Predict-Then-Optimize by Proxy: Learning Joint Models of Prediction and Optimization

James Kotary
University of Virginia
jk4pn@virginia.edu

Vincenzo Di Vito
University of Virginia
eda8pc@virginia.edu

Jacob Christopher
University of Virginia
csk4sr@virginia.edu

Pascal Van Hentenryck
Georgia Institute of Technology
pvh@isye.gatech.edu

Ferdinando Fioretto
University of Virginia
fioretto@virginia.edu

Abstract

Many real-world decision processes are modeled by optimization problems whose defining parameters are unknown and must be inferred from observable data. The Predict-Then-Optimize framework uses machine learning models to predict unknown parameters of an optimization problem from features before solving. Recent works show that decision quality can be improved in this setting by solving and differentiating the optimization problem in the training loop, enabling end-to-end training with loss functions defined directly on the resulting decisions. However, this approach can be inefficient and requires handcrafted, problem-specific rules for backpropagation through the optimization step. This paper proposes an alternative method, in which optimal solutions are learned directly from the observable features by 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, accurate, and flexible 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:

$$\begin{aligned} \boldsymbol{x}^\star(\boldsymbol{\zeta}) = \underset{\boldsymbol{x}}{\arg\min} \;\; & f(\boldsymbol{x}, \boldsymbol{\zeta}) \\ \textit{such that:} \;\; & \boldsymbol{g}(\boldsymbol{x}) \leq 0, \;\; \boldsymbol{h}(\boldsymbol{x}) = 0, \end{aligned} \tag{1}$$

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 (Elmachtoub & Grigas, 2021). 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 models trained to predict local transit times 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 overall commutes, with respect to the true city-block transit times $\boldsymbol{\zeta}$.

However, direct training of predictions from observable features to problem parameters tends to generalize poorly with respect to the ground-truth optimality achieved by a subsequent decision model (Mandi et al., 2023; Kotary et al., 2021b). To address this challenge, *End-to-end Predict-Then-Optimize* (EPO) (Elmachtoub & Grigas, 2021) has emerged as a transformative paradigm in data-driven decision making in which predictive models are trained by directly minimizing loss functions defined on the downstream optimal solutions $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$.

On the other hand, EPO implementations 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 (Amos & Kolter, 2017; Agrawal et al., 2019a). Furthermore, difficult decision models involving nonconvex or discrete optimization may not admit well-defined backpropagation rules.

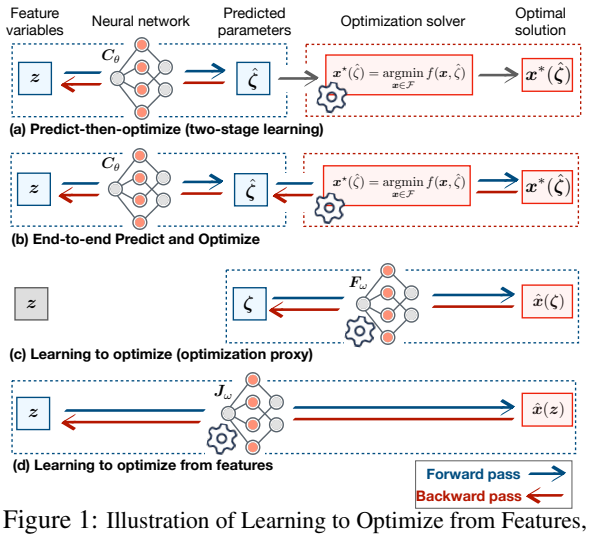

Figure 1: Illustration of Learning to Optimize from Features, in relation to other learning paradigms.

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. Such 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 (Vesselinova et al., 2020; Fioretto et al., 2020a). Several LtO methods specialize in training proxies to solve difficult problem forms, especially those involving nonconvex optimization.

The proposed methodology of this paper, called *Learning to Optimize from Features* (LtOF), recognizes that existing Learn-to-Optimize 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, directly transferring a pretrained optimization proxy into the training loop of an EPO model leads to poor accuracy, as shown in Section 3, due to the inability of LtO proxies to generalize outside their training distribution. To circumvent this distributional shift issue, this paper shows how to adapt the LtO methodology to learn optimal solutions directly from features.

