Neural Compression of Long ADMM Trajectory for Multiparametric Quadratic Program

Liang Wu[1,*] **Honghui Zheng**[1,*,†] **Bo Yang**[2,*] **Xu Yang**[2,*] **Yilin Mo**[2] **Ján Drgoňa**[1]
[1]Johns Hopkins University [2]Tsinghua University
{wliang14, hzheng39, jdrgona1}@jh.edu
{yang-b21, yangx21}@mails.tsinghua.edu.cn, ylmo@tsinghua.edu.cn

Abstract

Solving large-scale multiparametric quadratic programs (mpQPs) in real time often requires thousands of iterative optimization updates, creating a major computational bottleneck in model predictive control and learning-enabled systems. This paper proposes *TQPNet*, a self-supervised neural compression framework that learns to compress long ADMM optimization trajectories into a compact neural warm start followed by a few refinement iterations. We derive a low-dimensional ADMM scheme operating on the primal–dual variables (x, β) and show that its iteration admits an equivalent ReLU-layer representation, termed *ADMM(ReLU)*. Building on this insight, TQPNet combines a multilayer perceptron with ReLU activations (*MLP(ReLU)*) and a small number of unrolled *ADMM(ReLU)* refinement layers. The network is trained offline using an optimality-condition-informed residual loss, eliminating the need for solver-generated labels. The proposed residual-based framework enables both self-supervised training and online solution certification through adaptive *ADMM(ReLU)* refinement. Experiments on large-scale mpQPs and real-time MPC applications demonstrate that *TQPNet* achieves high accuracy, strong generalization, and substantially faster inference than conventional optimization-based methods.

1 Introduction and Problem Statement

Quadratic programming (QP) is a fundamental optimization paradigm widely used across control, operations research, machine learning, finance, and related fields [1, 2]. Specifically, the widely used optimization-based control approach, model predictive control (MPC), can often be recast as a QP with time-varying parameters, referred to as a multiparametric QP (mpQP) [3].

This paper focuses on a convex mpQP problem as shown in (1), where $p \in \mathcal{P} \subset \mathbb{R}^{n_p}$ denotes the parametric vector. The problem data, including $Q = Q^\top \in \mathbb{R}^{n\times n} \succeq 0$, $A \in \mathbb{R}^{m\times n}$, $q(p) \in \mathbb{R}^n$ and $c(p) \in \mathbb{R}^m$ dependent on p, and $x \in \mathbb{R}^n$ denotes the decision vector. We assume that QP (1) has an optimal solution set $\mathcal{X}^*$, namely the set $\{x \in \mathbb{R}^n : Ax \geq c(p)\}$ is not empty. Note that this assumption is not restrictive, as infeasible instances can be equivalently handled via a slack-variable reformulation. Specifically, introducing slack variables $\xi \in \mathbb{R}^m$, we consider the penalized problem $\min_{x,\xi}\ \frac{1}{2}x^\top Qx + x^\top q(p) + \frac{\alpha}{2}\xi^\top\xi, \text{s.t. } Ax + \xi \geq c(p),\ \xi \geq 0$ where $\alpha > 0$ is a penalization scalar, which can be written as the standard mpQP formulation (1) by recasting $x \leftarrow \text{col}(x, \xi)$, $Q \leftarrow \begin{bmatrix} Q & 0_{n\times m} \\ 0_{m\times n} & \alpha I_m \end{bmatrix}$, $q(p) \leftarrow \begin{bmatrix} q(p) \\ 0_m \end{bmatrix}$, $A \leftarrow \begin{bmatrix} A & I_m \\ 0_{m\times n} & I_m \end{bmatrix}$, and $c(p) \leftarrow \begin{bmatrix} c(p) \\ 0_m \end{bmatrix}$.

$$\min_{x\in\mathbb{R}^n}\ \frac{1}{2}x^\top Qx + q(p)^\top x,\ \text{s.t. } Ax \geq c(p) \quad (1)$$

*Equal Contributions. †Corresponding Author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

The mpQP (1) arises from MPC formulations for both linear and nonlinear systems (e.g., via Koopman operator [4]). Despite the maturity of QP theory and algorithms [5, 6], solving large-scale mpQPs in real time often requires thousands of iterative optimization updates, which remain a major computational bottleneck in modern control, robotics, and learning-enabled decision-making systems.

This challenge becomes even more pronounced in scenarios requiring massively parallel optimization, such as reinforcement-learning-MPC pipelines and differentiable control architectures, where a large number of mpQPs must be solved simultaneously and repeatedly. In such settings, there is a growing need for optimization methods that simultaneously achieve: *(i)* fast inference, *(ii)* scalability in both offline and online phases, *(iii)* batch-parallel execution, *(iv)* differentiability for end-to-end learning integration, and *(v)* trustworthy solution refinement with optimality certification.

Related work. Existing approaches for solving mpQPs can be broadly categorized into several classes. *Online optimization-based solvers*, such as using interior-point [7, 8, 5, 9, 10], active-set [11, 12], and first-order methods [13, 14, 15, 16], offer convergence guarantees and optimality certificates, but do not exploit offline computation to accelerate online inference. *Offline explicit region-based methods* [3, 17, 18] rely on offline computation to precalculate the piecewise-affine solution mapping, enabling extremely fast online evaluation, but their offline construction suffers from the well-known combinatorial growth of critical regions, which quickly becomes prohibitive for high-dimensional problems. This limits their applicability to large-scale MPC. *Supervised neural approaches* [19, 20, 21] learn mappings from problem parameters to optimal decisions using solver-generated labeled data. While enabling fast inference, they require expensive offline supervision and generally lack mechanisms for adaptive refinement or solution certification. More recently, *learning-to-optimize and self-supervised approaches* [22, 23, 24, 25, 26, 27, 28, 29, 30] have explored learning iterative optimization structures, warm-start strategies, or optimization dynamics without requiring labeled optimal solutions. Finally, *differentiable optimization approaches* [31, 32, 33, 34, 35], enable end-to-end gradient propagation through optimization layers, but often rely on implicit differentiation involving large Hessian-related linear systems, which can incur substantial computational and memory overhead.

Motivated by these challenges, this paper proposes *TQPNet*, a self-supervised neural compression framework for mpQPs, illustrated in Fig. 1. The central idea is to learn to compress long ADMM algorithm trajectories into a compact neural warm start followed by only a few optimization refinement iterations.

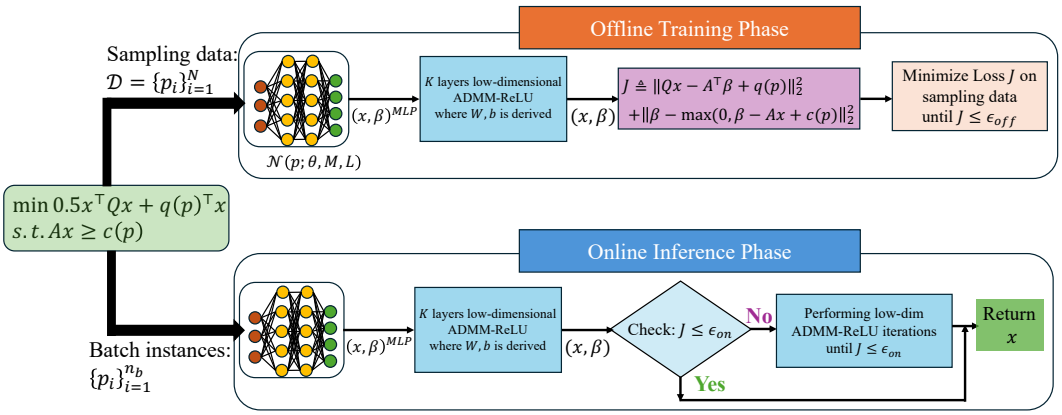


Figure 1: Architecture of *TQPNet* based on the hybrid *MLP(ReLU)-ADMM(ReLU)* neural network.

To enable this formulation, we first derive a reduced-state ADMM iteration scheme operating directly on the primal–dual variables (x, β). We further show that the resulting iteration admits an equivalent ReLU-layer representation, termed *ADMM(ReLU)*, enabling a neural-friendly and naturally batch-parallel implementation. Building on this insight and motivated by the strong expressive power of deep multilayer perceptrons with ReLU activations (*MLP(ReLU)*) for representing piecewise-affine structures, which naturally arise in mpQP solution mappings, TQPNet combines a multilayer perceptron with ReLU activations (MLP(ReLU)) and a small number of unrolled *ADMM(ReLU)* refinement layers. The *MLP(ReLU)* component learns a compact approximation of long optimization trajectories, while the *ADMM(ReLU)* layers provide optimization-based refinement and adaptive

certification. Rather than relying on solver-generated labels, the *MLP(ReLU)* parameters are trained offline in a self-supervised manner using an optimality-condition-informed residual loss over sampled parameter instances. The proposed residual-based formulation provides a unified mechanism for both offline training supervision and online solution verification. When the predicted solution does not satisfy a prescribed optimality tolerance, additional *ADMM(ReLU)* refinement iterations are triggered automatically to improve solution accuracy.

