NLPOpt–Net: A Learning Method for Nonlinear Optimization with Feasibility Guarantees

Anonymous authors
Paper under double-blind review

Abstract

We introduce the Nonlinear Parametric Optimization Network (NLPOpt–Net), an unsupervised learning architecture for solving parametric nonlinear programs (NLPs) across varying parameter instances while guaranteeing constraint feasibility. The architecture consists of a backbone neural network (NN) followed by a multilayer (k-layered) projection to enforce nonlinear constraints. NLPOpt–Net employs an inversion-free, modified Chambolle–Pock algorithm for rapid projection during the forward pass, leveraging local quadratic approximations of the original NLP, and uses the implicit function theorem for efficient backpropagation. Numerical experiments suggest that the projection improves solution quality by driving the predictions closer to optimality while preserving feasibility. NLPOpt–Net solves convex multiparametric QP, QCQP, and NLP problems with near zero optimality gap with constraint violations reduced to machine precision, thereby enabling a scalable approach for multiparametric programming. While a first-order Taylor approximation provides a feasible projection for convex problems, we observe that, with appropriate linearization, NLPOpt–Net can also solve nonconvex problems.We consider a nonconvex collision-free fleet trajectory optimization problem to demonstrate the ability of NLPOpt–Net to incorporate large number of constraints. While the models are trained using JAX in Python, the fixed structure of the projection further allows compiling a trained model in C. The compiled models provide significant improvement in inference time.

1 Introduction

We consider the following parametric nonlinear optimization (NLP) problem:

$$\min f(x,y), \quad \text{s.t. } h(x,y)=0, \quad g(x,y)\le 0, \quad l(x)\le y\le u(x), \tag{P}$$

where, y are the decision variables and x are the parameters. We aim to find the optimal solutions of P for different realizations of x. Formally, our goal is to develop a projection layer that projects a neural network (NN) output, $\hat{y}=\Phi_{NN}(x;\Theta)$, onto the constraint set such that the projected output $\tilde{y}$ always belongs to the feasible set $\mathcal{S}$, i.e., $\tilde{y}\in\mathcal{S}$, where $\mathcal{S}=\{y \mid h(x,y)=0,\ g(x,y)\le 0,\ l(x)\le y\le u(x)\}$. Here, h represents a set of parameterized equality constraints $h_i(x,y)=0,\ \forall i\in\mathcal{N}_E$; g represents a set of parameterized inequality constraints $g_j(x,y)=0,\ \forall j\in\mathcal{N}_I$; and l and u represent the parameterized lower and upper bounds of the variables y, respectively. Lastly, Θ represents the aggregated tunable parameters of the NN. We make the following assumptions: (i) the number of equality constraints, $|\mathcal{N}_E|\le n$, (ii) each equality constraint, h, is affine in y and linearly independent and the constraint g is convex nonlinear in y. Moreover, $h,\ g$ are continuously differentiable, and (iii) f is twice continuously differentiable and convex and its gradient is Lipschitz continuous.

Parametric NLPs arise in systems governed by nonlinear objectives and constraints for a broad range of applications such as robotics, model predictive control, operations research, materials science and many others. There are well-established techniques (see Nocedal & Wright (2006)) and packages (e.g., IPOPT (Wächter & Biegler, 2006), CONOPT (Drud, 1985)) for solving NLPs. However, they are primarily designed to solve a single instance, and are not well-suited for repeatedly solving the same problem with varying parameters. One way to address this challenge is by deriving the solution maps *a priori* across varying parameters (Bank et al., 1982; Pistikopoulos et al., 2020). While explicit parameter-to-solution mappings exist for certain problem classes, such as linear and quadratic

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

programs (Bemporad et al., 2002), conventional approaches for repeatedly solving parametric NLPs are computationally intensive and often lack scalability. Deep NNs can be trained over a distribution of parameters or problem instances to "learn" the solution maps. However, NNs inherently lack mechanisms to ensure explicit constraint satisfaction, a limitation that motivates a large body of existing works.

Penalizing constraint violations through the NN loss function is a simple and practical approach often applied in parametric optimization (Drgona et al., 2023) and physics informed neural networks (PINNs) (Raissi et al., 2019; Erichson et al., 2019). Despite being flexible and generalizable, soft-constrained methods suffer from an inherent tradeoff between prediction accuracy and constraint feasibility (Cai et al., 2021). Manual tuning is required to adjust the weights of the penalty terms, which can alter the solution (Grontas et al., 2026). Even with high penalties, constraint violation can still persist (Wang et al., 2022; Ma et al., 2022). In parametric optimization, when constraints are treated softly, the prediction may admit solutions that are not physically realizable, especially for problems arising in systems that are well defined by physical laws. Hard-constrained NNs focus on methodological approaches that can ensure constraint feasibility within a tolerance. These can be broadly classified into two categories. The first is *predict-and-complete* method (Beucler et al., 2021; Zamzam & Baker, 2020) where a subset of the outputs is predicted through the NN and the rest is computed by solving the system of equality constraints. Donti et al. (2021) in their work `DC3`, introduced a framework that uses predict-and-complete method for equality constraints and a correction strategy to satisfy the inequality constraints. The second method is *projection* that incorporates a differentiable projection layer to project the outputs of an NN into the feasible constraint manifold. Early works such as OptNet (Amos & Kolter, 2017) introduced quadratic programs (QP) as differentiable layers, later extended to broader classes of disciplined convex programs (Agrawal et al., 2019). Chen et al. (2024) proposed a Karush-Kuhn-Tucker (KKT)-based orthogonal projection with explicit analytical form for enforcing linear equality constraints. Min & Azizan (2024) proposed `HardNet`, a closed form projection approach to satisfy affine equality and inequality constraints. Iftakher et al. (2025) developed `KKT-HardNet`, a framework that finds a stationary point for the KKT system through projection to satisfy both nonlinear equality and inequality constraints. This approach is later extended to satisfy differential-algebraic (DAE) constraints in `DAE-HardNet` (Golder et al., 2027). Grontas et al. (2025) introduced `ΠNet`, an operator splitting based projection method that can incorporate convex constraints. Constante-Flores et al. (2025) proposed decision rule-based networks to incorporate linear inequality constraints. Lastrucci & Schweidtmann (2025) used an adaptive neural projection that recursively linearizes nonlinear equality constraints. A common trait of the projection-based methods is that the projection is based on Euclidean distance minimization. For learning the solutions of constrained optimization problems, this approach tends to move away from the true optima. To the best of our knowledge, no existing method addresses this limitation.

In this work, we introduce `NLPOpt-Net`, a general deep learning method and toolbox for solving parametric convex NLPs. The architecture consists of a backbone NN followed by a multilayer (k-layered) projection. While the NN drives toward optimality through a loss function consisting of a modified Lagrangian augmented with a consistency loss, the projection ensures feasibility in the original constraint manifold. Specifically, we linearize the constraints at NN output and, instead of traditional distance minimization based projection, we quadratically approximate the objective such that the quadratic coefficient remains diagonal. This approximated QP is then solved with an inversion-free method. This approximation in the projection guides the NN to find locally optimal solution. For affine constraints, a single approximation is sufficient. For general nonlinear constraints, we repeat this process for k times and we provide convergence guarantees after k layers of projection to attain a guaranteed feasible solution. The k-layered projection can be incorporated with any standard deep learning architectures. Moreover, the projection has a computationally cheap forward pass and the backward propagation is expedited with gradient computation through implicit function theorem.

Our key methodological contributions include: (1) a deep learning framework, NLPOpt-Net, for learning the solutions of parametric nonlinear optimization problems with a k-layered projection that ensures constraint feasibility; (2) a first order primal-dual algorithm for solving the projection problems with high efficiency; and (3) an ahead-of-time (AOT) compilation strategy for the trained model to accelerate the single instance inference time for practical deployment. Computational studies demonstrate considerable success of NLPOpt-Net over other methods for various classes of convex parametric problems with superior results in terms of both feasibility and optimality.

2 NLPOPT-NET: ARCHITECTURE AND ALGORITHMS

The `NLPOpt-Net` (**N**on**L**inear **P**arametric **Opt**imization **Net**work) framework is illustrated in Fig. 1. The architecture of NLPOpt-Net includes a backbone NN followed by a differentiable projection layer with one or multiple layers. We utilize automatic differentiation for backward propagation of the backbone and custom defined backward gradient for the projection layers. In the forward pass the projection is solved with a modified Chambolle-Pock (CP) algorithm (Chambolle & Pock, 2011). The Jacobian is computed with automatic differentiation. All the computations can be parallelized in GPUs. Our framework is able to train directly from the problem specification, i.e., the framework is fully unsupervised. Furthermore, when the NN prediction becomes close, the loss becomes self-supervised due to the consistency loss. We incorporate a first order primal dual method, the Chambolle-Pock algorithm to run the projection layer.

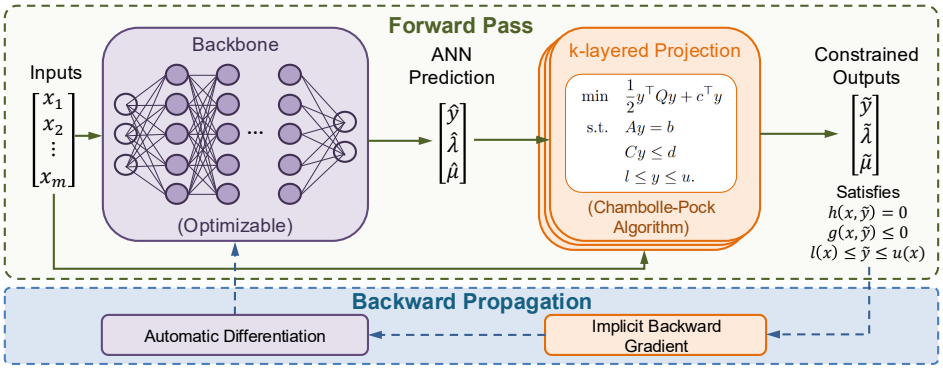

Figure 1: The `NLPOpt-Net` framework: the forward pass includes NN followed by k-layered projection and the backward pass uses custom vector Jacobian product (VJP) route. The NN is trained using a loss function that minimizes the the original objective of the parametric NLP along with a regularized soft loss and a consistency loss.

2.1 BACKBONE NEURAL NETWORK

Any standard NN can be used as the backbone NN. The NN takes x, the parameters as input and predicts the primal ($\hat{y}$) and dual ($\hat{\lambda}$ and $\hat{\mu}$) variables. The primal and dual variables are then projected to find a feasible solution. The loss function $\mathcal{L}$ is the modified Lagrangian on the projected primal ($\tilde{y}$) and projected dual ($\tilde{\lambda}$ and $\tilde{\mu}$) variables of the NLP augmented with a consistency term:

$$\mathcal{L} = f(\tilde{y}) + \|\frac{1}{|\mathcal{N}_E|}\tilde{\lambda}^\top h(x,\tilde{y})\|^2 + \frac{1}{|\mathcal{N}_E|}\tilde{\mu}^\top \mathrm{ReLU}\big(g(x,\tilde{y})\big) + \frac{\alpha}{M}\|\hat{z} - \tilde{z}\|^2, \tag{1}$$

where, $z = [y^\top, \lambda^\top, \mu^\top]^\top$, is a vector containing all the primal and dual variables. The only user defined tuning parameter is α which enforces the consistency loss. The consistency loss considers the minor linearization error and keeps the unconstrained predictions in the neighborhood of the original solution. The loss function pushes the NN prediction to get close to the optimal solution of (P). However, due to the soft embedding of the constraints, the NN alone does not guarantee feasibility. For this reason, we next opt for projection which ensures feasibility.

2.2 PROJECTION

The projection-based constraint satisfaction can be done by projecting the backbone NN prediction, $\hat{y}$ onto the feasible constraint manifold $\mathcal{S}$. While projecting with distance minimization ensures feasibility, it may move $\tilde{y}$ from the optimal solution y^*. Instead of solving a distance minimization problem ($\min_{y\in\mathcal{S}}\|y - \hat{y}\|^2$) in the projection layer, our projection involves solving a local quadratic approximation of the objective subject to the constraints as shown in PR:

$$\tilde{y} = \arg\min_{y\in\mathcal{S}} \quad f(x,\hat{y}) + J^\top(y - \hat{y}) + \frac{1}{2}(y - \hat{y})^\top H_d(y - \hat{y}), \tag{PR}$$

where, $J = \nabla f(x, \hat{y})$, $H_d = \rho \operatorname{diag}\left(\nabla^2 f(x, \hat{y})\right)$ and ρ is a parameter. Here, we notice the similarity to a proximal second order approximation of the objective used in augmented Lagrangian methods (Neal et al., 2011). However, unlike proximal approximation, we rely on different weight multipliers for different variables which are Hessian dependent. It is worthwhile to mention when $H_d = I$ and $J = 0$, the projection becomes Euclidean distance minimization. The approximation at the optimal point share the same primal-dual solution as the original problem P (see proof in Appendix H).

k-Layered Projection. A local solution of the nonlinear problem PR can be obtained by solving its KKT system. However, finding the KKT solution involves solving a system of nonlinear equations Nocedal & Wright (2006). To that end, we locally approximate the nonlinear projection problem (PR) using the following (SP):

$$\min \quad \frac{1}{2} y^\top Q y + c^\top y, \quad \text{s.t. } Ay = b,\ Cy \leq d,\ l \leq y \leq u. \tag{SP}$$

Here, the objective of SP is second order approximation of the original objective with only diagonal Hessian elements, and the constraints are first order approximation at point (x, y^i), where $i = 1, ..., k$ in a k-layered projection. Specifically, $Q = \rho \operatorname{diag}(\nabla^2 f(x, y^i))$, $c = \nabla f(x, y^i) - \rho \operatorname{diag}(\nabla^2 f(x, y^i)) y^i$, $A = J_h(x, y^i)$, $b = J_h(x, y^i) y^i - h(x, y^i)$, $C = J_g(x, y^i)$, $d = J_g(x, y^i) y^k - g(x, y^i)$, $l = l(x)$ and $u = u(x)$. We start from the backbone prediction, i.e., $y^0 = \hat{y}$. We sequentially solve SP$_i$ approximated at (x, y^i) in each projection layer i for $i = 1, ..., k$. Note that, for convex problems, the approximated QP remains feasible whereas for nonconvex cases, special linearization or additional treatment is required.

