DEEP FLEXQP: ACCELERATED NONLINEAR PROGRAMMING VIA DEEP UNFOLDING

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose an always-feasible "flexible" quadratic programming (QP) optimizer, FlexQP, which is based on an exact relaxation of the QP constraints. If the original constraints are feasible, then the optimizer finds the optimal solution to the original QP. On the other hand, if the constraints are infeasible, the optimizer identifies a solution that minimizes the constraint violation in a sparse manner. FlexQP scales favorably with respect to the problem dimension, is robust to both feasible and infeasible QPs with minimal assumptions on the problem data, and can be effectively warm-started. We subsequently apply deep unfolding to improve our optimizer through data-driven techniques, leading to an accelerated version called Deep FlexQP. By learning dimension-agnostic feedback policies for the parameters from a small number of training examples, Deep FlexQP generalizes to problems with larger dimensions and can optimize for many more iterations than it was initially trained for. Our approach outperforms two recently proposed state-of-the-art accelerated QP approaches on a suite of benchmark systems including portfolio optimization, classification, and regression problems. We provide guarantees on the expected performance of our deep QP optimizer through PAC-Bayes generalization bounds. These certificates are used to design an accelerated SQP solver that solves nonlinear optimal control and predictive safety filter problems faster than traditional approaches. Overall, our approach is very robust and greatly outperforms existing non-learning and learning-based optimizers in terms of both runtime and convergence to the optimal solution across multiple classes of NLPs.

1 INTRODUCTION

Nonlinear programming (NLP) is a key technique for both large-scale decision making, where difficulty arises due to the sheer number of variables and constraints, as well as real-time embedded systems, which need to solve many NLPs with similar structure quickly and robustly. Within NLP, quadratic programming (QP) plays a fundamental role as many real-world problems such as optimal control (Anderson & Moore, 2007), portfolio optimization (Markowitz, 1952; Boyd et al., 2013; 2017), and machine learning (Huber, 1964; Cortes & Vapnik, 1995; Tibshirani, 1996; Candes et al., 2008) problems can be represented as QPs. Furthermore, sequential quadratic programming (SQP) methods utilize QP as a submodule to solve much more complicated problems where the objective and constraints may be nonlinear and non-convex, such as in nonlinear model predictive control (Diehl et al., 2009; Rawlings et al., 2020), state estimation (Aravkin et al., 2017), and power grid optimization (Montoya et al., 2019). SQP itself can even be used as a subproblem for solving mixed-integer NLPs (Leyffer, 2001) and large-scale partial differential equations (Fang et al., 2023).

However, a common difficulty with SQP methods occurs when the linearization of the constraints results in an infeasible QP subproblem, and a large amount of research has focused on how to repair or avoid these infeasibilities, e.g., (Fletcher, 1985; Izmailov & Solodov, 2012), among others. A significant advantage of SNOPT (Gill et al., 2005), one of the most well-known SQP-based methods, is in its infeasibility detection and reduction handling. These considerations necessitate a fast yet robust QP solver that works under minimal assumptions on the problem parameters.

To this end, we propose **FlexQP**, a *flexible* QP solver that is always-feasible, meaning that it can solve any QP regardless of the feasibility of the constraints. Our method is based on an exact relaxation of the QP constraints: if the original QP was feasible, then FlexQP will identify the

optimal solution. On the other hand, if the original QP was infeasible, instead of erroring or failing to return a solution, FlexQP automatically identifies the infeasibilities while simultaneously finding a point that minimizes the constraint violation. This allows FlexQP to be a robust QP solver in and of itself, but its power shines when used a submodule in an SQP-type method.

Moreover, through the relaxation of the constraints, multiple hyperparameters are introduced that can be difficult to tune and have a non-intuitive effect on the optimization. To address this shortcoming, we use deep unfolding (Monga et al., 2021) to design lightweight feedback policies for the parameters based on actual problem data and solutions for QP problems of interest, leading to an accelerated version titled **Deep FlexQP**. Learning the parameters in a data-driven fashion avoids the laborious process of tuning them by hand or designing heuristics for how they should be updated from one iteration to the next. Meanwhile, these data-driven rules have been shown to strongly outperform the hand-crafted ones, such as in the works by Ichnowski et al. (2021) and Saravanos et al. (2025).

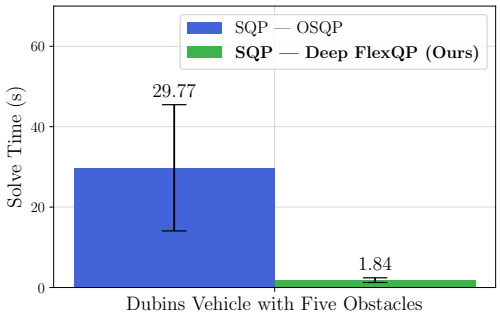


Figure 1: SQP with Deep FlexQP can solve highly-constrained nonlinear optimizations over 15x faster than SQP with OSQP (averaged over 100 problems).

We thoroughly benchmark Deep FlexQP against traditional and learned QP optimizers on multiple QP problem classes including machine learning, portfolio optimization, and optimal control problems. Moreover, we certify the performance of Deep FlexQP through probably approximately correct (PAC) Bayes generalization bounds, which provide a guarantee on the mean performance of the optimizer. We propose a log-scaled training loss that better captures the performance of the optimizer when the residuals are very small. Finally, we deploy Deep FlexQP to solve nonlinearly-constrained trajectory optimization and predictive safety filter problems (Wabersich & Zeilinger, 2021). Overall, Deep FlexQP can produce an order-of-magnitude speedup over OSQP when deployed as a subroutine in an SQP-based approach (Fig. 1), while also robustly handling infeasibilities that may occur due to a poor linearization or an over-constrained problem.

2 Related Work

Deep unfolding is a learning-to-optimize approach (Chen et al., 2022b) that has roots in the signal and image processing domains (Gregor & LeCun, 2010; Wang et al., 2015). It constitutes the state-of-the-art approach for sparse recovery and video reconstruction (De Weerdt et al., 2024). Furthermore, deep unfolding has been recently applied to accelerate QPs. Saravanos et al. (2025) use an analogy to closed-loop control and learn feedback policies for the parameters of a deep unfolded variant of the operator splitting QP (OSQP) solver (Stellato et al., 2020), which is a first-order method based on the alternating direction method of multipliers (ADMM) (Boyd et al., 2011). Their method can achieve orders-of-magnitude improvement in wall-clock time compared to OSQP, and they also propose a decentralized version for quickly solving QPs with distributed structure. Their idea is similar in vein to that of Ichnowski et al. (2021), who use reinforcement learning to train a policy that outputs the optimal parameters for OSQP, with the goals of accelerating the optimizer. Another related approach learns to warm-start a Douglas-Rachford splitting QP solver, with the goal of improving convergence speed (Sambharya et al., 2023).

3 FlexQP: An Always-Feasible Quadratic Program Solver

Our proposed QP optimizer, FlexQP, transforms the original QP constraints using an exact relaxation and then solves the resultant optimization using an operator splitting inspired by OSQP (Stellato et al., 2020). We will formalize what we mean by *exact* when referring to the relaxation of the constraints and prove that FlexQP solves the original QP, if it was feasible.

We will assume the reader is familiar with ADMM; a good overview is provided by Boyd et al. (2011).

3.1 Quadratic Programming

We are interested in solving QPs of the general form:

$$\text{minimize} \quad \frac{1}{2}x^\top Px + q^\top x, \tag{1a}$$

$$\text{subject to} \quad Gx \leq h, \quad Ax = b, \tag{1b}$$

where $x \in \mathbb{R}^n$ is the decision variable. The objective is defined by the symmetric positive semidefinite quadratic cost matrix $P \in \mathbb{S}^n_+$ and the linear cost vector $q \in \mathbb{R}^n$. The inequality constraints are defined by the matrix $G \in \mathbb{R}^{m\times n}$ and the vector $h \in \mathbb{R}^m$. Similarly, the equality constraints are defined by the matrix $A \in \mathbb{R}^{p\times n}$ and vector $b \in \mathbb{R}^p$.

Notably, throughout this work we will avoid making any assumptions on the rank of G or A, meaning that the constraints may be redundant, and in the worst case, there may not exist a feasible x for the optimization. This allows for a robust way to handle infeasibilities when optimizations of the form Eq. (1) are embedded as a subproblem in, e.g., SQP.

3.2 Elastic Formulation

To make the inequality constraints easier to optimize, we start by expressing Eq. (1) in the equivalent form

$$\text{minimize} \quad \frac{1}{2}x^\top Px + q^\top x, \tag{2a}$$

$$\text{subject to} \quad Gx + s - h = 0, \quad Ax - b = 0, \tag{2b}$$

$$s \geq 0, \tag{2c}$$

by introducing slack variables $s \in \mathbb{R}^m$. By Eq. (2b), $s = -(Gx - h) \geq 0$, thus recovering the original constraint $Gx - h \leq 0$.

Next, we relax the set of equality constraints Eq. (2b) using ℓ_1 penalty functions:

$$\text{minimize} \quad \frac{1}{2}x^\top Px + q^\top x + \mu_I \left\|Gx + s - h\right\|_1 + \mu_E \left\|Ax - b\right\|_1, \tag{3a}$$

$$\text{subject to} \quad s \geq 0, \tag{3b}$$

with elastic penalty parameters $\mu_I, \mu_E > 0$. This relaxation approach is known as *elastic programming* (Brown & Graves, 1975), and one of the most well-known SQP-based solvers, SNOPT, uses this technique in order to reduce the infeasibility of a QP subproblem (Gill et al., 2005). This relaxation is also a fundamental step in the Sℓ_1QP method of Fletcher (1985). Notably, if Eq. (1) has a feasible solution and the elastic penalty parameters are sufficiently large, then the solutions to Eq. (1) and Eq. (3) are identical — this is why the relaxation is "exact." On the other hand, if the original QP Eq. (1) is infeasible, then solving Eq. (3) finds a point that minimizes the constraint violation (Nocedal & Wright, 2006). This is formalized through the following theorem, which also describes what we mean by a "sufficiently large" penalty parameter.

Theorem 3.1. *Let (x^*, y_I^*, y_E^*) solve Eq.* (1)*. Let $\mu_I^* = \left\|y_I^*\right\|_\infty$ and $\mu_E^* = \left\|y_E^*\right\|_\infty$. Then, for all $\mu_I \geq \mu_I^*$ and $\mu_E \geq \mu_E^*$, the minimizers of Eq.* (1) *and Eq.* (3) *coincide.*

This theorem is a generalization of prior results (Han & Mangasarian, 1979) that shows we can select a different penalty parameter for the inequality vs. equality constraints. The proof, provided in Appendix A relies on two simple facts: the optimality conditions of Eq. (1) and the convexity of the objective. The proof also illuminates the fact that it is possible to select a *vector* of penalty parameters, as long as each obeys the individual constraint $\mu_i \geq |y_i|$.

What happens when the penalty parameters μ do not satisfy the conditions of Theorem 3.1? Using the dual interpretation of the Lagrange multipliers y_i representing a "cost" of a constraint i, if $\mu_i \geq y_i$ then the penalty on violating constraint i in Eq. (3) is large enough such that Theorem 3.1 holds. On the other hand, if $\mu_i < y_i$, then this constraint i is not being penalized strong enough and

so the solution to Eq. (3) will violate this constraint, with the amount of violation proportional to the difference between μ_i and y_i. We use this in Section 4 to design feedback policies that select the best penalty parameters as a function of the optimizer state and enforce the condition $\mu_i \geq y_i$ during learning using a supervised loss that includes the Lagrange multipliers (see also Theorem 3.2).

3.3 Operator Splitting and ADMM

The optimization in Eq. (3) has only simple bounds on the decision variables but is difficult to optimize due to the Gx and Ax terms appearing in the ℓ_1 penalties. Therefore, we express Eq. (3) in a simpler form

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^\top Px + q^\top x + \mu_I \left\|z_I\right\|_1 + \mu_E \left\|z_E\right\|_1, && \text{(4a)} \\ \text{subject to} \quad & z_I = Gx + s - h, \quad z_E = Ax - b, && \text{(4b)} \\ & s \geq 0, && \text{(4c)} \end{aligned}$$

by introducing extra decision variables $z_I \in \mathbb{R}^m$, $z_E \in \mathbb{R}^p$. These variables capture the constraint violation and can therefore be viewed as a certificate of feasibility if $z_I^* = z_E^* = 0$ and infeasibility if $z_I^* \neq 0$ or $z_E^* \neq 0$. While it may seem tempting to apply ADMM to this formulation, the resultant updates will not have a closed-form solution no matter how the variable splitting is performed. Therefore, we perform a final transformation for ADMM:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top \tilde{x} + \mathcal{I}_I(\tilde{x}, \tilde{s}, \tilde{z}_I) + \mathcal{I}_E(\tilde{x}, \tilde{z}_E) + \mathcal{I}_s(s) + \mu_I \left\|z_I\right\|_1 + \mu_E \left\|z_E\right\|_1, && \text{(5a)} \\ \text{subject to} \quad & (\tilde{x}, \tilde{s}, \tilde{z}_I, \tilde{z}_E) = (x, s, z_I, z_E), && \text{(5b)} \end{aligned}$$

where $\mathcal{I}_I$, $\mathcal{I}_E$, and $\mathcal{I}_s$ are the indicator functions

$$\mathcal{I}_I(x, s, z_I) = \begin{cases} 0 & z_I = Gx + s - h, \\ +\infty & \text{otherwise}, \end{cases} \qquad \mathcal{I}_E(x, z_E) = \begin{cases} 0 & z_E = Ax - b, \\ +\infty & \text{otherwise}, \end{cases} \tag{6a}$$

$$\mathcal{I}_s(s) = \begin{cases} 0 & s \geq 0, \\ +\infty & \text{otherwise}. \end{cases} \tag{6b}$$

The ADMM updates for solving Eq. (5) are given by

$$\begin{aligned} \tilde{x}^{k+1}, \tilde{s}^{k+1}, \tilde{z}_I^{k+1}, \tilde{z}_E^{k+1} = \underset{\tilde{x}, \tilde{s}, \tilde{z}_I, \tilde{z}_E}{\arg\min} \quad & \frac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top \tilde{x} + \mathcal{I}_I(\tilde{x}, \tilde{s}, \tilde{z}_I) + \mathcal{I}_E(\tilde{x}, \tilde{z}_E) && \text{(7a)} \\ & + \frac{\sigma_x}{2}\left\|\tilde{x} - x^k + \frac{w_x^k}{\sigma_x}\right\|_2^2 + \frac{\sigma_s}{2}\left\|\tilde{s} - s^k + \frac{w_s^k}{\sigma_s}\right\|_2^2 \\ & + \frac{\rho_I}{2}\left\|\tilde{z}_I - z_I^k + \frac{y_I^k}{\rho_I}\right\|_2^2 + \frac{\rho_E}{2}\left\|\tilde{z}_E - z_E^k + \frac{y_E^k}{\rho_E}\right\|_2^2, \\ x^{k+1} = \alpha\tilde{x}^{k+1} + (1-\alpha)x^k + \sigma_x^{-1}w_x^k, & && \text{(7b)} \\ s^{k+1} = \left(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k + \sigma_s^{-1}w_s^k\right)_+, & && \text{(7c)} \\ z_I^{k+1} = S_{\mu_I/\rho_I}\left(\alpha\tilde{z}_I^{k+1} + (1-\alpha)z_I^k + \rho_I^{-1}y_I^k\right), & && \text{(7d)} \\ z_E^{k+1} = S_{\mu_E/\rho_E}\left(\alpha\tilde{z}_E^{k+1} + (1-\alpha)z_E^k + \rho_E^{-1}y_E^k\right), & && \text{(7e)} \\ w_x^{k+1} = w_x^k + \sigma_x(\tilde{x}^{k+1} - x^{k+1}), & && \text{(7f)} \\ w_s^{k+1} = w_s^k + \sigma_s(\tilde{s}^{k+1} - s^{k+1}), & && \text{(7g)} \\ y_I^{k+1} = y_I^k + \rho_I(\tilde{z}_I^{k+1} - z_I^{k+1}), & && \text{(7h)} \\ y_E^{k+1} = y_E^k + \rho_E(\tilde{z}_E^{k+1} - z_E^{k+1}), & && \text{(7i)} \end{aligned}$$

where $\sigma_x, \sigma_s, \rho_I, \rho_E > 0$ are the augmented Lagrangian penalty parameters, $\alpha \in (0, 2)$ is the ADMM relaxation parameter, $(s)_+ = \max(s, 0)$ is the rectified linear unit (ReLU) activation function, and $S_\kappa(z) = (z - \kappa)_+ - (-z - \kappa)_+$ is the soft thresholding operator, which is the proximal operator of the ℓ_1 norm (Boyd et al., 2011). Note that by Eqs. (7b) and (7f), we have that $w_x^{k+1} = 0$

for all $k \geq 0$, so the w_x variable and the update Eq. (7f) can be disregarded. The first block update Eq. (7a) is the most computationally-demanding step of the algorithm and requires the solution of an equality-constrained QP. We show how to solve this QP using either a direct or indirect method in Appendix B; the indirect method becomes the only suitable choice for large-scale problems where the dimension can be very large. The final algorithm is summarized in Algorithm 1 of Appendix C.