As a result, *TQPNet* combines the efficiency of amortized optimization with the reliability of iterative optimization refinement. Extensive experiments demonstrate that the proposed framework can compress thousands of ADMM iterations into a compact *MLP(ReLU)* followed by only a few *ADMM(ReLU)* refinement steps, leading to substantial acceleration on large-scale mpQPs and real-time MPC applications while preserving strong solution accuracy and generalization performance.

Contributions. Our main contributions can be summarized as follows,

- To the best of the authors' knowledge, this is among the first works to propose a low-dimensional ADMM with lower per-iteration cost and empirically faster convergence than standard ADMM and show its equivalence to a ReLU layer, enabling a differentiable and batch-parallel *ADMM(ReLU)* implementation.
- We propose *TQPNet*, a hybrid *MLP(ReLU)–ADMM(ReLU)* architecture that learns to compress long optimization trajectories into compact neural warm starts followed by a few optimization refinement iterations. *TQPNet* is an optimality-condition-informed self-supervised learning framework that eliminates the need for solver-generated labels while enabling adaptive online refinement and solution verification.
- More broadly, this work suggests a new paradigm for designing neural compressions of deep iterative algorithms and integrating them with the underlying optimization iterations to achieve faster online computation.

2 Method

2.1 Preliminary: full-dimensional ADMM algorithm

Different variable-splitting schemes lead to different ADMM algorithms. One standard choice is introducing an auxiliary variable $z = Ax \in \mathbb{R}^m$ and QP (1) can be reformulated as follows,

$$\min_{x,z} \ \frac{1}{2} x^\top Q x + q(p)^\top x + \mathcal{I}_{\mathcal{Z}}(z), \text{ s.t. } z - Ax = 0 \tag{2}$$

where $\mathcal{Z} = \{z \in \mathbb{R}^m : z \geq c(p)\}$ and $\mathcal{I}_{\mathcal{Z}}(\cdot)$ is the indicator function for the set $\mathcal{Z}$. Then, its augmented Lagrangian (with the scaled-form Lagrangian variable) is given by $\mathcal{L}_\rho(x, z, u) = \frac{1}{2} x^\top Q x + q(p)^\top x + \mathcal{I}_{\mathcal{Z}}(z) + \frac{\rho}{2}\|z - Ax + u\|_2^2$, where $u \in \mathbb{R}^m$ denotes the scaled dual Lagrangian variable for the equality constraint. Based on Ref. [36], the corresponding $(k+1)$-th ADMM update is shown in Eqn. (3).

$$x^{k+1} = \left(Q + \rho A^\top A\right)^{-1} \left(- q(p) + \rho A^\top (z^k + u^k)\right) \tag{3a}$$

$$z^{k+1} = \max\left(c(p), Ax^{k+1} - u^k\right) \tag{3b}$$

$$u^{k+1} = u^k + z^{k+1} - Ax^{k+1} \tag{3c}$$

Remark 2.1. ***(Requirements of Problem Data for Well-posedness of ADMM).*** *The x-update* (3a) *is well-defined if $Q + \rho A^\top A \succ 0$, or equivalently, $\text{null}(Q) \cap \text{null}(A) = \{0\}$, which ensures the uniqueness of the x-update* (3a)*.*

Lemma 2.2. *In the ADMM iteration* (3)*, the x-, z-, and u-updates are* ***performed sequentially****. To improve parallelizability and get faster computation speed, we can rewrite* (3) *as the following* ***ReLU-like*** *layer,*

$$\begin{bmatrix} x^{k+1} \\ z^{k+1} \\ u^{k+1} \end{bmatrix} = \text{ReLU}_{z \geq c(p)} \left(W_3 \begin{bmatrix} x^k \\ z^k \\ u^k \end{bmatrix} + b_3 \right), \tag{4}$$

where

$$W_3 \triangleq \begin{bmatrix} -\rho H(A^\top A) & 2\rho HA^\top & \rho HA^\top \\ A - \rho AH(A^\top A) & 2\rho AHA^\top - I & \rho AHA^\top - I \\ -A & I & I \end{bmatrix}, \quad b_3 \triangleq \begin{bmatrix} -Hq(p) \\ -AHq(p) \\ 0 \end{bmatrix}, \quad \textit{and}$$

$$\mathrm{ReLU}_{z \geq c(p)} \left(\begin{bmatrix} x \\ z \\ u \end{bmatrix} \right) \triangleq \begin{bmatrix} x \\ \max(c(p), z) \\ u \end{bmatrix}, \textit{ with } H \triangleq \left(Q + \rho A^\top A\right)^{-1}.$$

Proof. See Appendix A.1. □

Remark 2.3. *The ADMM-ReLU formulation* (4) *is* ***full-dimensional*** *over* $(x, z, u) \in \mathbb{R}^{n+2m}$*, with computational complexity* $O((n+2m)^2)$*, as demonstrated in [37].*

To simultaneously eliminate the data requirements noted in Remark 2.1 and reduce the per-iteration computational cost of ADMM (i.e., the ADMM-ReLU layer) discussed in Remark 2.3, we propose a computationally efficient **low-dimensional** ADMM-ReLU scheme.

2.2 Low-dimensional ADMM-ReLU layer

Before presenting the proposed low-dimensional ADMM-ReLU, we introduce a decomposition of A, termed the *Canonical Outer-Product Expansion* (COPE), as an analytical tool. Notably, the resulting low-dimensional formulation does not rely on the COPE decomposition and does not perform the decomposition in practice.

Definition 2.4. ***(COPE)****. Let* $A \in \mathbb{R}^{m \times n}$ *be a matrix with* $\mathrm{nnz}(A) = o$ *nonzero entries. Let* $\{(i_k, j_k, A_{i_k j_k})\}_{k=1}^{o}$ *be any enumeration of the nonzero entries of* A*, where* $A_{i_k j_k} \neq 0$*,* $i_k \in \{1, \ldots, m\}$*, and* $j_k \in \{1, \ldots, n\}$*. Then* A *admits the* canonical outer-product expansion

$$A = \sum_{k=1}^{o} A_{i_k j_k} \, e_{i_k}^m (e_{j_k}^n)^\top. \tag{5}$$

where $\{e_i^m\}$ *and* $\{e_j^n\}$ *are the canonical basis vectors in* $\mathbb{R}^m$ *and* $\mathbb{R}^n$*, respectively. Denote*

$$U \triangleq \sum_{k=1}^{o} A_{i_k, j_k} e_{i_k}^m (e_k^o)^\top \in \mathbb{R}^{m \times o}, \quad V \triangleq \sum_{k=1}^{o} e_{j_k}^n (e_k^o)^\top \in \mathbb{R}^{n \times o} \tag{6}$$

where $\{e_k^o\}$ *are the canonical basis vectors in* $\mathbb{R}^o$*, then*

$$UV^\top = A_{i_1, j_1} e_{i_1}^m (e_1^o)^\top e_1^o (e_{j_1}^n)^\top + \cdots + A_{i_o, j_o} e_{i_o}^m (e_o^o)^\top e_o^o (e_{j_o}^n)^\top = A,$$

which is the matrix form of COPE.

Lemma 2.5. *Let* $A \in \mathbb{R}^{m \times n}$ *be a (sparse) matrix each of whose rows and columns contains at least one nonzero entry, and let* (U, V) *defined in* (6) *be its COPE representation. Then, the matrices* $D_U \triangleq UU^\top \in \mathbb{R}^{m \times m}$ *and* $D_V \triangleq VV^\top \in \mathbb{R}^{n \times n}$ *are positive diagonal matrices, with*

$$(D_U)_{ij} = \begin{cases} A_{i,:} A_{i,:}^\top, & i = j, \\ 0, & i \neq j, \end{cases} \; i, j = 1, \ldots, m, \quad (D_V)_{ij} = \begin{cases} \mathrm{nnz}(A_{:,i}), & i = j, \\ 0, & i \neq j, \end{cases} \; i, j = 1, \ldots, n, \tag{7}$$

which indicates that the i*-th diagonal value of* (D_U) *is the sum of the squares of all non-zero elements in* $A_{i,:}$ *(the* i*-th row of* A*), and the* i*-th diagonal value of* (D_V) *is the number of non-zero elements in* $A_{:,i}$ *(the* i*-th column of* A*).*

Proof. See Appendix A.2. □

Note that the subsequent analysis does not require a full COPE decomposition, but only the inexpensive and trivial computation of D_V and D_U. Then, using the COPE decomposition $A = UV^\top$ and introducing the auxiliary variable $y \in \mathbb{R}^o$: $y = V^\top x$, we can equivalently reformulate QP (1) as the following optimization:

$$\min_{x,y} \; \frac{1}{2} x^\top Q x + q(p)^\top x + \mathcal{I}_{\mathcal{Y}}(y), \text{ s.t. } y - V^\top x = 0 \tag{8}$$

where the set $\mathcal{Y} \triangleq \{y \in \mathbb{R}^o : Uy \geq c(p)\}$ and $\mathcal{I}_{\mathcal{Y}}(\cdot)$ is the indicator function for the set $\mathcal{Y}$.