We note that a single layer is sufficient if all the constraints are affine. However, for nonlinear problems, multi-layer projection is used in a sequential manner to get to the feasible solution. We define an operator $\mathcal{P}_s$ such that $\tilde{y} = \mathcal{P}_s(x, \hat{y})$ is the solution to the approximated subproblem SP at the point of approximation being $(x, \hat{y})$. We define an overall projection layer represented by $\mathcal{P}_o$. It is trivial to show that under assumptions made earlier, SP remains feasible and has a primal-dual solution of SP. A single $\mathcal{P}_s$ may not ensure feasibility in the original nonlinear constraint manifold. When the backbone prediction is close to the original solution, the number of required sequential subproblems becomes minimal. The goal is to train the backbone to predict such neighborhood point so that with minimal change feasibility and optimality can be restored. We define the complete projection $\mathcal{P}_o$ as a composition of sequential operators $\mathcal{P}_s$ implemented in sequential layers, such that, $\mathcal{P}_o = \mathcal{P}_s^1 \circ \mathcal{P}_s^2 \circ \cdots \mathcal{P}_s^k$, where, $\mathcal{P}_o$ (k-layered Projection) represents solving PR as QP, and $\mathcal{P}_s^i$ represents solving the subproblem SP in ith layer. Given an arbitrarily small scalar ϵ and $k \in \mathbb{N}$, $\tilde{y}$ is computed by shifting the backbone prediction from$\hat{y}$ to $\tilde{y} = \mathcal{P}_o(x, \hat{y})$, which converges to an optimal solution of PR with linear convergence rate under constraint smoothness conditions such that $\hat{y} \in \mathcal{S}$ with a maximum absolute violation of ϵ or less.

Linear convergence of the projection. The k-layered projection achieves a local linear convergence rate when the Jacobian approximation error $\|M^{(k)}\| < 1/2$, where $M^{(k)} = (J^{(k)})^{-1} E^{(k)}$. For an initial iterate sufficiently close to z^*, the error satisfies$\|z^{(k+1)} - z^*\| \leq q \|z^{(k)} - z^*\|$ where $0 < q < 1$. The condition $\|M^{(k)}\| < 1/2$ is sufficient but not necessary for convergence. In particular, when all constraints are affine and the objective is quadratic, the projection subproblem coincides with the original problem, allowing an exact solution to be recovered in a single layer. (proof in Appendix F).

While many methods exist for solving QPs of the form in SP, we current implementation utilizes the efficiency of the Chambolle-Pock (CP) algorithm (Chambolle & Pock, 2011) (see Algorithm 1) exploiting the specific structured QP in each $\mathcal{P}_s^i$. The CP algorithm is a first-order primal dual algorithm that has resemblance to Douglas Rachford (DR) splitting algorithm and can be understood as a preconditioned version of the alternating direction method of multipliers (ADMM) (Neal et al., 2011). We provide the step by step updates, the termination criteria and convergence of the CP algorithm for the problem defined in Eq. 30 in Appendix G.

2.3 Backward gradient computation

We predict both primal and dual variables and use them in the training loss. With $z = [y^\top, \lambda^\top, \mu^\top]^\top$, the loss function $\mathcal{L}(x, \tilde{z})$, the projection layer $\mathcal{P}_o(x, \hat{z})$, and the NN $\Phi(x; \Theta)$, the loss function can be generalized into as a composite function of Θ and x such that $\mathcal{L}(x, \tilde{z}) =$

Algorithm 1: CP for QPs in projection layer

ChambollePock:

Input: Initial values y^0, λ^0, μ^0, p

Problem: $\min \frac{1}{2} y^\top Q y + c^\top y; \quad \text{s.t. } Ay = b;\ Cy \leq d;\ l \leq y \leq u$

Extract: $Q, c, A, b, C, d, l, u \leftarrow p$

Parameter: $\tau > 0, \sigma > 0, \theta = 1$ while $\left(\frac{1}{\tau} - L_f\right) \frac{1}{\sigma} < \|K\|^2$

compute $P = \mathrm{diag}(1/(1 + \tau q_i))$

initialize $k \leftarrow 0, y^0 \leftarrow y^0, \lambda^0 \leftarrow \lambda^0, \mu^0 \leftarrow \mu^0, \bar{y}^0 \leftarrow y^0$

repeat

$\lambda^{k+1} = \lambda^k + \sigma(A\bar{y}^k - b); \quad \mu^{k+1} = \max(0, \mu^k + \sigma(C\bar{y}^k - d))$

$\nu^{k+1} = P\big(y^k - \tau(A^\top \lambda^{k+1} + C^\top \mu^{k+1}) - \tau c\big); \quad y^{k+1} = \Pi_{[l,u]}(\nu^{k+1})$

$\bar{y}^{k+1} = y^{k+1} + \theta(y^{k+1} - y^k)$

$k \leftarrow k + 1$

until *termination criteria is satisfied*

Output: $z \leftarrow y^k, \lambda^k, \mu^k$

$\mathcal{L}(x, \mathcal{P}_o(x, \Phi(x; \Theta)))$. To train the backbone NN with backpropagation, the loss $\mathcal{L}$ has to be differentiated with respect to backbone NN parameters Θ. Using chain rule, we get

$$\frac{\mathrm{d}\mathcal{L}(x, \mathcal{P}_o(x, \Phi(x; \Theta)))}{\mathrm{d}\Theta} = \frac{\partial \mathcal{L}(x, z)}{\partial z}\bigg|_{z=\tilde{z}} \frac{\partial \mathcal{P}_o(x, z)}{\partial z}\bigg|_{z=\hat{z}} \frac{\partial \Phi(x; \vartheta)}{\partial \vartheta}\bigg|_{\vartheta=\Theta}, \tag{2}$$

where, $\hat{z} = \Phi(x; \Theta)$ and $\tilde{z} = \mathcal{P}_o(x, \hat{z})$. The projection layer is solved in an iterative manner. While the first and last terms are standard and efficiently computed with automatic differentiation, computing the backward gradient with automatic differentiation for the second term suffers due to the potentially large number of iterations in the projection layer. To that end we utilize the implicit function theorem to compute the backward gradient of the CP algorithm. The detailed derivation of the backward gradient for the projection is given in Appendix I. The use of the implicit function theorem in backward gradient is not new for iterative processes. For example, Optnet (Amos & Kolter, 2017) and Π-net (Grontas et al., 2025) used implicit backward gradient for iterative schemes such as the Newton's method and the Douglas-Rachford method.

2.4 Training NLPOpt-Net

We train NLPOpt-Net with standard algorithms (i.e., Adam (Kingma & Ba, 2014)). The heuristic use of the dual variables in the loss function trains the model to lower the optimality gap. The training/testing and the overall NLPOpt-Net framework are given in Algorithms 2 and 3. Particularly, Algorithm 2 provides all the steps in NLPOpt-Net framework including the forward pass and backward pass with the implicit backward gradient computation. Algorithm 3 provides the steps to train an NLPOpt-Net model including the testing evaluations.

3 Numerical Experiments

We consider a standard artificial neural network (ANN) with ReLU activation function. We deploy NLPOpt-Net to learn the solution of several convex parametric optimization problems that include (quadratic programs (QP), quadratically constrained quadratic programs (QCQPs), and a family of nonlinear programs (NLPs)). Despite the convergence being established for convex cases, we also test the framework on NLPs with nonconvex objective. All the optimization problems have 100 variables, 50 equality and 50 inequality constraints and 50 parameters. After training, we load the models and evaluate on all the training and testing instances. Additionally, we test NLPOpt-Net for collision–free fleet trajectory generation problem which consider thousands of constraints.

Baseline. We compare NLPOpt-Net against the several state-of-the-art solvers as well as other constrained neural network methods. Specifically, we use `OSQP`(Stellato et al., 2020) for QPs, `SCS`(O'Donoghue et al., 2023) for QCQPs and NLPs) and `SLSQP`(Virtanen et al., 2020) as optimization solvers. We compare the performance of regular artificial neural network (ANN), `Eq.NN`

Algorithm 2: NLPOpt-Net

`Forward`:

Input: $x, \Theta, n_{\max}$

$\hat{z} \leftarrow \Phi(x;\Theta),\ z^0 \leftarrow \hat{z}$

for $k = 0$ **to** $n_{\max} - 1$ **do**

$p^k \leftarrow \mathcal{M}(x, z^k)$

$(y^0, \lambda^0, \mu^0) \leftarrow z^k$

$z^{k+1} \leftarrow$ `ChambollePock`$(y^0, \lambda^0, \mu^0, p^k)$

$\tilde{z} \leftarrow z^k$

Output: $\hat{z}, \tilde{z}$

`Backward`:

Input: $\tilde{z},\ \hat{z},\ x,\ g_z = \dfrac{\partial \mathcal{L}}{\partial \tilde{z}}$

$v \leftarrow \texttt{Solve}\big((I - J_F(z^*)^\top)v = g_z\big)$

$\xi \leftarrow v^\top \dfrac{\partial F}{\partial p}\Big|_{p=\mathcal{M}(x,\hat{z})}$

$w \leftarrow \xi \dfrac{\partial \mathcal{M}(x,z)}{\partial z}\Big|_{z=\hat{z}}$

$\dfrac{d\mathcal{L}}{d\Theta} \leftarrow w \dfrac{\partial \Phi(x;\Theta)}{\partial \Theta}$

Output: $\dfrac{d\mathcal{L}}{d\Theta}$

Algorithm 3: Training/Testing

Input: Chosen/tuned hyperparameters

`Training`:

initialize Θ_1

for $t = 1$ **to** n_{epochs} **do**

$x \leftarrow$ sample from training set

$\hat{z}, \tilde{z} \leftarrow \texttt{Forward}(x, \Theta_t, n_{\max})$

compute $\mathcal{L}(x, z)$

$\dfrac{\partial \mathcal{L}}{\partial \Theta_\ell} \leftarrow \texttt{Backward}\left(\tilde{z}, \hat{z}, x, \dfrac{\partial \mathcal{L}}{\partial \tilde{z}}\right)$

$\Theta_{\ell+1} \leftarrow \text{Update}(\texttt{adam})$

`Testing`:

$x \leftarrow$ sample from test set

$\hat{z}, \tilde{z} \leftarrow \texttt{Forward}(x, \Theta_{n_{\text{epochs}}}, n_{\max})$

Output: $\Theta_{n_{\text{epochs}}},\ \tilde{z}$

(Zamzam & Baker, 2020), and `DC3` (Donti et al., 2021). Eq.NN is a supervised learning predicts and completes the full set of variables with equality constraints. DC3 uses predict and completion method to satisfy the equality constraints and a internal gradient descent step for inequality correction.

Comparison metrics. For convex problems, we compare the performances based on *Optimality*: Quality of the solution as compared to the solver solution, *Feasibility:* Quality of the solution compared with constraint violation, and *Speed:* Inference time in seconds or milliseconds for batch and single inference (see Appendix C for details on metrics and methods compared).

All experiments are executed on a computing system with 48 CPU cores from dual Intel Xeon 6248R 3.0GHz processors, 1 NVIDIA A100 40 GB GPU node, and 360 GB RAM allocated per run. For cases that are run sequentially in CPU, we divide the total time for each batch by the total number of problem instances in a batch considering full parallelization. We run all synthetic optimization experiments 5 times for 2000 epochs with different seeds (with a train/test ratio of 0.8:0.2). The detailed problem generation descriptions are given in Appendix A. Additionally, we provide the code as supplementary material along with all the hyperparameters stated in Appendix B.

3.1 Various classes of optimization problems

Convex quadratic programs (QPs). We learn the solution of the parametric convex quadratic programs with a quadratic objective and affine constraints:

$$\min_{y} \quad \frac{1}{2} y^\top Q y + c^\top y \text{ , s.t. } Ay = b + Bx,\ Cy \leq d,\ l + Lx \leq y \leq u + Ux, \tag{3}$$

where, $Q \in \mathbb{R}^{n \times n} \succeq 0$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n_{\text{eq}} \times n}$, $b \in \mathbb{R}^{n_{\text{eq}}}$, $B \in \mathbb{R}^{n_{\text{eq}} \times p}$, $C \in \mathbb{R}^{n_{\text{ineq}} \times n}$, $d \in \mathbb{R}^{n_{\text{ineq}}}$, $l \in \mathbb{R}^n$, $L \in \mathbb{R}^{n \times p}$, $u \in \mathbb{R}^n$, and $U \in \mathbb{R}^{n \times p}$. The goal is to find optimal $y \in \mathbb{R}^n$ for different parameter realizations of $x \in \mathbb{R}^p$.

Convex quadratically constrained quadratic programs (QCQPs). QCQPs are required to solve in many real life scenarios, including as portfolio optimization and robotics (Boyd & Vandenberghe, 2004) . In this example, we test the performance of NLPOpt-Net for the following parameterized

convex QCQP:

$$\min_y \ \frac{1}{2}y^\top Qy + c^\top \ , \text{s.t. } Ay = b + Bx,\ y^\top C_i y + d_i^\top y \le \beta_i + E_i x,\ l + Lx \le y \le u + Ux. \quad (4)$$

where, $Q \in \mathbb{R}^{n\times n} \succeq 0$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n_{\text{eq}}\times n}$, $b \in \mathbb{R}^{n_{\text{eq}}}$, $B \in \mathbb{R}^{n_{\text{eq}}\times p}$, $C_i \in \mathbb{R}^{n\times n} \succeq 0$, $d_i \in \mathbb{R}^n$, $\beta_i \in \mathbb{R}$, $E_i \in \mathbb{R}^{1\times p}$ for $i = 1, 2, \ldots, n_{\text{ineq}}$, $l \in \mathbb{R}^n$, $L \in \mathbb{R}^{n\times p}$, $u \in \mathbb{R}^n$, and $U \in \mathbb{R}^{n\times p}$. We learn to approximate the optimal solution $y \in \mathbb{R}^n$ given a parameter realization $x \in \mathbb{R}^p$.

Convex nonlinear programs (NLPs). We consider the following convex nonlinear problem with nonlinear constraints and quadratic objective function:

$$\min_y \ \frac{1}{2}y^\top Qy + c^\top y \ , \text{s.t. } Ay = b + Bx,\ a_i^\top \exp(y) + y^\top W y \le \beta_i + E_i x,\ l + Lx \le y \le u + Ux. \quad (5)$$

where, $Q \in \mathbb{R}^{n\times n} \succeq 0$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n_{\text{eq}}\times n}$, $b \in \mathbb{R}^{n_{\text{eq}}}$, $B \in \mathbb{R}^{n_{\text{eq}}\times p}$, $a_i \in \mathbb{R}^n$, $W \in \mathbb{R}^{n\times n} \succeq 0$, $\beta_i \in \mathbb{R}$, $E_i \in \mathbb{R}^{1\times p}$ for $i = 1, 2, \ldots, n_{\text{ineq}}$, $l \in \mathbb{R}^n$, $L \in \mathbb{R}^{n\times p}$, $u \in \mathbb{R}^n$, and $U \in \mathbb{R}^{n\times p}$.

Nonconvex programs. We consider a simple non-convex problem introduced by Donti et al. (2021):

$$\min_y \quad \frac{1}{2}y^\top Qy + p^\top \sin(y) \quad \text{s.t.} \quad Ay = x,\ Gy \le h, \quad (6)$$

where, $Q \in \mathbb{R}^{n\times n} \succeq 0$, $p \in \mathbb{R}^n$, $A \in \mathbb{R}^{n_{\text{eq}}\times n}$, $G \in \mathbb{R}^{n_{\text{ineq}}\times n}$ and $h \in \mathbb{R}^{n_{\text{ineq}}}$. $\sin(y)$ denotes componentwise application of the sine function to the vector y. We generated the problems using the similar steps followed in (Donti et al., 2021). We consider `SLSQP` as the optimizer for this nonconvex case.