Contributions. In summary, this paper makes the following novel contributions: **(1)** It investigates the use of pretrained LtO proxy models as a means to approximate the decision-making component of the PtO pipeline, and demonstrates a distributional shift effect between prediction and optimization models that leads to loss of accuracy in end-to-end 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 distribution 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, under complex feature-to-parameter mappings. Besides the performance improvement over two-stage approaches, *the results show that difficult nonconvex optimization components can be incorporated into PtO pipelines naturally*, extending the flexibility and expressivity of PtO models.

2 Problem Setting and Background

In the Predict-then-Optimize (PtO) setting, a (DNN) prediction model $\boldsymbol{C}_\theta : \mathcal{Z} \to \mathcal{C} \subseteq \mathbb{R}^k$ first takes as input a feature vector $\boldsymbol{z} \in \mathcal{Z}$ to produce predictions $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$. The model $\boldsymbol{C}$ is itself parametrized by learnable weights θ. The predictions $\hat{\boldsymbol{\zeta}}$ are used to parametrize an optimization model of the form (1), which is then solved to produce optimal decisions $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}) \in \mathcal{X}$. We call these two components, respectively, the *first* and *second* stage models. Combined, their goal is to produce decisions $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$ which minimize the ground-truth objective value $f(\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}), \boldsymbol{\zeta})$ given an observation of $\boldsymbol{z} \in \mathcal{Z}$. Concretely, assuming a dataset of samples $(\boldsymbol{z}, \boldsymbol{\zeta})$ drawn from a joint distribution Ω, the goal is to learn a model $\boldsymbol{C}_\theta : \mathcal{Z} \to \mathcal{C}$ producing predictions $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$ which achieves

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

This optimization is equivalent to minimizing expected *regret*, defined as the magnitude of suboptimality of $\boldsymbol{x}^\star(\hat{\zeta})$ with respect to the ground-truth parameters:

$$\textit{regret}(\boldsymbol{x}^\star(\hat{\zeta}), \zeta) = f(\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}), \boldsymbol{\zeta}) - f(\boldsymbol{x}^\star(\boldsymbol{\zeta}), \boldsymbol{\zeta}). \tag{3}$$

Two-stage Method. A common approach to training the prediction model $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$ is the *two-stage* method, which trains to minimize the mean squared error loss $\ell(\hat{\boldsymbol{\zeta}}, \boldsymbol{\zeta}) = \|\hat{\boldsymbol{\zeta}} - \boldsymbol{\zeta}\|_2^2$, without taking into account the second stage optimization. While directly minimizing the prediction errors is confluent with the task of optimizing ground-truth objective $f(\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}), \boldsymbol{\zeta})$, the separation of the two stages in training leads to error propagation with respect to the optimality of downstream decisions, due to misalignment of the training loss with the true objective (Elmachtoub & Grigas, 2021).

End-to-End Predict-Then-Optimize. Improving on the two-stage method, the End-to-end Predict-end-Optimize (EPO) approach trains directly to optimize the objective $f(\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}}), \boldsymbol{\zeta})$ by gradient descent, which is enabled by finding or approximating the derivatives through $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$. This allows for end-to-end training of the PtO goal (2) directly as a loss function, which consistently outperforms two-stage methods with respect to the evaluation metric (2), especially when the mapping $\boldsymbol{z} \to \boldsymbol{\zeta}$ is *difficult to learn* and subject to significant prediction error. Such an integrated training of prediction and optimization is referred to as *Smart Predict-Then-Optimize* (Elmachtoub & Grigas, 2021), *Decision-Focused Learning* (Wilder et al., 2019), or End-to-End Predict-Then-Optimize (EPO) (Tang & Khalil, 2022). This paper adopts the latter term throughout, for consistency. Various implementations of this idea have shown significant gains in downstream decision quality over the conventional two-stage method. 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 the topic is reported in Appendix 6.

