Learning to Solve Constrained Bilevel Control Co-Design Problems

James Kotary
Pacific Northwest National Laboratory
james.kotary@pnnl.gov

Himanshu Sharma
Pacific Northwest National Laboratory
himanshu.sharma@pnnl.gov

Ethan King
Pacific Northwest National Laboratory
ethan.king@pnnl.gov

Draguna Vrabie
Pacific Northwest National Laboratory
draguna.vrabie@pnnl.gov

Ferdinando Fioretto
University of Virginia
fioretto@virginia.edu

Jan Drgona
Johns Hopkins University
jdrgona1@jh.edu

Abstract

We propose a learning to optimize (L2O) method for solving *constrained parametric bilevel problems* that arise in control co-design, where upper-level design variables are coupled with lower-level optimal control through explicit *coupling constraints*. Our self-supervised framework comprises: (i) a *differentiable optimization layer* to enforce lower-level optimality, and (ii) a *differentiable gradient-based projection* routine that iteratively reduces coupling-constraint violation while maintaining feasibility of upper-level constraints. A soft penalty is used during training to initialize predictions near feasibility, enabling stable end-to-end learning. On bilevel QPs with certified optima, our learned models achieve 10^{-2} relative optimality gaps while running $\sim 10^2\times$ faster than a mixed-integer programming (MIP) reformulation. On two optimal control co-design tasks, our approach yields 15–19% lower design cost and $\sim 10^4\times$ faster inference than a particle swarm optimization (PSO) baseline, while maintaining comparable constraint satisfaction. These results indicate that the proposed L2O method can deliver real-time, high-quality approximations for challenging bilevel programming problems that are computationally prohibitive using conventional methods.

1 Introduction

Bilevel optimization problems arise in a wide range of applications, from economics and game theory (Binmore, 2007), to operations management and logistics (Sadigh, Mozafari, and Karimi, 2012), as well as engineering system design (Bergonti et al., 2024). Unfortunately, bilevel problems are, in general, notoriously difficult to solve. They are typically NP-hard, and depending on their particular qualities, may even lack efficient frameworks for their approximate solution (Beck and Schmidt, 2021). This complexity challenges their use in applications requiring *real-time or repeated solutions*. Yet, many applications of bilevel optimization demand repeated solution of related problem instances, such as when engineering design decisions are considered across a variety of scenarios and objectives.

This paper introduces a novel framework for applying deep learning to solve a broad class of bilevel problems, which include *coupling constraints*, i.e., constraints that bind upper and lower level decision variables. The approach uses differentiable optimization solvers at the problem's

Preprint.

lower level, allowing for gradient-based training of neural networks to approximate the upper-level solution, as well as internal *correction* routines to enforce coupling constraints between the upper and lower-level problems. The resulting models are trained to map the parameters of a bilevel problem to its optimal solution. We motivate this approach primarily using problems from optimal control co-design: a setting which calls for the optimization of engineering systems, with respect to system design objectives at the upper level and subject to system dynamics determined by optimal control problems at the lower level. We demonstrate how the proposed framework can be used to learn solutions in response to varying design objectives and desiderata.

Contributions. The main contributions of this paper are: **(1)** We propose a novel Learning to Optimize method for bilevel optimization problems, whose core concept is based on using differentiable optimization to ensure optimality of the lower-level solution, while ensuring constraint satisfaction at the upper level via differentiable projections. **(2)** We show how differentiable optimization can also be used to compose gradient-based correction mechanisms for the satisfaction of *coupling constraints* - a significant challenge in bilevel problems. **(3)** We demonstrate the proposed framework on a collection of challenging constrained bilevel optimization and control-co-design problems, including those with *nonconvex* lower level problems and complex constraints. In particular, we demonstrate its ability to learn high-quality approximate parametric solutions for a distribution of problems. Full code is found at: https://github.com/anon-research-coding/Control-Design-Learning

2 Related Work

Bilevel optimization methods. Bilevel optimization problems are notoriously difficult to solve, and in general have no efficient solution methods except in special cases. Particularly when problems are sufficiently small and a convex structure is present, generic solution methods tend to focus on single-level reformulations (via the KKT conditions or the optimal value function), which can be solved by mixed-integer programming (Beck and Schmidt, 2021). When a special structure is present, such as linearity at both levels or a lack of coupling constraints, gradient methods have been proposed that tend to rely on penalty methods (Cerulli et al., 2021; Ghadimi and Wang, 2018; Solodov, 2007). Recently, (Sharifi et al., 2025; Abolfazli et al., 2025) solved a special class of bilevel problems using upper-level gradient descent with a quadratic programming (QP)-based safety filter for enforcing the KKT conditions of the lower-level problem. The complex structure possessed by more general bilevel problems has led to greater interest in metaheuristics, such as evolutionary algorithms (Sinha, Malo, and Deb, 2013; Bergonti et al., 2023) and particle swarm optimization (Li, Tian, and Min, 2006) over classical alternatives. A comprehensive review of modern approaches is found in Sinha, Malo, and Deb (2017).

Learning to Optimize. L2O is a subfield of ML concerned with learning *fast approximations* to challenging optimization problems (Van Hentenryck, 2025). One distinct branch focuses on learning information to accelerate the convergence of a classical solver (Bengio, Lodi, and Prouvost, 2020). In continuous optimization, these include prediction of active constraints (Ng et al., 2018), optimization problem parameters (Agrawal, Barratt, and Boyd, 2021), stepsizes (Amos et al., 2023), primal variables (Sambharya et al., 2024; Bertsimas and Stellato, 2022), and noneuclidean metrics (King et al., 2024), while for integer variables they include branching rules (Khalil et al., 2016), cutting planes (Tang, Agrawal, and Faenza, 2020), variable partitions in large neighborhood search (Song et al., 2020), or primal variables (Tang, Khalil, and Drgoňa, 2025; Huang et al., 2025). A separate branch aims at training ML models to predict optimal solutions directly from a representation of the problem (Kotary et al., 2021).

A key challenge for such an approach is to maintain the feasibility of the predicted solutions to arbitrary constraints. Proposals for addressing this aspect are based on differentiable projections (Wilder, Dilkina, and Tambe, 2019), reparametrization tricks (Frerix, Nießner, and Cremers, 2020; Konstantinov and Utkin, 2024), dual-variable estimation (Fioretto et al., 2020; Park and Van Hentenryck, 2023), and gradient-based constraint correction routines (Donti, Rolnick, and Kolter, 2021). Finally, some recent works have proposed to accelerate bilevel optimization with ML. (Dumouchelle et al., 2024) proposes in the non-parametric case to learn an optimal value reformulation from solved examples with a ReLU network, which is embedded into an MIP solver. (Shen et al., 2020; Andrychowicz et al., 2016) propose to learn solutions of a parametric bilevel program directly, albeit without any constraints at either level. *This paper extends the toolkit for direct learning of solutions to the case of bilevel optimization with continuous variables and a full set of constraints, including coupling constraints*.

3 Problem Setting

The goal of this paper is to learn to solve parametric bilevel optimization problems represented by (1). We take the convention that optimization variables $\boldsymbol{y}, \boldsymbol{z}$ are written as function arguments while problem parameters $\boldsymbol{p}$ are written as subscripts. We consider a pair of problems:

$$\begin{aligned} \mathcal{B}(\boldsymbol{p}) \coloneqq \underset{\boldsymbol{y}}{\operatorname{argmin}} \quad & \mathcal{L}_{\boldsymbol{p}}(\boldsymbol{y}, \boldsymbol{z}) && \text{(1a)} \\ \text{s.t.} \quad & \boldsymbol{z} \in O_{\boldsymbol{p}}(\boldsymbol{y}) && \text{(1b)} \\ & \boldsymbol{y} \in C_{\boldsymbol{p}} && \text{(1c)} \\ & \boldsymbol{U}_{\boldsymbol{p}}(\boldsymbol{y}, \boldsymbol{z}) \leq \boldsymbol{0} && \text{(1d)} \end{aligned}$$

$$\begin{aligned} O_{\boldsymbol{p}}(\boldsymbol{y}) \coloneqq \underset{\boldsymbol{z}}{\operatorname{argmin}} \quad & l_{\boldsymbol{p}}(\boldsymbol{y}, \boldsymbol{z}) && \text{(2a)} \\ \text{s.t.} \quad & \boldsymbol{z} \in \mathcal{S}_{\boldsymbol{p}}(\boldsymbol{y}). && \text{(2b)} \end{aligned}$$

The defining feature of problem (1) is its constraint (1b), which holds that the variables $\boldsymbol{z}$ be the solution to another optimization problem (2), which in turn depends on the variables $\boldsymbol{y}$. We call problem (1) the *upper-level problem*, and (2) is the *lower-level problem*, while $\boldsymbol{y}$ and $\boldsymbol{z}$ are the *upper* and *lower-level* variables, respectively. Our goal is to learn a fast approximator that solves the coupled problems (1,2), over a distribution of problem parameters denoted as $\mathcal{P}$.