Table 1: Results for parameterized optimization problems in terms of objective, constraint violation and time. The standard deviations across different runs are reported in the parenthesis. After training, both training and validation instances are tested. Values are rounded off to three decimals.

	Avg. Obj.	Max eq.	Mean eq.	Max ineq.	Mean ineq.	Time (s/batch)
Convex QP, α=10, k=1						
OSQP	-8.730 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.0039 (0.0000)
NN	-18.100 (0.055)	0.631 (0.000)	0.161 (0.000)	0.005 (0.000)	0.000 (0.000)	0.0052 (0.0000)
Eq. NN	5.132 (0.187)	0.000 (0.000)	0.000 (0.000)	2.356 (0.084)	0.003 (0.000)	0.0058 (0.0000)
DC3	-3.213 (1.441)	0.000 (0.000)	0.000 (0.000)	0.607 (0.190)	0.000 (0.000)	0.012 (0.0000)
NLPOpt-Net	-8.730 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.0357 (0.0000)
Convex QCQP, $\alpha = 10$, $k = 5$						
SCS	3.227 (0.055)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.0514 (0.0000)
NN	2.583 (0.161)	0.210 (0.110)	0.033 (0.000)	0.000 (0.000)	0.000 (0.000)	0.0070 (0.025)
Eq. NN	17.270 (0.929)	0.000 (0.000)	0.000 (0.000)	2.276 (0.699)	0.012 (0.045)	0.0080 (0.032)
DC3	7.186 (0.646)	0.000 (0.000)	0.000 (0.000)	0.306 (0.167)	0.000 (0.000)	0.0240 (0.035)
NLPOpt-Net	3.227 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.4260 (0.020)
Convex NLP, $\alpha = 100$, $k = 5$						
SCS	113.300 (0.000)	0.000 (0.000)	0.000 (0.000)	0.003 (0.000)	0.000 (0.000)	0.2510 (0.0000)
NN	113.200 (0.336)	0.017 (0.032)	0.003 (0.000)	0.378 (0.474)	0.000 (0.000)	0.0070 (0.010)
Eq. NN	296.600 (5.916)	0.000 (0.000)	0.000 (0.000)	18.250 (1.593)	2.517 (0.684)	0.0070 (0.010)
DC3	114.600 (0.677)	0.000 (0.000)	0.000 (0.000)	1.540 (0.639)	0.000 (0.000)	0.0280 (0.014)
NLPOpt-Net	113.300 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.4300 (0.036)
Nonconvex NLP, $\alpha = 10$, $k = 1$						
SLSQP	-3.833 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.0638 (0.0000)
NN	-4.098 (0.110)	0.853 (0.257)	0.177 (0.000)	0.000 (0.000)	0.000 (0.000)	0.0040 (0.010)
Eq. NN	-1.306 (1.008)	0.000 (0.000)	0.000 (0.000)	5.436 (1.347)	0.022 (0.122)	0.0050 (0.010)
DC3	-2.803 (0.736)	0.000 (0.000)	0.000 (0.000)	0.961 (1.043)	0.000 (0.000)	0.0110 (0.020)
NLPOpt-Net	-3.832 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.0780 (0.022)

Table 1 compares the performance of NLPOpt-Net with other methods for different cases. We find that both NLPOpt-Net and DC3 preserves feasibility. However, the maximum inequality violation is not zero for DC3 indicating that for a very small portion of instances DC3 is unable to preserve feasibility whereas NLPOpt-Net shows no infeasibility across different runs. Soft constrained NN includes infeasibility and goes below the true objectives. Eq-NN model preserves feasibility in terms of equality constraints, but not for inequality constraints. In terms of computational time, NLPOpt-Net requires same order of magnitude time for this case as DC3. In addition to the feasibility

assurance, a major improvement we find is in terms of optimality. While DC3 has the average optimality gap of more than 150% after training for 2000 epochs, NLPOpt-Net finds exact optimal solutions when rounded up to three decimals. For QCQP the results are similar to similar to QP case. While the mean inequality violation is zero for DC3, it struggles for a very few instance. However, for the QCQP test case, NLPOpt-Net requires one order of magnitude more time than DC3 and the optimizer. Although the work is done in batch and the optimizer time is taken as a average time for solving instances over a batch to ensure complete parallelization, the time required is around 9 times slower than the optimizer due to the number of layers (k=5) used in projection layer. Moreover, the best objectives are found for NLPOpt-Net across all the learning methods providing near optimal solution with feasibility. Table 1 also shows the results for the convex NLP test case. For $\alpha = 100$ and $k = 5$, NLPOpt-Net achieves near optimal solution compared to other learning methods and also retains complete feasibility across all instances. For this case, DC3 also achieves a near optimal solution. Even though the NLPOpt-Net framework is designed for convex cases we test the nonconvex case where NLPOpt-Net achieved best optimality among all the learning methods (see Table 1, optimality gap $\approx 0.026\%$). Due to the constraints being affine, a single layer in the projection is sufficient to satisfy the constraints resulting into efficient inference time. However, DC3 has $\approx 19\%$ variation in different runs. Additional experiments varying the number of parameters, constraints and active set agreements are provided in Appendix D.

3.2 Large scale optimization: Collision free fleet trajectory generation

We present an extended application of NLPOpt-Net to generate collision free trajectory generation for a fleet consists of multiple vehicles. This problem can be categorized as a control problem with a quadratic objective to minimize effort subject to discrete time dynamics, state constraints and collision avoidance constraints (nonconvex safety constraints). The input parameters to the NN backbone are the initial and final position of the vehicles.

Problem description. We follow the formulation provided by Grontas et al. (2025) for the dynamic and state constraints. However, the nonconvex collision avoidance constraints are adopted from the work of Augugliaro et al. (2012). We consider a fleet of N vehicles going from p^0 to p^f position in T time. We denote $p_i[t], v_i[t]$ and $a_i[t]$ as the position, velocity and acceleration vector for vehicle i at the discrete time $t \in 0, 1, \ldots, H$ with time step $h = T/H$. Discretized dynamic constraints are: $p_i[t+1] = p_i[t] + hv_i[t] + \frac{h^2}{2}a_i[t]$ and $v_i[t+1] = v_i[t] + ha_i[t]$ with the parameterized constraints being $p_i[0] = p_i^0$ and $p_i[H] = p_i^f$. The bound constraints on the variables and jerk are as follows: $p_{\min} \leq p_i[t] \leq p_{\max}$, $v_{\min} \leq v_i[t] \leq v_{\max}$, $a_{\min} \leq a_i[t] \leq a_{\max}$ and $J_{\min} \leq \frac{a_i[t+1]-a_i[t]}{h} \leq J_{\max}$. The nonconvex collision avoidance constraints are: $\|p_i[t] - p_j[t]\|_2 \geq R_c, \forall i < j, i \neq j, t \in \{1, \ldots, H-1\}$. The objective is minimizing the cumulative acceleration given to the system: $\texttt{minimize} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \|a_i[t]\|_2^2$.

Implementation Details. A Taylor approximation based linearization of the nonconvex collision avoidance constraints are considered for the sequential projection steps (Augugliaro et al., 2012) (for details see Appendix E.2). As this problem introduces a large number of variables and constraints, we consider sparse computation on the projection layer based on the known sparsity pattern. The projection mechanism has a significant advantage for a locally approximate solution. Hence, to expedite the training procedure we include a warm start of the NN backbone by training it for certain number of epochs with a soft loss (constraint violation) and then start the projection with the loss defined in Eq. 1. We consider $k = 3$ layers and $\alpha = 100$ for this experiment.

Results. We consider a 2D space with 15 vehicles and 30 discrete times. This produces an parametric optimization problem with 60 parameters, 2,640 variables and 11,970 constraints (1,800 equality, 4,890 inequality and 5,280 bounding constraints). We provide some of the trajectories generated by our trained model in Figure 2. Additionally, we provide more visualizations in Appendix E.1. As we include collision avoidance constraints, the optimization problem can not be decoupled among vehicles (see Grontas et al. (2025)) which shows that our model can be extended to a large number of constraints in practice even without having decoupling properties of the constraints. Furthermore, the constraint satisfaction is ensured to the similar level of optimization-based collision free trajectory generation.

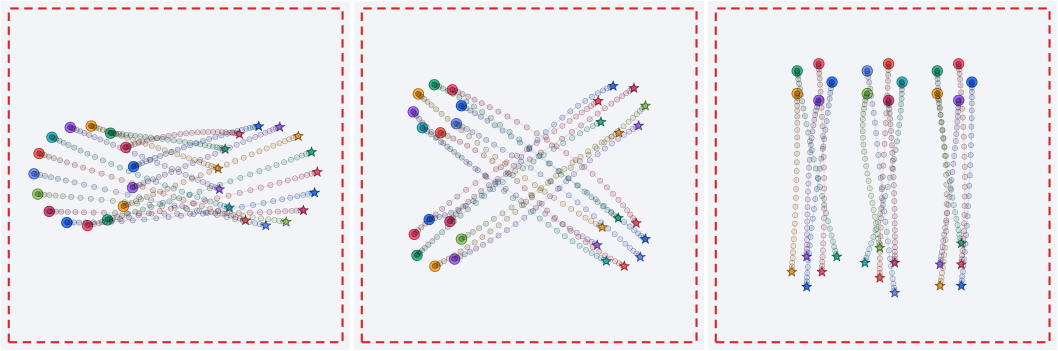

Figure 2: Example trajectories for 15-vehicles with different instances (different parameters) of vehicle initial and final configurations. From left to right, we show different clusters of vehicle moving together and avoiding collisions.

3.3 Practical deployment of NLPOpt-Net

Often in practice, parametric optimization is solved in sequence rather than solving in batches. In model predictive control, for example solvers are called rapidly and repetitively as parameters vary with time. To emphasize on the rapid and repetitive calls for optimization, we compare the inference performance of NLPOpt-Net with conventional single instance optimization solvers. For NLPOpt-Net we utilize the fixed projection mechanism to have an ahead-of-time (AOT) compiled model using C as well as just-in-tim (JIT) compilation using JAX, specialized for single inference for trained models. This experimentation is executed on a 13th Gen Intel® Core™ i7–13700 (24-core CPU) with 32 GB RAM running Linux OS for reflecting realistic deployment scenarios. Results in Table 2 shows that 2–10 times speed up can be achieved during inference with AOT compared to the solvers. Furthermore, solvers tend to have a very large standard deviation across different calls.

Table 2: Inference performance of trained NLPOpt-Net models compared to the optimizers for single samples. Average inference time is presented in milliseconds and standard deviation in parentheses.

Problem	Optimizer (ms)	JIT (ms)	AOT (ms)
QP	6.88 (2.65)	2.81 (0.03)	1.35 (0.01)
QCQP	52.71 (20.12)	38.16 (0.11)	20.81 (0.06)
NLP	236.91 (75.03)	36.76 (0.53)	17.91 (0.24)
Nonconvex	29.37 (5.20)	6.15 (0.03)	2.95 (0.02)

4 Conclusions

Contributions. We present NLPOpt-Net, a neural network architecture that learns parameter-to-solution mapping of parametric optimization problems in the constraint manifold. Our approach includes a k-layered projection step where k can vary to achieve feasibility. In the k-layered projection, we employ a case specific version of Chambolle-Pock algorithm to solve a structured quadratic program. We find that with the objective based projection, NLPOpt-Net yields superior results in terms of optimality than other deep learning based solvers. In general, NLPOpt-Net is designed for convex problems. However, we find that in some specific cases, it also yields superior results for nonconvex problems. Furthermore, we show that NLPOpt-Net can be applied to large scale problems with thousands of variables and constraints.

Limitations. While the current form of NLPOpt-Net considers fixed number of layers, during inference, adaptive number of layers can be deployed to achieve user-specified feasibility tolerance.. Depending on the number of layers in projection, the computational expense and feasibility may vary. While a multiparameteric solver, such as `PPOPT` (Kenefake et al., 2023), can detect infeasible parameter space, our method requires feasible parametric space to start training. The framework can be further improved exploiting specific constraint structure, problem instances, and sparse computation to handle large problems.

AI USE STATEMENT

In this work, we used generative AI tools to create or edit software code and for grammatical check. We have not used generative AI to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims and generate synthetic data sets, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, or interpret results. We have reviewed all AI-assisted work. All LLM-generated code was verified and tested for correctness by all authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The research conducted in this paper conforms in every respect to the ICLR Code of Ethics (https://iclr.cc/public/CodeOfEthics). NLPOpt-Net is a computational framework with experimental studies performed on synthetic optimization problems, and it does not pose any societal impact at it's current stage of development. This statement does not consider any future release of other software using this framework.

REPRODUCIBILITY STATEMENT

To support reproducibility, the main text presents the complete formulation of NLPOpt-Net, including k-layered projection, implicit differentiation procedure, and training and evaluation methodology. The appendices provide detailed algorithms, mathematical derivations, benchmark formulations, implementation choices, neural architectures with hyperparameter settings, and the preconditioning procedure. Additionally we report the complete problem-instance generation procedures, dataset splits, random seeds, computational environments, and evaluation metrics used in all experiments. We also provide the code as supplementary material containing the JAX implementation, native C inference code, configuration files, trained model parameters, and scripts required to reproduce the reported tables and figures.

REFERENCES

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

Brandon Amos and J Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pp. 136–145. PMLR, 2017.

Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pp. 1917–1922. IEEE, 2012.

Bernd Bank, Jürgen Guddat, Diethard Klatte, Bernd Kummer, and Klaus Tammer. *Nonlinear parametric optimization*, volume 58. Walter de Gruyter GmbH & Co KG, 1982.

Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

Tom Beucler, Michael Pritchard, Stephan Rasp, Jordan Ott, Pierre Baldi, and Pierre Gentine. Enforcing analytic constraints in neural networks emulating physical systems. *Physical review letters*, 126(9):098302, 2021.

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

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. `https://github.com/jax-ml/jax`, 2018. Version 0.3.13.

Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6):060801, 2021.

Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40(1):120–145, 2011.

Antonin Chambolle and Thomas Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1):253–287, 2016.

Hao Chen, Gonzalo E Constante Flores, and Can Li. Physics-informed neural networks with hard linear equality constraints. *Computers & Chemical Engineering*, 189:108764, 2024.

Gonzalo E Constante-Flores, Hao Chen, and Can Li. Enforcing hard linear constraints in deep learning models with decision rules. *arXiv preprint arXiv:2505.13858*, 2025.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 2016. URL `https://stanford.edu/~boyd/papers/pdf/cvxpy_paper.pdf`. To appear.

Priya Donti, David Rolnick, and J Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.

Jan Drgona, A Tuor, J Koch, M Shapiro, B Jacob, and D Vrabie. NeuroMANCER: Neural modules with adaptive nonlinear constraints and efficient regularizations. *URL https://github.com/pnnl/neuromancer*, 2023.