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

Despite their advantages over the two-stage, EPO methods 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 or provide useful descent directions, 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, at each iteration of training, which is often inefficient even for simple optimization problems.

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 $\boldsymbol{\zeta} \rightarrow \boldsymbol{x}^\star(\boldsymbol{\zeta})$ 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 optimal solutions are learned end-to-end from the observable features $\boldsymbol{z}$.

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 that 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 *Learning-to-Optimize (LtO)* (Kotary et al., 2021b). Here the goal is to learn a mapping $\mathbf{F}_\omega : \mathcal{C} \rightarrow \mathcal{X}$ 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* Ω^F of problem parameters $\boldsymbol{\zeta}$.

A variety of LtO methods have been proposed, many of which specialize in learning to solve problems of a specific form. Some are based on supervised learning, in which case 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 $\boldsymbol{F}_\omega$:

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

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 (Fioretto et al., 2020a; Park & Van Hentenryck, 2023), or designing $\boldsymbol{\mathcal{S}}$ to restore feasibility (Donti et al., 2021), as reviewed in Appendix 5.1. 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 (Kotary et al., 2021b).

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:

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

in which the solver output $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$ of problem (2) is replaced with the prediction $\hat{\boldsymbol{x}}$ obtained by LtO model $\boldsymbol{F}_\omega$ on input $\hat{\boldsymbol{\zeta}}$ (gray color highlights that the model is pretrained, before freezing its weights ω).

However, a fundamental 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

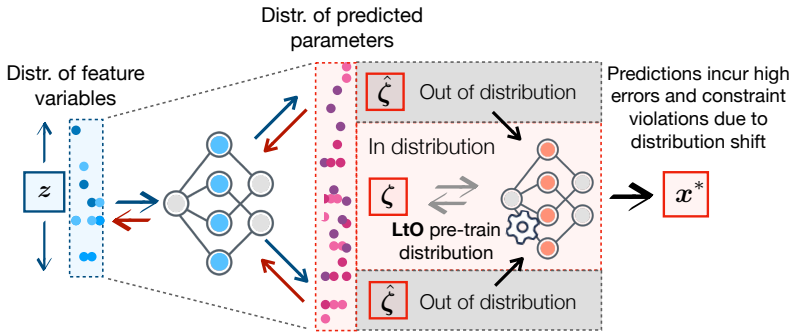

Figure 2: A distribution shift between the training distribution of a LtO proxy and the parameter predictions during training leads to inaccuracies in the proxy solver.

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, adversely affecting 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 $\boldsymbol{\zeta} \sim \Omega^F$ (shown in black). The experimental results presented in Tables 2, 3, and 4 reinforce these observations. While each proxy solver performs well within its training distribution, their effectiveness deteriorates sharply when utilized as described in equation 5. This degradation is observed irrespective of any normalization applied to the proxy's input parameters during EPO training.

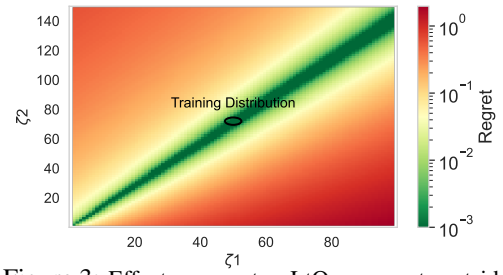

Figure 3: Effect on regret as LtO proxy acts outside its training distribution.

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:

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

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 us to the 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 model $\boldsymbol{J}_\phi : \mathcal{Z} \to \mathcal{X}$ is trained by Learning-to-Optimize 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 direct adaptation of the Learn-to-Optimize framework to the Predict-then-Optimize setting. The key difference from the typical LtO setting, described in Section 3, 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 separate 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 the paper. 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, illustrated 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 $\boldsymbol{\zeta}$.