The following theorem establishes the relationship between the FlexQP solution and the solution to the original QP and can be proven using the definition of soft thresholding (Appendix D).

Theorem 3.2. *For any $\mu_I > 0$ and $\mu_E > 0$, let $(\hat{x}, \hat{y}_I, \hat{y}_E)$ solve the relaxed QP Eq.* (3) *using Algorithm 1. Then, $\hat{y}_{I,i} \leq \mu_I$ and $\hat{y}_{E,i} \leq \mu_E$ for all constraints i. Furthermore, let (x^*, y_I^*, y_E^*) solve Eq.* (1) *if it is feasible. If the conditions of Theorem 3.1 hold, then $(\hat{x}, \hat{y}_I, \hat{y}_E) = (x^*, y_I^*, y_E^*)$. Otherwise, for any infeasible constraint i with associated dual variable y_i, the FlexQP solution satisfies $|\hat{y}_i| = \mu_i$.*

Finally, we summarize the key roles of the different hyperparameters of our algorithm. These insights are important for understanding the parameterization of our deep unfolded architecture presented in the next section.

Role of Elastic Penalty Parameters: The elastic penalty parameters μ_I and μ_E only appear in a single step of the algorithm during the Eqs. (7d) and (7e) as part of the soft thresholding in the second block ADMM updates. Larger elastic penalties μ result in a larger threshold, meaning that a larger amount of constraint violation will be zeroed out. The choice of μ is key for satisfying the conditions of Theorem 3.1.

Role of Augmented Lagrangian penalty parameters: The primary role of the parameter σ_x is to regularize the quadratic cost matrix P, and allows the equality-constrained QP Eq. (7a) to admit a unique solution even if P is not positive definite (P can even be zero to capture linear programs). We tested multiple fixed values of σ_x along with adaptive and learned rules, but a fixed $\sigma_x = 1\text{e}{-6}$ appears to work very well in practice. This is similar to the choice of the σ parameter in OSQP (Stellato et al., 2020).

The parameter σ_s plays the role of quadratic cost on the slack variable s when solving Eq. (7a). It also plays a small role in regularizing the constraint matrix G. In practice, tuning this parameter is the most difficult as the optimal value appears to depend strongly on the scaling of the objective and the constraints. This motivates our adaptive data-driven approach described in Section 4.

The penalty parameters ρ_I and ρ_E are perhaps the most important, as they play two key roles in the algorithm. First, they regularize the constraint matrices G and A so that Eq. (7a) is solvable regardless of the rank of G or A. Second, they weight the noise level in the soft thresholding operations Eqs. (7d) and (7e) playing an inverse role to μ_I and μ_E, where a larger ρ results in a smaller threshold. Determining the optimal values of these parameters by hand is unintuitive as they can have varying effects on the optimization, further motivating the deep unfolding approach presented in the next section.

4 Accelerating Quadratic Programming through Deep Unfolding

We focus our study on two recently proposed data-accelerated QP optimizers. The deep centralized QP optimizer from Saravanos et al. (2025) is a version of deep-unfolded OSQP where the penalty parameters ρ and relaxation parameter α are learned as feedback policies on the problem residuals using an analogy to feedback control. In our comparisons, we refer to their method as **Deep OSQP**. The main limitation of their approach is that only scalar penalty parameters are learned, but it could be the case that different penalty parameters should be applied to different constraints to more effectively accelerate the optimizer. This was the main motivation for the deep QP method proposed by Ichnowski et al. (2021), where a policy that outputs a vector of penalty parameters is learned using reinforcement learning. The vector policy is applied across the constraint dimensions so it is dimension-agnostic and generalizes across different problem classes. The authors show that the vector policy outperforms the scalar one across a suite of QP benchmarks. However, unlike Saravanos et al. (2025), the authors do not learn the relaxation parameter α, which can greatly improve the convergence of ADMM (Boyd et al., 2011). We implement the approach from Ichnowski et al.

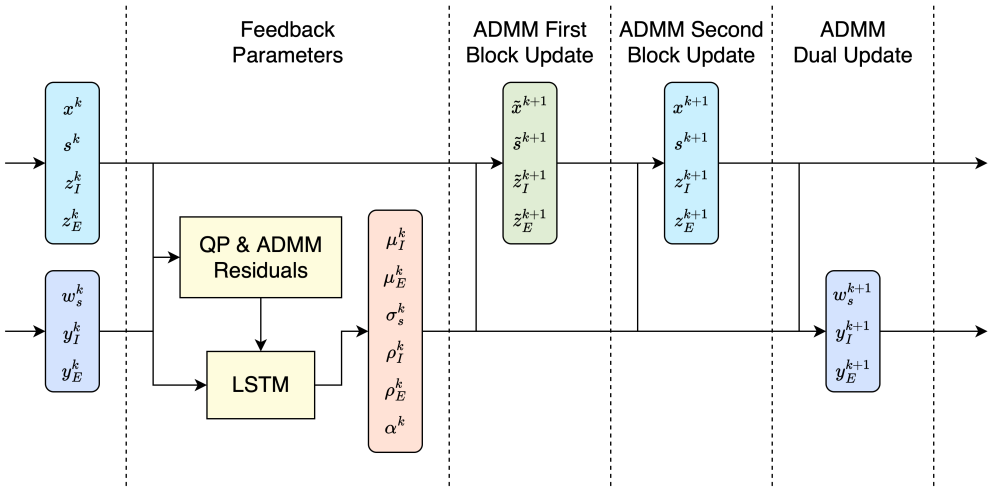


Figure 2: One layer of our proposed Deep FlexQP architecture. We learn dimension-agnostic feedback policies for the parameters while the propagation from one layer to the next is defined by the ADMM updates Eq. (7).

(2021) and train it using the supervised learning scheme from Saravanos et al. (2025), leading to the baseline **Deep OSQP — RLQP Parameterization**. Finally, we implement a "best-of-both-worlds" approach that learns a vector feedback policy for the penalty parameters ρ while also learning a policy for the ADMM relaxation parameter α, which we call **Deep OSQP — Improved**.

4.1 Deep FlexQP Architecture

Our proposed **Deep FlexQP** learns feedback policies for the algorithm parameters as a function of the current state of the optimizer as well as the QP and ADMM residuals, see Fig. 2. Based on the successes of the Deep OSQP methods discussed above, we learn separate policies π_I, π_E, and π_α for the parameters related to the inequality constraints, equality constraints, and the relaxation parameter, respectively. Furthermore, the π_I and π_E policies are designed so that the resultant architecture is independent of problem size and permutation by applying the policies in a batched fashion per constraint coefficient. Note that the variables s, z_I, w_s and y_I are one-to-one with μ_I, σ_s, and ρ_I, and that z_E and y_E are one-to-one with μ_E and ρ_E. We therefore use s, z_I, w_s, y_I along with their associated ADMM residuals, as well as the (relaxed) QP residual $Gx + s - h - z_I$ as inputs to the inequality policy π_I. We also include the infinity norm of the QP dual residual $\zeta_{\text{dual}} = Px + q + G^\top y_I + A^\top y_E$ as a scale-invariant measure of optimality. This leads to a total of ten inputs and three outputs corresponding to the coefficients of μ_I, σ_s, and ρ_I. Likewise, the equality constraint policy π_E learns μ_E and ρ_E as a function of the variables z_E and y_E, along with their associated ADMM residuals, the relaxed QP residual $Ax - b - z_E$, and the dual residual ζ_{dual}, leading to six total inputs. Finally, the policy π_α learns the relaxation parameter α as a function of the infinity norms of each of the QP and ADMM residuals, which provide a scale-invariant measure of how well and fast the optimizer is converging. Full expressions for the residuals and policies are given in Appendix E.

All policies are parameterized by long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997), with the hypothesis that learning long-term dependencies can aid the selection of the optimal parameters. This furthers the idea from Saravanos et al. (2025) that time-varying feedback on the current "nominal" parameters can provide a large improvement. In our case, we are applying feedback based on a latent state capturing the optimization history. Our results show that LSTMs provide the most benefit for problems where the active constraints might change many times over the course of the optimization, see Appendix J.

4.2 Supervised Learning

For training Deep OSQP variants, we adopt the supervised learning approach from Saravanos et al. (2025). The training loss is the weighted sum of the optimality gaps between the iterates x^k and the true optimal solution x^*:

$$\min_{\theta} \sum_{k=1}^{K} \gamma_k \left\| x^k(\theta) - x^* \right\|_2, \tag{8}$$

where $\gamma_k = \exp((k-K)/5)$ is a per-iteration scaling factor.

For training Deep FlexQP, we adopt a similar loss, but generalize it to incorporate the optimal Lagrange multipliers based on the discussion in Section 3. We also use the normalized optimality gaps instead of the unnormalized ones so that the scale is automatically determined based on the distance from the optimal solution:

$$\min_{\theta} \sum_{k=1}^{K} \left\| \xi^k(\theta) - \xi^* \right\|_2 / \left\| \xi^* \right\|_2, \tag{9}$$

where $\xi = (x, y_I, y_E)$. Jointly taking the norm over all the variables x, y_I, and y_E keeps the loss scale-invariant between the problem dimensions n, m, and p, as well as across the problem classes themselves. Furthermore, by including the Lagrange multipliers here, we are able to enforce the Deep FlexQP optimizer to select penalty parameters that meet the conditions of Theorem 3.1, namely that $\mu_I \geq \|y_I^*\|_\infty$ and $\mu_E \geq \|y_E^*\|_\infty$. This is due to the fact that the Lagrange multipliers of Deep FlexQP $y_I(\theta)$ and $y_E(\theta)$ are upper-bounded (in absolute value) by the current selection of μ (see Theorem 3.2). An ablation studying the effect of this loss is provided in Appendix K.

4.3 PAC-Bayes Generalization Bounds

Recent approaches have been proposed for establishing generalization bounds for guaranteeing the performance of learning-to-optimize methods, including a binary loss approach from Sambharya & Stellato (2025) as well as a more informative "progress" metric by Saravanos et al. (2025), given as:

$$\ell(\theta) = \min(\|x^K(\theta) - x^*\|_2 / \|x^0(\theta) - x^*\|_2, 1). \tag{10}$$

These approaches can be used to construct PAC-Bayes generalization bounds on the mean performance of the optimizer that hold with high probability. Nevertheless, a limitation of the resulting PAC-Bayes bound from Eq. (10) is that it assumes that the losses can fall anywhere within in the range $[0, 1]$, despite the fact that, in practice, most of the final optimality gaps fall very close to 0 (on the order of $1\mathrm{e}{-2}$ and smaller). In other words, the loss in Eq. (10) does not properly account for the scale of the errors, and as a result, obtaining a meaningful bound might require exponentially more training samples. For example, Fig. 4a shows that training for this generalization bound loss results in a bound that is uninformative since it sits above even the vanilla optimizers and does not capture the behavior well at very small errors.

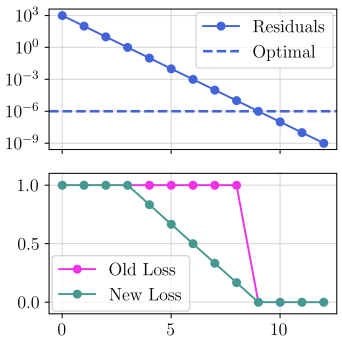
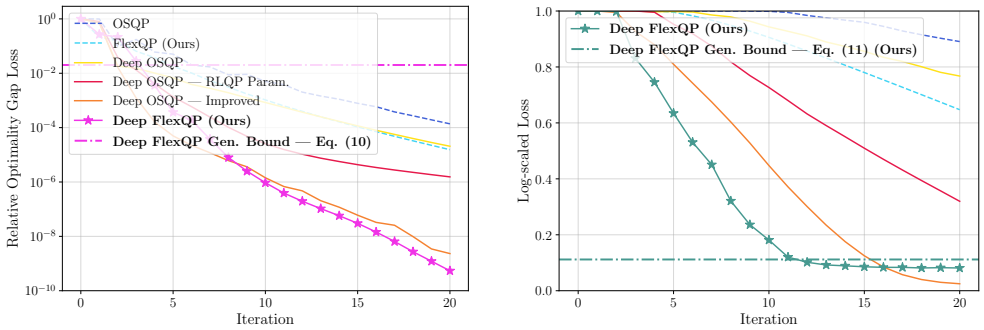


Figure 3: Log-scaled loss better captures small errors when the solution is close to the optimal.

To address this issue, we propose a log-scaled loss that is zero when the learner performs as well as or better than the optimal solution, and increases linearly as the performance decreases on a log-scale; see Fig. 3 for a visualization. Furthermore, we penalize distance from the optimal solution with respect to the norm of the QP residuals, which better takes into account the scale of the problem:

$$\ell(\theta) = \text{clip}(1 - \log\|r(\xi^K(\theta))\|_2 / \log\|r(\xi^*)\|_2, 0.0, 1.0), \tag{11}$$

where $\xi = (x, y_I, y_E)$ and $r(\xi) := (Px + q + G^\top y_I + A^\top y_E, \max(Gx - h, 0), Ax - b)$ computes the residuals of the original QP in Eq. (1). As intended, training for this loss better captures the performance when the residuals are very small (Fig. 4b). Further results are presented in Appendix I.



(a) Deep FlexQP trained for the generalization bound loss Eq. (10).

(b) Deep FlexQP trained for our proposed generalization bound loss Eq. (11).

Figure 4: Optimizer comparison on 1000 test LASSO problems. Training using our log-normalized loss Eq. (11) results in a substantially more informative performance guarantee.