Arne Drud. CONOPT: A grg code for large sparse dynamic nonlinear optimization problems. *Mathematical programming*, 31(2):153–191, 1985.

N Benjamin Erichson, Michael Muehlebach, and Michael W Mahoney. Physics-informed autoencoders for lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866*, 2019.

Rahul Golder, Bimol Nath Roy, and MM Faruque Hasan. Dae-hardnet: A physics informed neural network enforcing differential-algebraic hard constraints. *Computers & Chemical Engineering*, pp. 109881, 2027.

Panagiotis D Grontas, Antonio Terpin, Efe C Balta, Raffaello D'Andrea, and John Lygeros. Pinet: Optimizing hard-constrained neural networks with orthogonal projection layers. *arXiv preprint arXiv:2508.10480*, 2025.

Panagiotis D Grontas, Anastasios Tsiamis, and John Lygeros. Operator splitting for convex constrained markov decision processes. *IEEE Transactions on Automatic Control*, 2026.

Ashfaq Iftakher, Rahul Golder, Bimol Nath Roy, and M M Faruque Hasan. Physics-informed neural networks with hard nonlinear equality and inequality constraints. *Computers & Chemical Engineering*, pp. 109418, 2025.

Dustin Kenefake, Iosif Pappas, Nikolaos A Diangelakis, Styliani Avraamidou, Richard Oberdieck, and Efstratios N Pistikopoulos. Multiparametric linear and quadratic programming. In *Encyclopedia of Optimization*, pp. 1–5. Springer, 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Giacomo Lastrucci and Artur M Schweidtmann. ENFORCE: Nonlinear constrained learning with adaptive-depth neural projection. *arXiv preprint arXiv:2502.06774*, 2025.

Kaiwen Ma, Nikolaos V Sahinidis, Satyajith Amaran, Rahul Bindlish, Scott J Bury, Devin Griffith, and Sreekanth Rajagopalan. Data-driven strategies for optimization of integrated chemical plants. *Computers & Chemical Engineering*, 166:107961, 2022.

Youngjae Min and Navid Azizan. Hardnet: Hard-constrained neural networks with universal approximation guarantees. *arXiv preprint arXiv:2410.10807*, 2024.

Parikh Neal, Chu Eric, Peleato Borja, and Eckstein Jonathan. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

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

Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. SCS: Splitting conic solver, version 3.2.11. `https://github.com/cvxgrp/scs`, November 2023.

Efstratios N Pistikopoulos, Nikolaos A Diangelakis, and Richard Oberdieck. *Multiparametric optimization and control*. John Wiley & Sons, 2020.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Daniel Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical report, CM-P00040415, 2001.

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.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1): 25–57, 2006.

Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

Ahmed S Zamzam and Kyri Baker. Learning optimal solutions for extremely fast ac optimal power flow. In *2020 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm)*, pp. 1–6. IEEE, 2020.

A PROBLEM INSTANCE GENERATION

We provide the details of the problem generation for different computational studies here:

QP: We consider $n = 100, p = 50, n_{\text{eq}} = 50, n_{\text{ineq}} = 50$, and the parameter bounds are taken as $x \in [-10, 10]^{50}$. During training, the parameter entries were drawn from the uniform distribution on $[-10, 10]$ and we consider 2000 problem instances. For the Q matrix, we generate a random Gaussian matrix M and consider $Q = (M^\top M)/n + \text{diag}(\delta)$. We add the diagonal shift δ between $[1.2, 1.8]$ ensuring Q is positive definite, so the objective is always strictly convex. The vector c is created elementwise and all the entries are drawn from the uniform distribution on $[0.5, 1.5]$. The matrix A is generated from a QR factorization of a random Gaussian matrix, where the first n_{eq} orthonormal directions are selected and scaled row-wise to ensure full row rank and good numerical conditioning. To ensure feasibility of the equality constraints for all parameter realizations, $x \in [x_L, x_U]$, we construct an affine mapping of the form $y_f(x) = y_c + Tx$, where $y_c \in \mathbb{R}^n$ and $T \in \mathbb{R}^{n\times p}$ that are randomly generated with controlled magnitude. Then other constant vector and matrix for equality constraints are defined as $b = Ay_c$ and $B = AT$. For inequality constraints, each row of matrix C is generated with sparse support and normalized coefficients. Vector d is defined as $d_i = C_i y_c + \max_{x\in[x_L,x_U]} C_i T x$ which guarantees feasibility for all admissible x. Finally, the bound constraints are defined with $L = U = T$, $l = y_c - rI$, and $u = y_c + rI$ for some $r > 0$. This ensures feasibility of the constraints for any realization of x within the bounds.

QCQP: The objective and equality constraints are generated in same way as the quadratic case. For quadratic inequality constraints, we construct each constraint in a structured manner. Specifically, for each i, we generate a direction vector $r_i \in \mathbb{R}^n$ and define $C_i = \alpha_i r_i r_i^\top$ and $d_i = r_i$, where $\alpha_i > 0$ is chosen to control the curvature and ensure numerical stability. Thus, each constraint takes the form

$$\frac{1}{2}\alpha_i (r_i^\top y)^2 + r_i^\top y \le \beta_i + E_i x. \tag{7}$$

The right-hand side is constructed to guarantee feasibility of the affine mapping $y_f(x)$ for all admissible x. In particular, β_i is chosen as

$$\beta_i = \max_{x\in[x_L,x_U]} \left(\frac{1}{2}\alpha_i (r_i^\top y_f(x))^2 + r_i^\top y_f(x) - E_i x \right), \tag{8}$$

which ensures that $y_f(x)$ satisfies all quadratic inequalities over the parameter domain. The parameter-dependent term $E_i x$ is introduced by selecting E_i along representative directions in the parameter space, enabling nontrivial coupling between x and the constraints. Finally, the bound constraints are defined with $L = U = T$ and $l = y_c - rI$ and $u = y_c + rI$ for some $r > 0$. This ensures feasibility of the constraints for any realization of x within the bound.

Convex NLP: For nonlinear inequality constraints, we construct each constraint in a structured manner. Specifically, for each i, we generate a vector $a_i \in \mathbb{R}^n$ and a positive semi-definite matrix $W_i \in \mathbb{R}^{n\times n}$ with sparse diagonal structure. The nonlinear constraint takes the form

$$a_i^\top \exp(y) + y^\top W_i y \le \beta_i + E_i x. \tag{9}$$

The coefficients a_i and W_i are constructed by selecting a small subset of variables and assigning nonzero weights to emphasize nonlinear behavior along those directions. The right-hand side β_i is defined as

$$\beta_i = \max_{x\in[x_L,x_U]} \left(a_i^\top \exp(y_f(x)) + y_f(x)^\top W_i y_f(x) - E_i x \right) + \epsilon, \tag{10}$$

where $\epsilon > 0$ is a margin parameter. This ensures that the affine mapping $y_f(x)$ satisfies all nonlinear inequality constraints for every x in the parameter domain. The parameter-dependent term $E_i x$ is constructed by selecting directions in the parameter space and scaling them to introduce coupling between x and the nonlinear constraints. Rest of the vectors and matrices are generated as per previous instructions.

Nonconvex: We used the methods provided by Donti et. al.(Donti et al., 2021) to generate the problems.

B DETAILS OF HYPERPARAMETERS

We performed hyperparameter tuning via coordinate search over relevant parameters while parameters such as the neural network size and parameters in Chambolle-Pock algorithm were selected heuristically. Final parameters were chosen to prioritize feasibility. For the methods used for comparison we use the default values obtained from their repository. Following are the parameters that were kept fixed across all experiments:

- Problem generation seed: 42
- No. of samples: 2000 for simple optimization and 5000 for fleet trajectory
- Train–test ratio: 0.8–0.2
- Train seeds: 42, 43, 44, 45, 46
- No. of epochs: 2000 for simple optimization cases
- Number of hidden layers: 3
- Hidden layer size: 128
- Batch size: 400
- CP max iterations: 500
- CP tolerance: 1×10^{-09}
- Norm seed: 42

We used the fixed parameter variation of the Chambolle-Pock algorithm for solving the projection problems. In the fixed parameter variation we used a fixed parameter, `safety=0.95` to scale down all the other parameters (τ, σ) to ensure convergence.

B.1 SIMPLE OPTIMIZATION PROBLEMS

We list ranges over which different hyperparameters were tuned and the specific parameter value which are considered heuristically. Final parameters are marked in bold.

Parameter	**QP**	**QCQP**	**NLP**	**Nonconvex**
Learning rate	10^{-2}, $\mathbf{10^{-3}}$, 10^{-4}	10^{-2}, $\mathbf{10^{-3}}$, 10^{-4}	10^{-2}, $\mathbf{10^{-3}}$, 10^{-4}	10^{-2}, $\mathbf{10^{-3}}$, 10^{-4}
Consistency weight, α	1, **10**, 100	1, **10**, 100	1, 10, **100**	1, **10**, 100
k projection layers	1	3, 4, **5**	3, 4, **5**	1
Norm estimation iterations	25	25	25	25
Ruiz equilibration iterations	10	10	10	10
Adjoint maximum iterations	25	25	25	25

B.2 FLEET TRAJECTORY PROBLEM

We list ranges over which different hyperparameters were tuned and the specific parameter value which are considered heuristically. Here due to the inclusion of the warm train with soft loss we list some additional parameters for fleet trajectory experiment. Final parameters are marked in bold.

- Epochs trained with soft loss: 5000, 10000, **20000**
- Soft loss weight: equality: 0.1, 1, **10**; inequality: 0.1, **1**, 10
- Epochs trained with NLPOpt-Net loss: 500, 1000, **2500**
- Consistency weight, α : 1, 10, **100**
- Learning rate: 10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$
- k projection layers: 2, **3**
- Norm estimation iterations: 50
- Ruiz equilibration iterations: 20
- Adjoint maximum iterations: 30

C METRICS AND METHODS COMPARED

Followings are details of the metrics we compared in the computational experiments:

Optimality: Optimality refers to how close the final prediction is to the true optimal solution. We measure the average optimality gap (AOG) as the following formula:

$$\text{AOG} = \frac{\text{Average Objective Value (}\textit{Model Objective}\text{)} - \text{Best Bound (}\textit{Optimizer Objective}\text{)}}{|\text{Average Objective Value}|} \times 100.$$

Feasibility: Feasibility is measured by the constraint violation computed as the absolute residual for equality constraint, i.e, $|h(x, y)|$ and $g_+(x, y)$ for inequality where $g_+ = \max(0, g)$.

Speed: Speed refers to the time required for inference after the training is finished. The inference time is given for each batch. For optimizer we consider full parallelization and for a fare comparison we divide the total solve time by total number of instances in a batch.

We compare NLPOpt-Net against the following methods:

Optimizer: A traditional optimization solver. We use `CVXPY`(Diamond & Boyd, 2016) environment (for convex problem, solver: `OSQP`(Stellato et al., 2020) for QPs, `SCS`(O'Donoghue et al., 2023) for QCQPs and NLPs) and `SLSQP`(Virtanen et al., 2020) (the SciPy implementation using `scipy.optimize` for nonconvex objective problems).

NN: A regular ANN model with soft loss for constraint violation.

Eq. NN: This supervised learning (Zamzam & Baker, 2020) predicts and completes the full set of variables with equality constraints. The model is trained on minimizing the mean squared error between the optimal solutions of the Optimizer and the model completed solution.

DC3: The deep constraint completion and correction (`DC3`) (Donti et al., 2021) method is state-of-the-art that uses predict and completion method to satisfy the equality constraints. The inequality correction is then done by mapping infeasible points to feasible points using an internal gradient descent step during training. We use the github codespace for DC3 including default tunable parameters for comparison.

D ADDITIONAL EXPERIMENTS

In the additional experiments we compare NLPOpt-Net with Optimizer and DC3 for different settings on the QPs. In the additional experiments, we run QP with 100 variables as the number of equality constraint (Table 4), inequality constraints (Table 5), number of parameters (Table 3) and training sample size varies (Table 6). Additionally we provide the active set agreement analysis for different settings. This shows how from the dual variables we can detect the active sets. For inequality constraint set I_{ineq}, we consider a constraint $g_i \, \forall i \in I_{\text{ineq}}$ is active when the dual variable $\mu_i > 0$ and we denote the set of active constraints as $I_{\text{A,ineq}}$. We compare across all the runs and instances by one to one mapping, what fraction of the active sets were predicted correctly with the true active sets from the optimizer run. The activity agreements are summarized in Table 7. We take the default model specific hyperparameters for DC3 and the hyperparameter tuning may yield better results in terms of optimality and constraint violation. We train all the models for 2000 epochs. We observe that DC3 objective was still improving. Providing more training epoch may yield better objective with less variation. For NLPOpt-Net the only hyperparameter we consider for all the cases is weight for consistency loss (α).

Remarks on Additional Experiments. (see Appendix D) We find that the time requirement for NLPOpt-Net during inference increases with equality constraints and sometimes with inequality constraints. The underlying reason could be the sparsity pattern of the constraint matrices. As the Chambolle-Pock algorithm rely on matrix multiplications in each step and on norm calculation, the denser the matrix the more time it will take. In our problems we generated fully dense matrix. In practical setting constraints are sparse and techniques can be employed to make the matrix multiplications efficient by detecting the sparsity pattern. Grontas et al. (2025) shows such case where by considering sparsity similar first order methods can be used to deal with very large problems. We find from Table 6, NLPOpt-Net deviates from optimal solution very steadily when the training

sample is smaller while DC3 struggles to find a good solution when the training samples are limited. To the best of our knowledge, only NLPOpt-Net provides idea on the active sets of the constraints and Table 7 shows a near accurate prediction of active sets.

Table 3: Results on QP task for 100 variables, 50 equality and 50 inequality constraints as the number of parameters varies as 10, 25, 50, 75, 100. We compare the performance of NLPOpt-Net with DC3 and the optimizer according to objective value, maximum/mean equality and inequality violation and inference time per batch. Standard deviation across 5 runs are shown in parenthesis.

Model	Metric	10	25	50	75	100
Optimizer	Obj. val.	-8.026 (0.089)	-8.352 (0.089)	-8.730 (0.000)	-8.471 (0.000)	-8.892 (0.000)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/sample)	0.004 (0.000)	0.004 (0.000)	0.004 (0.000)	0.004 (0.000)	0.003 (0.000)
DC3	Obj. val.	-1.536 (1.770)	1.082 (3.576)	-3.213 (1.441)	-3.364 (1.025)	-2.652 (1.777)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.988 (0.228)	1.673 (0.837)	0.607 (0.190)	0.696 (0.302)	1.468 (0.219)
	Mean ineq.	0.001 (0.000)	0.001 (0.001)	0.000 (0.000)	0.000 (0.000)	0.001 (0.000)
	Time (s/batch)	0.014 (0.000)	0.013 (0.000)	0.012 (0.000)	0.013 (0.000)	0.014 (0.000)
NLPOpt-Net	Obj. val.	-8.024 (0.000)	-8.350 (0.000)	-8.730 (0.000)	-8.468 (0.000)	-8.888 (0.000)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/batch)	0.036 (0.000)	0.036 (0.000)	0.036 (0.000)	0.035 (0.000)	0.033 (0.000)

Table 4: Results on QP task for 100 variables, 50 parameters and 50 inequality constraints as the number of equality constraints varies as 10, 30, 50, 70, 90.