Then, its augmented Lagrangian (with the scaled-form Lagrangian variable) is given by $\mathcal{L}_\rho(x,y,u) = \frac{1}{2}x^\top Qx + q(p)^\top x + \mathcal{I}_\mathcal{Y}(y) + \frac{\rho}{2}\|y - V^\top x + u\|_2^2$, where $u \in \mathbb{R}^o$ denotes the scaled dual Lagrangian variable for the equality constraint. Based on Ref. [36], the corresponding $(k+1)$-th ADMM update is shown in (9).

$$y^{k+1} = \underset{y}{\operatorname{argmin}} \frac{1}{2}\|y - (V^\top x^k - u^k)\|_2^2, \text{ s.t. } Uy \geq c(p) \quad (9a)$$

$$x^{k+1} = (Q + \rho D_V)^{-1}\left(\rho V(y^{k+1} + u^k) - q(p)\right) \quad (9b)$$

$$u^{k+1} = u^k + y^{k+1} - V^\top x^{k+1} \quad (9c)$$

Lemma 2.6. *The y-update* (9a) *admits the following closed-form solution*

$$\beta^k = \max\left(0, D_U^{-1}(-Ax^k + Uu^k + c(p))\right) \quad (10a)$$

$$y^{k+1} = V^\top x^k - u^k + U^\top \beta^k \quad (10b)$$

Proof. See Appendix A.3. □

Theorem 2.7. ***(Low-dimensional ADMM-ReLU Layer).*** *The COPE-based ADMM iteration* (9) *can be equivalently rewritten as a low-dimensional ADMM-ReLU layer as follows,*

$$\begin{bmatrix} x^{k+1} \\ \beta^{k+1} \end{bmatrix} = \mathrm{ReLU}_{\beta\geq 0}\left(W_2 \begin{bmatrix} x^k \\ \beta^k \end{bmatrix} + b_2\right) \quad (11)$$

where $H_V \triangleq (Q + \rho D_V)^{-1}$, $\mathrm{ReLU}_{\beta\geq 0}\left(\left[\begin{smallmatrix} x \\ \beta \end{smallmatrix}\right]\right) \triangleq \left[\begin{smallmatrix} x \\ \max(0,\beta) \end{smallmatrix}\right]$, and

$$W_2 \triangleq \begin{bmatrix} \rho H_V D_V & \rho H_V A^\top \\ -D_U^{-1}A(2\rho H_V D_V - I) & I - 2\rho D_U^{-1} A H_V A^\top \end{bmatrix},$$

$$b_2 \triangleq \begin{bmatrix} -H_V q(p) \\ 2D_U^{-1} A H_V q(p) + D_U^{-1} c(p) \end{bmatrix},$$

with computational complexity $O((n+m)^2)$ per layer or iteration.

Proof. See Appendix A.4. □

In the proposed low-dimensional ADMM-ReLU (11), (x, β) can be represented as a single affine-ReLU operator evaluation, enabling a parallelizable and neural-friendly implementation that naturally supports batched computations. Furthermore, the unrolled form of (11) is inherently differentiable, allowing seamless end-to-end integration with learning-based modules. However, the ADMM iteration scheme may require many iterations when applied to ill-conditioned mpQPs. This motivates the development of a compact neural proxy to learn the solution–parameter mapping of mpQP in the offline phase.

2.3 Self-supervised Learning of Trustworthy Neural Proxy

A well-known result is that the solution of the mpQP (1) partitions the parameter vector space into critical regions (CRs), where a CR is the set of all parameter vectors p for which a certain combination of constraints is active at the optimizer of the multiparametric QP (1). For each CR, the optimizer x^* is a piecewise affine (PWA) function of p, see [3], i.e.,

$$\textit{Explicit PWA: } x^* = \begin{cases} K_1 p + f_1, & \text{if } p \in \mathcal{R}_1 \triangleq \{p \in \mathbb{R}^{n_p} | G_1 p \leq g_1\} \\ \vdots, & \vdots \\ K_{n_r} p + f_{n_r}, & \text{if } p \in \mathcal{R}_{n_r} \triangleq \{p \in \mathbb{R}^{n_p} | G_{n_r} p \leq g_{n_r}\} \end{cases} \quad (12)$$

One major drawback of the *explicit PWA* representation (12) of the multiparametric QP (1) is that the number of regions n_r required for an exact representation can grow exponentially with the problem dimension, leading to large memory requirements and rendering *explicit PWA* approaches (12) non-scalable.

2.3.1 Hybrid *MLP(ReLU)-ADMM(ReLU)* neural network structure

A scalable and memory-efficient alternative is a deep MLP with ReLU activation functions, motivated by the expressive power results in [38]. In this paper, we adopt the following *MLP(ReLU)-ADMM(ReLU)* parameterization as a proxy for the solution of multiparametric QP (1):

$$\begin{pmatrix} x^* \\ \beta^* \end{pmatrix} = \text{ADMM-ReLU}_{K,\rho}\Big(\mathcal{N}(p;\theta,M,L)\Big) \tag{13}$$

where, $\mathcal{N}(p;\theta,M,L) = F_{L+1}\circ G_L \circ F_L \circ \cdots \circ G_1 \circ F_1(p) : \mathbb{R}^{n_p} \to \mathbb{R}^n$ with $F_l(\xi_{l-1}) = W_l\xi_{l-1}+b_l$, and $G_l(\xi) = \max(0,\xi)$, for all $l = 1,\ldots,L$, (where $L \geq 2$); $\text{ADMM-ReLU}_{K,\rho}(\cdot)$ denotes the proposed low-dimensional ADMM-ReLU formulation in (11), with a fixed number of iterations (or layers) K and a penalty parameter ρ (can be learnable). The learnable parameter is given by $\theta = \{W_1, b_1, \ldots, W_{L+1}, b_{L+1}\}$, where M denotes the width of each hidden layer and L denotes the number of hidden layers.

Lemma 2.8 (see [38, Thm. 4, Cor. 5])**.** *Every neural network $\mathcal{N}(p;\theta,M,L)$ with ReLUs as activation functions and $M \geq n_p$ represents a PWA function. In addition, a lower bound on the maximal number of affine regions that the neural network represents is given by the following expression:*

$$\left(\prod_{l=1}^{L-1}\left\lfloor\frac{M}{n_p}\right\rfloor^{n_p}\right)\sum_{j=0}^{n_p}\begin{pmatrix} L \\ j \end{pmatrix} \sim \Omega\left(\left(\frac{M}{n_p}\right)^{(L-1)n_p} M^{n_p}\right).$$

Lemma 2.8 shows that the number of linear regions of a MLP grows exponentially with L and polynomially with M. Furthermore, the bound in Lemma 2.8 can be slightly improved when $M \geq 3n_p$; see [39].

2.3.2 Self-supervised learning loss design

Rather than using solver-generated data to build a supervised loss, this paper introduces an **optimality-condition-informed neural network** that constructs the loss within a self-supervised learning framework. Define

$$r(x,\beta;p) \triangleq \begin{bmatrix} Qx - A^\top\beta + q(p) \\ \beta - \max\left(0, \beta - Ax + c(p)\right) \end{bmatrix} \tag{14}$$

where it is immediate that $\|r(x,\beta;p)\|_2^2$ is piecewise-quadratic and convex on the solution manifold.