5 Applications

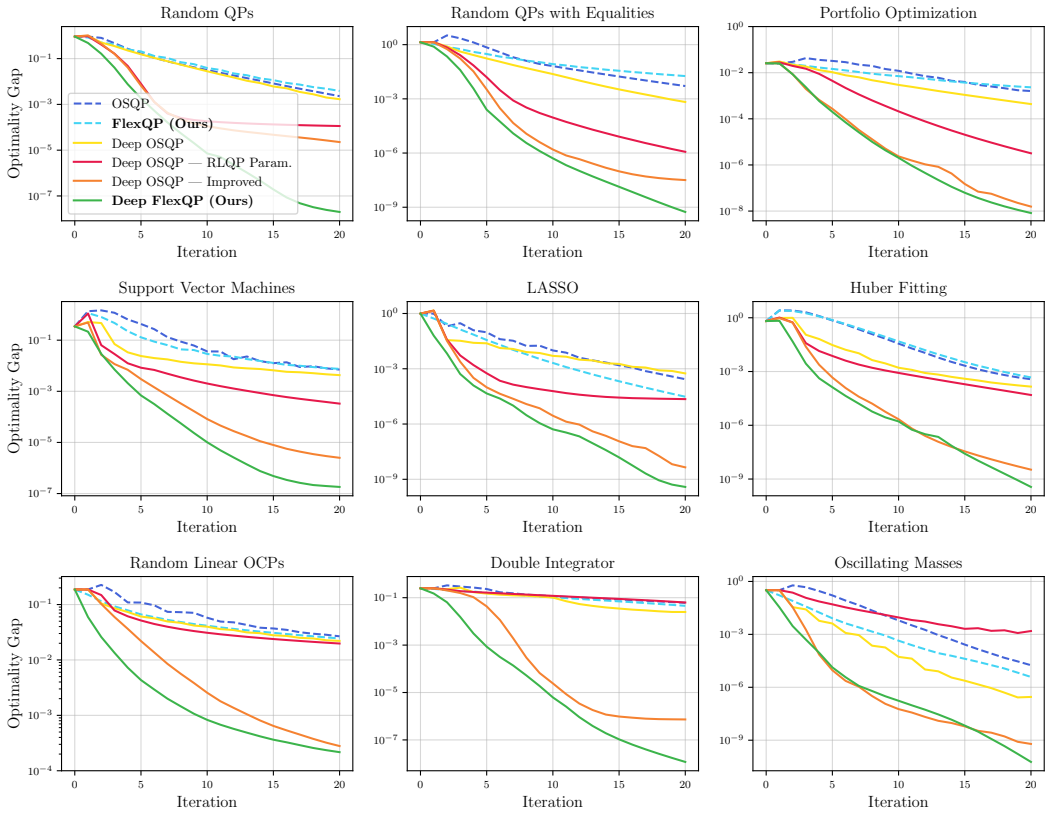


Figure 5: Performance comparison of learned deep optimizers and their non-learned counterparts. Our improved version of Deep OSQP outperforms the baselines, while Deep FlexQP consistently surpasses the rest of the methods in terms of convergence to the optimal QP solution.

We apply our deep-unfolded methodology to a benchmark suite of QPs adapted from Stellato et al. (2020) and Saravanos et al. (2025), with results presented in Fig. 5. These consist of various problem classes, including synthetic QPs that are feasible by construction, portfolio optimization problems from finance, classification and regression problems from machine learning, and linear optimal control problems. Details on the problem representations as well as the data generation processes are

provided in Appendix F. In all plots, **OSQP** is the best performing version of OSQP, found using a hyperparameter search over the following configurations: fixed parameters for all iterations, adaptive penalty parameters using the OSQP rule, or adaptive penalty parameters using the ADMM rule. Similarly, **FlexQP** is the best performing version of FlexQP, found using a hyperparameter search over the following configurations: fixed parameters for all iterations, adaptive penalty parameters using an OSQP-like rule, or adaptive penalty parameters using the ADMM rule. **Deep OSQP** is the approach from Saravanos et al. (2025), **Deep OSQP — RLQP Parameterization** is the parameterization from Ichnowski et al. (2021), and **Deep OSQP — Improved** is an improved "best-of-both-worlds" version of deep-unfolded OSQP, as described in Section 4. Finally, **Deep FlexQP** is our proposed deep-unfolded FlexQP optimizer with LSTM policy parameterization and trained using the loss Eq. (9). We also perform an extensive timing comparison and analysis between all of the above optimizers, presented in Appendix H.

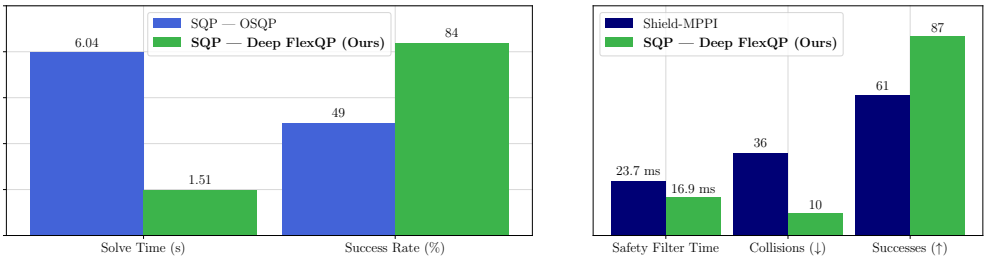


Figure 6: Comparison of our approach vs. traditional optimizer baselines on quadrotor trajectory optimization problems (left) and nonlinear predictive safety filter problems (right). Ours is faster than the baselines while vastly improving the task completion rate and safety.

Lastly, we use our Deep FlexQP optimizer as a submodule in SQP to solve two important classes of nonlinear optimizations. These are nonlinear optimal control problems and predictive safety filter problems generated using control barrier functions. Training for generalization bounds Eq. (11) yields a numerical certificate of performance that we use when designing the SQP method. The results are summarized in Fig. 1 and Fig. 6, with supplementary information provided in Appendix G.

6 Conclusion

We present FlexQP, a flexible QP solver that can solve any QP irregardless of any assumptions on the constraints. FlexQP always returns a solution that minimizes the constraint violation, and thus can be used as a robust QP solver in SQP. Our accelerated variant, Deep FlexQP, outperforms other traditional optimizers and learned approaches in terms of convergence, both in number of iterations and solve time. Generalization bounds provide a numerical certificate of performance, and we use these bounds to design SQP solvers for nonlinear optimal control and predictive safety filters. Using Deep FlexQP as a submodule in SQP provides a substantial speedup over traditional approaches, while also allowing for a graceful recovery when an infeasible QP subproblem is encountered. Some potential future extensions include learning warm-starts for our solver and applying it to the distributed QP setting explored in Saravanos et al. (2025).

References

Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, pp. 3420–3431. Ieee, 2019.

Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

Aleksandr Aravkin, James V Burke, Lennart Ljung, Aurelie Lozano, and Gianluigi Pillonetto. Generalized kalman smoothing: Modeling and algorithms. *Automatica*, 86:63–86, 2017.

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.

Stephen Boyd, Mark T Mueller, Brendan O'Donoghue, Yang Wang, et al. Performance bounds and suboptimal policies for multi–period investment. *Foundations and Trends® in Optimization*, 1 (1):1–72, 2013.

Stephen Boyd, Enzo Busseti, Steve Diamond, Ronald N Kahn, Kwangmoo Koh, Peter Nystrup, Jan Speth, et al. Multi-period trading via convex optimization. *Foundations and Trends® in Optimization*, 3(1):1–76, 2017.

Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

G Brown and G Graves. Elastic programming: a new approach to large-scale mixed integer optimization. In *ORSA/TIMS Conference, Las Vegas*, 1975.

Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd. Enhancing sparsity by reweighted $\ell 1$ minimization. *Journal of Fourier analysis and applications*, 14(5):877–905, 2008.

Steven W Chen, Tianyu Wang, Nikolay Atanasov, Vijay Kumar, and Manfred Morari. Large scale model predictive control with neural networks and primal active sets. *Automatica*, 135:109947, 2022a.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022b.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Brent De Weerdt, Yonina C. Eldar, and Nikos Deligiannis. Deep unfolding transformers for sparse recovery of video. *IEEE Transactions on Signal Processing*, 72:1782–1796, 2024. doi: 10.1109/TSP.2024.3381749.

Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In *Nonlinear model predictive control: towards new challenging applications*, pp. 391–417. Springer, 2009.

Liang Fang, Stefan Vandewalle, and Johan Meyers. An sqp-based multiple shooting algorithm for large-scale pde-constrained optimal control problems. *Journal of Computational Physics*, 477: 111927, 2023.

R Fletcher. An l1 penalty method for nonlinear constraints. *Numerical optimization*, 1984:26–40, 1985.

Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proc. International Conference on Machine learning (ICML'10)*, 2010.

S-P Han and Olvi L Mangasarian. Exact penalty functions in nonlinear programming. *Mathematical programming*, 17(1):251–269, 1979.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Peter J Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35 (1):73–101, 1964.

PJ Huber. Robust statistics. *Wiley series in probability and mathematical statistics*, 1981.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. *Advances in Neural Information Processing Systems*, 34:21043–21055, 2021.

Alexey F Izmailov and Mikhail V Solodov. Stabilized sqp revisited. *Mathematical programming*, 133(1):93–120, 2012.

Sven Leyffer. Integrating sqp and branch-and-bound for mixed integer nonlinear programming. *Computational optimization and applications*, 18(3):295–309, 2001.

Anirudha Majumdar, Alec Farid, and Anoopkumar Sonar. PAC-Bayes control: learning policies that provably generalize to novel environments. *The International Journal of Robotics Research*, 40 (2-3):574–593, 2021.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2975974.

Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

Oscar Danilo Montoya, Walter Gil-González, and Alejandro Garces. Sequential quadratic programming models for solving the opf problem in dc grids. *Electric Power Systems Research*, 169: 18–23, 2019.

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.

James Blake Rawlings, David Q Mayne, Moritz Diehl, et al. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2020.

Francesco Sabatino. Quadrotor control: modeling, nonlinearcontrol design, and simulation. 2015. URL https://api.semanticscholar.org/CorpusID:61413561.

Rajiv Sambharya and Bartolomeo Stellato. Data-driven performance guarantees for classical and learned optimizers. *Journal of Machine Learning Research*, 26(171):1–49, 2025.

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.

Augustinos D Saravanos, Hunter Kuperman, Alex Oshin, Arshiya Taj Abdul, Vincent Pacelli, and Evangelos Theodorou. Deep distributed optimization for large-scale quadratic programming. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=hzuumhfYSO.

Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Mobile Robots*, pp. 469–521. Springer London, London, 2009. ISBN 978-1-84628-642-1. doi: 10.1007/978-1-84628-642-1_11. URL https://doi.org/10.1007/978-1-84628-642-1_11.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, 2020.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

Kim Peter Wabersich and Melanie N Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129:109597, 2021.

Zhaowen Wang, Ding Liu, Jianchao Yang, Wei Han, and Thomas Huang. Deep networks for image super-resolution with sparse prior. In *Proceedings of the IEEE international conference on computer vision*, pp. 370–378, 2015.

Ji Yin, Charles Dawson, Chuchu Fan, and Panagiotis Tsiotras. Shield model predictive path integral: A computationally efficient robust mpc method using control barrier functions. *IEEE Robotics and Automation Letters*, 8(11):7106–7113, 2023.

A Proof of Theorem 3.1

First, we state an equivalent representation of the relaxed QP:

Lemma A.1. *The relaxed QP in Eq.* (3) *can equivalently be expressed as the following optimization:*

$$\min_x \quad \phi(x;\mu_I,\mu_E) := \frac{1}{2}x^\top Px + q^\top x + \mu_I \left\|(Gx-h)_+\right\|_1 + \mu_E \left\|Ax-b\right\|_1. \tag{12}$$

We will also require the optimality conditions for Eq. (1):

Lemma A.2. *Let* (x^*, y_I^*, y_E^*) *solve Eq.* (1)*. Then, the following conditions hold:*

$$Px^* + q + G^\top y_I^* + A^\top y_E^* = 0, \tag{13a}$$
$$y_{I,i}^*(g_i^\top x^* - h_i) = 0, \quad \forall i = 1,\dots,m, \tag{13b}$$
$$Ax^* - b = 0, \tag{13c}$$
$$Gx^* - h \le 0, \tag{13d}$$
$$y_I^* \ge 0. \tag{13e}$$

Proof Sketch. The sketch of the proof of Theorem 3.1 is as follows. We will show for any $\mu_I \ge \mu_I^* = \|y_I^*\|_\infty$ and for any $\mu_E \ge \mu_E^* = \|y_E^*\|_\infty$ that

1. if x^* solves Eq. (1), then x^* solves Eq. (12), and
2. if $\hat{x}$ solves Eq. (12) then $\hat{x}$ solves Eq. (1).

Step 1: Show that x^* solves Eq. (12). To start, for any $x \in \mathbb{R}^n$ we have that

$$\phi(x;\mu_I,\mu_E) = \frac{1}{2}x^\top Px + q^\top x + \mu_I \left\|(Gx-h)_+\right\|_1 + \mu_E \left\|Ax-b\right\|_1 \tag{14a}$$
$$= \frac{1}{2}x^\top Px + q^\top x + \mu_I \sum_{i=1}^m (g_i^\top x - h_i)_+ + \mu_E \sum_{i=1}^p |a_i^\top x - b_i| \tag{14b}$$
$$\ge \frac{1}{2}x^\top Px + q^\top x + \|y_I^*\|_\infty \sum_{i=1}^m (g_i^\top x - h_i)_+ + \|y_E^*\|_\infty \sum_{i=1}^p |a_i^\top x - b_i| \tag{14c}$$
$$\ge \frac{1}{2}x^\top Px + q^\top x + \sum_{i=1}^m y_{I,i}^*(g_i^\top x - h_i)_+ + \sum_{i=1}^p y_{E,i}^*|a_i^\top x - b_i| \tag{14d}$$
$$\ge \frac{1}{2}x^\top Px + q^\top x + \sum_{i=1}^m y_{I,i}^*(g_i^\top x - h_i) + \sum_{i=1}^p y_{E,i}^*(a_i^\top x - b_i) \tag{14e}$$
$$= \frac{1}{2}x^\top Px + q^\top x + \sum_{i=1}^m y_{I,i}^*(g_i^\top x^* - h_i + g_i^\top(x-x^*)) \tag{14f}$$
$$+ \sum_{i=1}^p y_{E,i}^*(a_i^\top x^* - b_i + a_i^\top(x-x^*))$$
$$= \frac{1}{2}x^\top Px + q^\top x + \sum_{i=1}^m y_{I,i}^* g_i^\top(x-x^*) + \sum_{i=1}^p y_{E,i}^* a_i^\top(x-x^*) \tag{14g}$$
$$= \frac{1}{2}x^\top Px + q^\top x + (G^\top y_I^* + A^\top y_E^*)^\top(x-x^*) \tag{14h}$$
$$= \frac{1}{2}x^\top Px + q^\top x - (Px^* + q)^\top(x-x^*) \tag{14i}$$
$$= \frac{1}{2}x^{*\top} Px^* + q^\top x^* + \frac{1}{2}(x-x^*)^\top P(x-x^*) \tag{14j}$$
$$\ge \frac{1}{2}x^{*\top} Px^* + q^\top x^* \tag{14k}$$

$$= \frac{1}{2}{x^*}^\top P x^* + q^\top x^* + \mu_I \left\|(Gx^* - h)_+\right\|_1 + \mu_E \left\|Ax^* - b\right\|_1 \tag{14l}$$

$$= \phi(x^*; \mu_I, \mu_E). \tag{14m}$$

The step from Eq. (14b) to Eq. (14c) follows from the assumption that $\mu_I \geq \|y_I^*\|_\infty$ and $\mu_E \geq \|y_E^*\|_\infty$. Going from Eq. (14c) to Eq. (14d) follows from $\|y_I^*\|_\infty \geq y_{I,i}^*$ and $(g_i^\top x - h_i)_+ \geq 0$ for all $i = 1, \ldots, m$, in addition to $\|y_E^*\|_\infty \geq y_{E,i}^*$ and $|a_i^\top x - b_i| \geq 0$ for all $i = 1, \ldots, p$. Obtaining Eq. (14g) follows from the conditions in Eq. (13) and in particular, $y_{I,i}^*(g_i^\top x_i^* - h_i) = 0$ for all $i = 1, \ldots, m$, and $a_i^\top x_i^* - b_i = 0$ for all $i = 1, \ldots, p$. Getting Eq. (14i) follows from Eq. (13a). Finally, obtaining Eq. (14k) follows from $(x - x^*)^\top P(x - x^*) \geq 0$ since $P \in \mathbb{S}_+^n$.