Model	Metric	10	30	50	70	90
Optimizer	Obj. val.	-8.772 (0.000)	-8.730 (0.000)	-8.730 (0.000)	-8.365 (0.000)	3.814 (0.000)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/sample)	0.003 (0.000)	0.004 (0.000)	0.004 (0.000)	0.004 (0.000)	0.004 (0.000)
DC3	Obj. val.	-3.757 (1.141)	-3.288 (1.361)	-3.213 (1.441)	-2.177 (1.894)	4.102 (1.928)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.395 (0.141)	0.557 (0.195)	0.607 (0.190)	0.977 (0.319)	1.123 (0.594)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.001 (0.000)	0.001 (0.000)
	Time (s/batch)	0.013 (0.000)	0.013 (0.000)	0.012 (0.000)	0.013 (0.000)	0.013 (0.000)
NLPOpt-Net	Obj. val.	-8.772 (0.000)	-8.730 (0.000)	-8.730 (0.000)	-8.365 (0.000)	3.819 (0.000)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/batch)	0.018 (0.000)	0.027 (0.000)	0.036 (0.000)	0.040 (0.000)	0.045 (0.000)

Table 5: Results on QP task for 100 variables, 50 parameters and 50 equality constraints as the number of inequality constraints varies as 10, 30, 50, 70, 90.

Model	Metric	10	30	50	70	90
Optimizer	Obj. val.	-19.310 (0.000)	-11.020 (0.000)	-8.730 (0.000)	-6.551 (0.084)	-8.363 (0.063)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/sample)	0.002 (0.000)	0.003 (0.000)	0.004 (0.000)	0.005 (0.000)	0.005 (0.000)
DC3	Obj. val.	-3.757 (1.141)	-3.288 (1.361)	-3.213 (1.441)	-2.177 (1.894)	-2.119 (1.928)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.395 (0.141)	0.557 (0.195)	0.607 (0.190)	0.977 (0.319)	1.123 (0.594)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.001 (0.000)	0.001 (0.000)
	Time (s/batch)	0.013 (0.000)	0.013 (0.000)	0.012 (0.000)	0.013 (0.000)	0.013 (0.000)
NLPOpt-Net	Obj. val.	-19.310 (0.000)	-11.020 (0.000)	-8.730 (0.000)	-6.548 (0.000)	-8.363 (0.000)
	Max eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/batch)	0.019 (0.000)	0.026 (0.000)	0.036 (0.000)	0.044 (0.000)	0.026 (0.000)

Table 6: Results on QP task for 100 variables, 50 parameters, 50 inequality constraints and 50 parameters as the number of samples taken for training is varying. We consider total 2000 instances and for training we vary the training fraction as 0.8, 0.6, 0.4, 0.2.

Model	Metric	0.8	0.6	0.4	0.2
Optimizer	Objective	-8.730 (0.000)	-8.730 (0.095)	-8.730 (0.095)	-8.730 (0.095)
	Max. Eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean Eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max. Ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean Ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/sample)	0.004 (0.000)	0.004 (0.000)	0.004 (0.000)	0.004 (0.000)
DC3	Objective	-3.213 (0.845)	-0.922 (1.348)	3.189 (1.478)	23.650 (2.317)
	Max. Eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean Eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max. Ineq.	0.607 (0.276)	1.132 (0.656)	1.099 (0.409)	3.201 (0.770)
	Mean Ineq.	0.000 (0.000)	0.000 (0.000)	0.001 (0.000)	0.006 (0.045)
	Time (s/batch)	0.012 (0.000)	0.014 (0.032)	0.015 (0.045)	0.015 (0.045)
NLPOpt-Net	Objective	-8.730 (0.000)	-8.726 (0.000)	-8.662 (0.095)	-8.271 (0.130)
	Max. Eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean Eq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Max. Ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Mean Ineq.	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Time (s/batch)	0.035 (0.000)	0.039 (0.000)	0.034 (0.000)	0.027 (0.000)

Table 7: Performance across problem classes and parameter settings. The first row provides the active set agreement for the results shown in Section 3. Other rows shows results on QP with 100 variables and the common structure with 50 equality, 50 inequality and 50 parameters with varying one of the property at a time. We cap the dual variables to 1×10^{-6}

Class	Problem type	QP	QCQP	NLP	Nonconvex	
	Active set agreement	0.998	0.997	0.999	0.999	
p (QP)	No. of parameters	10	25	50	75	10
	Active set agreement	0.998	0.998	0.998	0.999	0.998
n_{eq} (QP)	No. of equality	10	30	50	70	90
	Active set agreement	0.997	0.997	0.998	0.999	1.000
n_{ineq} (QP)	No. of inequality	10	30	50	70	90
	Active set agreement	0.997	0.998	0.998	0.998	0.998

E ADDITIONAL DETAILS ON THE FLEET TRAJECTORY PROBLEM

We provide additional trajectories for different configurations in Figure 3. We were able to generate collision free trajectory for the fleet even when the fleets were moving in clusters. It is worthwhile to mention the illustrated configurations were not included as parameter samples in the training sets. We use the first order Taylor approximation for the linearization of the nonconvex collision avoidance constraints. The mathematical form of the linearization is shown in E.2.

E.1 ADDITIONAL VISUALIZATION OF THE TRAJECTORY GENERATION PROBLEM

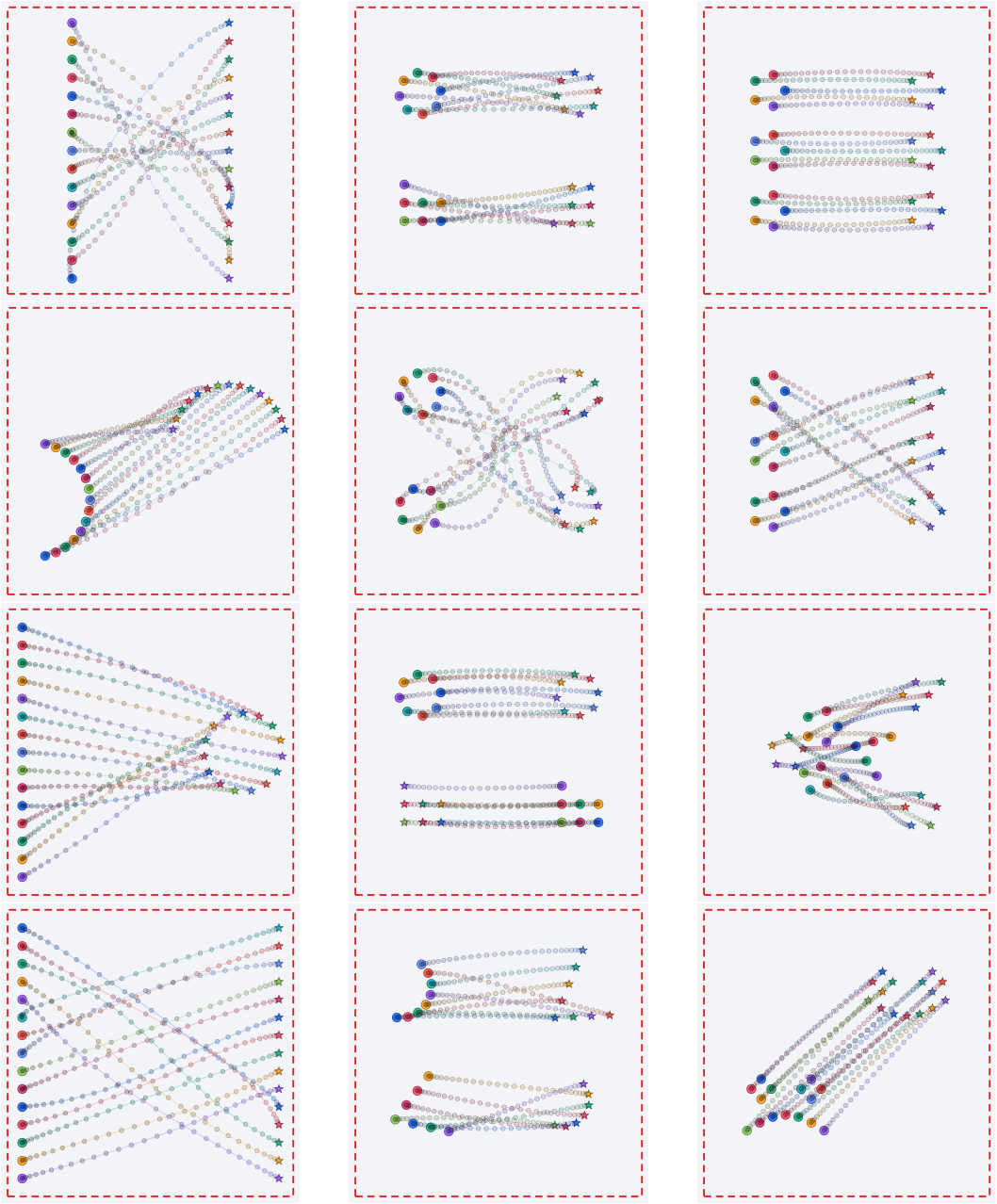

Figure 3: Additional synthesized collision free trajectories for 15-vehicles. Each figure indicates a different instance of initial and final configurations of the vehicles.

E.2 LINEARIZATION OF COLLISION AVOIDANCE CONSTRAINTS

While the vehicle dynamics and bounding constraints are linear (i.e. convex), the collision avoidance constraints are nonconvex; $\|p_i[t] - p_j[t]\|_2 \geq R_{\mathrm{c}}, \forall i < j, i \neq j, t \in \{1, \ldots, H-1\}$. A first order Taylor expansion of the constraints at point $\bar{p}_i[t]$ and $\bar{p}_j[t]$ can be written in the following form:

$$\|\bar{p}_i[t] - \bar{p}_j[t]\|_2 + \eta^\top \left[(p_i[t] - p_j[t]) - (\bar{p}_i[t] - \bar{p}_j[t])\right] \geq R_{\mathrm{c}}, \tag{11}$$

where,

$$\eta = \frac{\bar{p}_i[t] - \bar{p}_j[t]}{\|\bar{p}_i[t] - \bar{p}_j[t]\|_2}. \tag{12}$$

Eq. 11 can be cast into an linear form and added to the matrix A in the projection (SP). The linearization for the first projection layer is done at the neural network output and the consequent linearization is done at the solution of previous projection step.

Sparse computation. We exploit the sparse nature of the constraints of the trajectory problem. Particularly, we consider storing all the matrix and vectors in a shared-pattern `COO` format. We use a preliminary version of sparse computation algorithm through only considering nonzero elements by considering the predefined nonzero row and columns instead of any special sparse algorithm. We then use the sparse matrix vector products in projection, norm calculation and in implicit backward gradient.

F LOCAL CONVERGENCE RATE OF THE PROJECTION LAYER

The k-layer projection is an iterative procedure to polish the solution to get back feasibility. This can be seen as a similar technique like sequential quadratic programming (SQP). However, we do not imply the second order information of the Lagrangian. Instead we refine the quadratic objective. The convergence rate of the method can be analyzed from SQP theory(Nocedal & Wright, 2006). A similar type of convergence analysis is discussed for nonlinear equality constraints in AdaNP Projection(Lastrucci & Schweidtmann, 2025).

The k-layer projection at the kth layer ($\mathcal{P}_s^k$) solves the subproblem of the form:

$$\begin{aligned} \min_{y \in \mathcal{B}(x)} \quad & f(x, \hat{y}) + J^\top (y - \hat{y}) + \frac{1}{2}(y - \hat{y})^\top H_d (y - \hat{y}) \\ \text{s.t.} \quad & J_h(x, y^k)(y - y^k) + h(x, y^k) = 0 \\ & J_g(x, y^k)(y - y^k) + g(x, y^k) \leq 0 \end{aligned} \tag{$\mathrm{P}_{\mathrm{approx}}$}$$

which approximates the original nonlinear problem:

$$\begin{aligned} \min_{y \in \mathcal{B}(x)} \quad & f(x, \hat{y}) + J^\top (y - \hat{y}) + \frac{1}{2}(y - \hat{y})^\top H_d (y - \hat{y}) \\ \text{s.t.} \quad & h(x, y) = 0 \\ & g(x, y) \leq 0 \end{aligned} \tag{$\mathrm{P}_{\mathrm{orig}}$}$$

where, $\mathcal{B}$ defines the box constraint on the decision variables. While the hessian of the approximated objective includes the hessian of the Lagrangian, we keep original objective as it is quadratic already. As we discussed, the original problem is convex, therefore we assume the existence of y^*, a global solution to the original problem at which the following conditions (C1-C3) hold:

- C1 The objective function and the constraints are twice differentiable in a neighborhood of y^* with Lipschitz continuous second order derivatives.
- C2 Linear independence constraint qualification and Slater's condition hold at y^*, therefore, the KKT conditions are satisfied at y^* with a valid set of Lagrangian multipliers λ^* (equality) and μ^* (inequality) with strong duality.

C3 Second-order sufficient conditions hold at (y^*, λ^*, μ^*).