Classes of constraints. We distinguish three sets of constraints at the upper level. The condition (1c) constrains only the upper-level variables, while (1b) prescribes $\boldsymbol{z}$ as a solution to the lower-level problem (2) resulting from $\boldsymbol{y}$. Additionally, the *coupling constraints* (1d) significantly complicate the solution of (1). They impose additional conditions on the relationship between upper and lower-level variables in (1), preventing solution concepts based on their separation or decoupling (Beck and Schmidt, 2021). A large portion of algorithms for bilevel programming cannot accommodate problems with coupling constraints (Sinha, Malo, and Deb, 2017), and to the best of our knowledge, *no previous work has ventured to propose an L2O framework for learning to solve them parametrically*.

Conditions on the problem form. The proposed framework aims to learn solutions to a broad class of problems having the form (1,2). However, it depends on a key condition: for all $\boldsymbol{y} \in C_{\boldsymbol{p}}$ and $\boldsymbol{p} \sim \mathcal{P}$, the solution to the lower-level problem (2) must be *unique* whenever it *exists* - existence is not required but assumed to hold without loss of generality until it is relaxed using a reformulation trick, introduced in Section 4.3. This leads $O_{\boldsymbol{p}}(\boldsymbol{y})$ to define a function, which we further suppose to be *differentiable*. These conditions are not met in all bilevel problems. However, they are satisfied in many important cases, such as when (2) is a *Model Predictive Control* (MPC) problem. Several recent works have proposed efficient techniques for differentiating through MPC (Amos et al., 2018; Dinev et al., 2022), which generally have unique solutions given sufficiently specified objectives. The functions $\mathcal{L}_{\boldsymbol{p}}$ and $\boldsymbol{U}_{\boldsymbol{p}}$ should also be differentiable. Importantly, though, a.e. or pseudo-differentiability is often sufficient for effective gradient descent optimization, as in the case of ReLU activations. Following common practice in L2O and machine learning, these conditions are treated as practical guidelines for applying the proposed architecture, rather than formal requirements.

4 Learning to Solve Bilevel Optimization

This section presents a self-supervised method for learning to solve the parametric bilevel optimization problems described in Section 3. In particular, it trains an ML model to approximate the mapping (1), from problem parameters $\boldsymbol{p}$ to *upper-level* solutions $\boldsymbol{y}^\star = \mathcal{B}(\boldsymbol{p})$. Let $\hat{\mathcal{B}}_\theta$ denote that hypothetical model, with weights θ. Assume a training set of n_p problem instances, each specified by a vector of problem parameters $\{\boldsymbol{p}_{(i)}\}_{i=1}^{n_p}$ which are drawn from the distribution $\mathcal{P}$. A training procedure for $\hat{\mathcal{B}}_\theta$ should minimize the objective function $\mathcal{L}_{\boldsymbol{p}}(\hat{\boldsymbol{y}})$ attained by its predicted solutions $\hat{\boldsymbol{y}} = \hat{\mathcal{B}}_\theta(\boldsymbol{p})$ in expectation over $\mathcal{P}$, resulting in the empirical risk minimization (ERM) (3). Constraints (3c, 3b, 3d) require that each such output and its resulting pair $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ is a feasible solution to the bilevel problem (1,2), where $\hat{\boldsymbol{y}} \coloneqq \hat{\mathcal{B}}_\theta(\boldsymbol{p})$. Learning solutions subject to such complex constraints is inherently challenging, as any predicted $\hat{\boldsymbol{y}}$ and its corresponding $\hat{\boldsymbol{z}}$ are unlikely to satisfy the coupling constraint (3d) after solving (3b), even if $\hat{\boldsymbol{y}} \in C_{\boldsymbol{p}}$ as required in (3c). In this section, we propose an architecture for the model $\hat{\mathcal{B}}_\theta$, along with a method for training it to approximate the ERM goal (3).

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \mathbb{E}_{\boldsymbol{p} \sim \mathcal{P}}\Big[\mathcal{L}_{\boldsymbol{p}}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})\Big] && \text{(3a)} \\ \text{s.t.} \quad & \hat{\boldsymbol{z}} \in O_{\boldsymbol{p}}(\hat{\boldsymbol{y}}) \quad \forall \boldsymbol{p} \sim \mathcal{P}, && \text{(3b)} \\ & \hat{\boldsymbol{y}} \in C_{\boldsymbol{p}} \quad \forall \boldsymbol{p} \sim \mathcal{P}, && \text{(3c)} \\ & \boldsymbol{U}_{\boldsymbol{p}}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}) \leq \boldsymbol{0} \quad \forall \boldsymbol{p} \sim \mathcal{P}. && \text{(3d)} \end{aligned}$$

4.1 Satisfying Constraints with Differentiable Optimization Modules

The proposed architecture employs differentiable optimization to implement two of its main components, prefaced below. In general, the derivatives between a problem's parameters and its optimal solution can be found by implicit differentiation, either of the KKT conditions (Amos and Kolter, 2017; Gould, Hartley, and Campbell, 2021; Agrawal et al., 2019a,b) or a fixed-point condition (Sambharya et al., 2024). This proposal is agnostic as to which implementation is used.

Differentiable solution of the lower-level problem. For a given $\boldsymbol{p}$ and any predicted upper-level solution $\hat{\boldsymbol{y}}$, a differentiable solver of problem (2) produces $\hat{\boldsymbol{z}} = \boldsymbol{O_p}(\hat{\boldsymbol{y}})$ along with $\frac{\partial \hat{\boldsymbol{z}}}{\partial \hat{\boldsymbol{y}}}$. This provides a means by which feasible solution pairs (3b) can be computed and back-propagated as part of an end-to-end trainable model. In the applications of Section 5, it corresponds to generating an optimal control policy $\hat{\boldsymbol{z}}$ as a differentiable function of learned design parameters $\hat{\boldsymbol{z}}$.

Differentiable projection at the upper level. In addition to satisfying constraint (3b) with its solution pair, any candidate $\hat{\boldsymbol{y}}$ must also satisfy its own upper-level constraint (3c). This constraint is handled using a projection operator $\boldsymbol{\Pi}_C$. The defining problem (4) can be solved in a differentiable library such as *cvxpylayers* (Agrawal et al., 2019a), but automatic differentiation in PyTorch suffices when a closed form is available (e.g., the projection onto $\{\boldsymbol{x} : \boldsymbol{x} \geq 0\}$ is well-known to be a ReLU function). Differentiable projections are a mainstay tool for constraint satisfaction in L2O (Sambharya et al., 2023; King et al., 2024; Wilder, Dilkina, and Tambe, 2019). Besides projections, other mechanisms could be used to guarantee constraint satisfaction (Chen, Tanneau, and Van Hentenryck, 2023; Tordesillas, How, and Hutter, 2023). However, the projection operator is chosen for its role in the algorithm presented next, resulting in the *projected gradient descent* method, which has well-studied convergence properties (Beck, 2017).

$$\boldsymbol{\Pi}_C(\boldsymbol{y}) = \underset{\boldsymbol{w} \in C}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{w}\|_2^2 \quad (4)$$

4.2 End-to-End Trainable Architecture

We can now present the complete architecture and training routine of a model $\hat{\mathcal{B}}_\theta$ which learns to solve problem (3). Let $\mathcal{N}_\theta$ be a deep neural network with weights θ, which predicts initial estimates $\hat{\boldsymbol{y}} = \mathcal{N}_\theta(\boldsymbol{p})$ of an upper-level solution. Composition of $\mathcal{N}_\theta$ with $\boldsymbol{\Pi}_{C_p}$ ensures feasibility to (3c), and further composition with $\boldsymbol{O_p}$ produces a solution pair $(\hat{\boldsymbol{y}}, \boldsymbol{O_p}(\hat{\boldsymbol{y}}))$ which satisfies (3b). Therefore the function $\boldsymbol{O_p} \circ \boldsymbol{\Pi}_{C_p}$ can be viewed as one which maps infeasible upper-level estimates to solution pairs $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ satisfying (3b) and (3c), but not the coupling constraint $\boldsymbol{U_p}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}) \leq \boldsymbol{0}$ in (3d). We define the *Coupling Constraint Violation* as in (5), along with the gradient of its Euclidean norm in (6). Note that $\boldsymbol{y}$ is recognized as the independent variable and $\boldsymbol{z} = \boldsymbol{O_p}(\boldsymbol{y})$ as dependent.