Theorem 2.9. *For any $p \in \mathcal{P}$, the multiparametric QP* (1) *admits a nonempty solution set $\mathcal{S}(p)$. Then, there exists a constant $\kappa(p) > 0$ such that $\operatorname{dist}\big((x,\beta),\mathcal{S}(p)\big) \leq \kappa(p)\,\|r(x,\beta;p)\|_2$ for all (x,β) in a neighborhood of $\mathcal{S}(p)$. Consequently, minimizing the **optimality-condition-informed** residual norm $\|r(x,\beta;p)\|_2$ drives the (x,β) toward the solution set of the multiparametric QP* (1)*.*

Proof. See Appendix A.5. □

By Theorem 2.9, substituting the *MLP(ReLU)-ADMM(ReLU)*-parameterized proxy (13) into the residual $\|r(x,\beta;p)\|_2^2$ naturally induces a self-supervised learning framework for training the neural proxy. Because the ℓ_2-norm of the optimality condition residual may have several orders of magnitude across different samples, in practice, we design the following training loss,

$$\mathcal{L}(\theta) = \mathbb{E}_{p\sim\mathcal{D}}\left[\log\left(\left\|r\Big(\text{ADMM-ReLU}_{K,\rho}\big(\mathcal{N}(p;\theta,M,L);p\big);p\Big)\right\|_2^2 + 1\right)\right]. \tag{15}$$

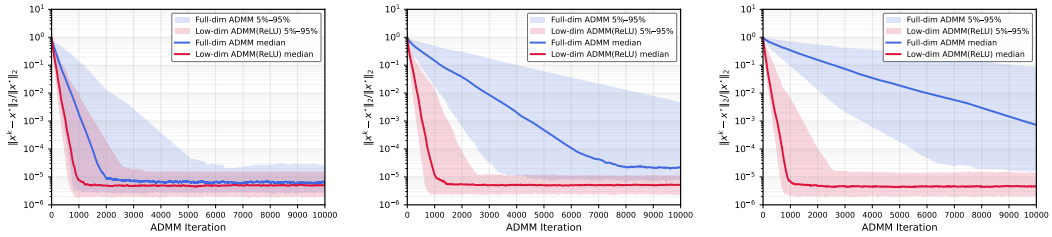
Minimizing $\mathcal{L}(\theta)$ is theoretically consistent with driving the prediction of the *MLP(ReLU)-ADMM(ReLU)*-parameterized proxy (13) toward the optimal primal-dual solution. The dataset $\mathcal{D}$ can be constructed by sampling the points $\{p_i\}_{i=1}^N$ uniformly over the domain $\mathcal{P}$. However, uniform sampling often provides insufficient coverage near the boundary of $\mathcal{P}$, even though these regions are particularly important for characterizing the behavior of the loss function and improving the accuracy of the surrogate model close to the boundary. To obtain better boundary coverage, this paper adopts a projected sampling strategy as shown in Algorithm 1. The unified algorithm for offline training and online inference with certification is presented in Algorithm 2.

3 Numerical Experiments

We evaluate *TQPNet* on four increasingly complex experiments on a single RTX 3080 (per-experiment unrolling depth K_{train} in Appendix C.3). Code: `https://github.com/SOLARIS-JHU/TQPNet`.

3.1 Advantages of *MLP(ReLU)-ADMM(ReLU)*

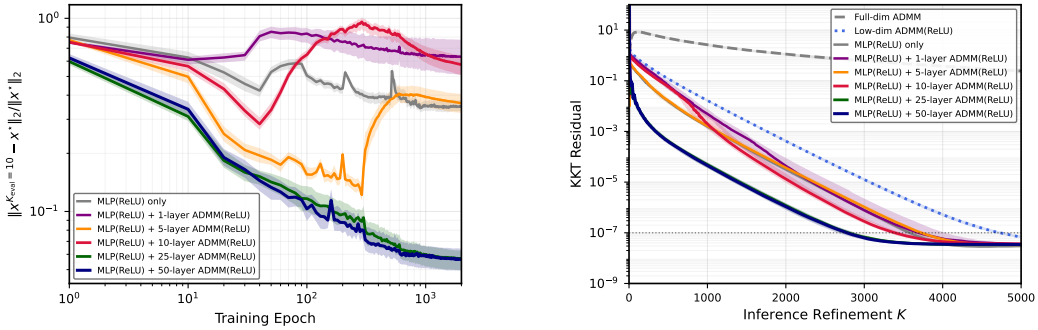
Empirically faster convergence of low-dimensional ADMM-ReLU. We evaluate the convergence behavior of the proposed low-dimensional ADMM-ReLU (9) against the standard full-dimensional ADMM (3) on 100 independently generated QP instances with dimensions $n = 500$ and $m = 1000$. The constraint matrix $A \sim \text{scale} \times \mathcal{U}([0,1]^{m \times n})$ is constructed as a sparse matrix with density 0.1 and the right-hand side is sampled as $b \sim \text{scale} \times \mathcal{U}([0,1]^m)$. Fig. 2 presents the convergence statistics on three different $\text{scale} = \{2, 4, 6\}$ settings across problem instances, including the median convergence trajectory and the corresponding 5%–95% percentile band over the 100 randomly generated QPs. The results indicate that the proposed low-dimensional ADMM-ReLU (9) consistently achieves faster convergence behavior than the standard full-dimensional ADMM (3) across problem instances. We **do not** aim to demonstrate that the proposed low-dimensional ADMM is **universally faster** than the standard full-dimensional ADMM, since both approaches may still require a large number of iterations in certain cases. Rather, this limitation motivates us to develop an offline self-supervised MLP(ReLU) neural compression of long ADMM iterations to achieve faster online inference.



(a) scale $= 2$ (b) scale $= 4$ (c) scale $= 6$

Figure 2: Convergence statistics over 100 random QPs for the low-dimensional ADMM-ReLU and the full-dimensional ADMM.

***MLP(ReLU)* compresses long *ADMM(ReLU)* iterations.** Even after the low-dimensional reformulation, a typical mpQP at n=100, m=200, n_p=20 still requires 4729 cold-start *ADMM(ReLU)* iterations to reach ε=10^{-7} (Fig. 3b). We amortize this overhead into a learned warm start: an *MLP(ReLU)* maps $p \mapsto (x_0, \beta_0)$, trained end-to-end through K_{train} unrolled *ADMM(ReLU)* layers under the KKT loss. The depth K_{train} governs whether back-propagation sees warm-start error or residual ADMM-convergence noise: shallow unrolling ($K_{\text{train}} \leq 10$) stays in the latter regime and fails to train, whereas K_{train}=25 crosses into the former and saturates the achievable accuracy (Fig. 3a; the required K_{train} grows with n, Appendix C). The trained warm start then cuts the inference budget by up to 1.8× (Fig. 3b, Tab. 2). We denote this *MLP(ReLU)-ADMM(ReLU)* hybrid (13) as *TQPNet*; the supervised baseline *LQPNet* (Label-supervised QP Net) shares the same MLP backbone but is trained directly on OSQP labels via MSE.



(a) Training. (b) Inference.

Figure 3: Ablation of K_{train}. (a) Test relative error vs. epoch at fixed K_{eval}=10; (b) post-training KKT residual vs. refinement steps K. $K_{\text{train}} \leq 10$ fails to compress; $K_{\text{train}} \geq 25$ stabilizes training and accelerates inference 1.7–1.8× over *ADMM(ReLU)*.

3.2 Generalizability and Scalability

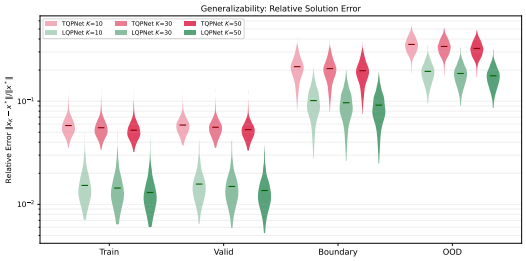


Figure 4: Relative solution error across parameter sets at n=100, m=200. *LQPNet* is more accurate in distribution, but the TQP–LQP gap shrinks under distribution shift.

Out-of-distribution robustness. A key concern for learned solvers is their robustness under distribution shift. We train *TQPNet* and *LQPNet* on n=100, m=200, n_p=20 random mpQPs with parameters sampled from $p \sim \mathcal{U}[-1,1]^{20}$, and evaluate on progressively shifted ranges (Train/Valid: $\mathcal{U}[-1,1]$, Boundary: $\mathcal{U}[-2,2]$, OOD: $\mathcal{U}[-3,3]$; 500 instances each). In distribution, *LQPNet* is more accurate (*TQPNet* 5.8×10^{-2} vs. *LQPNet* 1.5×10^{-2} at K=10; Table 3), but the TQP–LQP gap narrows monotonically under shift, from 3.9× in-distribution to 1.8× on OOD (Fig. 4): *TQPNet*'s KKT-residual training learns features more invariant to parameter shift than direct regression on $x^\star$. *TQPNet* additionally outputs the dual variable β as a per-instance KKT certificate and trains without OSQP labels.

MPC scalability. We benchmark MPC-structured QPs generated following Ref. [40] across nine scales (S0–S8, $n+m$ = 60–2800) using closed-loop simulations over T=200 steps. Fig. 5(a) shows that *TQPNet* (K=10) maintains near-constant latency (∼1.3 ms) across all problem sizes, whereas OSQP exhibits significant growth from 0.03 ms to 13 ms as the scale increases. The crossover occurs at S4 ($n+m$=875), beyond which *TQPNet* becomes faster, achieving a 7× speedup at S8 ($n+m$=2800).

Explicit MPC (using MPT3 [18]) already requires 3123 regions and incurs 4.1 ms latency at S1 ($n+m$=140), making it slower than *TQPNet* at small scales and infeasible beyond S2 due to exponential region growth. Table 4 further shows that both *TQPNet* and *LQPNet* match OSQP in tracking RMSE and closed-loop cost across all scales, while *ADMM(ReLU)* with cold initialization (K=50) exhibits a noticeable performance gap, highlighting the importance of learned warm starts.