Let $\mathcal{L}_{\text{orig}}$ is the Lagrangian of the original problem (P_{orig}). With slack variables and Fischer-Burmeister reformulation the KKT system can be expressed as (see (Iftakher et al., 2025) for details on the formulation):

$$F(z) = \begin{bmatrix} \nabla_y \mathcal{L}_{\text{orig}}(y, \lambda, \mu) \\ h(y) \\ g(y) + s \\ \phi(\mu, s) \end{bmatrix}, \quad \text{with} \quad z = \begin{bmatrix} y \\ \lambda \\ \mu \\ s \end{bmatrix}, \tag{13}$$

where, $\phi(\mu_j, s_j) = \left[\sqrt{\mu_j^2 + s_j^2} - \mu_j - s_j\right]$. We define the Jacobian of the KKT conditions of the original problem in a neighborhood of the local solution as:

$$J^{(k)} = \begin{bmatrix} \nabla^2_{yy} \mathcal{L}_{\text{orig}} & J_h^\top & J_g^\top & 0 \\ J_h & 0 & 0 & 0 \\ J_g & 0 & 0 & I \\ 0 & 0 & D_\mu & D_s \end{bmatrix}, \tag{14}$$

where, $D_\mu = \text{diag}(\mu_i / \sqrt{\mu_i^2 + s_i^2} - 1)$ and $D_s = \text{diag}(s_i / \sqrt{\mu_i^2 + s_i^2} - 1)$. Due to the assumptions we made earlier, the Jacobian at iteration k, $J^{(k)}$ is non-singular and invertible. Similarly the KKT of the approximated problem (P_{approx}) is:

$$\bar{F}(z) = \begin{bmatrix} \nabla_y \mathcal{L}_{\text{approx}}(y, \lambda, \mu) \\ J_h(x, y^k)(y - y^k) + h(x, y^k) \\ J_g(x, y^k)(y - y^k) + g(x, y^k) + s \\ \phi(\mu, s) \end{bmatrix}, \tag{15}$$

the Jacobian of the KKT conditions of the approximated problem in a neighborhood of the local solution is:

$$J^{(k)} = \begin{bmatrix} \nabla^2_{yy} \mathcal{L}_{\text{approx}} & J_h^\top & J_g^\top & 0 \\ J_h & 0 & 0 & 0 \\ J_g & 0 & 0 & I \\ 0 & 0 & D_\mu & D_s \end{bmatrix}, \tag{16}$$

The deviation of the projection from complete SQP step can be expressed through a matrix E where

$$E^{(k)} = J^{(k)} - \bar{J}^{(k)} = \begin{bmatrix} \nabla^2_{yy} \mathcal{L}_{\text{orig}} - \nabla^2_{yy} \mathcal{L}_{\text{approx}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{17}$$

where, $\nabla^2_{yy} \mathcal{L}_{\text{orig}} - \nabla^2_{yy} \mathcal{L}_{\text{approx}} = \sum_i \lambda_i^{(k)} \nabla^2 h_i(y^{(k)}) + \sum_j \mu_j^{(k)} \nabla^2 g_j(y^{(k)})$, the deviation comes from the second order information of the constraints. At kth layer, we define the residual $r^{(k)} = F(z^{(k)})$, the error $e^{(k)} = z^{(k)} - z^*$ and solve for the Newton's step $\Delta z^{(k)}$:

$$\begin{aligned} \bar{J} \Delta z^{(k)} &= -r^{(k)} \\ z^{(k+1)} &= z^{(k)} + \Delta z^{(k)} \\ e^{(k+1)} &= e^{(k)} + \Delta z^{(k)}. \end{aligned} \tag{18}$$

Using first order Taylor's expansion,

$$F(z^{(k)}) = J^{(k)} e^{(k)} + r^{(k)}, \tag{19}$$

where, $r^{(k)} = O(\|e^{(k)}\|)$. Therefore,

$$\bar{J}^{(k)}\Delta z^{(k)} = -r^{(k)} = -F(z^{(k)}) = -J^{(k)}e^{(k)} - r^{(k)}. \tag{20}$$

Since, $E^{(k)} = J^{(k)} - \bar{J}^{(k)}$,

$$\begin{aligned}
\left(J^{(k)} - E^{(k)}\right)\Delta z^{(k)} &= -J^{(k)}e^{(k)} - r^{(k)}, \\
(J^{(k)})^{-1}\left(J^{(k)} - E^{(k)}\right)\Delta z^{(k)} &= -(J^{(k)})^{-1}\left(J^{(k)}e^{(k)} - r^{(k)}\right), \\
(I - M)\Delta z^{(k)} &= -e^{(k)} - (J^{(k)})^{-1}r^{(k)}, \\
\Delta z^{(k)} &= -(I - M)^{-1}\left(e^{(k)} + (J^{(k)})^{-1}r^{(k)}\right),
\end{aligned} \tag{21}$$

where, $M = ((J^{(k)})^{-1}E^{(k)})$. Hence, the error at $(k+1)$th layer can be expressed as:

$$\begin{aligned}
e^{(k+1)} &= e^{(k)} + \Delta z^{(k)} \\
&= e^{(k)} - (I - M)^{-1}\left(e^{(k)} + (J^{(k)})^{-1}r^{(k)}\right) \\
&= \left(I - (I - M)^{-1}\right)e^{(k)} - (I - M)^{-1}(J^{(k)})^{-1}r^{(k)} \\
&= \left((I - M)(I - M)^{-1} - (I - M)^{-1}\right)e^{(k)} - (I - M)^{-1}(J^{(k)})^{-1}r^{(k)} \\
&= (I - M)^{-1}\left((I - M) - I\right)e^{(k)} - (I - M)^{-1}(J^{(k)})^{-1}r^{(k)} \\
&= -(I - M)^{-1}Me^{(k)} - (I - M)^{-1}(J^{(k)})^{-1}r^{(k)}.
\end{aligned} \tag{22}$$

Taking norms and using Banach's lemma ($\|(I - A)^{-1}\| \leq 1/(1 - \|A\|)$), and $r^{(k)} = \mathcal{O}(\|e^{(k)}\|)$ i.e., $\|r^{(k)}\| \leq C_1\|e^{(k)}\|^2$ the error can be estimated,

$$\|e^{(k+1)}\| \leq \frac{\|M^{(k)}\|}{1 - \|M^{(k)}\|}\|e^{(k)}\| + \frac{\|(J^{(k)})^{-1}\|C_1}{1 - \|M^{(k)}\|}\|e^{(k)}\|^2. \tag{23}$$

Suppose that, throughout a sufficiently small neighborhood of z^*, $\|M^{(k)}\| \leq \rho < \frac{1}{2}$,$\|(J^{(k)})^{-1}\| \leq K$ for some finite constant $K > 0$. Eq. 23 then implies

$$\|e^{(k+1)}\| \leq \frac{\rho}{1 - \rho}\|e^{(k)}\| + \frac{KC_1}{1 - \rho}\|e^{(k)}\|^2. \tag{24}$$

For $\|e^{(k)}\| \leq \delta$, choosing $\delta > 0$ sufficiently small such that

$$q = \frac{\rho + KC_1\delta}{1 - \rho} < 1. \tag{25}$$

It follows that

$$\|e^{(k+1)}\| \leq q\|e^{(k)}\|, \qquad 0 < q < 1. \tag{26}$$

Thus, the projection converges locally at linearly. If the constraints are affine $E^{(k)} = M^{(k)} = 0$. Moreover, the linearized constraints coincide with the original constraints. For a quadratic objective, the projection subproblem is then identical to the original problem, and its exact solution is recovered in a single layer. When $M \neq 0$ and $\|M\| < 1/2$, linear convergence is guaranteed. However, when $\|M\| \geq 1/2$, second order corrections might require to converge to the solution. Note that, $\|M\| < 1/2$ is a sufficient condition for linear convergence but not necessary.

G CHAMBOLLE-POCK ALGORITHM

The Chambolle-Pock (CP) algorithm (Chambolle & Pock, 2011) for solving the generic saddle-point problem of the following form

$$\begin{aligned} \textit{Primal Problem:} \quad & \min_{y \in Y} F(Ky) + G(y) \\ \textit{Dual Problem:} \quad & \max_{\lambda \in \Lambda} -(G^*(-K^*\lambda) + F^*(\lambda)) \end{aligned} \tag{SPP}$$

where, the map $K : Y \mapsto \Lambda$ is a continuous linear operator, $G : Y \mapsto [0, +\infty)$, $F^* : \Lambda \mapsto [0, +\infty)$ are proper, convex, lower semi-continuous functions while F^* is the convex conjugate of the convex lower semi-continuous function F. The algorithm assumes the SPP has at least a solution $(\hat{y}, \hat{\lambda}) \in Y \times \Lambda$, which satisfies

$$\begin{aligned} K\hat{y} &\in \partial F^*(\hat{\lambda}) \\ -(K^*\hat{\lambda}) &\in \partial G(\hat{y}) \end{aligned} \tag{27}$$

where ∂F^* and ∂G are the subgradient of F^* and G. To solve the problem defined in Eq. The update of the CP algorithm for solving the SPP is

$$\begin{cases} \lambda^{n+1} = (I + \sigma \partial F^*)^{-1}(\lambda^n + \sigma K \bar{y}^n) \\ y^{n+1} = (I + \tau \partial G)^{-1}(y^n - \tau K^* \lambda^{n+1}) \\ \bar{y}^{n+1} = y^{n+1} + \theta(y^{n+1} - y^n) \end{cases} \tag{28}$$

where, $\tau, \sigma > 0$ and $\theta \in [0, 1]$. The algorithm converges when $\theta = 1$ and $\tau\sigma\|K\|^2 \leq 1$. See (Chambolle & Pock, 2011) for details. We consider the QP of the following form:

$$\begin{aligned} \min \quad & \frac{1}{2} y^\top Q y + c^\top y \\ \text{s.t.} \quad & Ay = b \\ & Cy \leq d \\ & l \leq y \leq u \end{aligned} \tag{29}$$

where, Q is a diagonal matrix such that $Q = \text{diag}(q_1, q_2, \cdots, q_n) \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{n_{\text{eq}} \times n}$, $b \in \mathbb{R}^{n_{\text{eq}}}$, $C \in \mathbb{R}^{n_{\text{ineq}} \times n}$, $d \in \mathbb{R}^{n_{\text{ineq}}}$, $l \in (R \cup \{-\infty\})^n$, and $u \in (R \cup \{\infty\})^n$. By introducing the dual variables $\lambda \in \mathbb{R}^{n_{\text{eq}}}$, $\mu \in \mathbb{R}^{n_{\text{ineq}}}$ and $\alpha, \beta \in \mathbb{R}^n$ corresponding to the equality, inequality and bound constraints respectively the saddle-point problem becomes

$$\min_{l \leq y \leq u} \quad \max_{\lambda,\, \mu \geq 0} \quad \frac{1}{2} y^\top Q y + c^\top y + \lambda^\top (Ay - b) + \mu^\top (Cy - d). \tag{30}$$

For a general quadratic problem defined by Eq. 30 the steps are generated as follows where the combined constraint matrix K, combined coefficient vector h and combined multipliers w are defined as $K^\top = (A^\top, C^\top)$, $h^\top = (b^\top, d^\top)$ and $w^\top = (\lambda^\top, \mu^\top)$. Multipliers for the bound constraints are obtained analytically.

Step 1: *Dual update*
Updating the dual variable involves solving the convex quadratic problem (in w)

$$w^{k+1} = \arg\min_w \frac{1}{2}\|w - (w^k + \sigma K \bar{y}^k)\|^2 + \sigma h^\top w. \tag{31}$$

By setting the gradient to zero yields, $w - (w^k + \sigma K \bar{y}^k) + \sigma h = 0$. The update step becomes,

$$w^{k+1} = w^k + \sigma(K\bar{y}^k - h). \tag{32}$$

To retain the complementary conditions ($\mu \geq 0$) in the dual variables, the update in Eq. 32 can be split into

$$\begin{aligned} \lambda^{k+1} &= \lambda^k + \sigma(A\bar{y}^k - b) \\ \mu^{k+1} &= \max(0, \mu^k + \sigma(C\bar{y}^k - d)) \end{aligned} \tag{33}$$

Step 2: *Primal Update*
Updating the primal variable involves solving the convex quadratic problem (in y)

$$\nu^{k+1} = \arg\min_{\nu} \frac{1}{2}\|\nu - (y^k - \tau K^\top w^{k+1})\|^2 + \tau\left(\frac{1}{2}\nu^\top Q\nu + c^\top \nu\right). \tag{34}$$

By setting the gradient to zero yields, $\nu - (y^k - \tau K^\top w^{k+1}) + \tau Q\nu + \tau c = 0$ i.e. $(I + \tau Q)\nu = y^k - \tau K^\top z^{k+1} - \tau c$. The update step becomes,

$$\nu^{k+1} = P\left(y^k - \tau(A^\top \lambda^{k+1} + C^\top \mu^{k+1}) - \tau c\right) \tag{35}$$

where, $P = (I + \tau Q)^{-1}$. It is worthwhile to mention that this is true for any convex QP. However, if the $Q = \text{diag}(q_1, q_2, \cdots, q_n) = \text{diag}(q_i)$ then $(I + \tau Q)^{-1} = \text{diag}(1/(1 + \tau q_i))$. Hence, computational cost involved in computing P becomes negligible which requires minimal binary operations only.

To retain the box constraint we apply the box projection $\Pi_{[l,u]}$ on ν.

$$y^{k+1} = \Pi_{[l,u]}(\nu^{k+1}) \tag{36}$$

where, $\Pi_{[l,u]}(v) = \min(u, \max(l, v))$ is the projection to keep the primal feasibility on the box constraints.

Step 3: *Update initialization*

$$\bar{y}^{k+1} = y^{k+1} + \theta(y^{k+1} - y^k) \tag{37}$$

Termination Criteria: As SP remains convex, the primal and dual feasibility and primal-dual gap fit perfectly as termination criteria to get the optimal solution of the layer $\mathcal{P}_s^i$. We set the termination criteria when the primal residuals, the dual residuals and the primal-dual gap are smaller than some predefined tolerance levels $\epsilon_{\text{prim}} > 0, \epsilon_{\text{dual}} > 0$, and $\epsilon_{\text{gap}} > 0$ respectively, i.e.,

$$\|r^k_{\text{prim}}\|_\infty \le \epsilon_{\text{prim}}, \quad \|r^k_{\text{dual}}\|_\infty \le \epsilon_{\text{dual}}, \quad \|r^k_{\text{gap}}\|_\infty \le \epsilon_{\text{gap}}, \tag{38}$$

where,

$$\begin{aligned} r^k_{\text{prim}} &= \begin{bmatrix} Ay^k - b \\ \max(Cy^k - d,\ 0) \end{bmatrix}, \quad r^k_{\text{dual}} = Qy^k + c + A^\top \lambda^k + C^\top \mu^k - \alpha^k + \beta^k, \\ r^k_{\text{gap}} &= (\lambda^k)^\top (Ay^k - b) + (\mu^k)^\top (Cy^k - d) + (\alpha^k)^\top (l - y^k) + (\beta^k)^\top (y^k - u). \end{aligned} \tag{39}$$

We emphasize that the dual gap (r^k_{dual}) computation may include errors due to an iterative projection on the box. The box projection multiplier values may not represent the actual multipliers until converged. Therefore during implementation, we only enforce primal feasibility and primal-dual gap. As the box projection is always feasible, the last two terms in the primal-dual gap ($(\alpha^k)^\top (l - y^k)$ and $(\beta^k)^\top (y^k - u)$) are always zero. Taking the primal residual and primal-dual gap provides plausible criteria for termination.

Convergence: With the criteria on the algorithm parameters that for ($\theta = 1$) and

$$\left(\frac{1}{\tau} - L_f\right)\frac{1}{\sigma} < \|K\|^2, \tag{40}$$

where, L_f is the Lipschitz constant (in our case the maximum value of the Q matrix), $\|K\|$ is the spectral norm of K ($K^\top = (A^\top, C^\top)$), the CP iterates converge to a primal-dual solution and the ergodic primal-dual gap decreases as $\mathcal{O}(1/k)$ where k is the number of iteration (Chambolle & Pock, 2016). The spectral norm of K is computed with power iteration algorithm (provided in the Appendix G.2). The Algorithm 1 can be further accelerated for strongly convex problems with less number of iterations by updating the parameters. We provide the accelerated method in the Appendix G.1. However, due to change in parameters each iteration may require additional

computation which limits the accelerated method. Additionally, we discuss the conditioning for the matrices in CP algorithm in Appendix G.3. In practice, conditioning reduces the number of iterations to converge the CP algorithm. Note that, when $Q \succeq 0$, the CP algorithm converges to the optimal solution.