$$\nu(\boldsymbol{y}) \coloneqq \textsc{ReLU}\left(\boldsymbol{U}(\,\boldsymbol{y}, \boldsymbol{O_p}(\boldsymbol{y})\,)\right), \quad (5) \qquad \nabla_y \|\nu(\boldsymbol{y})\|^2 = 2\nu(\boldsymbol{y})\frac{d\nu}{d\boldsymbol{y}} = 2\nu(\boldsymbol{y})\frac{d\nu}{d\boldsymbol{U}}\left[\frac{\partial \boldsymbol{U}}{\partial c} + \frac{\partial \boldsymbol{U}}{\partial \boldsymbol{O_p}}\frac{\partial \boldsymbol{O_p}}{\partial \boldsymbol{y}}\right]. \quad (6)$$

The nontrivial component of (6) is the Jacobian $\frac{\partial \boldsymbol{O_p}}{\partial \boldsymbol{y}}$. This information represents backpropagation through the lower-level problem, which can be obtained from one of the differentiable solvers discussed in the previous section, while automatic differentiation in PyTorch (Paszke et al., 2017) is sufficient to complete the remaining chain rule calculations in (6). We also define a function which reduces $\|\nu(\boldsymbol{y})\|^2$ by performing a gradient descent step of size γ:

$$\mathcal{G}(\boldsymbol{y}) \coloneqq \boldsymbol{y} - \gamma \nabla_y \|\nu(\boldsymbol{y})\|_2^2. \quad (7)$$

Importantly, this function can also be rendered differentiable by leveraging functionality for *back-propagating gradient calculations* (in this case, equation (6)) in automatic differentiation libraries such as PyTorch. The result of function (7) is generally infeasible to (3c), and this can be addressed by a (differentiable) projection back onto C_p, completing one step of an end-to-end differentiable *Coupling Constraint Correction* routine:

$$\boldsymbol{y}_{k+1} = \boldsymbol{\Pi}_{C_p}\left(\boldsymbol{y}_k - \gamma \nabla \|\nu(\boldsymbol{y}_k)\|_2^2\right). \quad (8)$$

Letting $\boldsymbol{y}_0 = \mathcal{N}_\theta(\boldsymbol{p})$, our architecture for $\hat{\mathcal{B}}_\theta$ corrects the neural network with m steps of (8). Explicitly,

$$\hat{\mathcal{B}}_\theta(\boldsymbol{p}) = \left[\left(\boldsymbol{\Pi}_{C_p} \circ \mathcal{G}\right)^m \circ \mathcal{N}_\theta\right](\boldsymbol{p}). \quad (9)$$

By construction, this model is end-to-end differentiable, and maintains feasibility to (3b) and (3c) while iterating toward satisfaction of (3c). Furthermore, the process (8) can be recognized as the classical projected gradient descent method on $\|\nu(\boldsymbol{y})\|_2^2$. This method is well-known to converge to

local minima of convex and nonconvex functions, provided certain conditions on those functions' properties, their feasible set as well as the stepsize (Beck, 2017). The correction routine (8) is illustrated in Figure 1, in which blue arrows represent gradient steps (7), and the alternating green arrows represent projections (4) back onto C_p. The entire chain of operations is composed with the neural network $\mathcal{N}_\theta$, and unrolled in backpropagation to update its predictions.

Penalty loss function. The proposed architecture builds on the concept of a gradient-based constraint correction mechanism, popularized by the DC3 algorithm (Donti, Rolnick, and Kolter, 2021). Gradient descent methods are not guaranteed to converge in general, and it follows that DC3 corrections are not guaranteed to yield zero violations. However, as noted in (Donti, Rolnick, and Kolter, 2021), when *initialized* close to an optimum, they are highly effective in practice (Busseti, Moursi, and Boyd, 2019; Lee et al., 2019). Following Donti, Rolnick, and Kolter (2021), we use a penalty loss:

$$\mathcal{L}^{\text{SOFT}}_{\boldsymbol{p}_{(i)}}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}) := \mathcal{L}_{\boldsymbol{p}_{(i)}}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}) + \lambda \|\nu(\hat{\boldsymbol{y}})\|_2^2. \qquad (10)$$

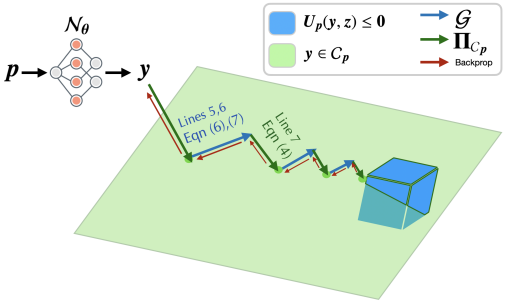

Figure 1: Illustrating the end-to-end trainable model with algorithm line and equation numbers.

This leads to upper-level predictions from $\mathcal{N}_\theta$ which are initialized close to satisfying the coupling constraint (1d), so that its end-to-end training with the differentiable correction (8) tends to produce feasible solutions to the full problem (1).

Training Routine. An overall training scheme is summarized in Algorithm 1, in terms of one epoch of stochastic gradient descent. Each data input sample $\boldsymbol{p}_{(i)}$ represents a distinct instance of problem (1), for which an initial solution estimate $\hat{\boldsymbol{y}}$ is predicted at line 3. The sequence (8) makes up lines 4-8. For each of m *correction steps*, $\hat{\boldsymbol{y}}$ is iteratively refined by taking a step toward feasibility of the coupling constraint (1d). The gradient $\hat{\boldsymbol{g}}$ at line 6 is calculated per equation (6). The loss (10) is then evaluated w.r.t. the refined estimate $\hat{\boldsymbol{y}}$ and its lower-level pair. If needed, at test time, more than m iterations may be applied in the correction routine. Note that Algorithm 1 describes only the *forward pass* of the training routine. Line 11 encapsulates backpropagation through all components of the model, implemented with a combination of automatic differentiation and implicit differentiation via the differentiable solvers, as described above.

4.3 Satisfying Lower-Level Constraints

So far, this section has assumed that the lower-level problem (2) had at least one feasible solution for any $\boldsymbol{p}$ and $\boldsymbol{y} \in C_p$ (see the comments on existence and uniqueness in Section 3). Otherwise, the optimization problem at line 5 of Algorithm 1 may be infeasible. In such cases, we address this issue by reformulating the overall bilevel problem so that the property is satisfied. To do so, we simply identify the lower-level constraints that prevent feasibility and *lift* them to the upper level where they become coupling constraints. Let the lower level's feasible set be partitioned as $\mathcal{S}_p(\boldsymbol{y}) = \mathcal{F}_p(\boldsymbol{y}) \cup \mathcal{F}_p^C(\boldsymbol{y})$, such that $\mathcal{F}_p(\boldsymbol{y})$ is nonempty for all $\boldsymbol{p}$ and $\boldsymbol{y} \in C_p$. We reformulate problems (1) and (2) so that $\mathcal{S}_p(\boldsymbol{y})$ is replaced by $\mathcal{F}_p(\boldsymbol{y})$ in problem (2), while $\boldsymbol{z} \in \mathcal{F}_p^C(\boldsymbol{y})$ is promoted to the upper level problem (1). As an upper-level constraint relating $\boldsymbol{y}$ and $\boldsymbol{z}$, it is absorbed into the *coupling constraints* (1d). This technique is applied in the experiments of Section 5.2, and detailed in Appendices D and E.