Thus, we have shown that $\phi(x^*; \mu_I, \mu_E) \leq \phi(x; \mu_I, \mu_E)$ for any x, which implies that x^* minimizes $\phi(x; \mu_I, \mu_E)$ and therefore solves Eq. (12).

Step 2: Show that $\hat{x}$ solves Eq. (1). Next, let $\hat{x}$ solve Eq. (12). If $x^* \neq \hat{x}$ solves Eq. (1), then we have that

$$\phi(\hat{x}; \mu_I, \mu_E) = \frac{1}{2}\hat{x}^\top P \hat{x} + q^\top \hat{x} + \mu_I \left\|(G\hat{x} - h)_+\right\|_1 + \mu_E \left\|A\hat{x} - b\right\|_1 \tag{15}$$

$$\leq \frac{1}{2}{x^*}^\top P x^* + q^\top x^* + \mu_I \left\|(Gx^* - h)_+\right\|_1 + \mu_E \left\|Ax^* - b\right\|_1 \tag{16}$$

$$= \frac{1}{2}{x^*}^\top P x^* + q^\top x^*. \tag{17}$$

Now, assume that $\hat{x}$ is not feasible for Eq. (1). Then

$$\frac{1}{2}\hat{x}^\top P \hat{x} + q^\top \hat{x} \geq \frac{1}{2}{x^*}^\top P x^* + q^\top x^* + (Px^* + q)^\top (\hat{x} - x^*) \tag{18}$$

$$= \frac{1}{2}{x^*}^\top P x^* + q^\top x^* - \sum_{i=1}^m y_{I,i}^* g_i^\top (\hat{x} - x^*) - \sum_{i=1}^p y_{E,i}^* a_i^\top (\hat{x} - x^*) \tag{19}$$

$$= \frac{1}{2}{x^*}^\top P x^* + q^\top x^* - \sum_{i=1}^m y_{I,i}^* (g_i^\top \hat{x} - h_i - (g_i^\top x^* - h_i)) - \sum_{i=1}^p y_{E,i}^* (a_i^\top \hat{x} - b_i - (a_i^\top x^* - b_i)) \tag{20}$$

$$= \frac{1}{2}{x^*}^\top P x^* + q^\top x^* - \sum_{i=1}^m y_{I,i}^* (g_i^\top \hat{x} - h_i) - \sum_{i=1}^p y_{E,i}^* (a_i^\top \hat{x} - b_i) \tag{21}$$

$$\geq \frac{1}{2}{x^*}^\top P x^* + q^\top x^* - \mu_I \sum_{i=1}^m g_i^\top \hat{x} - h_i - \mu_E \sum_{i=1}^p a_i^\top \hat{x} - b_i \tag{22}$$

$$\geq \frac{1}{2}{x^*}^\top P x^* + q^\top x^* - \mu_I \sum_{i=1}^m (g_i^\top \hat{x} - h_i)_+ - \mu_E \sum_{i=1}^p |a_i^\top \hat{x} - b_i|. \tag{23}$$

Rearranging, we have that

$$\frac{1}{2}\hat{x}^\top P \hat{x} + q^\top \hat{x} + \mu_I \left\|(G\hat{x} - h)_+\right\|_1 + \mu_E \left\|Ax - b\right\|_1 \geq \frac{1}{2}{x^*}^\top P x^* + q^\top x^*, \tag{24}$$

but either $\hat{x} = x^*$ or this contradicts the fact that $\hat{x}$ minimized $\phi(\cdot; \mu_I, \mu_E)$ in Eq. (17). Thus, $\hat{x}$ is feasible for Eq. (1).

Therefore, by Eq. (17) we have that

$$\frac{1}{2}\hat{x}^\top P \hat{x} + q^\top \hat{x} \leq \frac{1}{2}{x^*}^\top P x^* + q^\top x^*, \tag{25}$$

so $\hat{x}$ minimizes the objective Eq. (1a) and thus solves Eq. (1), completing the proof.

B FLEXQP — FIRST BLOCK ADMM UPDATE

The most computationally demanding step of FlexQP is the first block update Eq. (7a), which is an equality-constrained QP:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top\tilde{x} + (\sigma_x/2)\left\|\tilde{x} - x^k\right\|_2^2 + (\sigma_s/2)\left\|\tilde{s} - s^k + \sigma_s^{-1}w_s^k\right\|_2^2 && \text{(26a)} \\ & + (\rho_I/2)\left\|\tilde{z}_I - z_I^k + \rho_I^{-1}y_I^k\right\|_2^2 + (\rho_E/2)\left\|\tilde{z}_E - z_E^k + \rho_E^{-1}y_E^k\right\|_2^2, \\ \text{subject to} \quad & \tilde{z}_I = G\tilde{x} + \tilde{s} - h, && \text{(26b)} \\ & \tilde{z}_E = A\tilde{x} - b. && \text{(26c)} \end{aligned}$$

The optimality conditions for this QP are given by

$$\begin{aligned} P\tilde{x} + q + \sigma_x(\tilde{x} - x^k) + G^\top\tilde{\nu}_I + A^\top\tilde{\nu}_E &= 0, && \text{(27a)} \\ \sigma_s(\tilde{s} - s^k) + w_s^k + \tilde{\nu}_I &= 0, && \text{(27b)} \\ \rho_I(\tilde{z}_I - z_I^k) + y_I^k - \tilde{\nu}_I &= 0, && \text{(27c)} \\ \rho_E(\tilde{z}_E - z_E^k) + y_E^k - \tilde{\nu}_E &= 0, && \text{(27d)} \\ G\tilde{x} + \tilde{s} - \tilde{z}_I - h &= 0, && \text{(27e)} \\ A\tilde{x} - \tilde{z}_E - b &= 0, && \text{(27f)} \end{aligned}$$

where $\tilde{\nu}_I \in \mathbb{R}^m$ and $\tilde{\nu}_E \in \mathbb{R}^p$ are the Lagrange multipliers for the constraints Eq. (26b) and Eq. (26c), respectively. Solving this QP as-is would be expensive since it requires solving a linear system of size $n + 3m + 2p$. However, we can eliminate $\tilde{s}$, $\tilde{z}_I$, and $\tilde{z}_E$ using Eqs. (27b) to (27d) above, so the linear system simplifies to

$$\begin{bmatrix} P + \sigma_x I & G^\top & A^\top \\ G & -(\sigma_s^{-1} + \rho_I^{-1})I & 0 \\ A & 0 & -\rho_E^{-1}I \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{\nu}_I \\ \tilde{\nu}_E \end{bmatrix} = \begin{bmatrix} \sigma_x x^k - q \\ h - s^k + \sigma_s^{-1}w_s^k + z_I^k - \rho_I^{-1}y_I^k \\ b + z_E^k - \rho_E^{-1}y_E^k \end{bmatrix}, \tag{28}$$

with the eliminated variables recoverable using

$$\begin{aligned} \tilde{s} &= s^k - \sigma_s^{-1}w_s^k - \sigma_s^{-1}\tilde{\nu}_I, && \text{(29a)} \\ \tilde{z}_I &= z_I^k - \rho_I^{-1}y_I^k + \rho_I^{-1}\tilde{\nu}_I, && \text{(29b)} \\ \tilde{z}_E &= z_E^k - \rho_E^{-1}y_E^k + \rho_E^{-1}\tilde{\nu}_E. && \text{(29c)} \end{aligned}$$

The coefficient matrix in the linear system Eq. (28) is always full rank due to the positive parameters σ_x, σ_s, ρ_I, and ρ_E introduced through the ADMM splitting. This linear system can be solved using a direct method such as an $LDL^\top$ factorization requiring $O((n+m+p)^3)$ time, the same as OSQP using the direct method. On the other hand, for large-scale QPs, i.e., when $n + m + p$ is very large, factoring this matrix can be prohibitively expensive. In this case, we can use an indirect method to solve the reduced system

$$\begin{aligned} (P + \sigma_x I + \bar{G}^\top G + \bar{A}^\top A)\tilde{x} = \sigma_x x^k - q + \bar{G}^\top(h - s^k + \sigma_s^{-1}w_s^k + z_I^k - \rho_I^{-1}y_I^k) \\ + \bar{A}^\top(b + z_E^k - \rho_E^{-1}y_E^k), \end{aligned} \tag{30}$$

where $\bar{G} = (\sigma_s^{-1} + \rho_I^{-1})^{-1}G$ and $\bar{A} = \rho_E A$. This can be obtained by eliminating $\tilde{\nu}_I$ and $\tilde{\nu}_E$ from the linear system Eq. (28). These variables are recoverable using

$$\begin{aligned} \tilde{\nu}_I &= (\sigma_s^{-1} + \rho_I^{-1})^{-1}(G\tilde{x} + s^k - \sigma_s^{-1}w_s^k - z_I^k + \rho_I^{-1}y_I^k - h), && \text{(31a)} \\ \tilde{\nu}_E &= \rho_E(A\tilde{x} - z_E^k + \rho_E^{-1}y_E^k - b). && \text{(31b)} \end{aligned}$$

The coefficient matrix in Eq. (30) is always positive definite, so the linear system can be solved using an iterative algorithm such as the conjugate gradient (CG) method. The linear system is of size n, matching the complexity of OSQP using the indirect method. In this work, we consider a supervised learning setting where we will need to compute derivatives of the solution $\tilde{x}$ with respect to the parameters σ_x, ρ_I, etc. While each iteration of the CG method is very fast, it can require many iterations to converge to a low-error solution. It would be very inefficient to backpropagate through all these iterations of the CG method, the main issue being the high memory cost since

the entire compute graph needs to be stored and then differentiated through during the backward pass. We instead adopt the approach from Saravanos et al. (2025, Theorem 2) using differentiable optimization in order to compute these derivatives in a more efficient manner. In practice, this means we can compute the derivatives by solving a new linear system with the same coefficient matrix but different right-hand side during the backward pass.

C FLEXQP ALGORITHM

Algorithm 1: FlexQP

Input : Initialization $x^0, s^0, z_I^0, z_E^0, w_s^0, y_I^0, y_E^0$ and parameters $\mu_I, \mu_E, \sigma_x, \sigma_s, \rho_I, \rho_E > 0$
Output: Solution x^*, y_I^*, y_E^*
while *termination criterion not satisfied* **do**
$\tilde{x}^{k+1}, \tilde{\nu}_I^{k+1}, \tilde{\nu}_E^{k+1} \leftarrow$ Solve the linear system Eq. (28)
$\tilde{s}^{k+1} = s^k - \sigma_s^{-1} w_s^k - \sigma_s^{-1} \tilde{\nu}_I^{k+1}$
$\tilde{z}_I^{k+1} = z_I^k - \rho_I^{-1} y_I^k + \rho_I^{-1} \tilde{\nu}_I^{k+1}$
$\tilde{z}_E^{k+1} = z_E^k - \rho_E^{-1} y_E^k + \rho_E^{-1} \tilde{\nu}_E^{k+1}$
$x^{k+1} = \alpha \tilde{x}^{k+1} + (1-\alpha) x^k$
$s^{k+1} = \left(\alpha \tilde{s}^{k+1} + (1-\alpha) s^k + \sigma_s^{-1} w_s^k\right)_+$
$z_I^{k+1} = S_{\mu_I/\rho_I}\left(\alpha \tilde{z}_I^{k+1} + (1-\alpha) z_I^k + \rho_I^{-1} y_I^k\right)$
$z_E^{k+1} = S_{\mu_E/\rho_E}\left(\alpha \tilde{z}_E^{k+1} + (1-\alpha) z_E^k + \rho_E^{-1} y_E^k\right)$
$w_s^{k+1} = w_s^k + \sigma_s(\tilde{s}^{k+1} - s^{k+1})$
$y_I^{k+1} = y_I^k + \rho_I(\tilde{z}_I^{k+1} - z_I^{k+1})$
$y_E^{k+1} = y_E^k + \rho_E(\tilde{z}_E^{k+1} - z_E^{k+1})$
end

D PROOF OF THEOREM 3.2

The proof follows from the definition of the soft thresholding operator. First, we consider the z_I and z_E updates from Eq. (7d) and Eq. (7e) as well as the dual variable updates for y_I and y_E (Eq. (7h) and Eq. (7i)). Assume w.l.o.g. that $\alpha = 1$. These updates have the general form:

$$z^{k+1} = S_{\mu/\rho}\left(\tilde{z}^{k+1} + y^k/\rho\right), \tag{32}$$

$$y^{k+1} = y^k + \rho(\tilde{z}^{k+1} - z^{k+1}). \tag{33}$$

Now, there are three cases the consider based on the output of the soft thresholding operation:

1. **Positive constraint violation:** If $\tilde{z}^{k+1} + y^k/\rho > \mu/\rho$, then $z^{k+1} = \tilde{z}^{k+1} + y^k/\rho - \mu/\rho$. Substituting into Eq. (33) yields $y^{k+1} = \mu$.
2. **No constraint violation:** If $|\tilde{z}^{k+1} + y^k/\rho| \leq \mu/\rho$, then $z^{k+1} = 0$. This further implies $\rho|\tilde{z}^{k+1} + y^k/\rho| \leq \mu$ and by Eq. (33) this implies $|y^{k+1}| \leq \mu$.
3. **Negative constraint violation:** If $\tilde{z}^{k+1} + y^k/\rho < \mu/\rho$, then $z^{k+1} = \tilde{z}^{k+1} + y^k/\rho + \mu/\rho$. Substituting into Eq. (33) yields $y^{k+1} = -\mu$.

Combining these three cases, we have that $|y^{k+1}| \leq \mu$, for any $\tilde{z}^{k+1}, y^k$ and therefore $|\hat{y}| \leq \mu$. This proves the first and last statement of the theorem. Applying Theorem 3.1 shows the second statement.

E Deep FlexQP Policy Parameterization

The residuals for Eq. (4) are given by

$$\zeta_{\text{dual}} = Px + q + G^\top y_I + A^\top y_E, \tag{34a}$$
$$\zeta_I = Gx + s - h - z_I, \tag{34b}$$
$$\zeta_E = Ax - b - z_E. \tag{34c}$$

The ADMM residuals for Eq. (5) are given by

$$\bar{\zeta}_s = s^{\text{prev}} - s, \tag{35a}$$
$$\bar{\zeta}_I = z_I^{\text{prev}} - z_I, \tag{35b}$$
$$\bar{\zeta}_E = z_E^{\text{prev}} - z_E, \tag{35c}$$
$$\tilde{\zeta}_s = \tilde{s} - s, \tag{35d}$$
$$\tilde{\zeta}_I = \tilde{z}_I - z_I, \tag{35e}$$
$$\tilde{\zeta}_E = \tilde{z}_E - z_E, \tag{35f}$$

where "bar" quantities denote the ADMM dual residuals and "tilde" quantities denote the ADMM primal residuals (Boyd et al., 2011). We ignore the residuals corresponding to x since it is unconstrained in the second ADMM block (so the primal residual is not very meaningful) and we have already captured the dual optimality through Eq. (34).

