagation of parametric quadratic programming problems was introduced by [3], which implicitly differentiates the solution via its KKT conditions of optimality [8]. Agrawal et al. [2] followed by proposing a differentiable cone programming solver, which uses implicit differentiation of problem-specific optimality conditions. That framework is leveraged by [1] to solve and differentiate general convex programs, by pairing it with a symbolic system for conversion of convex programs to canonical convex cone programs.

For many practical problems with discrete structure, such as linear programs, the mapping defined by (1) does not have well-defined derivatives, necessitating a suitable approximation in EPO training. [13, 19] propose a surrogate loss function for (2) in cases where f is linear, which admits useful subgradients for stochastic gradient descent training. [32] proposes backpropagation through linear programs by adding a smooth quadratic term to the objective and differentiating the resulting QP problem via [3]. [7] also propose backpropagation through linear programs but by smoothing the mapping (1) through random noise perturbations to the objective function. [25] form approximate derivatives through linear optimization of discrete variables, by using a finite difference approximation between a pair of solutions with perturbed input parameters. Finally, [18] use the barrier function of an interior point method with early stopping to provide a smoothed surrogate model for differentiable linear programming.

A few recent works have addressed the topic of PtO learning without computing or approximating derivatives through optimization. In [28], [29], [33] neural networks are trained to function as surrogate models of solution regret, for use in EPO training. In [20], PtO is recast as a learning to rank (LTR) problem and solved with various LTR methods, in favor of EPO training.

7 Discussion and Conclusions

While the typical role of Learning to Optimize is to accelerate the solution of optimization problems, this paper demonstrates a novel use case: solving problems in the Predict-Then-Optimize scope. The adaptations of LtO described in this paper bring distinct advantages in the PtO setting, including real-time inference and enhanced ability to handle some PtO problems with nonconvex optimization.

Another advantage of the Learning to Optimize from Features approach to PtO settings is its generic framework, which enables it to leverage a variety of existing techniques from the LtO literature. On the other hand, as such, a particular implementation of LtOF may inherit any limitations of the specific LtO method that it adopts. Future work should focus on understanding to what extent a broader variety of LtO methods can be applied to PtO settings; given the large variety of existing works in the area, such a task is beyond the scope of this paper. In particular, this paper does not investigate of the use of *combinatorial* optimization proxies in learning to optimize from features. Such methods tend to use a distinct set of approaches from those studied in this paper, often relying on training by reinforcement learning [6, 15, 22], and are not suited for capturing broad classes of optimization problems. As such, this direction is left to future work.

One *disadvantage* inherent to LtOF, compared to EPO, is the inability to recover parameter estimations from the predictive model, since optimal solutions are predicted directly from features. Although it is not required for the PtO problem setting, this may pose a challenge if transferring the learned parameters to external solvers is desirable. Furthermore, LtOF cannot be applied to PtO problems whose optimization component does not have an effective LtO solution.

By showing that effective Predict-Then-Optimize models can consist solely of Learning-to-Optimize methods, this paper has aimed to provide a unifying perspective on these as-yet distinct problem settings. The flexibility of its approach has been demonstrated by showing superior performance over PtO baselines with diverse problem forms. As the advantages of LtO are often best realized in combination with application-specific techniques, it is hoped that future work can build on these findings to maximize the practical benefits offered by LtO in settings that require data-driven decision-making.

8 Acknowledgements

This research is partially supported by NSF grants RI-2007164, RI-2232054, EPCN-2242931, AI Institute-2112533, and CAREER-2143706. Its views and conclusions are those of the authors only.

References

[1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

[2] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi. Differentiating through a cone program. *arXiv preprint arXiv:1904.09043*, 2019.

[3] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[4] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 2019.

[5] H. Attouch, J. Bolte, and B. F. Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–seidel methods. *Mathematical Programming*, 137(1):91–129, 2013.

[6] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv:1611.09940*, 2017.

[7] Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

[8] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1): 1–122, 2011.

[10] C. Coffrin, D. Gordon, and P. Scott. Nesta, the nicta energy system test case archive. *arXiv preprint arXiv:1411.0359*, 2014.

[11] S. Diamond and S. Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.

[12] P. L. Donti, D. Rolnick, and J. Z. Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

[13] A. N. Elmachtoub and P. Grigas. Smart "predict, then optimize". *Management Science*, 2021.

[14] F. Fioretto, P. V. Hentenryck, T. W. Mak, C. Tran, F. Baldo, and M. Lombardi. Lagrangian duality for constrained deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 118–135. Springer, 2020.

[15] W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[16] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder. End-to-end constrained optimization learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4475–4482, 2021. doi: 10.24963/ijcai.2021/610. URL https://doi.org/10.24963/ijcai.2021/610.

[17] J. Kotary, M. H. Dinh, and F. Fioretto. Backpropagation of unrolled solvers with folded optimization. *arXiv preprint arXiv:2301.12047*, 2023.