4.1 Sources of Error

Both the PtO and LtOF methods yield solutions subject to *regret*, which measures suboptimality relative to the true parameters $\boldsymbol{\zeta}$, as defined in Equation 3. However, while in end-to-end and, especially, in the two-stage PtO approaches, the regret in $\boldsymbol{x}^\star(\hat{\boldsymbol{\zeta}})$ arises from imprecise parameter predictions $\hat{\boldsymbol{\zeta}} = \boldsymbol{C}_\theta(\boldsymbol{z})$ (Mandi et al., 2023), in LtOF, the regret in the inferred solutions $\hat{\boldsymbol{x}}(\boldsymbol{z}) = \boldsymbol{J}_\phi(\boldsymbol{z})$ arises due to imperfect learning of the proxy optimization. This error is inherent to the LtO methodology used to train the joint prediction and optimization model $\boldsymbol{J}_\phi$, and persists even in typical LtO, in which $\boldsymbol{\zeta}$ are precisely known. In principle, a secondary source of error can arise from imperfect learning of the implicit feature-to-parameter mapping $\boldsymbol{z} \to \boldsymbol{\zeta}$ within the joint model $\boldsymbol{J}_\phi$. However, these two sources of error are indistinguishable, as the prediction and optimization steps are learned jointly. Finally, depending on the specific LtO procedure adopted, a further source of error arises when small violations to the constraints occur in $\hat{\boldsymbol{x}}(\boldsymbol{z})$. In such cases, restoring feasibility (e.g, through projection or heuristics steps) often induces slight increases in regret (Fioretto et al., 2020a).

Despite being prone to optimization error, Section 5 shows that Learning to Optimize from Features greatly outperforms two-stage methods, and is competitive with EPO training based on exact differentiation through $\boldsymbol{x}^\star(\boldsymbol{\zeta})$, when the feature-to-parameter mapping $\boldsymbol{z} \to \boldsymbol{\zeta}$ is complex. This is achieved *without* any access to exact optimization solvers, nor models of their derivatives. This feat can be explained by the fact that by learning optimal solutions end-to-end directly from features, LtOF does not directly depend on learning an accurate representation of the underlying mapping from $\boldsymbol{z}$ to $\boldsymbol{\zeta}$.

4.2 Efficiency Benefits

Because the primary goal of the Learn-to-Optimize methodology is to achieve *fast solving times*, the LtOF approach broadly inherits this advantage. While these benefits in speed may be diminished when constraint violations are present and complex feasibility restoration are required, efficient feasibility restoration is possible for many classes of optimization models Beck (2017). This enables the design of *accelerated* PtO models within the LtOF framework, as shown in Section 5.

4.3 Modeling Benefits

While EPO approaches require the implementation of problem-specific backpropagation rules, the LtOF framework allows for the utilization of existing LtO methodologies in the PtO setting, on a problem-specific basis. A variety of existing LtO methods specialize in learning to solve convex

and nonconvex optimization (Fioretto et al., 2020a; Park & Van Hentenryck, 2023; Donti et al., 2021), combinatorial optimization (Bello et al., 2017; Kool et al., 2019), and other more specialized problem forms (Wu & Lisser, 2022). The experiments of this paper focus on the scope of continuous optimization problems, whose LtO approaches share a common set of solution strategies.

5 Experiments

This section evaluates three distinct LtO methods adapted to the LtOF setting, on three different Predict-Then-Optimize tasks, where each task involves 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 quadratic programming variant, and a nonconvex AC-Optimal Power Flow problem, to demonstrate the general utility of the framework. First, the section's three LtOF methods are briefly described.

5.1 Learning to Optimize Methods

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

5.1.1 Lagrangian Dual Learning (LD)

Fioretto et al. (2020a) uses the following modified Lagrangian loss function for training $\hat{\boldsymbol{x}} = \boldsymbol{F}_\omega(\boldsymbol{\zeta})$:

$$\mathcal{L}_{\mathbf{LD}}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) = \|\hat{\boldsymbol{x}} - \boldsymbol{x}^\star(\boldsymbol{\zeta})\|_2^2 + \boldsymbol{\lambda}^T \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})\right]_+ + \boldsymbol{\mu}^T \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}). \tag{8}$$

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 $\boldsymbol{x}^\star(\boldsymbol{\zeta})$.