Fixed parameter variation: We use the following fixed parameter values for the projection,

$$\begin{aligned} L = \max(\|K\|, 10^{-12}), \qquad L_f = \max_i(Q)_{ii} \\ \sigma = \frac{\texttt{safety}}{L}, \quad \tau = \frac{\texttt{safety}}{L_f + L}, \theta = 1 \end{aligned} \tag{41}$$

where $0 < \texttt{safety} \leq 1$ to ensure convergence, however `safety 0` provides asymptotic convergence. We keep `safety=0.95` in all experiments.

Handling Non-Affine Constraints: The CP algorithm is stated for more general case rather than only affine constraints. This leads us to have appropriate proximal operator (inequality multiplier update in Step 1) for convex constraint of specific form. For example, convex quadratic inequality constraints can be reformulated as second order conic constraint and can be solved exactly in the CP algorithm with a few change. We provide such a modified version of Algorithm 1 for quadratic inequality constraint which has exploitable convex structure (see Appendix G.4). If the nonlinear constraints have exploitable structure then similar modification can be made in the proximal operator for dual update. The advantage of this is that it satisfies the nonlinear constraints in single subproblem i.e., $k = 1$, so that the computational burden reduces from solving multiple subproblems to one subproblem.

G.1 Accelerated Chambolle-Pock algorithm

When, G is uniformly convex in SPP, the rate of convergence of the Chambolle-Pock algorithm can be improved to $\mathcal{O}(1/k^2)$, by choosing the parameters of the algorithm iteratively instead of a fixed value (Chambolle & Pock, 2016). Algorithm 4 is the accelerated version of the CP algorithm. Here, γ is the uniform convexity constant, for QP which is the $\lambda_{\min}(Q) = \min(Q)$ as Q is diagonal.

Algorithm 4: Accelerated Chambolle-Pock for QPs

`AcceleratedChambollePock`:

Input: Initial values y^0, λ^0, μ^0, p
Problem: $\min \frac{1}{2} y^\top Q y + c^\top y; \quad \text{s.t. } Ay = b;\ Cy \leq d;\ l \leq y \leq u$
Extract: $Q, c, A, b, C, d, l, u \leftarrow p$
Parameter: $\tau > 0, \sigma > 0, \theta = 1$ while $\left(\frac{1}{\tau} - L_f\right) \frac{1}{\sigma} < \|K\|^2$
compute $P = \text{diag}(1/(1 + \tau q_i)), \gamma = \max\left(\min_i q_i, 0\right)$
initialize $k \leftarrow 0, y^0 \leftarrow y^0, \lambda^0 \leftarrow \lambda^0, \mu^0 \leftarrow \mu^0, \bar{y}^0 \leftarrow y^0$
repeat
  $\lambda^{k+1} = \lambda^k + \sigma(A\bar{y}^k - b)$
  $\mu^{k+1} = \max(0, \mu^k + \sigma(C\bar{y}^k - d))$
  $\nu^{k+1} = P\left(y^k - \tau(A^\top \lambda^{k+1} + C^\top \mu^{k+1}) - \tau c\right)$
  $y^{k+1} = \Pi_{[l,u]}(\nu^{k+1})$
  $\bar{y}^{k+1} = y^{k+1} + \theta(y^{k+1} - y^k)$
  if $\gamma > 0$ **then**
    $\theta \leftarrow \dfrac{1}{\sqrt{1 + \gamma\tau}}$
    $\tau \leftarrow \theta\tau$
    $\sigma \leftarrow \dfrac{\sigma}{\theta}$
    $P \leftarrow \text{diag}\left(\dfrac{1}{1 + \tau q_i}\right)$
  $k \leftarrow k + 1$
until *termination criteria is satisfied*
Output: $z \leftarrow y^k, \lambda^k, \mu^k$

G.2 Power iteration for spectral norm computation

To estimate the operator norm required for step-size selection, we employ a randomized power iteration procedure on the matrix $K^\top K$, where

$$K = \begin{bmatrix} A \\ C \end{bmatrix}. \tag{42}$$

The method starts from a randomly initialized normalized vector and repeatedly applies the map $x \mapsto K^\top Kx = A^\top Ax + C^\top Cx$. At each iteration, the vector is normalized, and the corresponding Rayleigh quotient is used to approximate the dominant eigenvalue of $K^\top K$. The spectral norm is then estimated as $\|K\| \approx \sqrt{\lambda_{\max}(K^\top K)}$. This avoids explicitly forming $K^\top K$ and provides an efficient procedure for estimating the quantity needed in the primal-dual step size condition. Algorithm 5 algorithmic steps to estimate the norm.

Algorithm 5: Power Iteration for Estimating $\|K\|$

`EstimateKNorm`:

Input: $A,\ C$, random key, number of iterations N
Parameter: $\varepsilon > 0$
initialize Sample a random vector x^0
initialize $k \leftarrow 0$, $x^0 \leftarrow \frac{x^0}{\|x^0\|+\varepsilon}$
repeat
  $a^k \leftarrow Ax^k$
  $c^k \leftarrow Cx^k$
  $y^k \leftarrow A^\top a^k + C^\top c^k$
  $\lambda^k \leftarrow (x^k)^\top y^k$
  $x^{k+1} \leftarrow \dfrac{y^k}{\|y^k\| + \varepsilon}$
  $k \leftarrow k + 1$
until $k = N$
Output: $\|K\| \leftarrow \sqrt{\max(\lambda^{k-1}, 0)} + \varepsilon$

G.3 Conditioning of the matrices in Chambolle-Pock

Preconditioning such as Ruiz's equilibration(Ruiz, 2001) is often used as a heuristic for primal-dual algorithms(Stellato et al., 2020). Ruiz equilibration is applied as a preprocessing step to improve the numerical conditioning of the quadratic program before running the Chambolle–Pock iterations. The idea is to iteratively rescale the columns and rows of the problem data so that their magnitudes become more balanced. In the present setting, column scaling is applied to the primal variables, while separate row scalings are applied to the equality and inequality constraint matrices. This produces a scaled problem with better-balanced coefficients, which typically reduces the number of iterations in Chambolle-Pock algorithm. After solving the scaled problem, the primal and dual variables are mapped back to the original coordinates using the accumulated scaling factors.

Let $S \in \mathbb{R}^{n\times n}$ denote the diagonal column-scaling matrix for the primal variables, and let $R_{\text{eq}} \in \mathbb{R}^{n_{\text{eq}}\times n_{\text{eq}}}$ and $R_{\text{ineq}} \in \mathbb{R}^{n_{\text{ineq}}\times n_{\text{ineq}}}$ denote the diagonal row-scaling matrices for the equality and inequality constraints, respectively. The scaled quadratic program is written in terms of the transformed primal variable y_s, defined through $y = Sy_s$. Under this transformation, the scaled problem data become:

$$Q_s = SQS, \quad c_s = Sc, \quad A_s = R_{\text{eq}}AS, \quad b_s = R_{\text{eq}}b, \quad C_s = R_{\text{ineq}}CS, \quad d_s = R_{\text{ineq}}d, \tag{43}$$

with the bound vectors transformed as

$$l_s = S^{-1}l, \qquad u_s = S^{-1}u. \tag{44}$$

The Chambolle-Pock iterations are then performed on the scaled problem to obtain a solution $(y_s^*, \lambda_s^*, \mu_s^*)$. After convergence, the solution is mapped back to the original coordinates by reversing the applied scalings. In particular, the primal variable is recovered as $y^* = Sy_s^*$, while the dual variables associated with the equality and inequality constraints are recovered as $\lambda^* = R_{\text{eq}}\lambda_s^*$ and

$\mu^* = R_{\text{ineq}}\mu_s^*$. Thus, the optimization is carried out entirely in the equilibrated space, but the final primal and dual solutions are returned in the original problem coordinates. Algorithm 6 provides the step by step procedure we used for equilibration.

Algorithm 6: Modified Ruiz Equilibration for Box-Constrained QPs

`RuizEquilibrate:`

Input: Q, c, A, b, C, d, l, u, number of iterations N, safeguard $\varepsilon > 0$

Problem: $\min\ \frac{1}{2}y^\top Qy + c^\top y \quad \text{s.t.} \quad Ay = b,\ Cy \le d,\ l \le y \le u$

compute $n \leftarrow \text{length}(Q),\ n_{\text{eq}} \leftarrow \text{rows}(A),\ n_{\text{ineq}} \leftarrow \text{rows}(C)$

initialize $Q_s \leftarrow Q,\ c_s \leftarrow c,\ A_s \leftarrow A,\ b_s \leftarrow b,\ C_s \leftarrow C,\ d_s \leftarrow d,\ l_s \leftarrow l,\ u_s \leftarrow u$
$S \leftarrow I_n,\ R_e \leftarrow I_{n_{\text{eq}}},\ R_i \leftarrow I_{n_{\text{ineq}}}$
$k \leftarrow 0$

repeat

`// Column equilibration`

compute $(m_{\text{col}})_j = \max\Big(\max_i |(A_s)_{ij}|,\ \max_i |(C_s)_{ij}|,\ |(Q_s)_j|\Big),\ \forall j$

compute $\delta_j = \dfrac{1}{\sqrt{\max((m_{\text{col}})_j, \varepsilon)}},\ \forall j$

$A_s \leftarrow A_s\,\text{diag}(\delta), C_s \leftarrow C_s\,\text{diag}(\delta), Q_s \leftarrow Q_s \odot \delta^2, c_s \leftarrow c_s \odot \delta, l_s \leftarrow l_s \oslash \delta, u_s \leftarrow u_s \oslash \delta$

$S \leftarrow S\,\text{diag}(\delta)$

`// Equality-row equilibration`

if $n_{\text{eq}} \neq 0$ **then**

compute $(m_{\text{eq}})_i = \max_j |(A_s)_{ij}|,\ \forall i$

compute $\eta_i = \dfrac{1}{\sqrt{\max((m_{\text{eq}})_i, \varepsilon)}},\ \forall i$

$A_s \leftarrow \text{diag}(\eta)\,A_s, b_s \leftarrow \eta \odot b_s$

$R_e \leftarrow \text{diag}(\eta)\,R_e$

`// Inequality-row equilibration`

if $n_{\text{ineq}} \neq 0$ **then**

compute $(m_{\text{ineq}})_i = \max_j |(C_s)_{ij}|,\ \forall i$

compute $\zeta_i = \dfrac{1}{\sqrt{\max((m_{\text{ineq}})_i, \varepsilon)}},\ \forall i$

$C_s \leftarrow \text{diag}(\zeta)\,C_s, d_s \leftarrow \zeta \odot d_s$

$R_i \leftarrow \text{diag}(\zeta)\,R_i$

$k \leftarrow k + 1$

until $k = N$

Output: $Q_s, c_s, A_s, b_s, C_s, d_s, l_s, u_s, S, R_e, R_i$

G.4 Reformulation for convex quadratic constraints

The projection layer requires solving k subproblems to get feasible solution. However, solving a single subproblem is sufficient for affine constraints as well as specific classes of nonlinear constraints, such as second order conic constraints because of having appropriate proximal operator in the solving step. Quadratic constraints can be reformulated as second order conic constraints. For each quadratic inequality

$$y^\top C_i y + d_i^\top y \le e_i - E_i x, \tag{45}$$

let $C_i = R_i^\top R_i$ be a factorization of C_i. Assume $d_i \in \text{Range}(C_i)$, and define $q_i := \frac{1}{2}(R_i^\dagger)^\top d_i$, where $R_i^\dagger$ denotes the Moore-Penrose pseudo-inverse. Then

$$y^\top C_i y + d_i^\top y = \|R_i y + q_i\|_2^2 - \|q_i\|_2^2. \tag{46}$$

Substituting equation 46 into equation 45 yields $\|R_i y + q_i\|_2^2 \le e_i - E_i x + \|q_i\|_2^2$. Defining the radius $r_i(x) := \sqrt{e_i - E_i x + \|q_i\|_2^2}$ assuming $r_i(x)^2 \ge 0$. Then the quadratic inequality becomes

$$\|R_i y + q_i\|_2 \le r_i(x), \qquad i = 1, \ldots, m. \tag{47}$$

Lets consider the parametric convex QCQP

$$\begin{aligned} \min_{y} \quad & \frac{1}{2} y^\top Q y + c^\top y \\ \text{s.t.} \quad & Ay = b + Bx, \\ & y^\top C_i y + d_i^\top y \le e_i - E_i x, \qquad i = 1, \dots, m, \\ & l + Lx \le y \le u + Ux, \end{aligned} \tag{48}$$

where $Q \succeq 0$ and $C_i \succeq 0$ for all $i = 1, \dots, m$. With the reformulation the problem in Eq. 48 can be rewritten as

$$\begin{aligned} \min_{y} \quad & \frac{1}{2} y^\top Q y + c^\top y \\ \text{s.t.} \quad & Ay = b + Bx, \\ & \|R_i y + q_i\|_2 \le r_i(x), \qquad i = 1, \dots, m, \\ & l + Lx \le y \le u + Ux. \end{aligned} \tag{49}$$

Primal-Dual Saddle Form

The indicator of the Euclidean ball $\mathbb{B}(r_i(x)) := \{z : \|z\|_2 \le r_i(x)\}$ has conjugate

$$I^*_{\mathbb{B}(r_i(x))}(\mu_i) = r_i(x)\|\mu_i\|_2. \tag{50}$$

Using this identity, the reformulated problem equation 49 admits the saddle-point representation

$$\begin{aligned} \min_{y \in [\, l+Lx,\, u+Ux \,]} \max_{\lambda, \mu_i} \quad & \frac{1}{2} y^\top Q y + c^\top y + \lambda^\top \big(Ay - (b + Bx) \big) \\ & + \sum_{i=1}^{m} \mu_i^\top (R_i y + q_i) - \sum_{i=1}^{m} r_i(x)\|\mu_i\|_2. \end{aligned} \tag{51}$$

The corresponding primal proximal map is $\operatorname{prox}_{\tau G}(v) = (I + \tau Q)^{-1}(v - \tau c)$, where $G(y) := \frac{1}{2} y^\top Q y + c^\top y$. If Q is diagonal with diagonal entries q_j, then the proximal map reduces to

$$P := \operatorname{diag}\left(\frac{1}{1 + \tau q_j} \right), \qquad \operatorname{prox}_{\tau G}(v) = P(v - \tau c). \tag{52}$$

For each dual block, the proximal operator of $\sigma r_i(x)\|\cdot\|_2$ is the vector soft-thresholding map

$$\operatorname{prox}_{\sigma r_i(x)\|\cdot\|_2}(v) = \begin{cases} \left(1 - \dfrac{\sigma r_i(x)}{\|v\|_2} \right) v, & \|v\|_2 > \sigma r_i(x), \\ 0, & \|v\|_2 \le \sigma r_i(x). \end{cases} \tag{53}$$

The Chambolle-Pock algorithm for equation 49 is given in Algorithm 7.