The policy $\pi_I : \mathbb{R}^{10} \to \mathbb{R}^3_+$ is given by

$$\mu_I, \sigma_s, \rho_I = \pi_I(s, z_I, w_s, y_I, \|\zeta_{\text{dual}}\|_\infty, \zeta_I, \bar{\zeta}_s, \bar{\zeta}_I, \tilde{\zeta}_s, \tilde{\zeta}_I), \tag{36}$$

where we have dropped the index by constraint i for clarity.

The policy $\pi_E : \mathbb{R}^6 \to \mathbb{R}^2_+$ is given by

$$\mu_E, \rho_E = \pi_E(z_E, y_E, \|\zeta_{\text{dual}}\|_\infty, \zeta_E, \bar{\zeta}_E, \tilde{\zeta}_E). \tag{37}$$

The policy $\pi_\alpha : \mathbb{R}^9 \to (0, 2)$ is given by

$$\alpha = \pi_\alpha(\|\zeta_{\text{dual}}\|, \|\zeta_I\|, \|\zeta_E\|, \|\bar{\zeta}_s\|, \|\bar{\zeta}_I\|, \|\bar{\zeta}_E\|, \|\tilde{\zeta}_s\|, \|\tilde{\zeta}_I\|, \|\tilde{\zeta}_s\|), \tag{38}$$

where the norm used is the infinity norm.

Computationally, we log-scale any small positive inputs like the infinity norms of the residuals. Following Ichnowski et al. (2021), we also predict log-transformed values $\log \mu_I$, $\log \rho_E$, etc. and then apply an exponential function so that it is easier to predict parameters across a wide scale of values. We then clamp the parameters (besides α) to the range $[1\text{e}{-6}, 1\text{e}6]$; $\alpha \in (0, 2)$ is enforced using a scaled sigmoid function.

F Further Details on Problem Classes

A summary of the problem sizes and training parameters of the different classes is presented in Table 1. We train all models for 500 epochs and evaluate using 1000 test samples. More training samples are used for random QPs following the setup by Saravanos et al. (2025) since the shared structure between these QPs is less clear and therefore harder to learn. All experiments were run on a system with an Intel i9-13900K processor, 64GB of RAM, and an NVIDIA RTX 4090 GPU.

Table 1: QP problem sizes and number of samples used for training.

Problem Class	n	m	p	Train Samples
Random QPs	50	40	0	2000
Random QPs with Equalities	50	25	20	2000
Portfolio Optimization	275	250	26	500
Support Vector Machine	210	400	0	500
LASSO	510	10	500	500
Huber Fitting	310	200	100	500
Random Linear OCPs	128	256	88	500
Double Integrator	62	124	42	500
Oscillating Masses	162	324	132	500
Car with Obstacles (SQP)	253	455	153	500
Quadrotor (SQP)	812	400	612	500
Car Safety Filter (SQP)	253	50	153	500

F.1 Random QPs

The first type of problems we study are random QPs of the form Eq. (1). These are helpful as we can freely adjust the number of constraints as well as the sparsity of the problem directly in order to benchmark the optimizers under different operating conditions.

Problem Instances: We adopt the problem generation procedure from Saravanos et al. (2025), where $P = M^\top M + \alpha I$ with $\alpha = 1$ and all elements of M, q, G, and A are standard normal distributed, i.e., each element $M_{ij}, q_i, G_{ij}, A_{ij} \sim \mathcal{N}(0, 1)$. The vectors h and b are generated using $h = G\xi$ and $b = A\zeta$ with ξ, ζ standard normal vectors.

We consider two classes of random QPs. The first class, **Random QPs**, contains only inequality constraints generated by setting the problem dimensions as $n = 50$, $m = 40$, and $p = 0$. The second class, **Random QPs with Equalities** contains a mix of inequality and equality constraints, and is generated using $n = 50$, $m = 25$, and $p = 20$.

F.2 Portfolio Optimization

Portfolio Optimization is a foundational problem in finance where the goal is to maximize the risk-adjusted return of a group of assets (Markowitz, 1952; Boyd et al., 2013; 2017). This can be represented as the following QP (Boyd & Vandenberghe, 2004; Stellato et al., 2020):

$$\max_{x} \quad \mu^\top x - \gamma(x^\top \Sigma x), \tag{39a}$$

$$\text{subject to} \quad \mathbf{1}^\top x = 1, \tag{39b}$$

$$x \geq 0, \tag{39c}$$

where $x \in \mathbb{R}^n$ is the portfolio, $\mu \in \mathbb{R}^n$ is the expected returns, $\gamma > 0$ is the risk aversion parameter, and $\Sigma \in \mathbb{S}^n_+$ is the risk model covariance.

QP Representation: We assume that $\Sigma = FF^\top + D$ where $F \in \mathbb{R}^{n \times k}$ is the rank-k factor loading matrix with $k < n$ and $D \in \mathbb{R}^{n \times n}$ is the diagonal matrix specifying the asset-specific risk. Using this assumption, the optimization problem can be converted into a more efficient QP representation:

$$\min_{x,y} \quad x^\top D x + y^\top y - \gamma^{-1} \mu^\top x, \tag{40a}$$

$$\text{subject to} \quad y = F^\top x, \tag{40b}$$
$$\mathbf{1}^\top x = 1, \tag{40c}$$
$$x \geq 0. \tag{40d}$$

This new QP has $n + k$ decision variables, $k + 1$ equality constraints, and n inequality constraints.

Problem Instances: We use the problem generation described in Saravanos et al. (2025), setting $n = 250$, $k = 25$, and $\gamma = 1.0$. The expected returns μ are sampled using $\mu_i \sim \mathcal{N}(0, 1)$. The factor loading matrix F has 50% non-zero elements sampled through $F_{ij} \sim \mathcal{N}(0, 1)$. The diagonal elements of D are generated as $D_{ii} \sim \mathcal{U}[0, \sqrt{k}]$.

F.3 Support Vector Machines

Support Vector Machines (SVMs) is a classical machine learning problem where the goal is to find a linear classifier that best separates two sets of points (Cortes & Vapnik, 1995):

$$\min_x x^\top x + \lambda \sum_{i=1}^{m} \max(0, b_i a_i^\top x + 1), \tag{41}$$

where $b_i \in \{-1, 1\}$ is the label and $a_i \in \mathbb{R}^n$ is the set of features for point i.

QP representation: The SVM problem Eq. (41) can be converted into an equivalent QP representation (Stellato et al., 2020):

$$\min_{x,t} \quad x^\top x + \lambda \mathbf{1}^\top t, \tag{42a}$$
$$\text{subject to} \quad t \geq \text{diag}(b) A x + 1, \tag{42b}$$
$$t \geq 0. \tag{42c}$$

This QP has $n + m$ decision variables and $2m$ inequalty constraints.

Problem Instances: We generate random problems using the rules from Stellato et al. (2020) with $n = 10$ features and $m = 200$ data points. The labels b are chosen using

$$b_i = \begin{cases} +1 & \text{if } i \leq m/2, \\ -1 & \text{otherwise,} \end{cases} \tag{43}$$

and the elements of A are chosen such that

$$A_{ij} \sim \begin{cases} \mathcal{N}(+1/n, 1/n) & \text{if } i \leq m/2, \\ \mathcal{N}(-1/n, 1/n) & \text{otherwise.} \end{cases} \tag{44}$$

F.4 LASSO

LASSO (least absolute shrinkage and selection operator) is a fundamental problem in statistics and machine learning (Tibshirani, 1996; Candes et al., 2008). The objective is to select sparse coefficients of a linear model that best match the given observations:

$$\min_x \|Ax - b\|_2^2 + \lambda \|x\|_1, \tag{45}$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ is the data matrix, $b \in \mathbb{R}^m$ are the observations, and $\lambda > 0$ is the weighting parameter.

QP Representation: LASSO can be represented as a QP by introducing two extra decision variables $y \in \mathbb{R}^m$ and $t \in \mathbb{R}^n$ which help simplify the objective (Stellato et al., 2020):

$$\min_{x,y,t} \quad y^\top y + \lambda \mathbf{1}^\top t, \tag{46a}$$
$$\text{subject to} \quad y = Ax - b, \tag{46b}$$
$$-t \leq x \leq t. \tag{46c}$$

Problem Instances: We use the data generation procedure from (Stellato et al., 2020), where A has 15% non-zero normally-distributed elements $A_{ij} \sim \mathcal{N}(0,1)$ and b is generated through $b = Av + \epsilon$ with

$$v_i \sim \begin{cases} 0 & \text{with probability } p = 0.5, \\ \mathcal{N}(0, 1/n) & \text{otherwise,} \end{cases} \tag{47}$$

and $\epsilon_i \sim \mathcal{N}(0,1)$. The parameter λ is chosen as $\lambda = (1/5)\|A^\top b\|_\infty$.

F.5 Huber Fitting

Huber Fitting is a robust least squares problem where the goal is to perform a linear regression with the assumption that outliers are present in the data (Huber, 1964; 1981):

$$\min_x \sum_{i=1}^m \phi_{\text{hub}}(a_i^\top x - b_i), \tag{48}$$

where the penalty function ϕ_{hub} penalizes the residuals quadratically when they are large and linearly when they are small:

$$\phi_{\text{hub}}(u) = \begin{cases} u^2 & \text{if } |u| \le \delta, \\ \delta(2|u| - \delta) & \text{if } |u| > \delta, \end{cases} \tag{49}$$

with $\delta > 0$ representing the slope of the linear term.

QP Representation: This robust least squares problem can be represented in the following QP form (Stellato et al., 2020):

$$\begin{aligned} \min_{x,u,r,s} \quad & u^\top u + 2\delta \mathbf{1}^\top (r + s), && \text{(50a)} \\ \text{subject to} \quad & Ax - b - u = r - s, && \text{(50b)} \\ & r, s \ge 0. && \text{(50c)} \end{aligned}$$

This QP has $n + 3m$ decision variables, $2m$ inequalities, and m equalities.

Problem Instances: We follow Stellato et al. (2020) and generate A with 15% nonzero elements with $A_{ij} \sim \mathcal{N}(0,1)$ and set $b = Av + \epsilon$ where

$$\epsilon_i = \begin{cases} \mathcal{N}(0, 1/4) & \text{with probability } p = 0.95, \\ \mathcal{U}[0, 10] & \text{otherwise.} \end{cases} \tag{51}$$

We let $M = 1$ and choose the problem dimensions as $n = 10$ features and $m = 10n = 100$ datapoints.

F.6 Linear Optimal Control

The goal in linear optimal control is to stabilize the system to the origin subject to dynamical constraints as well as polyhedral constraints on the states and controls.

$$\begin{aligned} \min_{x,u} \quad & \sum_{t=0}^{T-1} x_t^\top Q x_t + u_t^\top R u_t + x_T^\top Q_T x_T, && \text{(52a)} \\ \text{subject to} \quad & x_{t+1} = A_d x_t + B_d u_t, && \text{(52b)} \\ & A_u u_t \le b_u, && \text{(52c)} \\ & A_x x_t \le b_x, && \text{(52d)} \\ & x_0 = \bar{x}_0, && \text{(52e)} \end{aligned}$$

where $T > 0$ is the time horizon, $Q \in \mathbb{S}_+^{n_x}$ is the running state cost matrix, $R \in \mathbb{S}_{++}^{n_u}$ is the control cost matrix, $Q_T \in \mathbb{S}_+^{n_x}$ is the terminal state cost matrix, $A_d \in \mathbb{R}^{n_x \times n_x}$ and $B_d \in \mathbb{R}^{n_x \times n_u}$ define the dynamics of the system, $A_u \in \mathbb{R}^{m_u \times n_u}$ and $b_u \in \mathbb{R}^{m_u}$ define the polyhedral input constraints, $A_x \in \mathbb{R}^{m_x \times n_x}$ and $b_x \in \mathbb{R}^{m_x}$ define the polyhedral state constraints, and $\bar{x}_0 \in \mathbb{R}^{n_x}$ is the initial state.

We study three classes of linear optimal control problems (OCPs). The first, **Random Linear OCPs**, consists of randomly generated stabilizable dynamics along with random costs, constraints, and initial conditions. The second and third classes, **Double Integrator** and **Oscillating Masses**, are adapted from Chen et al. (2022a) and contain dynamics with true physical interpretations. The randomness in these problems is given by sampling varying initial conditions for the systems as in Saravanos et al. (2025).

F.6.1 Random Linear OCPs

We use the problem generation procedure similar to that in Stellato et al. (2020). We set the state dimension $n_x = 8$ and $n_u = 0.5n_x$. The dynamics are generated by $A_d = X^{-1}AX$, where $A = \text{diag}(a) \in \mathbb{R}^{n_x \times n_x}$ such that $a_i \sim \mathcal{U}(-1, 1)$ and $X \in \mathbb{R}^{n_x \times n_x}$ with elements generated by $X_{ij} \sim \mathcal{N}(0, 1)$, and $B_d \in \mathbb{R}^{n_x \times n_u}$ with $(B_d)_{ij} \sim \mathcal{N}(0, 1)$.

The running state cost $Q \in \mathbb{S}_+^{n_x}$ is generated by $Q = \text{diag}(q)$ where each element of the sparse vector q is generated by

$$q_i \sim \begin{cases} \mathcal{U}[0, 10] & \text{with probability } p = 0.7, \\ 0 & \text{otherwise,} \end{cases} \tag{53}$$

so that q has 70% nonzero values. We fix the control cost $R = 0.1I_u$ and the terminal cost Q_T is determined by solving the discrete algebraic Riccati for the optimal cost of a linear quadratic regulator applied to A, B, Q, and R. The state and control constraints are generated by

$$A_x = \begin{bmatrix} I_x \\ -I_x \end{bmatrix}, b_x = \begin{bmatrix} x^{\text{bound}} \\ -x^{\text{bound}} \end{bmatrix} \text{ s.t. } x_i^{\text{bound}} \sim \mathcal{U}(1, 2), \tag{54a}$$

$$A_u = \begin{bmatrix} I_u \\ -I_u \end{bmatrix}, b_u = \begin{bmatrix} u^{\text{bound}} \\ -u^{\text{bound}} \end{bmatrix} \text{ s.t. } u_i^{\text{bound}} \sim \mathcal{U}(0, 0.1). \tag{54b}$$

Note that we use I_x and I_u as a shorthand for I_{n_x} and I_{n_u}. Finally, we sample the initial state from $\bar{x}_0 \sim \mathcal{U}[-0.5x^{\text{bound}}, 0.5x^{\text{bound}}]$.

F.6.2 Double Integrator

For the double integrator, adapted from Chen et al. (2022a), we have $n_x = 2$, $n_u = 1$, and $T = 20$ timesteps. The dynamics are fixed with

$$A_d = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, B_d = \begin{bmatrix} 0.5 \\ 0.1 \end{bmatrix}. \tag{55}$$

We use cost matrices $Q = Q_T = I_x$ and $R = 1.0$. The state and control constraints have

$$A_x = \begin{bmatrix} I_x \\ -I_x \end{bmatrix}, b_x = \begin{bmatrix} 5 \\ 1 \\ 5 \\ 1 \end{bmatrix}, A_u = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, b_u = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}. \tag{56}$$

The initial state is sampled from $\bar{x}_0 \sim \mathcal{U}\left(\begin{bmatrix} -1 \\ -0.3 \end{bmatrix}, \begin{bmatrix} 1 \\ 0.3 \end{bmatrix}\right)$.