Fig. 5(b) compares offline preparation costs. *TQPNet*'s unsupervised training pipeline avoids expensive OSQP label generation and becomes more efficient than *LQPNet*'s supervised approach for $n+m \geq 2100$, demonstrating superior scalability in both online inference and offline training.

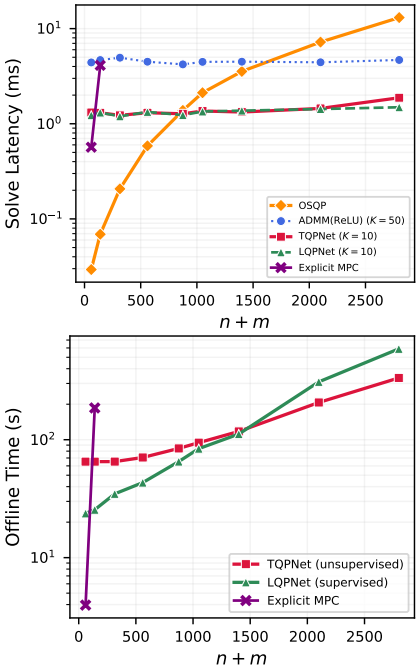


Figure 5: (a) Solve latency. (b) Offline time.

3.3 Application to Quadrupedal Locomotion

Complementing the single-UAV experiment in Appendix E, we now consider quadrupedal locomotion, where many QPs must be solved per control cycle. Quadrupedal locomotion control is formulated via convex MPC [41] running at 50 Hz over a prediction horizon of 16 steps with sampling period $\Delta t = 0.05\,\mathrm{s}$. At each control cycle, the MPC problem is parameterized by $n_p = 18$ parameters, including the body roll–pitch–yaw angles, center-of-mass height, desired linear and angular velocities, and gait-related parameters (e.g., contact schedule and swing-phase indicators), and reduces to a QP with $n = 192$ decision variables and $m = 384$ inequality constraints. *TQPNet* is trained offline on QP instances sampled from the locomotion parameter distribution over $\mathcal{P} \subset \mathbb{R}^{18}$, without any environment interaction or solver-generated labels, with training completing in under 5 minutes on a single RTX 3080. The trained proxy is subsequently deployed in massively parallel simulation using Isaac Gym [42], in which each simulated robot corresponds to an independent QP instance that is solved concurrently within a single control cycle.

Fig. 6 reports the wall-clock solve latency as a function of the number of parallel environments (equivalently, the number of simultaneously solved QP instances). *TQPNet* exploits GPU-native batch matrix operations inherited from the *MLP(ReLU)-ADMM(ReLU)* architecture, maintaining near-constant per-batch latency as the environment count grows. In contrast, qpth [31] incurs significantly higher latency due to its reliance on interior-point solvers, and *ADMM(ReLU)* with cold initialization requires substantially more refinement iterations without a learned warm start.

Fig. 7 demonstrates (a) relative cost error against the OSQP optimum and (b) maximum constraint violation, as functions of refinement steps K. *TQPNet* attains a cost error below 10^{-2} and a constraint violation below 10^{-2} within fewer than $K{=}20$ refinement steps, whereas cold-started *ADMM(ReLU)* requires more than $K{=}60$ iterations to reach a comparable accuracy. This gap directly quantifies the value of the learned warm start: the *MLP(ReLU)* initializer effectively compresses hundreds of ADMM iterations into a single forward pass while preserving certified solution quality. These results, combined with the batch-latency advantage in Fig. 6(b), show that *TQPNet* delivers high-fidelity QP solutions at the throughput required by parallel reinforcement-learning pipelines in robotics.

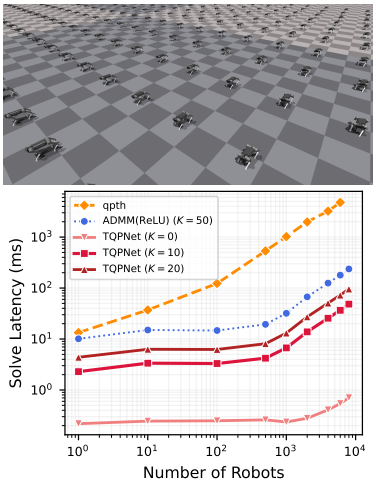


Figure 6: (a) 4096 robots concurrently controlled by *TQPNet*-based MPC in Isaac Gym. (b) Solve latency vs. number of parallel environments.

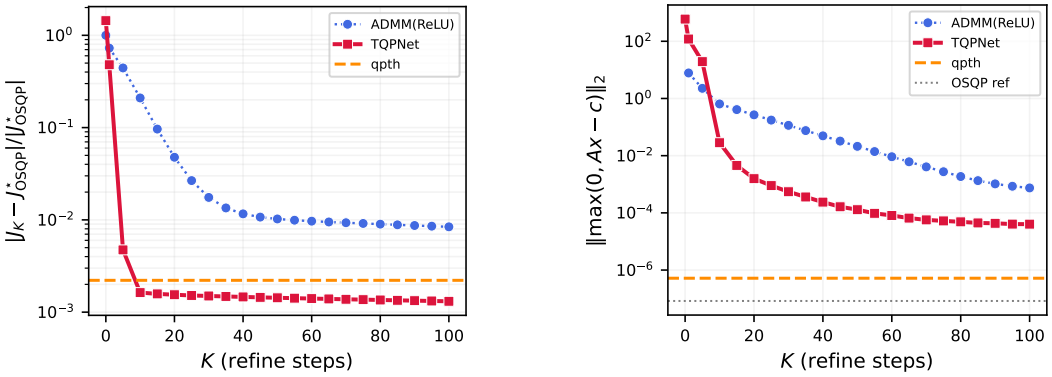


Figure 7: Solution quality vs. refinement steps K on the quadruped locomotion MPC. (a) Relative QP cost error vs. refinement steps K. (b) Constraint violation vs. refinement steps K.

4 Conclusion

This paper proposed *TQPNet*, a self-supervised neural compression framework for mpQP that compresses long ADMM optimization trajectories into a compact neural warm start followed by a few refinement iterations. By deriving a low-dimensional ADMM and its equivalent ReLU-layer representation, we use a hybrid *MLP(ReLU)-ADMM(ReLU)* architecture that combines amortized optimization with optimization-based refinement, enabling adaptive online certification. Experiments demonstrate substantial acceleration on large-scale mpQPs and real-time MPC applications while preserving strong solution accuracy and scalability. More broadly, this work suggests a new paradigm for neural compression of iterative optimization algorithms.

Limitations include the lack of a rigorous convergence analysis characterizing when the proposed low-dimensional ADMM outperforms the standard full-dimensional ADMM. In particular, since infinitely many COPE decompositions may exist for a given matrix, it remains unclear which decomposition yields the best convergence behavior.

Future work includes extending the proposed framework to more general classes of optimization, such as multiparametric mixed-integer QPs or nonlinear programs, as well as exploring hardware-aware implementations to further improve real-time performance.

Acknowledgments and Disclosure of Funding

This research was supported by the Ralph O'Connor Sustainable Energy Institute at Johns Hopkins University, and by the U.S. DOE Office of Science, ASCR program under the Scientific Discovery through Advanced Computing (SciDAC) Institute "LEADS: LEarning-Accelerated Domain Science." The authors declare no competing interests.

References

[1] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 2006.

[2] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[3] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[4] M. Korda and I. Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018.

[5] L. Wu, W. Xiao, and R. D. Braatz. EIQP: Execution-Time-Certified and Infeasibility-Detecting QP Solver. *IEEE Transactions on Automatic Control*, 71(5):2934–2949, 2026.

[6] L. Wu and R. D. Braatz. A Quadratic Programming Algorithm with $O(n^3)$ Time Complexity. *IEEE Transactions on Automatic Control*, 71(9):5709–5724, 2026.

[7] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[8] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11(1-4):451–484, 1999.

[9] L. Wu, Y. Che, R. D. Braatz, and J. Drgoňa. A Time-Certified Predictor-Corrector IPM Algorithm for Box-QP. *IEEE Control Systems Letters*, 9:3059–3064, 2025.

[10] A. Wächter and L. 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.

[11] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27(1):1–33, 1983.

[12] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[13] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[14] H. Lu and J. Yang. A practical and optimal first-order method for large-scale convex quadratic programming. *Mathematical Programming*, 215(1):771–808, 2026.

[15] L. Wu, B. Yang, J. Li, X. Yang, Y. Mo, Y. Shi, A. D. Ames, and J. Drgoňa. πMPC: A Parallel-in-horizon and Construction-free NMPC solver. *arXiv preprint arXiv:2601.14414*, 2026.

[16] P. Patrinos and A. Bemporad. An accelerated dual gradient-projection algorithm for embedded linear model predictive control. *IEEE Transactions on Automatic Control*, 59(1):18–33, 2014.

[17] P. Tøndel, T. A. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3):489–497, 2003.