Algorithm 1: Learning Bilevel Optimization with Coupling

1: **Input:** parameters $\{\boldsymbol{p}_{(i)}\}_{i=1}^N$,
   weights $\boldsymbol{\theta}$,
   learning rate α, correction stepsize γ
2: **for** $i = 1$ **to** N **do**
3: $\quad \hat{\boldsymbol{y}} \leftarrow \mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{p}_{(i)})$
4: $\quad$ **for** $k = 1$ **to** m **do**
5: $\quad\quad \hat{\boldsymbol{g}} \leftarrow \nabla \left\| \text{ReLU}\left(\boldsymbol{U}_{\boldsymbol{p}_{(i)}}(\hat{\boldsymbol{y}}, \boldsymbol{O}_{\boldsymbol{p}_{(i)}}(\hat{\boldsymbol{y}}))\right)\right\|^2$
6: $\quad\quad \hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} - \gamma \cdot \hat{\boldsymbol{g}}$
7: $\quad\quad \hat{\boldsymbol{y}} \leftarrow \boldsymbol{\Pi}_{C_{\boldsymbol{p}_{(i)}}}(\hat{\boldsymbol{y}})$
8: $\quad$ **end for**
9: $\quad \hat{\boldsymbol{z}} \leftarrow \boldsymbol{O}_{\boldsymbol{p}_{(i)}}(\hat{\boldsymbol{y}})$
10: $\quad \boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{SOFT}}_{\boldsymbol{p}_{(i)}}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$
11: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{g}$
12: **end for**

BQP	$\left\lvert\frac{\mathcal{L}(\hat{y},\hat{z})-\mathcal{L}(y^\star,z^\star)}{\mathcal{L}(y^\star,z^\star)}\right\rvert$	$\|v(y)\|_2$	Time (s)	YALMIP (s)
3×2	$9.2e-4\pm2e-3$	$5.9e-3\pm1e-2$	$\mathbf{6.4e-2}$	0.12
6×4	$2.0e-3\pm5e-3$	$2.8e-4\pm9e-4$	$\mathbf{6.4e-2}$	1.2
9×6	$1.1e-2\pm1e-2$	$4.0e-5\pm7e-4$	$\mathbf{6.7e-2}$	10.3

Table 1: BQP problems, Test Set Average

Model	$\mathcal{L}(y)$		$\|v(y)\|_2$		Time (s)	
	TT	HVAC	TT	HVAC	TT	HVAC
Learned (Alg. 1)	**0.122**	**1.23**	$\mathbf{1.4e-2}\pm2e-2$	$3.0e{-}2\pm5e-2$	$\mathbf{2.7e-2}$	$\mathbf{7.70e-2}$
PSO (baseline)	0.140	1.46	$1.6e-2\pm2e-2$	$3.0e{-}2\pm2e-2$	1268.7	1055.9

Table 2: On Control Co-Design, Test Set Average

5 Experiments

In this section, we evaluate the proposed methods' effectiveness in learning to solve several parametric bilevel optimization problems. In order to measure the optimality of its learned solutions, we require true optimal solutions for comparison. In light of Section 2, finding those solutions can pose significant challenges in itself. Most known methods either rely on exploiting problem-specific structure or otherwise employ metaheuristic methods which lack guarantees. Even if optimal solutions are found by such methods, their optimality often cannot be certified (Sinha, Malo, and Deb, 2017). For these reasons, Section 5.1 begins by learning solutions to small-scale synthetic problems for which open-source solvers can provide *certified optimal solutions for comparison*. Then, Section 5.2 extends the evaluation to more complex bilevel programs in engineering design. Those problems are significantly more difficult due to their much larger size and more complex forms, which include *nonconvex optimization* at the lower level. In those cases, we evaluate our learned solutions against those of a Particle Swarm Optimization (PSO), a metaheuristic framework commonly used in design optimization (Asaah, Hao, and Ji, 2021; Hou et al., 2015).

Evaluation criteria and conventions. We evaluate the ability of Algorithm 1 to perform the training task specified in Equation 3. Recall that constraints (3c) and (3b) are ensured by the construction of Algorithm 1; thus, our two main evaluation criteria are the *objective values* (3a), and potential *coupling constraint violation* (5). When true optimal solutions $(\boldsymbol{y}^\star, \boldsymbol{z}^\star)$ are available, we report the *relative optimality gap* relative to learned solutions $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$, which we define $\left\lvert\mathcal{L}(\hat{y},\hat{z})-\mathcal{L}(y^\star,z^\star)/\mathcal{L}(y^\star,z^\star)\right\rvert$ and illustrate *in blue* throughout. When the true optima are not known, we instead report the nominal objective value of the learned solutions, *in green*, alongside the solution produced by a baseline method for comparison. All metrics are reported on average over the respective test set. When a metric should ideally converge to zero, its standard deviation is also reported. Additional implementation details for each experiment, can be found in Appendices C, D, E .

5.1 Preliminary Problem: Learning Bilevel Quadratic Programming

We begin the experimental evaluation on a relatively simpler class of bilevel problems, the Bilevel Quadratic Programs (BQP). Both their upper and lower-level problems contain convex quadratic objective functions and only linear constraints. The upper and lower-level problems are respectively:

$$\mathcal{B}(\boldsymbol{c},\boldsymbol{d}) = \underset{\boldsymbol{y}}{\operatorname{argmin}} \quad \frac{1}{2}\boldsymbol{y}^T\boldsymbol{Q}\boldsymbol{y} + \boldsymbol{c}^T\boldsymbol{y} + \boldsymbol{d}^T\boldsymbol{z} + q \quad (11a)$$

$$s.t. \quad \boldsymbol{A}\boldsymbol{y} \le \boldsymbol{b} + \boldsymbol{E}\boldsymbol{z} \quad (11b)$$

$$\boldsymbol{z} \in O(\boldsymbol{y}), \quad (11c)$$

$$O(\boldsymbol{y}) = \underset{\boldsymbol{z}}{\operatorname{argmin}} \quad \frac{1}{2}\boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} + \boldsymbol{e}^T\boldsymbol{z} + \boldsymbol{f}^T\boldsymbol{y} + g \quad (12a)$$

$$s.t. \quad \boldsymbol{F}\boldsymbol{z} \le \boldsymbol{h} + \boldsymbol{G}\boldsymbol{y}. \quad (12b)$$

We train a neural network to learn its solutions as a function of the upper-level linear objective coefficients. That is, in the notation of Section 3, we have $\boldsymbol{p} \coloneqq (\boldsymbol{c}, \boldsymbol{d})$.

Experimental details. When problems (11) are sufficiently small, they can be solved using a mixed-integer programming reformulation. In this experiment, *we consider problems within a range of sizes such that ground-truth optimal solutions to the test-set problems can be computed within a reasonable time*. *This provides an initial setting* in which certified optimal solutions can be obtained, against which the optimality of our learned solutions can be measured objectively. We use the open-source YALMIP package to solve the instances by replacing their lower-level problem with KKT conditions, and solving the resulting single-level MIP with a branch-and-bound method (Lofberg, 2004).

We refer to a BQP problem with m and n variables at the upper and lower levels, respectively, as having size $m \times n$. Three sets of BQP problems having size 3×2, 6×4, and 9×6 are randomly generated - first by drawing the elements of each matrix $\boldsymbol{A}, \boldsymbol{E}, \boldsymbol{F}, \boldsymbol{G}$ and each vector $\boldsymbol{b}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{h}$ from a uniform distribution $U(0, 1)$. Positive-semidefinite $\boldsymbol{Q}$ and $\boldsymbol{H}$ are constructed by self-multiplication of such a matrix. Individual problem instances are generated by drawing vectors of linear objective coefficients $(\boldsymbol{c}, \boldsymbol{d})$ also from $U(0, 1)$. Each set is divided into validation and test portions, numbering

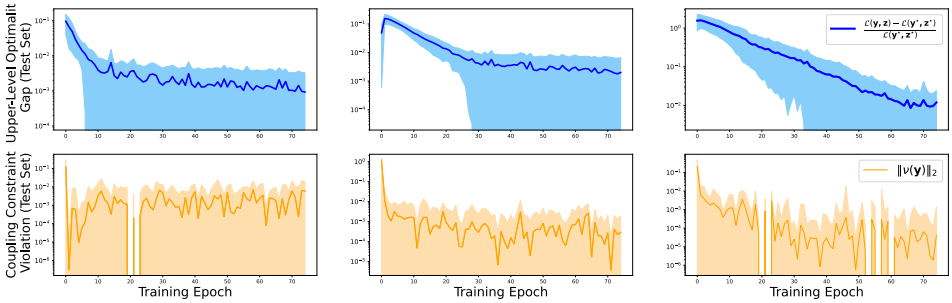

Figure 2: Optimality gap and coupling violation, on different-sized BQP problems from left to right: 3×2, 6×4, 9×6. Shows mean and standard deviation over the test set, at each training epoch.

1000 each. The prediction model $\hat{\mathcal{B}}_\theta$ is a 5-layer network followed by 20 steps of Algorithm 1. Solution and differentiation of problem (12) are implemented in *cvxpylayers* (Agrawal et al., 2019a).

Results. Figure 2 illustrates the two main evaluation metrics throughout Algorithm 1, in terms of mean and standard deviation over the test set for each parametric BQP. In each case, the relative optimality gap (in blue) is reduced by 2 orders of magnitude over 75 epochs to a value between 10^{-3} and 10^{-2}. The coupling violation (5) is rapidly diminished in the first epoch and then generally bounded below 10^{-2} within a full standard deviation throughout training. Test set metrics are also reported in Table 1. Together, these results demonstrate the ability of Algorithm 1 to learn bilevel optimization with negligible error on small-scale BQP problems. Beyond the problem sizes considered here, starting with 12×9, the time taken by YALMIP to fully solve the test instances *becomes intractable, thus we cannot benchmark against certified optimal solutions on larger instances*.