F.6.3 Oscillating Masses

For the oscillating masses problem, we have $n_x = 12$, $n_u = 3$, $T = 10$. For this problem, the discrete time dynamics matrices A_d and B_d are obtained through the Euler discretization of the continuous time dynamics of the oscillating masses system. That is,

$$A_d = I_x + A_c\Delta t, B_d = B_c\Delta t, \tag{57}$$

where $A_c \in \mathbb{R}^{n_x \times n_x}$ and $B_c \in \mathbb{R}^{n_x \times n_u}$ are the continuous-time dynamics matrices. We use the discretization $\Delta t = 0.5$. These dynamics are given by

$$A_c = \begin{bmatrix} 0_{6\times 6} & I_6 \\ aI_6 + c(L_6 + L_6^\top) & bI_6 + d(L_6 + L_6^\top) \end{bmatrix}, B_c = \begin{bmatrix} 0_{6\times 3} \\ F \end{bmatrix}, \tag{58}$$

where $c = 1$, $d = 0.1$, $a = -2c$, $b = 2$, $0_{m\times n}$ is the zero matrix in $\mathbb{R}^{m\times n}$, L_n is the lower shift matrix in $\mathbb{R}^{n\times n}$, and $F = [e_1 \quad -e_1 \quad e_2 \quad e_3 \quad -e_2 \quad e_3]^\top$, where e_1, e_2, and e_3 are the standard basis vectors in $\mathbb{R}^3$. We use cost matrices $Q = Q_T = I_x$ and $R = I_u$. The state and control constraints are given by

$$A_x = \begin{bmatrix} I_x \\ -I_x \end{bmatrix}, b_x = 4 \cdot \mathbf{1}_x, A_u = \begin{bmatrix} I_u \\ -I_u \end{bmatrix}, b_u = 0.5 \cdot \mathbf{1}_u. \tag{59}$$

Finally, we sample the initial state from $\bar{x}_0 \sim \mathcal{U}[-\mathbf{1}_x, \mathbf{1}_x]$.

G NONLINEAR OPTIMIZATIONS USING SQP

G.1 NONLINEAR OPTIMAL CONTROL

We consider nonlinear constrained optimal control problems of the following form:

$$\begin{aligned}
\text{minimize} \quad & \sum_{k=0}^{N-1} \ell(x_k, u_k) + \phi(x_N), && \text{(60a)}\\
\text{subject to} \quad & x_{k+1} = F(x_k, u_k), \quad \forall k = 0, \dots, N-1, && \text{(60b)}\\
& x_0 = \bar{x}_0, && \text{(60c)}\\
& h(x_k) \leq 0, \quad \forall k = 0, \dots, N, && \text{(60d)}\\
& g(u_k) \leq 0, \quad \forall k = 0, \dots, N-1, && \text{(60e)}
\end{aligned}$$

where $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ are the state and controls. The function $\ell(x_k, u_k) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the running cost and $\phi(x_N) \in \mathbb{R}^n \to \mathbb{R}$ is the terminal cost. The time horizon and initial condition are denoted as N and $\bar{x}_0$. The problem formulation in 60 includes control and state constraints as represented by the functions $h(x_k)$ and $g(u_k)$. In the next two subsections, we provide the specific nonlinear optimal control examples for the cases of the Dubins vehicle and quadrotor.

G.1.1 DUBINS VEHICLE

The Dubins vehicle is a dynamics model with a state $x = (p_x, p_y, \theta) \in \mathbb{R}^3$, where p_x and p_y are the vehicle's position in the Cartesian plane and θ is its orientation. We use the unicycle formulation of the continuous-time Dubins vehicle dynamics $\dot{x} = f(x, u)$ adapted from Siciliano et al. (2009), where the control is given by $u = (v, \omega) \in \mathbb{R}^2$. Here, v is the forward velocity of the vehicle and ω is the steering velocity of the vehicle.

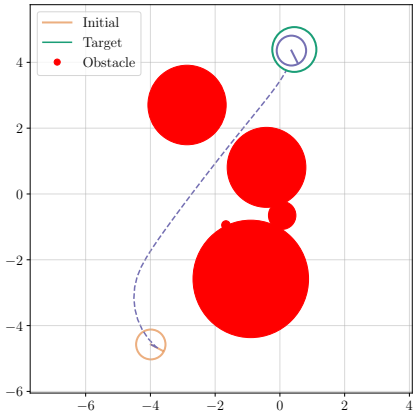


Figure 7: Visualization of a sample Dubins vehicle task. The goal is to reach the target state while avoiding obstacles and respecting the dynamics and input constraints.

We formulate a nonlinear optimal control problem following Eq. (60). We discretize the continuous-time dynamics using the Euler discretization $x_{k+1} = F(x_k, u_k) = x_k + f(x_k, u_k)\Delta t$ and use costs $\ell(x_k, u_k) = x_k^\top Q x_k + u_k^\top R u_k, \phi(x_N) = x_N^\top Q_N x_N$, where $Q = \text{diag}(1.0, 1.0, 0.1)$, $R = 0.1 \cdot I$, and $Q_N = 100 \cdot Q$. The initial and target state, x_0 and x_{target}, are sampled uniformly from $\mathcal{U}[-\bar{x}, \bar{x}]$ where $\bar{x} = (5.0, 5.0, \pi)$. The discretization of the dynamics uses $\Delta t = 0.033$ and the time horizon for trajectory optimization is $N = 50$ timesteps. We sample 5 circular obstacles uniformly at random in the region between the vehicle's initial and target positions, each with radius chosen uniformly from $\mathcal{U}[r_{\min}, r_{\max}]$ where $r_{\min} = 0.01 \cdot \|x_{\text{target}} - x_0\|_2$ and $r_{\max} = 0.2 \cdot \|x_{\text{target}} - x_0\|_2$. Each circular obstacle is then included in the optimization problem as a constraint given by

$$g_i(x_k) = r_i^2 - \|x_k - c_i\|_2^2 \leq 0, \quad \forall k = 0, \dots, N \tag{61}$$

where $r_i \sim \mathcal{U}[r_{\min}, r_{\max}]$ and $c_i \in \mathbb{R}^2$ are the radius and position of the center, respectively, of the i^{th} obstacle. The controls are constrained by $v \in [-10, 10]$ and $\omega \in [-5, 5]$. This leads to

a nonlinear optimization problem with 253 variables, 455 inequality constraints, and 153 equality constraints.

For generating the QP training data, we generate 500 QP subproblems by solving randomly generated Dubins vehicle problems with SQP using OSQP as the QP solver. For evaluation, we generate 100 random Dubins vehicle problems and solve them using SQP with OSQP or SQP with Deep FlexQP. Each algorithm is allowed 50 SQP iterations. Furthermore, each QP solver runs until convergence of $1\mathrm{e}{-3}$ is reached, with a max budget of 10 seconds and an unlimited number of iterations.

G.1.2 QUADROTOR

The quadrotor dynamics model consists of a state $x \in \mathbb{R}^{12}$ including the linear positions, angles, linear velocities, and angular velocities. The system is actuated by a four controls: collective thrust F and three torques, given by the vector $u = \begin{bmatrix} F & \tau_x & \tau_y & \tau_z \end{bmatrix}^\top \in \mathbb{R}^3$. We use the continuous-time dynamics model $\dot{x} = f(x, u)$ adapted from Sabatino (2015).

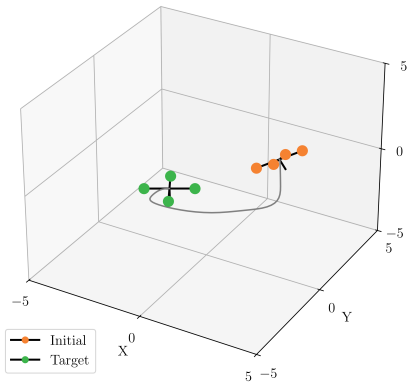


Figure 8: Visualization of a sample quadrotor task. The goal is to reach the target state from the initial state subject to dynamical constraints and input constraints.

Using the quadrotor dynamics model, we formulate a nonlinear optimal control problems as in Eq. (60). We discretize the dynamics through the Euler discretization $x_{k+1} = F(x_k, u_k) = x_k + f(x_k, u_k)\Delta t$. The cost in Eq. (60) is defined by $\ell(x_k, u_k) = x_k^\top Q x_k + u_k^\top R u_k$ and $\phi(x_N) = x_N^\top Q_N x_N$, where the cost matrices are given by

$$Q = \mathrm{diag}(1.0, 1.0, 1.0, 0.1, 0.1, 0.1, 1.0, 1.0, 1.0, 0.1, 0.1, 0.1), \tag{62a}$$

$R = 0.01 \cdot I$, and $Q_N = 1000 \cdot Q$. The initial and target state are sampled uniformly from $\mathcal{U}[-\bar{x}, \bar{x}]$ where $\bar{x} = (5, 5, 5, 1, 1, 1, \pi, \pi/2, \pi, 1, 1, 1)$. The discretization of the dynamics uses $\Delta t = 0.05$ and the time horizon for trajectory optimization is $N = 50$ timesteps. The controls are constrained in the range $F \in [0, 20]$ and $\tau_x, \tau_y, \tau_z \in [-10, 10]$. We use $m = 1.0$ kg for the mass of the quadrotor, $I_x = I_z = I_y = 1.0$ for its moments, and $g = 9.81$ for the acceleration due to gravity. This leads to a nonlinear optimization problem with 812 variables, 400 inequality constraints, and 612 equality constraints.

For generating the QP training data, we generate 500 QP subproblems by solving randomly generated quadrotor problems with SQP using OSQP as the QP solver. For evaluation, we generate 100 random quadrotor problems and solve them using SQP with OSQP or SQP with Deep FlexQP. Each algorithm is allowed 50 SQP iterations and success in Fig. 6 (left) is achieved when the norm of the SQP residuals falls below $1\mathrm{e}{-2}$. Each QP solver runs until convergence of $1\mathrm{e}{-3}$ is reached, with a max budget of 10 seconds and an unlimited number of iterations.

G.2 NONLINEAR PREDICTIVE SAFETY FILTERS

Finally, we apply our proposed approach to accelerate a predictive safety filter for nonlinear model predictive control. These methods are based on control barrier functions (CBFs) (Ames et al., 2019)

and filter a reference control u^{ref} so that it better respects safety constraints (Wabersich & Zeilinger, 2021). The following optimization is solved at every MPC step:

$$\begin{aligned}
\text{minimize} \quad & \sum_{k=0}^{N-1} \left\| u_k - u_k^{\text{ref}} \right\|_2^2, && \text{(63a)} \\
\text{subject to} \quad & x_{k+1} = F(x_k, u_k), \quad \forall k = 0, \ldots, N-1, && \text{(63b)} \\
& x_0 = \bar{x}_0 && \text{(63c)} \\
& (1-\beta)h(x_k) - h(x_{k+1}) \leq 0, \quad \forall k = 0, \ldots, N-1, && \text{(63d)} \\
& g(u_k) \leq 0, \quad \forall k = 0, \ldots, N-1, && \text{(63e)}
\end{aligned}$$

where $\beta \in (0, 1)$ is a parameter controlling the strength of the CBF constraint. This differs from Eq. (60) because the discrete CBF constraint is defined between two consecutive states x_k and x_{k+1} rather than assuming the state constraints are separable across time. Our method improves upon the Shield-MPPI method proposed by Yin et al. (2023) because our optimization explicitly incorporates the dynamics and input constraints while also minimizing the discrepancy from the reference control trajectory. Furthermore, using our accelerated Deep FlexQP ensures that the optimization can be run fast enough for real-time control. We use a version of Deep FlexQP with performance guarantees from minimizing the generalization bound loss (see Appendix I). Sample trajectories of our approach compared with Shield-MPPI are shown in Fig. 9.

G.2.1 SHIELD-MPPI

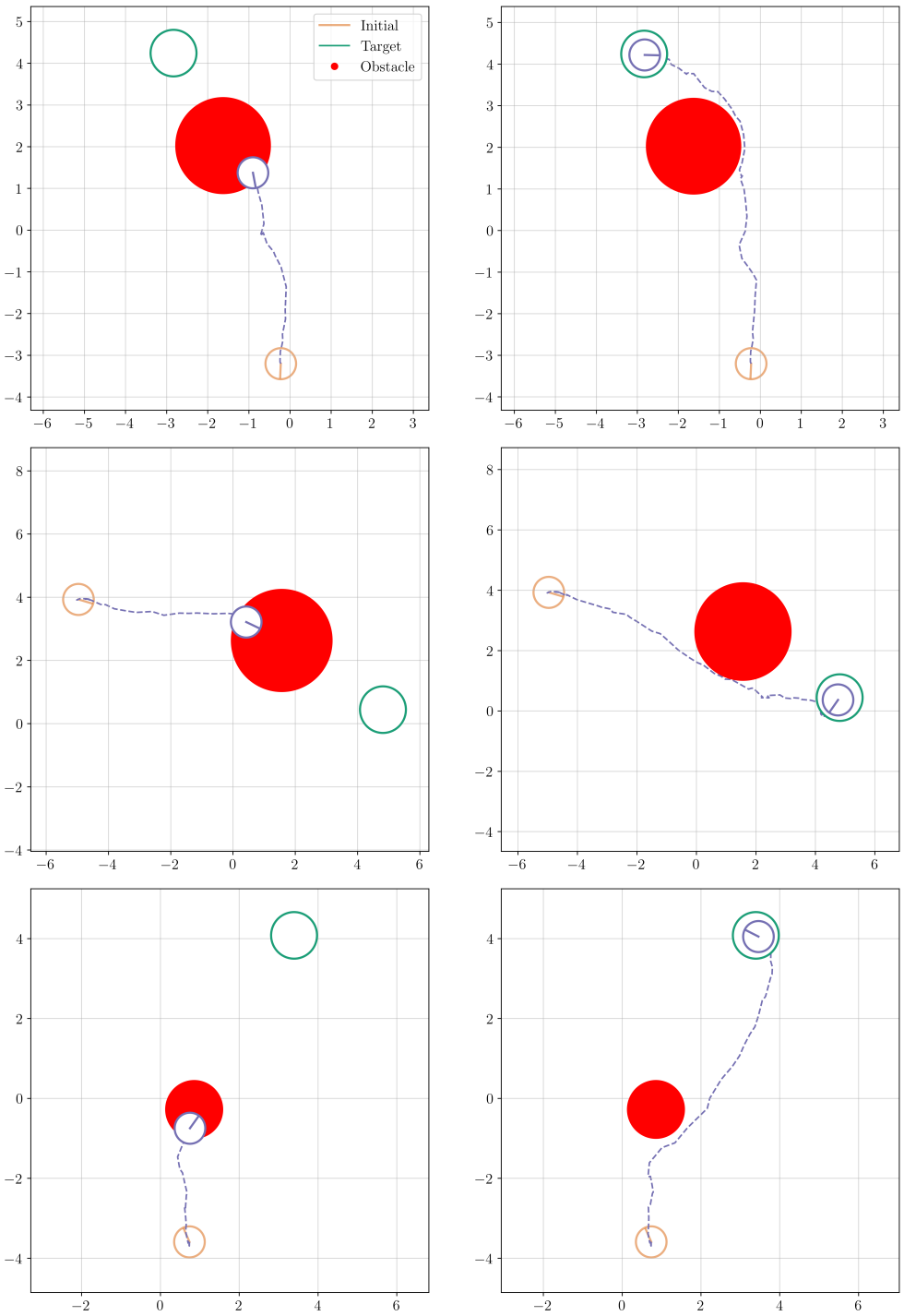
The method by Yin et al. (2023) approximately solves a nonlinear optimization at every MPC step to generate safe controls given a trajectory from a high-level planner such as a model predictive path integral (MPPI) controller. While the main motivation behind this approach is that it is computationally fast, unfortunately, there are a few flaws in that the method has no real guarantees of safety and that the MPPI trajectory is only used to warm-start this second optimization. The main bottleneck preventing us from solving a more complex optimization in real-time is the solver speed. Therefore, this is an application where accelerating optimizers using deep unfolding can shine.