5.1.2 Self-Supervised Primal-Dual Learning (PDL)

Park & Van Hentenryck (2023) use an augmented Lagrangian loss for self-supervised learning:

$$\mathcal{L}_{\mathbf{PDL}}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) = f(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \hat{\boldsymbol{\lambda}}^T \boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \hat{\boldsymbol{\mu}}^T \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \frac{\rho}{2} \left(\sum_j \nu(g_j(\hat{\boldsymbol{x}})) + \sum_j \nu(h_j(\hat{\boldsymbol{x}})) \right), \tag{9}$$

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 $\hat{\boldsymbol{x}} = \boldsymbol{F}_\omega(\boldsymbol{\zeta})$ to minimize equation 9. In addition to the primal network $\boldsymbol{F}_\omega$, a dual network $\mathcal{D}_{\boldsymbol{\omega}'}$ learns to store updates of the multipliers for each instance, and predict them as $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) = \mathcal{D}_{\boldsymbol{\omega}'}(\boldsymbol{\zeta})$ to the next iteration.

5.1.3 Deep Constraint Completion and Correction (DC3)

Donti et al. (2021) use the loss function

$$\mathcal{L}_{\mathbf{DC3}}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) = f(\hat{\boldsymbol{x}}, \boldsymbol{\zeta}) + \lambda \| \left[\boldsymbol{g}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})\right]_+ \|_2^2 + \mu \|\boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\zeta})\|_2^2 \tag{10}$$

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 $\boldsymbol{h}(\hat{\boldsymbol{x}}) = \boldsymbol{0}$ 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 $\boldsymbol{F}_\omega(\boldsymbol{\zeta}) = \boldsymbol{S} \circ \hat{\boldsymbol{F}}_\omega(\boldsymbol{\zeta})$.

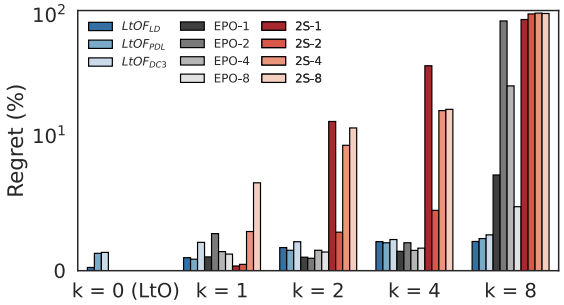

Figure 4: Comparison between **LtO** ($k=0$), **LtOF**, Two-stage (**2S**) and **EPO** ($k>1$) on the portfolio optimization. 2S(EPO)-m indicates that the prediction model of the respective PtO method is an m layer ReLU neural network.

	Method	Portfolio	N/conv. QP	AC-OPF
LtOF	LD it	**0.0003**	0.0000	**0.0004**
	LD fct	0.0000	0.0045	0.1573
	PDL it	**0.0003**	0.0000	0.0006
	PDL fct	0.0000	0.0045	0.1513
	DC3 it	0.0011	0.0001	-
	DC3 fct	0.0003	0.0000	-
PtO	PtO-1 et	0.0054	0.0122	0.1729
	PtO-2 et	0.0059	0.0104	0.1645
	PtO-4 et	0.0062	0.0123	0.1777
	PtO-8 et	0.0067	0.0133	0.1651

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.

While several other Learning-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 end-to-end LtO methodology as a whole to the Predict-Then-Optimize setting.

5.2 Experimental Settings

Feature generation. End-to-End Predict-Then-Optimize methods integrate learning and optimization to minimize the propagation of prediction errors–specifically, from feature mappings $\boldsymbol{z} \rightarrow \boldsymbol{\zeta}$ 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 (Elmachtoub & Grigas, 2021; Mandi et al., 2023). Accordingly, for each experiment, we generate feature datasets $(\boldsymbol{z}_1, \dots \boldsymbol{z}_N) \in \mathcal{Z}$ from ground-truth parameter sets $(\boldsymbol{\zeta}_1, \dots \boldsymbol{\zeta}_N) \in \mathcal{C}$ 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 k hidden layers where $k \in \{1, 2, 4, 8\}$. 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 $\mathbf{F}_\omega : \mathcal{C} \rightarrow \mathcal{X}$ learns a mapping from 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.