5.2 Learning Optimal Control Co-Design

This section introduces more challenging bilevel optimizations, in terms of both their size and their form. Optimal Control Co-Design is a bilevel problem setting in which an engineering system is designed to optimize an economic objective at the upper level, subject to conditions on its behavior under a known optimal control policy, which forms its lower level. The problems described below cannot hope to be solved by conventional methods with certificates of optimality; as an alternative, we compare our learned solutions to the results of a PSO-based metaheuristic method. The PSO framework is commonly applied to problems that lack efficient solution methods, and thus is a favored tool in design optimization (Asaah, Hao, and Ji, 2021; Hou et al., 2015). Details of the PSO baseline methods, along with illustrations of their results, can be found in Appendix A .

Nonconvex Bilevel Optimization: Control Co-design of a Nonlinear System

We consider a nonlinear control problem, in which two connected tanks are controlled by a single pump and a two-way valve. The system is a simplified model of a pumped-storage hydroelectricity, which is a form of energy storage used by electric power systems for load balancing. The system dynamics are described by nonlinear ODE's $\dot{\boldsymbol{x}} = f(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}, \boldsymbol{y})$:

$$\dot{x_1} = y_1(1-u_2)u_1 - y_2\sqrt{x_1}, \qquad (13a) \qquad \dot{x_2} = y_1 u_2 u_1 + y_2\sqrt{x_1} - y_2\sqrt{x_2}, \qquad (14a)$$

in which x_1, x_2 are the water levels in each tank. Control actions consist of u_1 and u_2, which are the pump modulation and valve opening. The nonlinear optimal control problem (16) seeks the control policy which minimizes energy expended to reach a desired terminal state $\boldsymbol{p}$. The function `ODESolve` represents Euler discretization of (13, 14) over $N = 20$ frames to a final time T. This yields new variables $\boldsymbol{x} = [\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}] \in \mathbb{R}^{N\times 2}$, $\boldsymbol{u} = [\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(N)}] \in \mathbb{R}^{N\times 2}$ bound by a sequence of nonlinear (i.e., *nonconvex*) equality constraints (16c) for $1 \le i < N$, while $dt = \frac{T}{N}$. The upper-level problem (15)

$$\begin{aligned} \mathcal{B}(\boldsymbol{p}) = \operatorname{argmin}_{\boldsymbol{y}} \quad & \boldsymbol{v}^T\boldsymbol{y} && (15a) \\ s.t. \quad & \boldsymbol{x}, \boldsymbol{u} = O_{\boldsymbol{p}}(\boldsymbol{y}) && (15b) \\ & \boldsymbol{y}_{min} \le \boldsymbol{y} \le \boldsymbol{y}_{max} && (15c) \\ & \boldsymbol{x}^{(N)} = \boldsymbol{p}, && (15d) \end{aligned}$$

$$\begin{aligned} O_{\boldsymbol{p}}(\boldsymbol{y}) \coloneqq \operatorname*{argmin}_{0\le \boldsymbol{x},\boldsymbol{u}\le 1} \quad & \sum_{k=1}^{N} \|\boldsymbol{u}^{(k)}\|_2^2 && (16a) \\ s.t. \quad & \boldsymbol{x}^{(N)} = \boldsymbol{p} && (16b) \\ & \boldsymbol{x}^{(i+1)} = \text{ODESolve}(f(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}, \boldsymbol{y})). && (16c) \end{aligned}$$

seeks to optimize the design of such a system in terms of its overall cost $\mathcal{L}(\boldsymbol{y}, \boldsymbol{z}) \coloneqq \boldsymbol{v}^T\boldsymbol{y}$, treating the inlet and outlet valve coefficients $\boldsymbol{y} = [y_1, y_2]$ as free design parameters. A feasible design demands

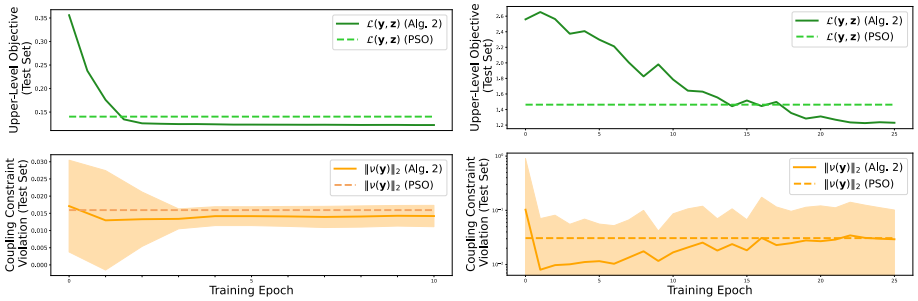

Figure 3: Test set metrics, learning control co-design of the two-tank (left) / HVAC (right) systems.

that upper and lower bounds on each element of $\boldsymbol{y}$ are satisfied per (15c). Additionally, the parametric end-state constraint (16b) is duplicated at the upper level in (15d) to emphasize its coupling role. The full reformulation per Section 4.3 is detailed in Appendix D. The initial condition is $\boldsymbol{x}^0 = \mathbf{0}$. Taken together, the coupled problems (15, 16) seek the parameters $\boldsymbol{y}$ which yield the minimal-cost system design that can be controlled to state $\boldsymbol{p}$ by time T under its control policy (16). In this experiment, our model $\hat{\mathcal{B}}_\theta$ is trained to perform a fast and accurate approximation to this design problem for any such scenario specified by a given $\boldsymbol{p} \in [0, 1]$. In the notation of Section 3, $\boldsymbol{z} := (\boldsymbol{x}, \boldsymbol{u})$.

Experimental details. We consider an experiment in which $T = 10s$ and $N = 20$, $\boldsymbol{c}_{min} = 0$, and $\boldsymbol{c}_{max} = \frac{1}{3}$. Problem instances correspond to reference states $\{\boldsymbol{p}\}$, which are randomly generated from $U(0, 1)$, but with $p_1 < p_2$ to ensure feasibility of problem (15). They are partitioned into training, validation, and test sets of sizes 10000, 1000, 1000. The model $\hat{\mathcal{B}}_\theta$ consists of an 8-layer network $\mathcal{N}_\theta$ followed by Algorithm 1. To implement Algorithm 1 is nontrivial, as it requires a differentiable solution of the nonconvex lower-level programs (16). For this, we employ the differentiable model predictive control solver of Amos et al. (2018), which differentiates problem (16) implicitly via the KKT conditions of the final convex subproblem of a sequential quadratic programming.

Results. Figure 3 (top two) illustrates the value of the design objective $\mathcal{L}(\mathbf{y}, \mathbf{z})$, as well as the coupling violation (5), over the test set throughout training. Overall metrics are also found in Table 2 under the header TT. Despite investing an average of 1286.7s of solution time per instance, the PSO baseline produces design solutions with 15 percent higher cost than those learned by Algorithm 1, which infers solutions in 0.027s on average. At the same time, it attains nearly identical satisfaction of the coupling constraint, on average. *This result is significant, since it demonstrates an ability to learn nonconvex bilevel optimization with high accuracy.*

Control Co-Design of a Building HVAC System

Finally, we consider the design and control of the heating, ventilation, and air conditioning (HVAC) system in a building. The control problem minimizes energy use while maintaining indoor temperature within prescribed bounds (18). The building consists of 2 zones, thermally connected to each other and the outside environment by a matrix of conductivity coefficients $\boldsymbol{A}$. State variables $\boldsymbol{x} = [\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}] \in \mathbb{R}^{N \times 8}$ consist of the temperatures of each zone's floor, walls, indoor air and exterior facade at each timestep k. Control actions $\boldsymbol{u} = [\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(N)}] \in \mathbb{R}^{N \times 2}$ induce heat flows into each zone, which affect the temperature states $\boldsymbol{x}$ via the actuator design variables $\boldsymbol{Y} \in \mathbb{R}^{8 \times 2}$. The states are also affected by random disturbances $\boldsymbol{d}$, which include heat transfer from occupants and solar irradiation. Thermal constraints (18c) demand that the indoor air temperature must remain within prescribed time-varying bounds $(\underline{\boldsymbol{p}}, \overline{\boldsymbol{p}})$. This condition couples the upper and lower problems, which we emphasize by its duplication at (17d). The design task (17) asks to learn $\boldsymbol{Y}$ which minimizes a linear cost function $Tr(\boldsymbol{V}^T \boldsymbol{Y})$ while allowing the system to be maintained within the thermal bounds using optimal control. In the notation of Section 3, we have $\boldsymbol{p} := (\underline{\boldsymbol{p}}, \overline{\boldsymbol{p}})$, $\boldsymbol{y} := \boldsymbol{Y}$ and $\boldsymbol{z} := (\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$:

$$\mathcal{B}(\boldsymbol{p}) = \operatorname{argmin} Tr(\boldsymbol{V}^T \boldsymbol{Y}) \quad (17a)$$
$$s.t. \quad \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w} = O_{\boldsymbol{p}}(\boldsymbol{Y}) \quad (17b)$$
$$\boldsymbol{Y} \succeq \mathbf{0} \quad (17c)$$
$$\underline{\boldsymbol{p}}^{(k)} \leq \boldsymbol{w}^{(k)} \leq \overline{\boldsymbol{p}}^{(k)} \quad (17d)$$

$$O_{\boldsymbol{p}}(\boldsymbol{Y}) = \operatorname*{argmin}_{\boldsymbol{x}, 0 \leq \boldsymbol{u} \leq 1, \boldsymbol{w}} \sum_{k \in \{1 \ldots N\}} \|\boldsymbol{u}^{(k)}\|_2^2 \quad (18a)$$
$$s.t. \quad \boldsymbol{w}^{(k)} = \boldsymbol{C}\boldsymbol{x}^{(k)} \quad (18b)$$
$$\underline{\boldsymbol{p}}^{(k)} \leq \boldsymbol{w}^{(k)} \leq \overline{\boldsymbol{p}}^{(k)} \quad (18c)$$
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{A}\boldsymbol{x}^{(k)} + \boldsymbol{Y}\boldsymbol{u}^{(k)} + \boldsymbol{E}\boldsymbol{d}^{(k)} \quad (18d)$$

Experimental details. Our experiment assumes $N = 30$ steps, the problem instances with for the thermal bounds $\underline{\boldsymbol{p}}$ are generated from a β-random walk along with $\overline{\boldsymbol{p}} = \underline{\boldsymbol{p}} + 2.0$, validation and test sets of size 10000, 1000, 1000. A fixed disturbance pattern $\boldsymbol{d}$ is generated from the building control test suite in `NEUROMANCER` (Drgona et al., 2023). The model $\mathcal{B}_\theta$ is a 6-layer ReLU network followed by 10 steps of Algorithm (1). Differentiable solution of (18) is implemented using `cvxpylayers`.

Results. Figure 3 (right) illustrates the value of the design objective $\mathcal{L}(\mathbf{y}, \mathbf{z})$, as well as the coupling violation (5), over the test set throughout training. Overall metrics are also found in Table 2 under HVAC. While our learned designs incur nearly identical coupling constraint violations on average, they are achieved at about 19 percent lower cost, and with orders of magnitude lower solving time.

6 Conclusion and Limitations

This paper has shown how the modern toolkit of differentiable optimization can be used to train machine learning models as fast and accurate approximators of parametric bilevel optimization with coupling constraints. Experiments on control co-design problems show that the proposed learning to optimize framework can accurately approximate even nonconvex programs. Multiple interesting avenues remain to extend the work. For example, extension to integer-valued decision variables would broaden its applicability. Second, when lower-level objectives are nondifferentiable with respect to the upper variables, extensions based on surrogate gradients may be investigated. Addressing these points would further broaden the scope of this work.

Acknowledgments and Disclosure of Funding

This research was supported by the Energy System Co-Design with Multiple Objectives and Power Electronics (E-COMP) Initiative, under the Laboratory Directed Research and Development (LDRD) Program at Pacific Northwest National Laboratory (PNNL). It was also supported by the Energy Earthshots Inititaive as part of the DOE Office of Biological and Environmental Research. PNNL is operated by DOE by the Battelle Memorial Institute under Contract DE-A06-76RLO 1830.

This work was also partially supported by the Ralph O'Connor Sustainable Energy Institute at Johns Hopkins University, as well as NSF grants 2242931, 2232054, and 2143706.

References

Abolfazli, N.; Sharifi, S.; Fazlyab, M.; and Hamedani, E. Y. 2025. Perturbed Gradient Descent via Convex Quadratic Approximation for Nonconvex Bilevel Optimization. arXiv:2504.17215.

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

Agrawal, A.; Barratt, S.; and Boyd, S. 2021. Learning Convex Optimization Models. *IEEE/CAA Journal of Automatica Sinica*, 8(8): 1355–1364.

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

Amos, B.; Jimenez, I.; Sacks, J.; Boots, B.; and Kolter, J. Z. 2018. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31.

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

Amos, B.; et al. 2023. Tutorial on amortized optimization. *Foundations and Trends® in Machine Learning*, 16(5): 592–732.

Andrychowicz, M.; Denil, M.; Colmenarejo, S. G.; Hoffman, M. W.; Pfau, D.; Schaul, T.; Shillingford, B.; and de Freitas, N. 2016. Learning to learn by gradient descent by gradient descent. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 3988–3996. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510838819.

Asaah, P.; Hao, L.; and Ji, J. 2021. Optimal placement of wind turbines in wind farm layout using particle swarm optimization. *Journal of Modern Power Systems and Clean Energy*, 9(2): 367–375.

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

Beck, Y.; and Schmidt, M. 2021. A gentle and incomplete introduction to bilevel optimization.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2020. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*.

Bergonti, F.; Nava, G.; Wüest, V.; Paolino, A.; L'Erario, G.; Pucci, D.; and Floreano, D. 2023. Co-Design Optimisation of Morphing Topology and Control of Winged Drones. *arXiv preprint arXiv:2309.13948*.

Bergonti, F.; Nava, G.; Wüest, V.; Paolino, A.; L'Erario, G.; Pucci, D.; and Floreano, D. 2024. Co-design optimisation of morphing topology and control of winged drones. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 8679–8685. IEEE.

Bertsimas, D.; and Stellato, B. 2022. Online Mixed-Integer Optimization in Milliseconds. *INFORMS Journal on Computing*, 34(4): 2229–2248.

Binmore, K. G. 2007. *Playing for real: a text on game theory*. Oxford university press.

Busseti, E.; Moursi, W. M.; and Boyd, S. 2019. Solution refinement at regular points of conic problems. *Computational Optimization and Applications*, 74: 627–643.

Cerulli, M.; Oustry, A.; d'Ambrosio, C.; and Liberti, L. 2021. Solving a class of bilevel programs with quadratic lower level.

Chen, W.; Tanneau, M.; and Van Hentenryck, P. 2023. End-to-end feasible optimization proxies for large-scale economic dispatch. *IEEE Transactions on Power Systems*, 39(2): 4723–4734.

Dinev, T.; Mastalli, C.; Ivan, V.; Tonneau, S.; and Vijayakumar, S. 2022. Differentiable optimal control via differential dynamic programming. *arXiv preprint arXiv:2209.01117*.

Donti, P. L.; Rolnick, D.; and Kolter, J. Z. 2021. DC3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*.

Drgona, J.; Tuor, A.; Koch, J.; Shapiro, M.; and Vrabie, D. 2023. NeuroMANCER: Neural Modules with Adaptive Nonlinear Constraints and Efficient Regularizations.

Dumouchelle, J.; Julien, E.; Kurtz, J.; and Khalil, E. B. 2024. Neur2BiLO: Neural Bilevel Optimization. *arXiv preprint arXiv:2402.02552*.

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

Frerix, T.; Nießner, M.; and Cremers, D. 2020. Homogeneous linear inequality constraints for neural network activations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 748–749.

Ghadimi, S.; and Wang, M. 2018. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*.

Gould, S.; Hartley, R.; and Campbell, D. 2021. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8): 3988–4004.

Hou, P.; Hu, W.; Soltani, M.; and Chen, Z. 2015. Optimized placement of wind turbines in large-scale offshore wind farm using particle swarm optimization algorithm. *IEEE Transactions on Sustainable Energy*, 6(4): 1272–1282.

Huang, W.; Isenberg, N. M.; Drgoňa, J.; Vrabie, D. L.; and Dilkina, B. 2025. Efficient Primal Heuristics for Mixed Binary Quadratic Programs Using Suboptimal Rounding Guidance. *Proceedings of the International Symposium on Combinatorial Search*, 18(1): 74–82.

Innocente, M. S.; and Sienz, J. 2021. Constraint-handling techniques for particle swarm optimization algorithms. *arXiv preprint arXiv:2101.10933*.