[18] J. Mandi and T. Guns. Interior point solving for lp-based prediction+optimisation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[19] J. Mandi, P. J. Stuckey, T. Guns, et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1603–1610, 2020.

[20] J. Mandi, V. Bucarey, M. M. K. Tchomba, and T. Guns. Decision-focused learning: Through the lens of learning to rank. In *International conference on machine learning*, pages 14935–14947. PMLR, 2022.

[21] J. Mandi, J. Kotary, S. Berden, M. Mulamba, V. Bucarey, T. Guns, and F. Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *arXiv preprint arXiv:2307.13565*, 2023.

[22] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pages 270–288, 2019.

[23] Nasdaq. Nasdaq end of day us stock prices. https://data.nasdaq.com/databases/EOD/documentation, 2022. Accessed: 2023-08-15.

[24] S. Park and P. Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4052–4060, 2023.

[25] M. V. Pogančić, A. Paulus, V. Musil, G. Martius, and M. Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations (ICLR)*, 2020.

[26] M. Rubinstein. Markowitz's" portfolio selection": A fifty-year retrospective. *The Journal of finance*, 57(3):1041–1045, 2002.

[27] R. Sambharya, G. Hall, B. Amos, and B. Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In *Learning for Dynamics and Control Conference*, pages 220–234. PMLR, 2023.

[28] S. Shah, K. Wang, B. Wilder, A. Perrault, and M. Tambe. Decision-focused learning without decision-making: Learning locally optimized decision losses. *Advances in Neural Information Processing Systems*, 35:1320–1332, 2022.

[29] S. Shah, A. Perrault, B. Wilder, and M. Tambe. Leaving the nest: Going beyond local loss functions for predict-then-optimize. 2024.

[30] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8:120388–120416, 2020.

[31] K. Wang, A. Perrault, A. Mate, and M. Tambe. Scalable game-focused learning of adversary models: Data-to-decisions in network security games. In *AAMAS*, pages 1449–1457, 2020.

[32] B. Wilder, B. Dilkina, and M. Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 1658–1665, 2019.

[33] A. Zharmagambetov, B. Amos, A. Ferber, T. Huang, B. Dilkina, and Y. Tian. Landscape surrogate: Learning decision losses for mathematical optimization under partial information. *Advances in Neural Information Processing Systems*, 36, 2024.

A Optimization Problems

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

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

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

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 $2\bar{e}$ 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 $2\bar{h}$ capture *Ohm's Law*. Finally, Constraint equation $2\bar{i}$ and equation 2j capture *Kirchhoff's Current Law* enforcing flow conservation at each bus.

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

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

$$\Delta \mathbf{x}_n = -\mathbf{J}^{-1}(\mathbf{x}_n)\mathbf{f}(\mathbf{x}_n) \tag{15}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta \mathbf{x}_n \tag{16}$$

Figure 7: Newton's method.

Feasibility restoration (AC-Optimal Power Flow) Being an approximation, a LtO solution $(\hat{\boldsymbol{p}}^g, \hat{v})$ may not satisfy the original constraints. Feasibility can be restored by applying Netwon's method, which is reported in Figure 7. It is an iterative method that produces better approximation to the root $\mathbf{x} \in \mathbb{R}^p$, of a function $f(x) \in \mathbb{R}^m$ by iteratively solving a non-linear system of equations. If solving for $\mathbf{x}_{n+1}$, given $\mathbf{x}_n$, the method requires to compute the inverse of the Jacobian $J(\mathbf{x}_n) \in \mathbb{R}^{m \times p}$. From Eq. 15 and 16, it can be noticed that $J(\mathbf{x}_n)\Delta\mathbf{x}_n = -f(\mathbf{x}_n)$, and so is possible to avoid computing the inverse of the Jacobian J of f, and solving a linear system of equation for the unknown $\Delta\mathbf{x}_n$. In the context of restoring feasibility of the LtO solution to the AC-Optimal Power Flow problem, f represents the set of inequality and equality constraint functions, from equation 2b to equation $2\bar{h}$, while $x = [v, \theta, p^g, q^g]^T$. Since the method requires each $f_i(x), i = 1, \ldots m$ to be an equality function, to construct a system of only equations, a $\mathbf{ReLU}(f(x)) = \max(0, f(x))$ is applied to each inequality function. For the AC-OPF experiment, the number of constraint function $m = 602$ while the number of variables $p = 472$; being $m > p$, the inverse J^{-1} of the Jacobian J is the generalized inverse $J^{+} = (J^T J)^{-1} J^T$, and $\Delta\mathbf{x}_n$ is the solution in the least square sense. The convergence of the method requires the starting point x_0 to be such that the 2-norm $\|f(x_0)\|_2 \ll 1$. In the experiments, we verified that such assumption holds as evidenced by the minimal Constraint Violation achieved by each LTO method adopted (see Table 2). We consider the method to have converged when the absolute value of each constraint function $|f_i(x_n)| < 1\text{e}{-6}$.