Algorithm 7: Chambolle-Pock for Convex QCQPs

Input: Initial values $y^0, \lambda^0, \mu_1^0, \ldots, \mu_m^0$
Problem: $\min \frac{1}{2} y^\top Q y + c^\top y; \quad \text{s.t. } Ay = b + Bx;\ \|R_i y + q_i\|_2 \le r_i(x),\ i = 1, \ldots, m;\ l + Lx \le y \le u + Ux$
Parameter: $\tau > 0, \sigma > 0, \theta \in [0, 1]$
compute $P = (I + \tau Q)^{-1}$ `// or P = diag(1/(1+τq_j)) if Q is diagonal`
initialize $k \leftarrow 0,\ y^0 \leftarrow y^0,\ \lambda^0 \leftarrow \lambda^0,\ \mu_i^0 \leftarrow \mu_i^0\ (i = 1, \ldots, m),\ \bar{y}^0 \leftarrow y^0$
repeat
  $\lambda^{k+1} = \lambda^k + \sigma\big(A\bar{y}^k - (b + Bx)\big)$
  for $i = 1, \ldots, m$ **do**
    $v_i^k = \mu_i^k + \sigma(R_i \bar{y}^k + q_i)$
    $$\mu_i^{k+1} = \begin{cases} \left(1 - \dfrac{\sigma r_i(x)}{\|v_i^k\|_2}\right) v_i^k, & \|v_i^k\|_2 > \sigma r_i(x), \\ 0, & \|v_i^k\|_2 \le \sigma r_i(x) \end{cases}$$
  $g^{k+1} = A^\top \lambda^{k+1} + \sum_{i=1}^m R_i^\top \mu_i^{k+1}$
  $y^{k+1} = \Pi_{[\,l+Lx,\ u+Ux\,]}\big(P(y^k - \tau g^{k+1} - \tau c)\big)$
  $\bar{y}^{k+1} = y^{k+1} + \theta(y^{k+1} - y^k)$
  $k \leftarrow k + 1$
until *termination criteria is satisfied*
Output: $y^k, \lambda^k, \mu_1^k, \ldots, \mu_m^k$

Remarks on the Algorithmic Structure: Compared with the Chambolle-Pock algorithm for linearly constrained QPs, the equality update preserves the same form, except that the right-hand side becomes parameter dependent through $b + Bx$. The box projection also preserves the same structure, with shifted bounds $l + Lx$ and $u + Ux$. The only essential modification occurs in the inequality dual step: the linear inequality update based on the componentwise positive-part operator is replaced by a blockwise proximal update equation 53, which arises from the conjugate of the Euclidean ball indicator after completing the square in each quadratic constraint.

If desired, the projection onto the parameterized box can be written componentwise as

$$\Pi_{[\,l+Lx,\ u+Ux\,]}(v) = \min\big(\max(v,\ l + Lx),\ u + Ux\big). \tag{54}$$

Although it is possible to incorporate different constraint directly in the CP algorithm, in our implementation we keep linearized constraints for generalization of the framework.

H Primal-Dual solution of approximated problem

Let, the convex (nonlinear) optimization problem is defined as

$$\begin{aligned} \min_{y \in \mathbb{R}^n} \quad & f(y) \\ \text{s.t.} \quad & h(y) = 0, \\ & g(y) \le 0, \end{aligned} \tag{P1}$$

where, f is convex $\mathcal{C}^2$, g is convex $\mathcal{C}^1$ and h is affine. We assume that Slater's condition holds and y^* is the global minimizer of this problem. With first order Taylor approximation of the constraints and the second order Taylor approximation of the objective at a arbitrary point y_k we form the subproblem:

$$\begin{aligned} \min_{y \in \mathbb{R}^n} \quad & f(y_k) + \nabla f(y_k)^\top (y - y_k) + \frac{1}{2}(y - y_k)^\top H_d(y_k)(y - y_k) \\ \text{s.t.} \quad & h(y) = 0, \\ & g(y_k) + \nabla g(y_k)^\top (y - y_k) \le 0, \end{aligned} \tag{P2}$$

where, instead of the exact Hessian of the objective, we take only the diagonal hessian elements multiplied by a positive constant i.e., $H_d(y_k) = \rho \times \mathrm{diag}(\nabla^2 f(y_k))$ with $\rho > 0$.

Lemma 1: Second order Taylor approximation of a convex function is also convex.

Proof: If $f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex then $\nabla^2 f(y) \succcurlyeq 0$ for all y. Second order Taylor approximation at y_0 of f is defined as

$$T_2(y) = f(y_0) + \nabla f(y_0)^\top (y - y_0) + \frac{1}{2}(y - y_0)^\top \nabla^2 f(y_0)(y - y_0) \tag{55}$$

The hessian of $T_2(y)$ is $\nabla^2 T_2(y) = \nabla^2 f(y_0)$. Since f is convex, $\nabla^2 f(y_0) \succcurlyeq 0$. Therefore $T_2(y)$ is convex. □

Proposition: The quadratic approximation of the second order Taylor approximation of a convex function with only the diagonal Hessian (The quadratic approximation of the objective in P2) is also convex.

Proof: As the hessian of the second order Taylor approximation is positive semi-definite (PSD), the diagonal hessian is also PSD, therefore convex.

Lemma 2: If y^* is a global solution of the convex problem P1 and λ^*, μ^* are the dual solution, the subproblem P2 has the same primal-dual solution when $y_k = y^*$.

Proof: According to lemma 1, the subproblem is also convex. Therefore if a any point satisfies KKT conditions then it would be sufficient for global optimality. At $y_k = y^*$ Lagrangian for the original and the subproblem becomes

$$\begin{aligned}
\mathcal{L}_{P1} =& f(y) + \lambda^\top \nabla h(y) + \mu^\top g(y) \\
\mathcal{L}_{P2} =& f(y^*) + \nabla f(y^*)^\top (y - y^*) + \frac{1}{2}(y - y^*)^\top H_d(y^*)(y - y^*) + \lambda^\top h(y) \\
&+ \mu^\top \left[g(y^*) + \nabla g(y^*)^\top (y - y^*) \right]
\end{aligned} \tag{56}$$

Assuming x^* is the global minimizer of P1 and there exists unique dual variables any λ^* and $\mu^* \geq 0$. Then the KKT conditions for P1 becomes

$$\begin{aligned}
\textit{Stationarity:} \quad & \nabla f(y^*) + \sum_{i=1}^{n_{\text{eq}}} \lambda_i^* \nabla h_i(y^*) + \sum_{j=1}^{n_{\text{ineq}}} \mu_j^* \nabla g_j(y^*) = 0 \\
\textit{Primal feasibility:} \quad & g(y^*) \leq 0, \quad h(y^*) = 0 \\
\textit{Dual feasibility:} \quad & \mu \geq 0 \\
\textit{Complementary slackness:} \quad & \mu^* \odot g(y^*) = 0.
\end{aligned} \tag{57}$$

Defining $g(y^*) + \nabla g(y^*)^\top (y - y^*) = \tilde{g}(y)$ the KKT conditions for the subproblem becomes

$$\begin{aligned}
\textit{Stationarity:} \quad & \nabla f(y^*) + H_d(y^*)(y - y^*) + \sum_{i=1}^{n_{\text{eq}}} \lambda_i \nabla h_i(y) + \sum_{j=1}^{n_{\text{ineq}}} \mu_j^* \nabla g_j(y^*) = 0 \\
\textit{Primal feasibility:} \quad & \tilde{g}(y) \leq 0, h(y) = 0 \\
\textit{Dual feasibility:} \quad & \mu \geq 0 \\
\textit{Complementary slackness:} \quad & \mu \odot \tilde{g}(y) = 0.
\end{aligned} \tag{58}$$

At $y = y^*$, $\tilde{g}(y^*) = g(y^*) + \nabla g(y^*)^\top (y^* - y^*) = g(y^*) \leq 0$, $h(y^*) = 0$ as y^* is a feasible and optimal solution of original problem. Putting $y = y^*$ in the KKT system (Eq. 58) satisfies the primal feasibility and yields a linear system for the dual variables. By inspection, λ^*, μ^* is the solution for the system. Therefore (y^*, λ^*, μ^*) is a KKT point of the subproblem. As the problem is convex it is sufficient that y^* is also a global minimizer for P2. Moreover, both the problems P1 and P2 attains the same dual variables λ^* and μ^*. □

The motivation behind the lemma is to show that when the projection is done at near optimal solution (when the NN predicts close neighborhood), the projection pushes for optimality with obtaining primal-dual variables.

Graphical interpretation: We illustrate the complex events happening in the `NLPOpt-Net` framework graphically in Fig 4. The illustrations represent the following phenomena: (left) relaxation of

the original feasible domain: linearization creates a polytope defining the feasible region; (middle) the neural network is trained to minimize the loss within the polytope. The training pushes towards finding a better solution inside the polytope. Due to local convergence after k-layer, the projected output will be on the facet of the polytope. Moreover, due to the descent property of the projection layer, the solution after projection will remain no worse than raw backbone prediction (in practice better in most cases); (right) an ϵ-optimal solution of the original problem P can be obtained if the approximation is done near optimal point.

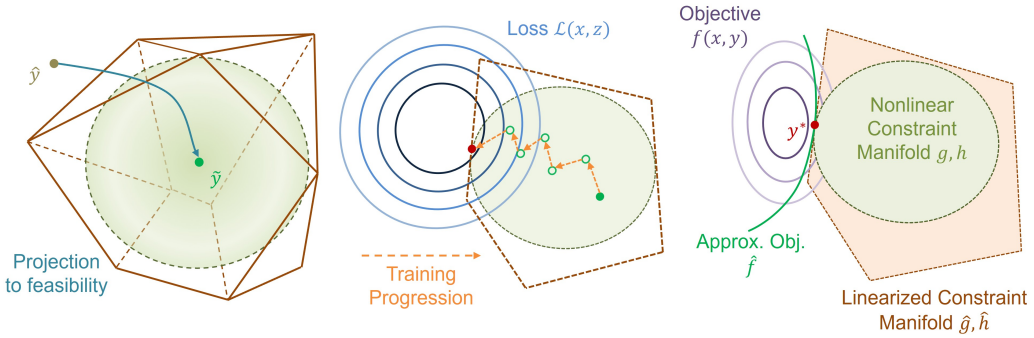

Figure 4: Graphical interpretation of the approximated layer. Left: the linearized model represents a polytope confining the original convex feasible domain. Middle: The neural network is learning to minimize the loss within the polytope, Right: the local exactness of the problem after linearization.

I BACKWARD GRADIENT OF THE PROJECTION

Projection as a Two Stage Mapping: The projection layer involves two tasks in sequence. The first is to create the matrices Q, A, C and the vectors c, b, d, l, u which define the QPs in SP. These matrices and vectors depend on the input x and network prediction $\hat{z}$. The second task is to solve the QPs using the CP algorithm. As the QPs in SP are convex, they converge with arbitrary initial points i.e., the solution $\tilde{z}$ does not directly depend on the initialization $\hat{z}$ but on $\mathcal{M}(x,z)$ where $\mathcal{M}(x,z) \equiv (Q,c,A,b,C,d,l,u)$. The operator $\mathcal{M}$ defines the conversion of the subproblem. The differentiability still holds due to the composition of the projection. Since $\mathcal{P}_o(x,z) = \mathcal{P}_o(\mathcal{M}(x,z))$, chain rule leads to

$$\left.\frac{\partial \mathcal{P}_o(x,z)}{\partial z}\right|_{z=\hat{z}} = \left.\frac{\partial \mathcal{P}_o(w)}{\partial w}\right|_{w=\mathcal{M}(x,\hat{z})} \left.\frac{\partial \mathcal{M}(x,z)}{\partial z}\right|_{z=\hat{z}}. \tag{59}$$

The second term of Eq. 59 can be obtained through automatic differentiation, which is not computationally expensive. We, therefore, provide the route for custom backward gradient computation of the first term.

Fixed-Point Formulation: Consider a fixed-point iteration of the form:

$$z^{k+1} = F(z^k; w), \tag{60}$$

where, $z \in \mathbb{R}^m$ denotes the vector of all primal and dual variables, $w = \mathcal{M}(x,z) \equiv (Q,c,A,b,C,d,l,u)$ denotes the collection of matrices and vectors defining the problem, and F is a continuously differentiable neighborhood of a fixed point. When the iteration converges to z^*, it satisfies $z^* = F(z^*; w)$. Defining the residual mapping G such that

$$G(z; w) := F(z; w) - z, \tag{61}$$

the fixed point satisfies $G(z^*; w) = 0$. The projection output can be then extracted as $\tilde{z}$, i.e., $\tilde{z} = [y^{*\top}, \lambda^{*\top}, \mu^{*\top}]^\top$.

Deploying Implicit Function Theorem: Assuming F is continuously differentiable and the Jacobian is non-singular, the implicit function theorem gives,

$$\left.\frac{\partial G(z,w)}{\partial z}\right. \left.\frac{\partial z}{\partial w}\right|_{z=z^*} + \left.\frac{\partial G(z,w)}{\partial w}\right|_{z=z^*} \implies \frac{\partial z^*}{\partial w} = -\left(\left.\frac{\partial G(z,w)}{\partial z}\right|_{z=z^*}\right)^{-1} \left.\frac{\partial G(z,w)}{\partial w}\right|_{z=z^*}. \tag{62}$$

Since $\frac{\partial G}{\partial z} = \frac{\partial F}{\partial z} - I$ (from Eq. 61), we obtain

$$\frac{\partial z^*}{\partial w} = \left(I - J_F(z^*)\right)^{-1} \frac{\partial F}{\partial w}, \tag{63}$$

where, $J_F(z^*) = \frac{\partial F}{\partial z}\Big|_{z=z^*}$. Now considering the loss function $\mathcal{L}(x, z^*)$ depend on the fixed point we get,

$$\frac{\partial \mathcal{L}}{\partial w} = g_z \frac{\partial z^*}{\partial w} \tag{64}$$

where, $g_z = \frac{\partial \mathcal{L}}{\partial z}\Big|_{z=\tilde{z}}$. Hence, the implicit expression gives,

$$\frac{\partial \mathcal{L}}{\partial w} = g_z^\top \left(I - J_F(z^*)\right)^{-1} \frac{\partial F}{\partial w} \tag{65}$$

Adjoint Formulation (VJP): The inversion may not be plausible in Eq. 65. Even if exists, in practice, constructing and computing the inverse may be expensive for high dimensions. Instead of inverting, we solve the following adjoint linear system:

$$\left(I - J_F(z^*)^\top\right) v = g_z. \tag{66}$$

The gradient with respect to p is then given by

$$\frac{\partial \mathcal{L}}{\partial w}\Big|_{w=\mathcal{M}(x,\hat{z})} = v^\top \frac{\partial F}{\partial w}. \tag{67}$$

We use the JAX (Bradbury et al., 2018) implementation of bi-conjugate gradient stable iteration to get the vector v.