Comparison to EPO with Pretrained Proxy. The end-to-end LtOF implementations are also compared against EPO models with pre-trained optimization proxies as a baseline, as described in Section 3.

All reported results are averages across 20 random seeds and the reader is referred to Appendix B for additional details regarding experimental settings, architectures, and hyperparamaters adopted.

5.3 Convex Quadratic Portfolio Optimization

A well-known problem combining prediction and optimization is the Markowitz Portfolio Optimization (Rubinstein, 2002). This task has as its optimization component a convex Quadratic Program:

$$\boldsymbol{x}^\star(\boldsymbol{\zeta}) = \underset{\boldsymbol{x} \geq \mathbf{0}}{\arg\max}\ \boldsymbol{\zeta}^T \boldsymbol{x} - \lambda \boldsymbol{x}^T \boldsymbol{\Sigma} \boldsymbol{x}, \quad \text{s.t. } \mathbf{1}^T \boldsymbol{x} = 1 \tag{11}$$

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

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 (Nasdaq, 2022) 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, possible 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` (Diamond & Boyd, 2016) is used as the optimization component in each PtO method.

Results. Figure 4 shows the percentage regret due to LtOF implementations 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 (11) as provided by the differentiable solver in `cvxpylayers` (Agrawal et al., 2019a) is competitive with LtOF until $k = 4$, after which point its best variant is outperformed by each LtOF variant. This result showcases the ability of LtOF models to reach high accuracy under complex feature mappings *without* access to optimization problem solvers *or* their derivatives, in training or inference, in contrast to conventional PtO frameworks. Full accuracy results are reported in Table 2, which includes constraint violation and regret of the inferred solutions before feasibility restoration.

Table 1 presents LtOF inference times (*it*) and feasibility correction times (*fct*), which are 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 obmitted. Notice how the LtOF methods are at least an order of magnitude faster than PtO methods. This efficiency has two key implications: firstly, the per-sample speedup can significantly accelerate training for PtO problems. Secondly, it is especially advantageous during inference, particularly if data-driven decisions are needed in real-time.

5.4 Nonconvex QP Variant

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

$$\begin{aligned} \mathbf{x}^\star(\boldsymbol{\zeta}) = \underset{\boldsymbol{x}}{\arg\min}\ \ & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{\zeta}^T \sin(\boldsymbol{x}) \\ \texttt{s.t.}\ \ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}, \end{aligned}$$

in which the $\sin$ function is applied elementwise. This formulation was used to evaluate the LtO methods proposed both in Donti et al. (2021) and in Park & Van Hentenryck (2023). Following

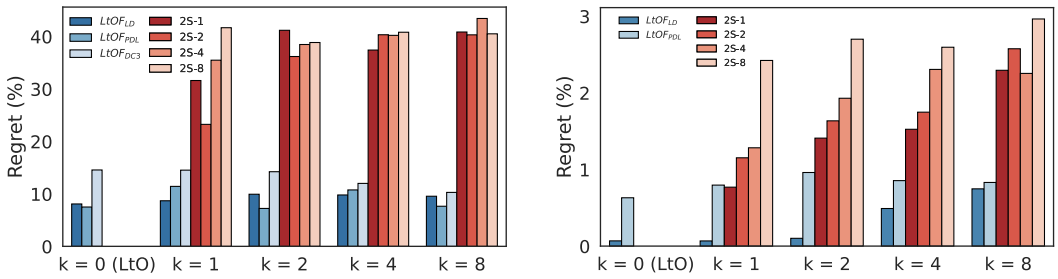

Figure 5: Comparison between LtO ($k = 0$), LtOF, and Two Stage Method (2S) on the nonconvex QP (**left**) and AC-OPF case (**right**). Right plot y-axis is in log-scale.