B Experimental Details

B.1 *Portfolio Optimization Dataset*

The stock return dataset is prepared exactly as prescribed in [27]. 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. 9, $\lambda = 2.0$.

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.

B.2 *Nonconvex Optimization Dataset*

The nonconvex optimization dataset has 2400 samples, divided into training, validation and test set, each consisting of 2000, 200 and 200 samples, respectively. With reference to 10 and 10a, the matrix $Q = \mu I$, where $\mu \in \mathbb{R}^n \sim \mathcal{U}(0, 1)$. The parameter $\zeta \sim \mathcal{U}(0, 5)$ and the matrix A and G are both drawn from the normal distribution $\mathcal{N}(0, 1)$. The right-hand side of the equality constraint $b \sim \mathcal{U}(-1, 1)$, while the right-hand side of the inequality constraint $h = \sum_{i=1}^{n} |M_{ij}|$, where $M = GA^{+}$ and $A^{+} = (A^T A)^{-1} A^T$.

B.3 Nonconvex AC-OPF Dataset

The nonconvex optimization dataset has 10000 samples, divided into training, validation and test set, each consisting of 8334, 833 and 833 samples, respectively. The Nonconvex AC-OPF Dataset is constructed by applying random perturbations of the cost values found in NESTA benchmark case 118. More specifically, a perturbation $\mu \in \mathcal{U}(0, 100)$ is applied to each generator cost value ζ_i.

B.4 Nonconvex EPO Baselines

The nonconvex QP variant (10) of Section 5 admits derivatives for EPO training by differentiation of the fixed-point conditions of a locally convergent solution method. Projected Gradient Descent is known to be locally convergent in nonconvex optimization [5], and it is found empirically to converge locally on the problem (10).

On a problem of form

$$\boldsymbol{x}^\star(\boldsymbol{\zeta}) = \arg\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}, \boldsymbol{\zeta})) \\ \text{s.t.} \quad \boldsymbol{x} \in \mathcal{S}$$

one step of the method follows

$$\boldsymbol{x}_{k+1} = \mathit{proj}_{\mathcal{S}}(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k, \boldsymbol{\zeta})) \tag{5}$$

leading to the fixed-point conditions

$$\boldsymbol{x}^\star = \mathit{proj}_{\mathcal{S}}(\boldsymbol{x}^\star - \alpha \nabla f(\boldsymbol{x}^\star, \boldsymbol{\zeta})) \tag{6}$$

whose implicit differentiation results in a linear system which can be solved for $\frac{\partial \boldsymbol{x}^\star}{\partial \boldsymbol{\zeta}}$:

$$\frac{\partial \boldsymbol{x}^\star}{\partial \boldsymbol{\zeta}} = \frac{\partial}{\partial \boldsymbol{x}^\star} \mathit{proj}_{\mathcal{S}}(\boldsymbol{x}^\star - \alpha \nabla f(\boldsymbol{x}^\star, \boldsymbol{\zeta})) \cdot \frac{\partial \boldsymbol{x}^\star}{\partial \boldsymbol{\zeta}} \tag{7a}$$

$$+ \frac{\partial}{\partial \boldsymbol{\zeta}} \mathit{proj}_{\mathcal{S}}(\boldsymbol{x}^\star - \alpha \nabla f(\boldsymbol{x}^\star, \boldsymbol{\zeta})) \tag{7b}$$

Differentiation of the inner projection step is performed by `cvxpy` [1], while the system (7) is constructed and solved by `fold-opt` [17].

B.5 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 chosen as the best among $\{5e-2, 1e-2, 5e-3, 1e-3, 5e-4, 1e-4\}$. For each task, an early stopping criteria based on the evaluation of the test-set percentage regret after restoring feasibility, is adopted to all the LtO(F) the proxies, the predictive EPO (w/o) proxy model, and pre-trained predictive model; 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 LtO proxies are 2-layers ReLU neural networks with dropout equal to 0.1 and batch normalization. All the LtOF 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 [14], [24] and [12] 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.

B.5.1 Convex Quadratic Optimization and Non Convex Quadratic Optimization

LD

Parameter	Values
$\lambda(0)$	**0.1**, 0.5, 1.0, 5.0, 10.0, 50.0
$\mu(0)$	0.1, **0.5**, 1.0, 5.0, 10.0, 50.0
Training epochs	50, 100, **200**, 300, 500
LD step size	1.0, 0.1, 0.01, **0.001**, 0.0001

PDL

Parameter	Values
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

Parameter	Values
$\lambda + \mu$	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

Non Convex AC-Optimal Power Flow

LD

Parameter	Values
$\lambda(0)$	**0.1**, 0.5, 1.0, 5.0, 10.0, 50.0
$\mu(0)$	**0.1**, 0.5, 1.0, 5.0, 10.0, 50.0
Training epochs	50, 100, 200, **300**, 500
LD step size	**1.0**, 0.1, 0.01, 0.001, 0.0001

PDL

Parameter	Values
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**