[18] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari. Multi-parametric toolbox 3.0. In *2013 European Control Conference (ECC)*, pages 502–510. IEEE, 2013.

[19] B. Karg and S. Lucia. Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics*, 50(9):3866–3878, 2020.

[20] B. Karg and S. Lucia. Deep learning-based embedded mixed-integer model predictive control. In *2018 European Control Conference (ECC)*, pages 2075–2080. IEEE, 2018.

[21] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and O. Lucia. Deep learning-based model predictive control for resonant power converters. *IEEE Transactions on Industrial Informatics*, 17(1):409–420, 2021.

[22] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

[23] J. Ichnowski, P. Jain, B. Stellato, G. Banjac, M. Luo, F. Borrelli, J. E. Gonzalez, I. Stoica, and K. Goldberg. Accelerating quadratic optimization with reinforcement learning. *Advances in Neural Information Processing Systems*, 34:21043–21055, 2021.

[24] R. Sambharya, G. Hall, B. Amos, and B. Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024.

[25] P. L. Donti, D. Rolnick, and J. Z. Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.

[26] L. Lüken and S. Lucia. Self-supervised learning of iterative solvers for constrained optimization. *Results in Control and Optimization*, 23:100751, 2026.

[27] E. King, J. Kotary, F. Fioretto, and J. Drgoňa. Metric learning to accelerate convergence of operator splitting methods. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 1553–1560. IEEE, 2024.

[28] H. T. Nguyen and P. L. Donti. FSnet: Feasibility-seeking neural network for constrained optimization with guarantees. *Advances in Neural Information Processing Systems*, 38, 2025.

[29] T. Tran, B. Nguyen, and T. X. Nghiem. HUANet: Hard-Constrained Unrolled ADMM for Constrained Convex Optimization. *arXiv preprint arXiv:2604.13179*, 2026.

[30] R. Wang, P. Patrinos, and A. Bemporad. Parametric nonconvex optimization via convex surrogates. *arXiv preprint arXiv:2604.05640*, 2026.

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

[32] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32, 2019.

[33] A. Butler and R. H. Kwon. Efficient differentiable quadratic programming layers: an ADMM approach. *Computational Optimization and Applications*, 84(2):449–476, 2023.

[34] M. Besançon, J. Dias Garcia, B. Legat, and A. Sharma. Flexible Differentiable Optimization via Model Transformations. *INFORMS Journal on Computing*, 36(2):456–478, 2024.

[35] A. Paulus, M. Rolinek, V. Musil, B. Amos, and G. Martius. CombOptNet: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 8443–8453, 2021.

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

[37] A. L. Bishop, J. Z. Zhang, S. Gurumurthy, K. Tracy, and Z. Manchester. ReLU-QP: A GPU-Accelerated Quadratic Programming Solver for Model-Predictive Control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13285–13292. IEEE, 2024.

[38] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems*, 27, 2014.

[39] T. Serra, C. Tjandraatmadja, and S. Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, 2018.

[40] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010.

[41] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim. Dynamic locomotion in the MIT Cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[42] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac Gym: High Performance GPU Based Physics Simulation for Robot Learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. NeurIPS, 2021.

[43] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[44] R. T. Rockafellar and R. J. B. Wets. *Variational analysis*. Springer, 1998.

[45] A. Bemporad and C. Rocchi. Decentralized hybrid model predictive control of a formation of unmanned aerial vehicles. *IFAC Proceedings Volumes*, 44(1):11900–11906, 2011.

A Proofs

A.1 Proof of Lemma 2.2

Proof. For the ADMM iteration (3), putting u-update before x- and z- updates, which is equivalent to a special initialization for u^0 at the cost of one iteration lag in u-update and still preserves the convergence guarantee, results in

$$u^{k+1} = u^k + z^k - Ax^k \tag{16a}$$
$$x^{k+1} = H\left(-q(p) + \rho A^\top \left(z^k + u^{k+1}\right)\right) \tag{16b}$$
$$z^{k+1} = \max\left(c(p), Ax^{k+1} - u^{k+1}\right) \tag{16c}$$

Substituting Eqn. (16a) into Eqn. (16b), and then substituting Eqns. (16a)–(16b) into Eqn. (16c), results in the ReLU-like layer (4), which completes the proof. □

A.2 Proof of Lemma 2.5

Proof. Based on the definition of U, V in (6), we have

$$\begin{aligned} UU^\top &= (A_{i_1,j_1})^2 e_{i_1}^m (e_1^o)^\top e_1^o (e_{i_1}^m)^\top + \cdots + (A_{i_o,j_o})^2 e_{i_o}^m (e_o^o)^\top e_o^o (e_{i_o}^m)^\top \\ &= \sum_{i_k=1} (A_{1,j_k})^2 e_1^m (e_1^m)^\top + \cdots + \sum_{i_k=m} (A_{m,j_k})^2 e_m^m (e_m^m)^\top \\ &= \begin{bmatrix} A_{1,:}A_{1,:}^\top & & & \\ & A_{2,:}A_{2,:}^\top & & \\ & & \ddots & \\ & & & A_{m,:}A_{m,:}^\top \end{bmatrix} \in \mathbb{R}^{m\times m} \end{aligned}$$

and

$$
\begin{aligned}
VV^\top &= e^n_{j_1}(e^o_1)^\top e^o_1(e^n_{j_1})^\top + e^n_{j_2}(e^o_2)^\top e^o_2(e^n_{j_2})^\top + \cdots + e^n_{j_o}(e^o_o)^\top e^o_o(e^n_{j_o})^\top \\
&= e^n_{j_1}(e^n_{j_1})^\top + e^n_{j_2}(e^n_{j_2})^\top + \cdots + e^n_{j_o}(e^n_{j_o})^\top \\
&= \sum_{j_k=1} e^n_1(e^n_1)^\top + \sum_{j_k=2} e^n_2(e^n_2)^\top + \cdots + \sum_{j_k=n} e^n_n(e^n_n)^\top \\
&= \begin{bmatrix} \mathrm{nnz}(A_{:,1}) & & & \\ & \mathrm{nnz}(A_{:,2}) & & \\ & & \ddots & \\ & & & \mathrm{nnz}(A_{:,n}) \end{bmatrix} \in \mathbb{R}^{n\times n},
\end{aligned}
$$

To clarify the concept, we offer an example: given a 3×5 matrix with 9 non-zero elements as follows,

$$
A = \begin{bmatrix} 1 & 0 & 4 & 6 & 8 \\ 0 & 0 & 5 & 0 & 9 \\ 2 & 3 & 0 & 7 & 0 \end{bmatrix},
$$

then its COPE decomposition is shown below:

$$
U = \begin{bmatrix} 1 & 0 & 0 & 4 & 0 & 6 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 9 \\ 0 & 2 & 3 & 0 & 0 & 0 & 7 & 0 & 0 \end{bmatrix},\ V = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}
$$

with

$$
D_U = UU^\top = \begin{bmatrix} 117 & 0 & 0 \\ 0 & 106 & 0 \\ 0 & 0 & 62 \end{bmatrix},\ D_V = VV^\top = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}.
$$

This completes the proof. □

A.3 Proof of Lemma 2.6

Proof. The KKT condition of the optimization problem (9a) is shown below:

$$
\begin{aligned}
& y - (V^\top x^k - u^k) - U^\top \beta = 0 \\
& s = Uy - c(p) \geq 0 \\
& s_i \beta_i = 0,\ s_i \geq 0,\ \beta_i \geq 0,\ i = 1, \cdots, m
\end{aligned}
$$

where $\beta \in \mathbb{R}^m$ and $s \in \mathbb{R}^m$ denote the Lagrangian and slack variables, respectively. Eliminating y via $y = V^\top x^k - u^k + U^\top \beta$ can result in a trivial linear complementarity problem as follows,

$$
\begin{aligned}
& s = U(V^\top x^k - u^k + U^\top \beta) - c(p) = Ax^k - Uu^k + D_U \beta - c(p) \geq 0 \\
& s_i \beta_i = 0,\ s_i \geq 0,\ \beta_i \geq 0,\ i = 1, \cdots, m
\end{aligned}
$$

which admits a closed-form solution $\beta = \max\left(0, D_U^{-1}(-Ax^k + Uu^k + c(p))\right)$, due to D_U being a positive diagonal matrix. This completes the proof. □

A.4 Proof of Theorem 2.7

Proof. By Lemma 2.6, we can replace (9a) with Eqn. (10) and eliminate y^{k+1}, which results in

$$
\begin{aligned}
\beta^k &= \max\left(0, D_U^{-1}(-Ax^k + Uu^k + c(p))\right) \\
x^{k+1} &= (Q + \rho D_V)^{-1}\left(\rho V(V^\top x^k - u^k + U^\top \beta^k + u^k) - q(p)\right) \\
&= (Q + \rho D_V)^{-1}\left(\rho D_V x^k + \rho A^\top \beta^k - q(p)\right) \\
u^{k+1} &= u^k + (V^\top x^k - u^k + U^\top \beta^k) - V^\top x^{k+1} = V^\top(x^k - x^{k+1}) + U^\top \beta^k
\end{aligned}
$$