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

those works, $\mathbf{0} \preccurlyeq \boldsymbol{Q} \in \mathbb{R}^{n \times n}$, $\boldsymbol{A} \in \mathbb{R}^{n_{\text{eq}} \times n}$, $\boldsymbol{b} \in \mathbb{R}^{n_{\text{eq}}}$, $\boldsymbol{G} \in \mathbb{R}^{n_{\text{ineq}} \times n}$ and $\boldsymbol{h} \in \mathbb{R}^{n_{\text{ineq}}}$ have elements drawn uniformly at random. Here it is evaluated as part of a Predict-Then-Optimize pipeline in which predicted coefficients occupy the nonconvex term. Feasibility is restored by a projection onto the feasible set, which is calculated by a more efficiently solvable *convex* QP. The problem dimensions are $n = 50$ $n_{\text{eq}} = 25$, and $n_{\text{ineq}} = 25$.

Results. Figure 5 (left) shows regret due to LtOF models based on *LD*, *PDL* and *DC3*, along with two-stage baseline PtO methods. No EPO baselines are available due to the optimization component's nonconvexity. The best two-stage models perform poorly for most values of k, implying that the regret is particularly sensitive to prediction errors in the oscillating term. Thus its increasing trend with k is less pronounced than in other experiments. The best LtOF models achieve over 4 times lower regret than the best baselines, suggesting strong potential for this approach in contexts which require predicting parameters of non-linear objective functions. Additionally, the fastest LtOF method achieves up to three order magnitude speedup over the two-stage, after restoring feasibility.

5.5 Nonconvex AC-Optimal Power Flow

Given a vector of marginal costs $\boldsymbol{\zeta}$ for each power 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 and more details are specified in Appendix A, where a quadratic cost objective is minimized subject to nonconvex physical and engineering power systems constraints. This experiment simulates a 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 overall 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 well-known NESTA energy system test case archive (Coffrin et al., 2014). Feasibility is restored for each LtOF model by a projection onto the nonconvex

	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)	-	0.7620	1.4090	1.5280	2.4740
	EPO Proxy Regret (Best)	-	431.7664	389.0421	413.8941	404.7452

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

feasible set. Optimal solutions to the AC-OPF problem, along with such projections, are obtained using state-of-the-art Interior Point OPTimizer IPOPT (Wächter & Laird, 2023).

6 Related Work

Predict-Then-Optimize While the idea is general and has broader applications, differentiation through the optimization of equation 1 is central to EPO approaches. Differentiation of quadratic programming problems was introduced by Amos & Kolter (2017), which implicitly differentiates the solution via its KKT equations of optimality, proposes its use for defining general-purpose learnable layers in neural networks. Agrawal et al. (2019b) proposes a more general differentiable cone programming solver, which 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 differentiate optimization problems by leveraging automatic differentiation through a single step of a convergent solution method to implicitly differentiate its fixed-point conditions. For discrete problems 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. (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), and Ferber et al. (2020) extends the technique to mixed-integer programs via the equivalent linear program found by cutting planes. (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 finite difference approximations.

Learning to Optimize 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. 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 not relevant 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 an offline solver. 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. The policy gradient method has been adapted to combinatorial problems such as vehicle routing Kool et al. (2018) and job scheduling Mao et al. (2019), and generally 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; 2021a). General frameworks for end-to-end learning of non-combinatorial problems have been proposed in

the works Fioretto et al. (2020b); Park & Van Hentenryck (2023); Donti et al. (2021), which are each reviewed and incorporated in the experiments of Section 5.

7 Limitations, Discussion, and Conclusions

The primary *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 and methods 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. For example, when the LtO method does not ensure feasibility, the ability to restore feasibility may be need as part of a PtO pipeline. 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 (Bello et al., 2017; Kool et al., 2019; Mao et al., 2019), and are not suited for capturing broad classes of optimization problems. As such, this direction is left to future work.

The main *disadvantage* inherent to any LtOF implementation, compared to end-to-end PtO, is the inability to recover parameter estimations from the predictive model, since optimal solutions are predicted end-to-end from features. Although it is not required in the canonical PtO problem setting, this may present a complication in situations where transferring the parameter estimations to external solvers is desirable. This presents an interesting direction for future work.