G.2.2 RANDOMIZED PROBLEM SCENARIOS

100 random scenarios are generated by first sampling a random initial and target state uniformly from $\mathcal{U}[(-5, -5, -\pi), (5, 5, \pi)]$. An obstacle is randomly sampled so its position falls between the initial and target state with a random radius r depending on the distance between the initial and target state: $r \sim \mathcal{U}[0.01, 2] * \max(|p_x^{\text{target}} - p_x^{\text{init}}|, |p_y^{\text{target}} - p_y^{\text{init}}|)$. The controls are constrained to be bounded in the interval $[(-10, -5), (10, 5)]$ enforced by clamping for Shield-MPPI and through the constraints of Eq. (63) for our SQP-based method. Fig. 9 shows the problem setup and sample trajectories for each algorithm.

The reference trajectory at every MPC step is given by running an MPPI controller that samples 10000 trajectories with a look-ahead horizon of 50 timesteps; with a dynamics discretization of $\Delta t = 0.05$, this corresponds to a planning horizon of 2.5 seconds ahead. The system experiences zero-mean Gaussian disturbances in its state at every MPC step with standard deviation $(0.05, 0.05, 0.01)$. Shield-MPPI is allowed to run up to 5 Gauss-Newton iterations per MPC step, while our SQP safety filter is allowed to run up to 5 SQP iterations per MPC step. These thresholds were determined by estimating the max number of iterations that would still allow for real-time control of the system.

Collisions in Fig. 6 (right) are counted if the state violates the CBF constraint (i.e., intersects the obstacle). Successes are counted if the vehicle reaches within a 0.1 radius of the target state.



(a) Shield-MPPI (b) SQP Safety Filter — Deep FlexQP

Figure 9: Sample trajectories comparing safety filter approaches on a nonlinear car system. The vehicle receives disturbances in the positions and orientation at every step. Our approach more effectively recovers from unsafe scenarios by better accounting for dynamic feasibility and constraints.

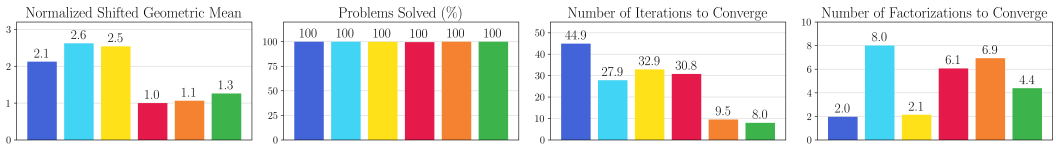
H Learned Optimizer Performance Comparisons

The vanilla and learned optimizers are benchmarked on 1000 test QPs from each problem class with the results summarized in Fig. 10. Problems are considered solved when the infinity norm of the QP residuals reaches below an absolute tolerance of $\varepsilon = 10^{-3}$. Optimizers are run with no limit on the max number of iterations until a timeout of 1 second (1000 ms) is reached. Timings are compared using the normalized shifted geometric mean, which is the factor at which a specific solver is slower than the fastest one (Stellato et al., 2020). We also compare the average number of iterations required to converge as well as the number of coefficient matrix factorizations required to converge to get a sense of where the optimizers are spending the most time. All methods use the direct method to solve their respective linear systems (i.e., equality-constrained QPs) at every iteration, and the factorization from the previous iteration is reused if the parameters have not changed by more than a factor of 5x following the heuristic used by OSQP (Stellato et al., 2020).

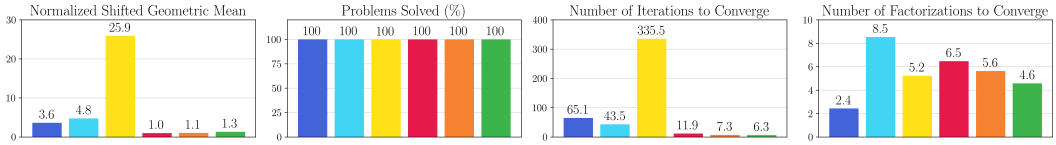
Key Takeaways:

1. Deep FlexQP and Deep OSQP — Improved solve QPs 2-5x faster than OSQP.
2. Deep FlexQP and Deep OSQP — Improved require upwards of 10x less iterations to converge than OSQP and require a comparable amount of matrix factorizations.
3. Deep OSQP — RLQP Parameterization struggles on problems with optimal control structure. This is not observed with Deep OSQP — Improved. Learning the ADMM relaxation parameter α seems to be crucial for these problems.

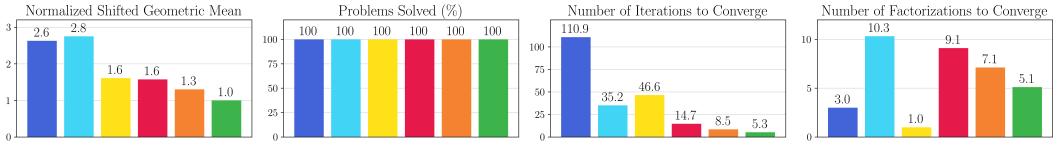
It is important to note that these results hold only when the direct method is used to solve the linear system. When using an indirect method such as the conjugate gradient (CG) method, converging in fewer iterations is actually a major benefit as each iteration across all the optimizers have roughly the same computational complexity (see Appendix B for discussion). Thus, we hypothesize that when using the indirect method, Deep FlexQP will substantially outperform OSQP.



(a) Random QPs

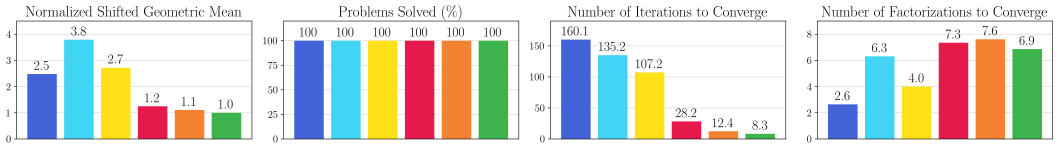


(b) Random QPs with Equalities

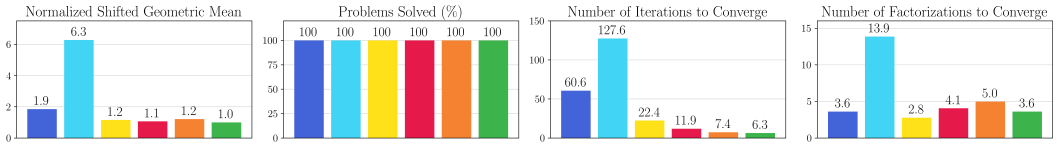


(c) Portfolio Optimization

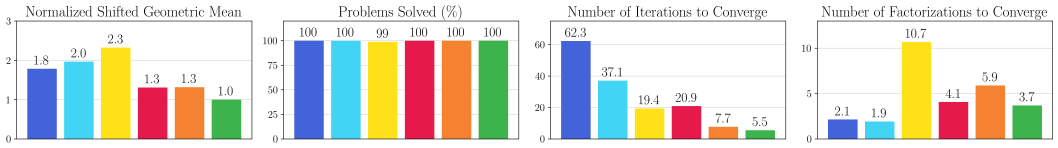
Figure 10: Performance comparison of vanilla vs. learned optimizers. Legend: OSQP, **FlexQP (Ours)**, Deep OSQP, Deep OSQP — RLQP Parameterization, Deep OSQP — Improved, **Deep FlexQP (Ours)**.



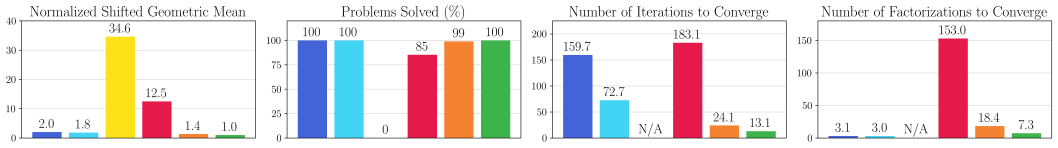
(d) Support Vector Machines



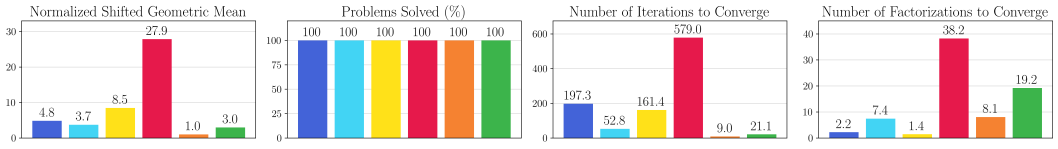
(e) LASSO



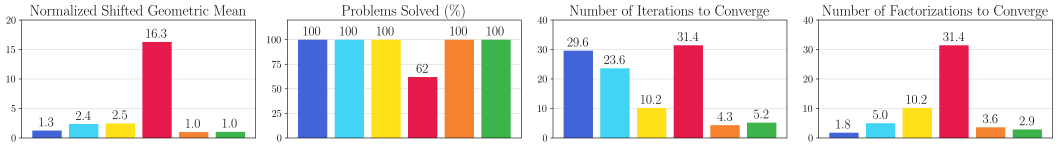
(f) Huber Fitting



(g) Random Linear OCPs



(h) Double Integrator



(i) Oscillating Masses

Figure 10: Performance comparison of vanilla vs. learned optimizers. Legend: OSQP, **FlexQP (Ours)**, Deep OSQP, Deep OSQP — RLQP Parameterization, Deep OSQP — Improved, **Deep FlexQP (Ours)**.

I SUPPLEMENTARY RESULTS FOR GENERALIZATION BOUNDS

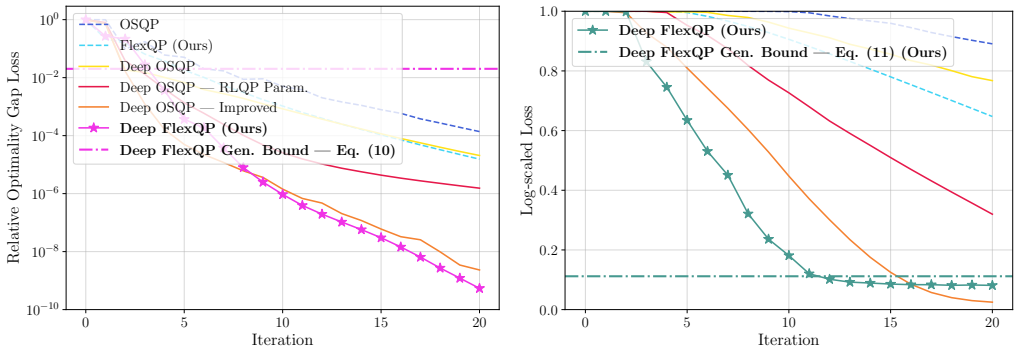


Figure 11: Generalization bounds for Deep FlexQP trained on 500 oscillating masses QPs. Following Fig. 4, this is another case where the generalization bound using the loss Eq. (10) is uninformative.

We provide an overview of the generalization bounds training procedure described by Saravanos et al. (2025), which in turn is adapted from the one described by Majumdar et al. (2021). Let $\boldsymbol{x} = (P, q, G, h, A, b, x^*, y_I^*, y_E^*) \sim \mathcal{D}$ denote a single sample from data distribution $\mathcal{D}$ and let $\mathcal{S} = \{\boldsymbol{x}_i\}_{i=1}^N$ be a dataset of N samples. Let $\ell(\boldsymbol{x}, \theta) \in [0, 1]$ be a bounded loss for hypothesis $\theta \sim \mathcal{P}$. The true expected loss is defined as

$$\ell_{\mathcal{D}}(\mathcal{P}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\mathbb{E}_{\theta\sim\mathcal{P}}[\ell(\boldsymbol{x}, \theta)], \tag{64}$$

and the sample loss is

$$\ell_{\mathcal{S}}(\mathcal{P}) = \mathbb{E}_{\theta\sim\mathcal{P}}\left[\frac{1}{N}\sum_{i=1}^{N}\ell(\boldsymbol{x}_i, \theta)\right]. \tag{65}$$

We rely on the following PAC-Bayes bounds that hold with probability $1-\delta$ (Majumdar et al., 2021, Corollary 1):

$$\ell_{\mathcal{D}}(\mathcal{P}) \leq \mathbb{D}^{-1}\left(\ell_{\mathcal{S}}(\mathcal{P})||\frac{\mathbb{D}_{KL}(\mathcal{P}||\mathcal{P}_0) + \log(2\sqrt{N}/\delta)}{N}\right) \leq \ell_{\mathcal{S}}(\mathcal{P}) + \sqrt{\frac{\mathbb{D}_{KL}(\mathcal{P}||\mathcal{P}_0) + \log(2\sqrt{N}/\delta)}{2N}}, \tag{66}$$

where $\mathbb{D}^{-1}(p||c) = \sup\{q \in [0,1] | \mathbb{D}_{KL}(\mathcal{B}(p)||\mathcal{B}(q) \leq c\}$ is the inverse KL-divergence for Bernoulli random variables $\mathcal{B}(p)$ and $\mathcal{B}(q)$. The first bound is tighter and is therefore useful for computing the generalization bounds as a numerical certificate of performance, while the second bound has the nice interpretation of a training loss plus a regularization term that depends on the size of the training set and penalizes being off from the prior $\mathcal{P}_0$. During training, we minimize the second bound using either the loss Eq. (10) or Eq. (11).

During the evaluation of the tighter generalization bound (after training is complete), since it is difficult to compute the expectation over $\theta \sim \mathcal{P}$ in Eq. (65), we instead estimate the sample loss using a large number of samples $\{\theta_j\}_{j=1}^M$ from $\mathcal{P}^*$:

$$\hat{\ell}_{\mathcal{S}}(\mathcal{P}^*) = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\ell(\boldsymbol{x}_i, \theta_j). \tag{67}$$

The following sample convergence bound holds with probability $1-\delta'$:

$$\bar{\ell}_{\mathcal{S}}(\mathcal{P}^*) \leq \mathbb{D}^{-1}\left(\hat{\ell}_{\mathcal{S}}(\mathcal{P}^*)||\frac{1}{M}\log(\frac{2}{\delta'})\right). \tag{68}$$

Combining these bounds results in a final version of the PAC-Bayes bound that holds with probability $1 - \delta - \delta'$ (Majumdar et al., 2021):

$$\ell_{\mathcal{D}}(\mathcal{P}^*) \leq \mathbb{D}^{-1}\left(\bar{\ell}_{\mathcal{S}}(\mathcal{P}^*)||\frac{\mathbb{D}_{KL}(\mathcal{P}^*||\mathcal{P}_0) + \log(2\sqrt{N}/\delta)}{N}\right). \tag{69}$$

This is the final bound that we report in our experiments.