Furthermore, we have

$$\begin{aligned}\beta^{k+1} &= \max\left(0, D_U^{-1}(-Ax^{k+1} + Uu^{k+1} + c(p))\right) \\ &\quad \left(\text{where } Uu^{k+1} = UV^\top(x^k - x^{k+1}) + UU^\top\beta^k = A(x^k - x^{k+1}) + D_U\beta^k\right) \\ &= \max\left(0, D_U^{-1}(-Ax^{k+1} + A(x^k - x^{k+1}) + D_U\beta^k + c(p))\right) \\ &= \max\left(0, \beta^k - D_U^{-1}A(2x^{k+1} - x^k) + D_U^{-1}c(p)\right)\end{aligned}$$

By adopting a special initialization $u^0 = 0$ and $\beta^0 = \max(0, D_U^{-1}(c(p) - Ax^0))$, we can eliminate the u-update and obtain a low-dimensional ADMM iteration

$$\begin{aligned}x^{k+1} &= (Q + \rho D_V)^{-1}\left(\rho D_V x^k + \rho A^\top\beta^k - q(p)\right) \\ \beta^{k+1} &= \max\left(0, \beta^k - D_U^{-1}A(2x^{k+1} - x^k) + D_U^{-1}c(p)\right)\end{aligned} \tag{17}$$

which can be easily rewritten as a ReLU formulation as shown in Eqn. (11). Moreover, Eqn. (11) involves the matrix-vector multiplication with the dimension $n + m$, thus the computation complexity is $O((n + m)^2)$. This completes the proof. □

A.5 Proof of Theorem 2.9

Proof. The Karush–Kuhn–Tucker (KKT) condition (see [43, Ch. 5]) for mpQP (1) is given by

$$Qx + q(p) - A^\top\beta = 0, \tag{18a}$$

$$y - Ax + c(p) = 0, \tag{18b}$$

$$y \geq 0, \quad \beta \geq 0, \quad y \odot \beta = 0, \tag{18c}$$

where $\odot$ denotes the Hadamard product. Eqn. (18c) can be equivalently rewritten as $\beta - \max(0, \beta - y) = 0 \Leftrightarrow \beta - \max(0, \beta - Ax + c(p)) = 0$ where the equivalence follows by substituting Eqn. (18b). Then, combining this expression with (18a) yields the KKT-informed residual defined in (14). The piecewise affine structure of the KKT mapping in (18) implies, through standard Hoffman-type error bounds and metric subregularity results [44], that the distance to the solution set is locally upper bounded by the norm of the KKT residual, so there exists a constant $\kappa(p) > 0$ such that $\operatorname{dist}\big((x, \beta), \mathcal{S}(p)\big) \leq \kappa(p) \left\|r(x, \beta; p)\right\|_2$, for all (x, β) in a neighborhood of $\mathcal{S}(p)$, which completes the proof. □

B Algorithms

Algorithm 1 Projected sampling strategy

Input: the given domain $\mathcal{P}$ and the required number of data points N.

1. Construct an enlarged domain $\tilde{\mathcal{P}}$ such that $\mathcal{P} \subseteq \tilde{\mathcal{P}}$;
2. Sample the data points $\{p_i\}_{i=1}^N$ uniformly from the enlarged domain $\tilde{\mathcal{P}}$;
3. Project the sampled data points onto the original domain $\mathcal{P}$ by solving a projection problem: $p_i^{\text{proj}} = \operatorname{argmin}_{p\in\mathcal{P}} \|p - p_i\|_2^2,\ \forall i = 1, \cdots, N$

Output: $\mathcal{D} = \{p_i^{\text{proj}}\}_{i=1}^N$;

Algorithm 2 Unified offline training and online inference

Offline training.
Input: $\mathcal{D}$ collected by Algorithm 1, learning rate η, batch size B, number of epochs N_{epoch}, offline stopping tolerance ϵ_{off}.

for epoch $= 1 : N_{\text{epoch}}$

1. Split the dataset $\mathcal{D}$ into mini-batches $\mathcal{B} \subset \mathcal{D}$ with $|\mathcal{B}| = B$ and calculate the mini-batch loss $\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{p \in \mathcal{B}} \log\left(\left\| r\big(\text{ADMM-ReLU}_{K,\rho}(\mathcal{N}(p;\theta,M,L);p);p\big)\right\|_2^2 + 1\right)$;
2. Update the neural network parameters using automatic differentiation and a gradient-based optimizer such as Adam or quasi-Newton (L-BFGS): $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\mathcal{B}}(\theta)$;
3. If $\mathcal{L}(\theta) \le \epsilon_{\text{off}}$, break;

end
Output: trained parameters θ.

Online inference (supporting batch scenario) with triggered trustworthy correction.
Input: trained parameters θ, matrix $P = \text{col}(p_1, \cdots, p_{n_b})$ consisting of a batch of parameter vectors, online tolerance ϵ_{on}.

1. Compute the batch of neural proxies **in parallel**:

$$\left(\begin{bmatrix} x_1 \\ \beta_1 \end{bmatrix}, \cdots, \begin{bmatrix} x_{n_b} \\ \beta_{n_b} \end{bmatrix}\right) = \left(\text{ADMM-ReLU}_{K,\rho}(\mathcal{N}(p_1;\theta,M,L);p_1), \cdots, \text{ADMM-ReLU}_{K,\rho}(\mathcal{N}(p_{n_b};\theta,M,L);p_{n_b})\right)$$

2. Compute the certificate value **in parallel**:

$$\boldsymbol{\varepsilon} = \left(\left\| r\big(x_1,\beta_1;p_1\big)\right\|_2^2, \cdots, \left\| r\big(x_{n_b},\beta_{n_b};p_{n_b}\big)\right\|_2^2\right)$$

3. If $\|\boldsymbol{\varepsilon}\|_\infty \le \epsilon_{\text{on}}$, accept the neural prediction and return;
4. Otherwise, find indices $\mathcal{I}$ such that $\boldsymbol{\varepsilon}_{\mathcal{I}} > \epsilon_{\text{on}}$;
5. Update $(\{x\}_{\mathcal{I}}, \{\beta\}_{\mathcal{I}})$ via low-dimensional ADMM-ReLU (11) in batch scenario;

Output: $(\{x\}_{1:n_b}, \{\beta\}_{1:n_b})$.

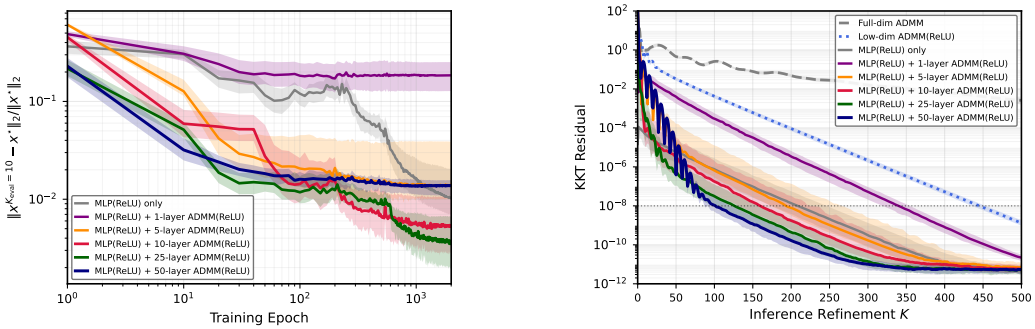
C Additional Ablation Results

C.1 Cross-scale K_{train} scaling rule

To verify that the unrolling depth must scale with problem size, we replicate the ablation at the smaller scale n=10, m=20, n_p=4. Table 1 reports the median test rel error (at fixed K_{eval}=10) across both scales and all six K_{train} values, and Fig. 8 shows the corresponding training and inference curves at n=10. The saturation point shifts from K_{train}=10 at n=10 (rel error $\sim 5\times10^{-3}$) to K_{train}=25 at n=100 (rel error $\sim 5\times10^{-2}$); shallower unrolling at n=100 remains in the unstable regime documented in the main text. This is consistent with the contraction rate of ADMM: the operator's contraction factor approaches 1 as the conditioning of $Q + \rho A^\top A$ grows with n, so a deeper unroll is needed for $\text{ADMM}_K(\text{MLP}(p))$ to enter the neighborhood of $x^\star$ where back-propagation reflects warm-start error. Empirically, $K_{\text{train}} \approx n/4$ suffices.