By showing that effective Predict-Then-Optimize models can be composed purely 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 Learning to Optimize in settings that require data-driven decision-making.

8 Acknowledgement

This research was partially supported by NSF grants 2232054, 2242931, and 2143706. Its views and conclusions are to be considered as of the authors only.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019a.

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *arXiv preprint arXiv:1904.09043*, 2019b.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Amir Beck. *First-order methods in optimization*. SIAM, 2017.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv:1611.09940*, 2017.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan 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.

Carleton Coffrin, Dan Gordon, and Paul Scott. Nesta, the nicta energy system test case archive. *arXiv preprint arXiv:1411.0359*, 2014.

Arnaud Deza and Elias B Khalil. Machine learning for cutting planes in integer programming: A survey. *arXiv preprint arXiv:2302.09166*, 2023.

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

Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 2021.

Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 1504–1511, 2020.

Ferdinando Fioretto, Pascal Van Hentenryck, Terrence WK Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. Lagrangian duality for constrained deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 118–135. Springer, 2020a.

Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 630–637, 2020b.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Learning hard optimization problems: A data generation perspective. *Advances in Neural Information Processing Systems*, 34:24981–24992, 2021a.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4475–4482, 2021b. doi: 10.24963/ijcai.2021/610. URL https://doi.org/10.24963/ijcai.2021/610.

James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Fast approximations for job shop scheduling: A lagrangian dual deep learning method. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7239–7246, 2022.

James Kotary, My H. Dinh, and Ferdinando Fioretto. Backpropagation of unrolled solvers with folded optimization. *arXiv preprint arXiv:2301.12047*, 2023.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *arXiv preprint arXiv:2307.13565*, 2023.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pp. 270–288, 2019.

Sidhant Misra, Line Roald, and Yeesian Ng. Learning for constrained optimization: Identifying optimal active constraint sets. *INFORMS Journal on Computing*, 34(1):463–480, 2022.

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

Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 4052–4060, 2023.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations (ICLR)*, 2020.

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

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In *Learning for Dynamics and Control Conference*, pp. 220–234. PMLR, 2023.

Bo Tang and Elias Boutros Khalil. Pyepo: A pytorch-based end-to-end predict-then-optimize library with linear objective function. In *OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop)*, 2022.

Natalia Vesselinova, Rebecca Steinert, Daniel F Perez-Ramirez, and Magnus Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8: 120388–120416, 2020.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NeurIPS*, pp. 2692–2700, 2015.

Andreas Wächter and Carl D. Laird. *IPOPT: Interior Point Optimizer, Version 3.1.4*. IBM and Carnegie Mellon University, 2023. Available online at: https://github.com/coin-or/Ipopt.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pp. 1658–1665, 2019.

Dawen Wu and Abdel Lisser. A deep learning approach for solving linear programming problems. *Neurocomputing*, 2022.

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

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

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

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

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

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.

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. However, when started with a good approximation it is typically much easier to solve than the AC-OPF Fioretto et al. (2020b).

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

Figure 7: AC Load Flow.

B Experimental Details

B.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. 11, $\lambda = 2.0$.

The covariance matrix $\boldsymbol{\Sigma}$ is constructed from historical price data and set as $\boldsymbol{\Sigma} = F\boldsymbol{\Sigma}_F F^T + D$, where $F \in \mathbb{R}^{n,l}$ is the factor-loading matrix, $\boldsymbol{\Sigma} \in \mathbb{S}_+^l$ 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 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 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. (2020a), 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.

B.2.1 Convex Quadratic Optimization and Non Convex Quadratic Optimization

LD

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

PDL

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

DC3

Parameter	Values
$\lambda + \mu$	0.1, 1.0, $\mathbf{10.0}$, 50.0, 100.0
$\frac{\lambda}{\lambda+\mu}$	0.1, 0.5, $\mathbf{0.75}$, 1
t_{test}	1, 2, $\mathbf{5}$, 10, 100
t_{train}	1, 2, $\mathbf{5}$, 50, 100

Non Convex AC-Optimal Power Flow (LD)

Parameter	Values
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