The prior $\mathcal{P}_0$ for all our models is a stochastic Deep FlexQP trained for 500 epochs on 500 QPs generated from the **Random QPs with Equalities** problem class using the supervised learning setup from Section 4. We train Deep FlexQP for the generalization bound loss using either Eq. (10) or Eq. (11) with $\delta = 0.009$. We fix a training set for the generalization bounds using 500 problems from the class of interest and train the model for 1000 epochs. We evaluate Eq. (69) using $M = 10000$ model samples and $\delta' = 0.001$. Our bounds therefore hold with 99% probability. We report results comparing Eq. (10) vs. Eq. (11) for **LASSO** in Fig. 4 and for **Oscillating Masses** in Fig. 11. The test loss is computed over 1000 new samples from the target problem class. Overall, the loss Eq. (10) results in a less informative bound as it is above all the optimizers, even though the performance in practice can be much better.

Finally, we train Deep FlexQP on 500 QP subproblems generated from a nonlinear predictive safety filter task described in Appendix G using the same procedure to minimize the generalization bound through the loss defined in Eq. (11). We use this Deep FlexQP as the QP solver for the predictive safety filter SQP approach described in Appendix G.2. This provides a numerical certificate on the performance that would not hold for a traditional optimizer.

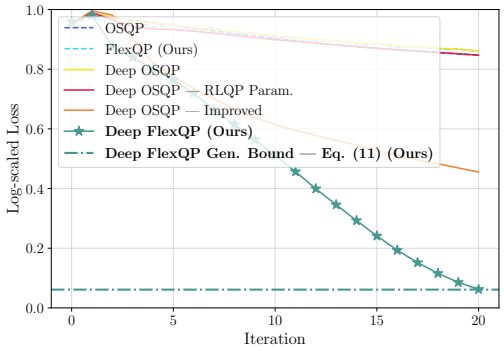


Figure 12: Generalization bound for Deep FlexQP trained on 500 QP subproblems generated from the nonlinear predictive safety filter task.

J LSTM VS. MLP POLICY COMPARISON

This section presents an ablation analysis on the use of LSTMs to parameterize the policies in both Deep OSQP and Deep FlexQP. The use of LSTMs further leverages the analogy between deep-unfolded optimizers and RNNs, as discussed in Monga et al. (2021). Our hypothesis is that the RNN hidden state can encode a compressed history or context from the past optimization variables and residuals, thereby leading to a better prediction of the algorithm parameters to apply at the current iteration. Using an MLP only provides access to information from the current iterate, which could lead to a myopic choice of parameters.

Our results show that LSTMs enhance performance on several problem classes (Fig. 13). LSTMs appear to help the most on problems where the active constraints might switch suddenly from one iteration to the next. These types of problems include the machine learning ones, such as SVM, LASSO, and Huber fitting, as well as some of the control problems, like the oscillating masses.

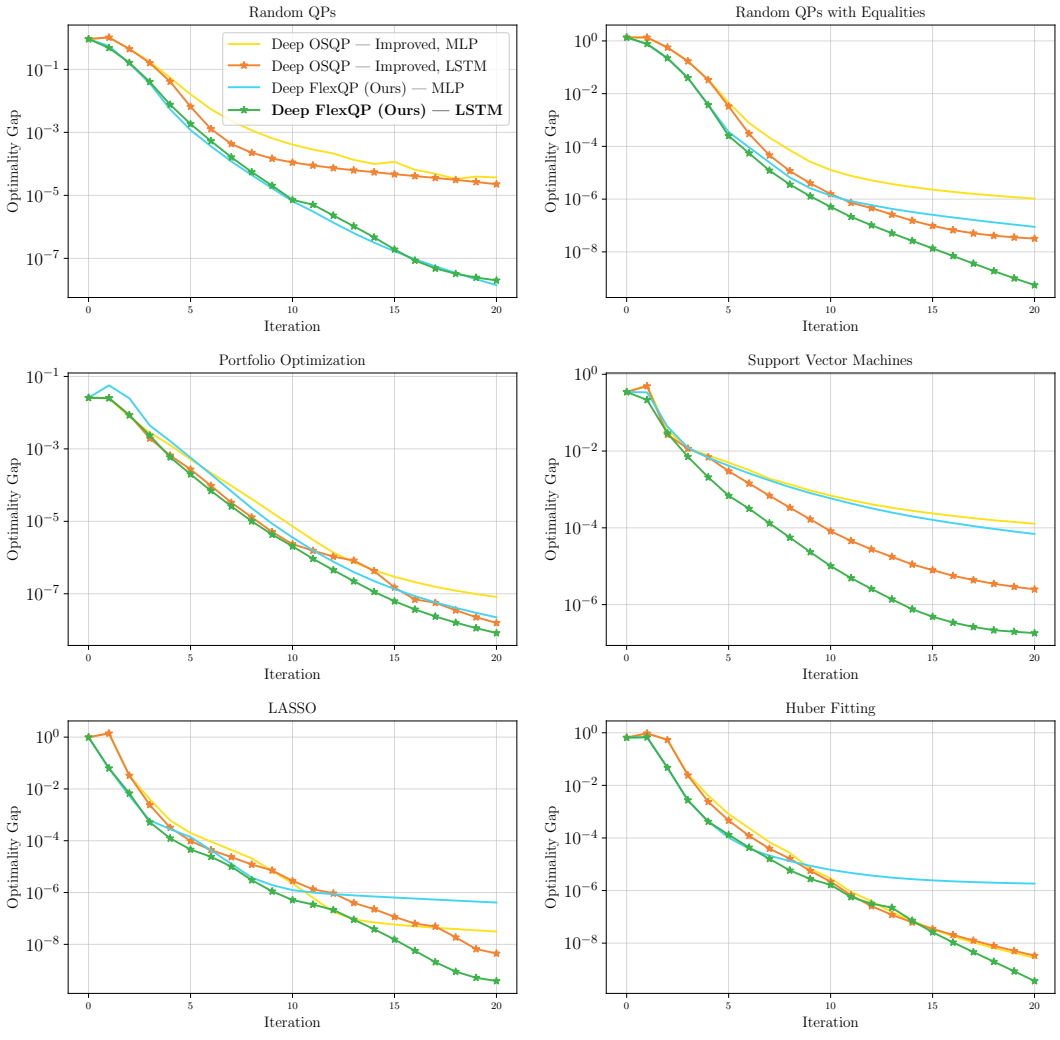


Figure 13: Comparison of MLP vs. LSTM policies.

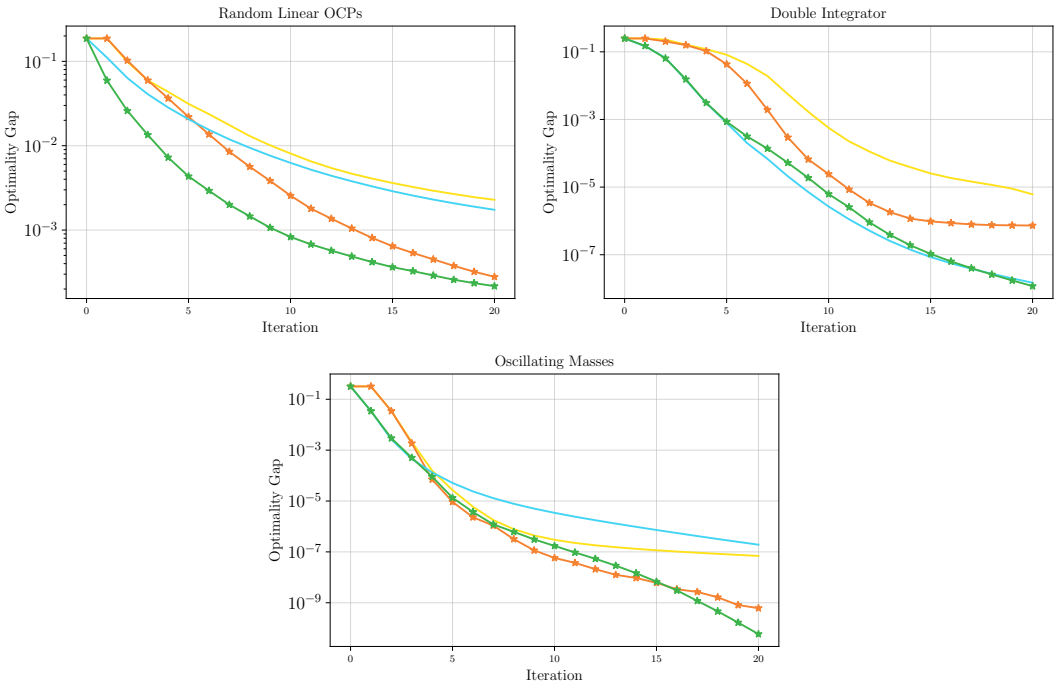


Figure 13: Comparison of MLP vs. LSTM policies.

K Ablation Analysis on the Use of Lagrange Multipliers in the Supervised Loss

In Fig. 14, we compare the performance of Deep FlexQP on different problem classes using the optimality gap loss from Eq. (8) and our proposed loss Eq. (9). It is evident that our loss including the Lagrange multipliers outperforms the optimality gap loss in all cases, except for the oscillating masses problem class. This could be simply due to the fact that the performance is already approacing $1\mathrm{e}{-10}$ at 20 iterations, and so small numerical differences play a bigger role. The overall increase in performance using the new loss can be explained by a stronger gradient signal given to Deep FlexQP to learn policies that ensure $\mu_I \geq \|y_I^*\|_\infty$ and $\mu_E \geq \|y_E^*\|_\infty$.

Surprisingly, however, the performance using both losses remains comparable. The ability of the optimality gap loss to perform nearly as well as the Lagrange multiplier loss likely stems from the coupling of μ_I with σ_s, ρ_I and that of μ_E with ρ_E in the Deep FlexQP architecture. That is, even with a weaker gradient signal from the optimality gap loss, the respective networks are able to learn a shared representation that allows effective learning of the penalty parameters μ as well.

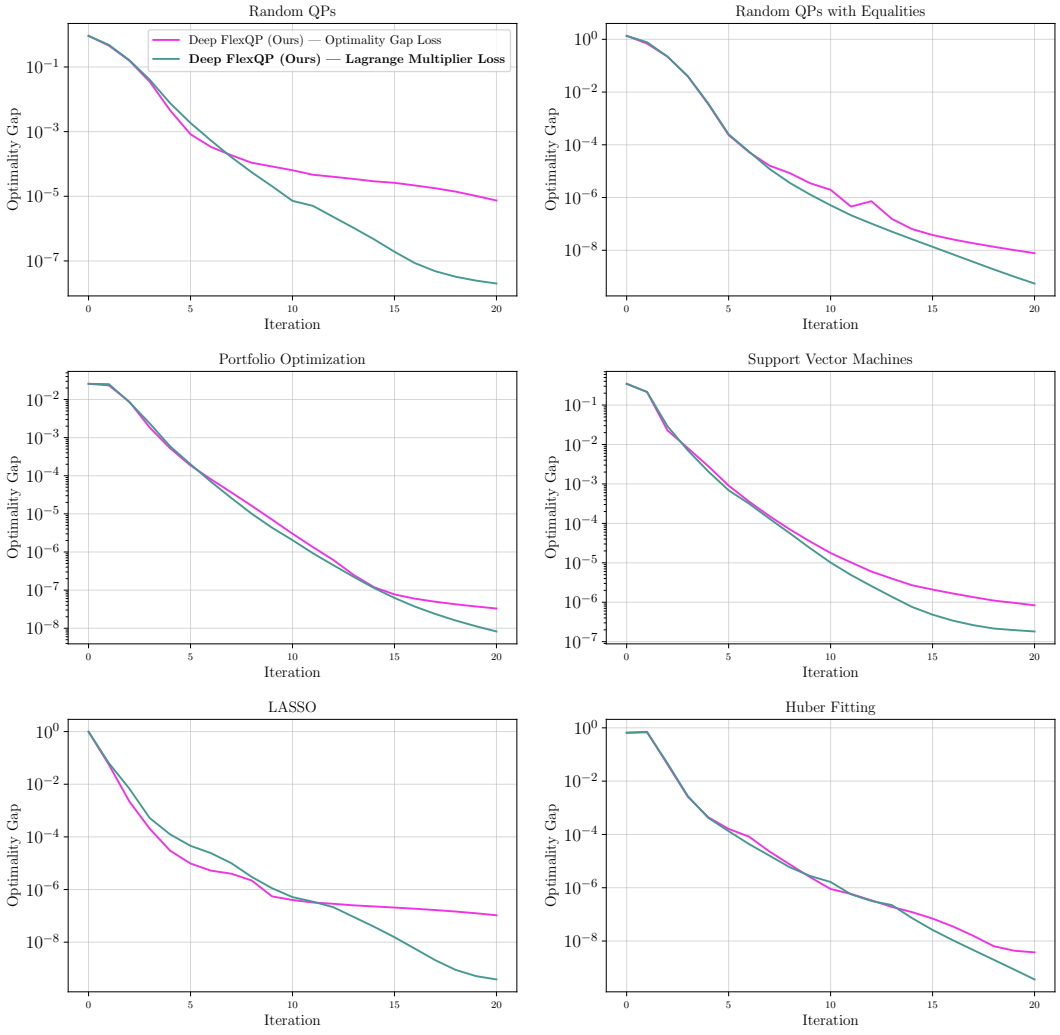


Figure 14: Using Lagrange multipliers in the supervised loss helps Deep FlexQP learn a more stable policy for the elastic penalty parameters μ.

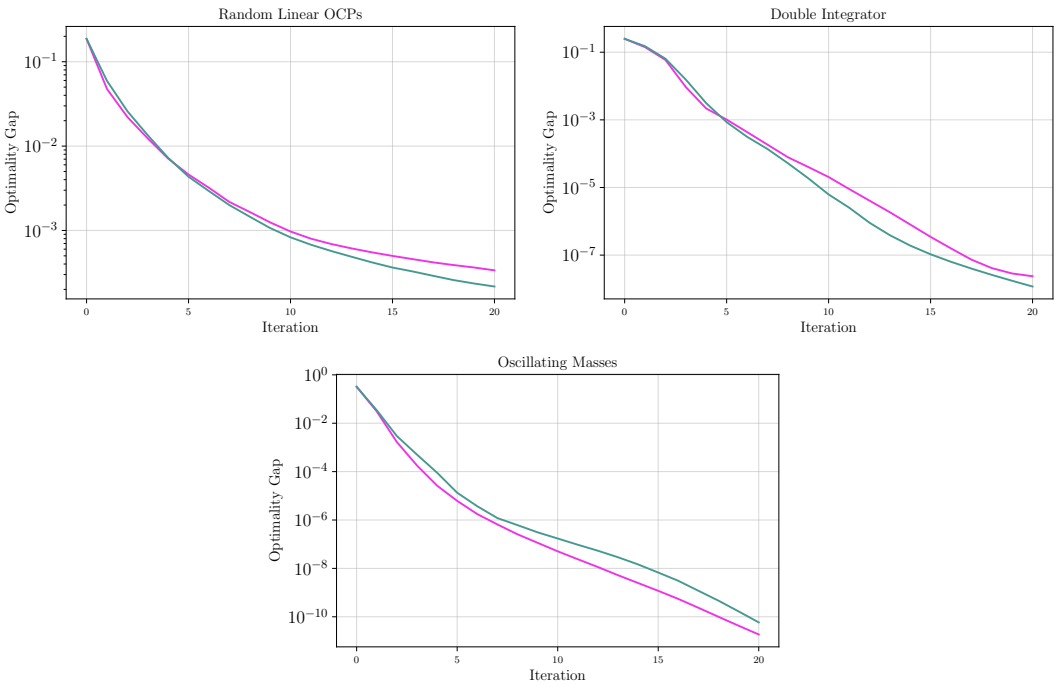


Figure 14: Using Lagrange multipliers in the supervised loss helps Deep FlexQP learn a more stable policy for the elastic penalty parameters μ.