Khalil, E.; Le Bodic, P.; Song, L.; Nemhauser, G.; and Dilkina, B. 2016. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

King, E.; Kotary, J.; Fioretto, F.; and Drgona, J. 2024. Metric Learning to Accelerate Convergence of Operator Splitting Methods for Differentiable Parametric Programming. *arXiv preprint arXiv:2404.00882*.

Konstantinov, A. V.; and Utkin, L. V. 2024. A new computationally simple approach for implementing neural networks with output hard constraints. In *Doklady Mathematics*, 1–9. Springer.

Kotary, J.; Fioretto, F.; Van Hentenryck, P.; and Wilder, B. 2021. End-to-End Constrained Optimization Learning: A Survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 4475–4482.

Lee, J. D.; Panageas, I.; Piliouras, G.; Simchowitz, M.; Jordan, M. I.; and Recht, B. 2019. First-order methods almost always avoid strict saddle points. *Mathematical programming*, 176: 311–337.

Li, X.; Tian, P.; and Min, X. 2006. A hierarchical particle swarm optimization for solving bilevel programming problems. In *International Conference on Artificial Intelligence and Soft Computing*, 1169–1178. Springer.

Lofberg, J. 2004. YALMIP: A toolbox for modeling and optimization in MATLAB. In *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*, 284–289. IEEE.

Miranda, L. J. 2018. PySwarms: a research toolkit for Particle Swarm Optimization in Python. *Journal of Open Source Software*, 3(21): 433.

Ng, Y.; Misra, S.; Roald, L. A.; and Backhaus, S. 2018. Statistical learning for DC optimal power flow. In *2018 Power Systems Computation Conference (PSCC)*, 1–7. IEEE.

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

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch.

Sadigh, A. N.; Mozafari, M.; and Karimi, B. 2012. Manufacturer–retailer supply chain coordination: A bi-level programming approach. *Advances in Engineering Software*, 45(1): 144–152.

Sambharya, R.; Hall, G.; Amos, B.; and Stellato, B. 2023. End-to-End Learning to Warm-Start for Real-Time Quadratic Optimization. In *Learning for Dynamics and Control Conference*, 220–234. PMLR.

Sambharya, R.; Hall, G.; Amos, B.; and Stellato, B. 2024. Learning to Warm-Start Fixed-Point Optimization Algorithms. *Journal of Machine Learning Research*, 25(166): 1–46.

Sharifi, S.; Abolfazli, N.; Hamedani, E. Y.; and Fazlyab, M. 2025. Safe Gradient Flow for Bilevel Optimization. arXiv:2501.16520.

Shen, J.; Chen, X.; Heaton, H.; Chen, T.; Liu, J.; Yin, W.; and Wang, Z. 2020. Learning a minimax optimizer: A pilot study. In *International Conference on Learning Representations*.

Sinha, A.; Malo, P.; and Deb, K. 2013. Efficient evolutionary algorithm for single-objective bilevel optimization. *arXiv preprint arXiv:1303.3901*.

Sinha, A.; Malo, P.; and Deb, K. 2017. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE transactions on evolutionary computation*, 22(2): 276–295.

Solodov, M. 2007. An explicit descent method for bilevel convex optimization. *Journal of Convex Analysis*, 14(2): 227.

Song, J.; lanka, r.; Yue, Y.; and Dilkina, B. 2020. A General Large Neighborhood Search Framework for Solving Integer Linear Programs. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 20012–20023. Curran Associates, Inc.

Tang, B.; Khalil, E. B.; and Drgoňa, J. 2025. Learning to Optimize for Mixed-Integer Non-linear Programming with Feasibility Guarantees. arXiv:2410.11061.

Tang, Y.; Agrawal, S.; and Faenza, Y. 2020. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, 9367–9376. PMLR.

Tordesillas, J.; How, J. P.; and Hutter, M. 2023. Rayen: Imposition of hard convex constraints on neural networks. *arXiv preprint arXiv:2307.08336*.

Van Hentenryck, P. 2025. Optimization Learning. arXiv:2501.03443.

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

Zhang, Z. 2018. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, 1–2. Ieee.

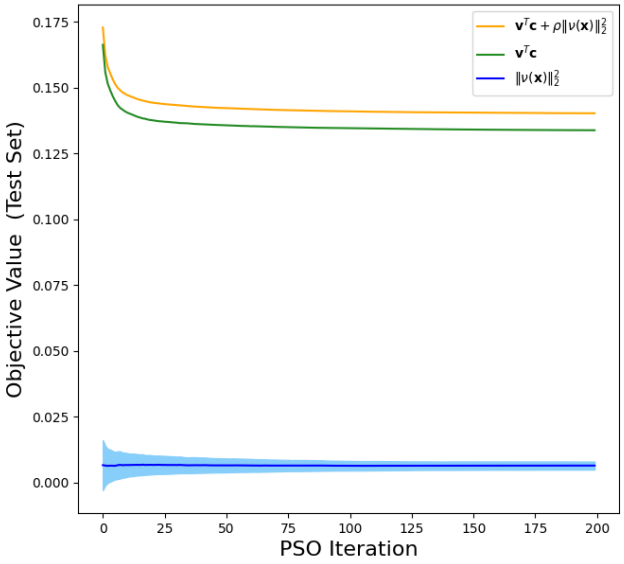

Figure 4: Best Objective Value per PSO Iteration on Two-Tank System Co-Design

A Experimental Details: Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a metaheuristic optimization method which works by having a population (called a swarm) of candidate solutions (called particles). Particles update their solutions using simple update rules based on their own best-known position in search space, as well as the entire swarm's best-known position. PSO is commonly applied to optimization problems with complex objective functions and simple constraints. While simple bounds can be handled naturally in PSO, more complex constraints are often handled using penalty functions. A survey of constraint-handling techniques in PSO is found in Innocente and Sienz (2021).

For both control co-design experiments, we implement PSO baseline methods using the open-source library *pyswarms* Miranda (2018). We adopt the penalty-function approach to handle coupling constraints in our PSO baseline methods, and since the remaining upper-level constraints take the form of variable bounds, those are handled natively in the PSO algorithm of pyswarms. As is the case in the paper's main proposal, the lower-level problem must be feasible relative to the upper-level solutions found at each iteration of PSO. The constraints preventing this condition are treated as coupling constraints and enforced with penalty functions in the lower-level problem. For the control co-design problems, the lower-level problem implementations are identical to (20) and (23). Overall, the PSO optimizes a relaxed upper-level objective function equal to

$$\mathcal{L}_p(\boldsymbol{y}) + \kappa\nu(\boldsymbol{y}), \tag{19}$$

subject to $\boldsymbol{y} \in C_p$, which are simple bounds on $\boldsymbol{y}$ in both of our experimental cases. Note that evaluation of $\nu(\boldsymbol{y})$ requires optimization at the lower level, at each step of PSO.

In each experiment, the pyswarms solver is given its default cognitive, social and inertia parameters $c_1 = 0.5, c_2 = 0.5, w = 0.9$, and run with 128 particles for 200 iterations. The penalty coefficient κ is chosen so that average coupling constraint violations over the test set are on the order of $1e-2$. This corresponds to $\kappa = 100.0$ in the two-tank experiment and $\kappa = 5.0$ in the HVAC experiment.

B Additional Results: Particle Swarm Optimization

We illustrate the evolution of the PSO objective throughout its solution of the test set instances. We plot the best objective values, among all particles, per iteration of PSO. The full PSO objective

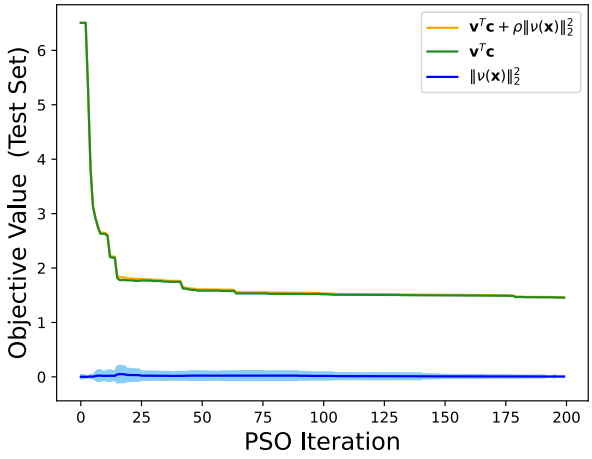

Figure 5: Best Objective Value per PSO Iteration on HVAC System Co-Design

is illustrated, and also is shown in terms of upper-level design objective and coupling constraint violation penalty. Each metric is reported on average over the test set. Figure 4 corresponds to the two-tank problem, and 5 corresponds to the HVAC problem.

C Experimental Details: Learning Bilevel Quadratic Programming

This section reports additional details on the experiments presented in Section 5.1.

C.1 Hyperparameters and Training

Results in Section 5 are shown from the model which achieves the lowest loss among independent training runs using all combinations of the following hyperparameters:

- Learning rates from among $[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}]$
- Correction stepsizes γ from $[10^{-2}, 10^{-3}, 10^{-4}]$
- $\mathcal{L}^{\texttt{SOFT}}$ penalty weights λ from $[10^2, 10^3]$

The best values are $10^{-3}, 10^{-4}, 10^2$ respectively. All models are trained using the Adam optimizer Zhang (2018) in PyTorch. In each training run, 10 correction steps are applied in training and 20 are applied at test time.

D Experimental Details: Learning Control Co-design of a Two-Tank System

This section reports additional details on the nonlinear system design experiments presented in Section 5.2.

D.1 Problem Reformulation

The two-tank system design and control problem (15, 16) is bound by the coupling constraint $\boldsymbol{x}^{(N)} = \boldsymbol{p}$, which is redundantly placed at both levels to emphasize its coupling effect. This coupling constraint expresses that a valid system design must be controllable to the end-state $\boldsymbol{p}$. We recognize that this condition may not be satisfiable for any design variable $\boldsymbol{y}$; for instance, some $\boldsymbol{y}$ may not allow sufficient throughput to fill the tanks from $\mathbf{0}$ to $\boldsymbol{p}$ by time step N.

In practice, we therefore reformulate the problem as follows, as prefaced in Section 4.3:

$$\mathcal{B}(\boldsymbol{p}) = \text{argmin}_{\boldsymbol{y}} \quad \boldsymbol{v}^T\boldsymbol{y} \tag{20a}$$
$$s.t. \quad \boldsymbol{x}, \boldsymbol{u} = O_p(\boldsymbol{y}) \tag{20b}$$
$$\boldsymbol{y}_{min} \leq \boldsymbol{y} \leq \boldsymbol{y}_{max} \tag{20c}$$
$$\boldsymbol{x}^{(N)} = \boldsymbol{p}. \tag{20d}$$

$$O_p(\boldsymbol{y}) \coloneqq \underset{0\leq \boldsymbol{x},\boldsymbol{u}\leq 1}{\text{argmin}} \quad \sum_{k=1}^{N} \|\boldsymbol{u}^{(k)}\|_2^2 + \rho\|\boldsymbol{x}^{(N)} - \boldsymbol{p}\|^2 \tag{21a}$$
$$s.t. \quad \boldsymbol{x}^{(i+1)} = \text{ODESolve}(f(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}, \boldsymbol{y})) \tag{21b}$$

In this reformulation, the constraint $\boldsymbol{x}^{(N)} = \boldsymbol{p}$ remains at the upper level as a *coupling* constraint, since it binds lower-level variables within the upper-level problem. Thus, it is treated by the coupling constraint correction in Algorithm 1. The constraint is absent however, from the lower level in this formulation and instead replaced with a penalty on the lower-level objective. We take $\rho = 100$ as the penalty weight in all experimental runs. While equivalent to the original bilevel problem (15, 16), the above formulation accomodates lower-level feasibility for any candidate design solution at the upper level.

D.2 Hyperparameters and Training

Results are shown from the model which achieves the lowest upper-level objective value on average, from among those whose average coupling constraint violation is less than or equal to that of the PSO baseline method after 10 epochs. The results are chosen from among independent training runs using all combinations of the following hyperparameters:

- Learning rates from among $[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}]$
- Correction stepsizes γ from $[10^{-2}, 10^{-3}, 10^{-4}]$
- $\mathcal{L}^{\texttt{SOFT}}$ penalty weights λ from $[10, 10^2, 10^3]$

The best values are $10^{-3}, 10^{-2}, 10$ respectively. All models are trained using the Adam optimizer in PyTorch. In each training run, 5 correction steps are applied in training and 10 are applied at test time. The upper-level objective function has linear coefficients $\boldsymbol{v}$ consisting of all ones in each problem instance, meaning that the inlet and outlet valve coefficients should be minimized with equal priority.

E Experimental Details: Learning Control Co-design of a Building HVAC System

This section reports additional details on the HVAC system design experiments presented in Section 5.2.

E.1 Problem Reformulation

The building HVAC design and control problem (17,18) is coupled by the thermal constraints $\underline{\boldsymbol{p}}^{(k)} \leq \boldsymbol{w}^{(k)} \leq \overline{\boldsymbol{p}}^{(k)}$, which appear at both levels to emphasize their coupling role. Before Algorithm 1 can be applied, we recognize that those constraints may not be satisfiable when design variables $\boldsymbol{Y}$ prevent heat flows from converting to temperature changes rapidly enough to stay within those changing bounds.

To arrive at an equivalent problem which ensures feasibility at the lower level for any $\boldsymbol{Y}$, we introduce slack variables $\underline{\boldsymbol{s}}^{(k)}$, $\overline{\boldsymbol{s}}^{(k)}$ to both sides of (18c) yielding

$$\underline{\boldsymbol{p}}^{(k)} - \underline{\boldsymbol{s}}^{(k)} \leq \boldsymbol{w}^{(k)} \leq \overline{\boldsymbol{p}}^{(k)} + \underline{\boldsymbol{s}}^{(k)},$$

along with a no-slack condition which maintains equivalence to the original problem:

$$\underline{\boldsymbol{s}}^{(k)} = \overline{\boldsymbol{s}}^{(k)} = \boldsymbol{0} \qquad \forall k.$$

It is held at the upper level, and replaced in the lower level by a penalty term:

$$\mathcal{B}(\boldsymbol{p}) = \text{argmin} \quad Tr(\boldsymbol{V}^T\boldsymbol{Y}) \tag{22a}$$
$$s.t. \quad \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w} = O_{\boldsymbol{p}}(\boldsymbol{Y}) \tag{22b}$$
$$\boldsymbol{Y} \geq \boldsymbol{0} \tag{22c}$$
$$\underline{\boldsymbol{s}}^{(k)} = \overline{\boldsymbol{s}}^{(k)} = \boldsymbol{0} \quad \forall k \tag{22d}$$

$$O_{\boldsymbol{p}}(\boldsymbol{Y}) = \underset{\boldsymbol{x}, 0 \leq \boldsymbol{u} \leq 1, \boldsymbol{w}, \boldsymbol{s}}{\text{argmin}} \quad \sum_{k \in \{1 \ldots N\}} \|\boldsymbol{u}^{(k)}\|_2^2 + \rho \left(\sum_{k \in \{1 \ldots N\}} \|\underline{\boldsymbol{s}}^{(k)}\|_2^2 + \sum_{k \in \{1 \ldots N\}} \|\overline{\boldsymbol{s}}^{(k)}\|_2^2 \right) \tag{23a}$$
$$s.t. \quad \boldsymbol{w}^{(k)} = \boldsymbol{C}\boldsymbol{x}^{(k)} \tag{23b}$$
$$\underline{\boldsymbol{p}}^{(k)} - \underline{\boldsymbol{s}}^{(k)} \leq \boldsymbol{w}^{(k)} \leq \overline{\boldsymbol{p}}^{(k)} + \underline{\boldsymbol{s}}^{(k)} \tag{23c}$$
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{A}\boldsymbol{x}^{(k)} + \boldsymbol{Y}\boldsymbol{u}^{(k)} + \boldsymbol{E}\boldsymbol{d}^{(k)}. \tag{23d}$$

In our implementation of Algorithm 1, coupling constraint corrections are applied to (22d). It is the operative coupling constraint in this reformulation, binding the upper-level problem to lower-level variables $\underline{\boldsymbol{s}}^{(k)}, \overline{\boldsymbol{s}}^{(k)}$.

E.2 Hyperparameters and Training

Results are shown from the model which achieves the lowest upper-level objective value on average, from among those whose average coupling constraint violation is less than or equal to that of the PSO baseline method after 25 epochs. The results are chosen from among independent training runs using all combinations of the following hyperparameters:

- Learning rates from among $[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}]$
- Correction stepsizes γ from $[10^{-4}, 10^{-5}, 10^{-6}]$
- $\mathcal{L}^{\texttt{SOFT}}$ penalty weights λ from $[10^2, 10^3]$

The best values are $10^{-3}, 10^{-4}, 10^2$ respectively. All models are trained using the Adam optimizer in PyTorch. In each training run, 5 correction steps are applied in training and 10 are applied at test time. The upper-level objective function has linear coefficients $\boldsymbol{V}$ consisting of all ones in each problem instance, meaning that all elements of the actuator design variable have equal cost.