Table 1: Median test-set rel error after fixed K_{eval}=10 refinement at two scales. All entries trained for 2000 epochs.

n	m	n_p	K=0	K=1	K=5	K=10	K=25	K=50
10	20	4	1.0×10^{-2}	1.8×10^{-1}	1.4×10^{-2}	5.3×10^{-3}	3.6×10^{-3}	1.4×10^{-2}
100	200	20	3.5×10^{-1}	6.3×10^{-1}	3.6×10^{-1}	5.8×10^{-1}	5.7×10^{-2}	5.6×10^{-2}



(a) Training. (b) Inference.

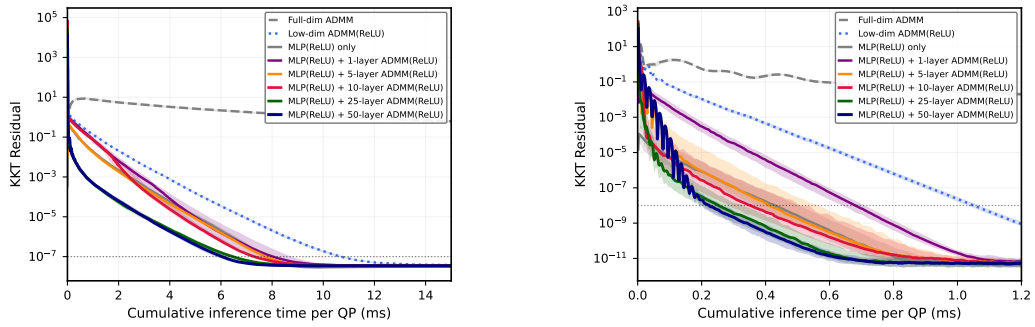
Figure 8: Ablation of K_{train} at n=10, m=20, n_p=4. (a) Test relative error vs. epoch at fixed K_{eval}=10; (b) post-training KKT residual vs. refinement steps K. K_{train}=10 already saturates, in contrast to the phase transition in Fig. 3.

C.2 Inference convergence detail

Table 2 reports per-method iteration count $K^\star$ to KKT residual $\leq 10^{-7}$, mean wall-clock per QP, and the implied speedup over cold-start ADMM(ReLU), corresponding to Fig. 3b. Fig. 9 re-plots the inference convergence in cumulative wall-clock per QP at both scales: $K_{\text{train}} \geq 25$ reaches $\varepsilon = 10^{-7}$ in roughly 6–7 ms at n=100 versus $\sim$ 11 ms for cold-start ADMM(ReLU), and the proportional speedup is even larger at n=10 ($\sim 5\times$) where the MLP forward overhead becomes negligible relative to the iteration savings.

Table 2: Iterations $K^\star$ and wall-clock to KKT residual $\leq 10^{-7}$, median over 100 test mpQPs at n=100, m=200, n_p=20. Speedup is relative to cold low-dim ADMM(ReLU); ΔK is the per-QP iteration reduction vs. that baseline. Full-dim ADMM does not reach the target within K=10^4.

Method	$K^\star$	Time (ms)	Speedup	ΔK
Full-dim ADMM	$>10^4$	>50.5	—	—
Low-dim ADMM(ReLU)	4729	10.689	1.0×	—
MLP only	3578	7.777	1.4×	1132
MLP + 1-layer ADMM(ReLU)	3692	8.032	1.3×	974
MLP + 5-layer ADMM(ReLU)	3728	7.751	1.4×	1000
MLP + 10-layer ADMM(ReLU)	3392	7.305	1.5×	1344
MLP + 25-layer ADMM(ReLU)	2800	6.460	1.7×	1930
MLP + 50-layer ADMM(ReLU)	2778	6.034	1.8×	1968



(a) n=100. (b) n=10.

Figure 9: Wall-clock view of inference convergence: KKT residual vs. cumulative time per QP. *TQPNet*'s wall-clock advantage over *ADMM(ReLU)* holds at both scales, with a larger proportional speedup at n=10.

C.3 Implementation details

Architecture. MLP(ReLU) backbone with depth $L{=}4$ and width $M{=}512$ at $n{=}100$ ($M{=}256$ at $n{=}10$); the x-head is linear and the β-head applies Softplus to enforce $\beta \geq 0$ ($\sim$1.0M params at $n{=}100$, $\sim$0.2M at $n{=}10$).

Unrolling depth. $K_{\text{train}}{=}25$ for the out-of-distribution study (Sec. 3), $K_{\text{train}}{=}10$ for the MPC scalability and quadruped studies, and $K_{\text{train}}{=}30$ for the UAV experiment (Appendix E); the choice of K_{train} is studied in Fig. 3 and Appendix C.1.

Optimization. AdamW (lr 10^{-3}, weight decay 10^{-5}), StepLR (step 300, $\gamma{=}0.5$), gradient clipping at norm 10, batch size 256, 2000 epochs. Training uses `torch.cholesky_solve` (numerically stable backward); inference uses pre-computed matmul.

Problem distribution. Sparse positive constraint matrix $A \sim \mathrm{Uniform}([0,3]^{m\times n})$ with column density 0.3; $c_0{=}Ax_{\mathrm{ref}}{+}1$ for a random reference x_{ref} (active-set tightness controlled by the $+1$ offset); $p{\sim}\mathrm{Uniform}([-1,1]^{n_p})$ for both training ($N{=}6000$) and test ($N{=}100$).

Reference solver. OSQP at `eps_abs`$=$`eps_rel`$=10^{-10}$ with polishing for the test-set $x^\star$.

Hardware. Single NVIDIA RTX 3080, PyTorch 2.10, CUDA 12.8.

D Tables of Numerical Examples

Table 3: Generalizability: median relative error ($n = 100$, $m = 200$, $n_p = 20$). Train on $p \sim \mathcal{U}[-1, 1]$.

Set	$K = 10$		$K = 30$		$K = 50$	
	TQPNet	LQPNet	TQPNet	LQPNet	TQPNet	LQPNet
Train	5.8e-02	1.5e-02	5.5e-02	1.4e-02	5.2e-02	1.3e-02
Valid	5.9e-02	1.6e-02	5.6e-02	1.5e-02	5.3e-02	1.4e-02
Boundary	2.2e-01	1.0e-01	2.1e-01	9.7e-02	2.0e-01	9.1e-02
OOD	3.5e-01	1.9e-01	3.4e-01	1.9e-01	3.2e-01	1.8e-01

Table 4: Scalability on random MPC problems (T=200 closed-loop steps with shared process noise $w \sim \mathcal{U}[-0.3, 0.3]^{n_s}$). Online latency in ms, offline training time in s. TQPNet/LQPNet use a fixed network size: hidden width M=512 and depth L=4 across all scales.

			OSQP			TQPNet (K=10)			LQPNet (K=10)			Offline		Explicit
$n+m$	n_p	Speedup	RMSE	Cost	Latency	RMSE	Cost	Latency	RMSE	Cost	Latency	TQP	LQP	Latency
60	3	0.0×	0.221	33	0.03	0.221	33	1.31	0.221	33	1.23	65	24	0.57
140	4	0.1×	0.218	45	0.07	0.218	45	1.30	0.218	45	1.31	65	25	4.11
315	6	0.2×	0.217	63	0.21	0.217	63	1.23	0.217	63	1.20	65	35	—
560	8	0.4×	0.263	124	0.59	0.264	124	1.31	0.263	124	1.33	71	43	—
875	10	1.1×	0.217	99	1.38	0.217	99	1.27	0.217	99	1.23	84	65	—
1050	12	1.6×	0.240	144	2.12	0.240	145	1.36	0.240	145	1.35	94	84	—
1400	16	2.7×	0.198	130	3.55	0.198	130	1.33	0.198	130	1.37	118	112	—
2100	24	5.0×	0.217	233	7.22	0.218	233	1.45	0.217	233	1.43	207	309	—
2800	32	7.0×	0.225	332	13.06	0.225	334	1.88	0.225	333	1.49	334	590	—

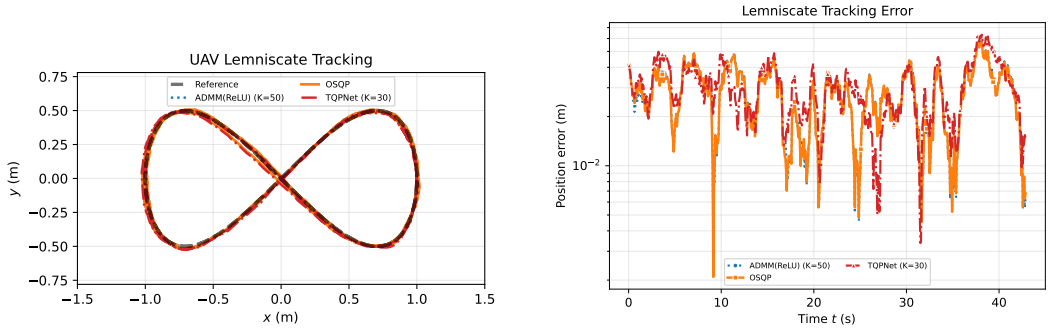
E Application to Quadrotor UAVs

TQPNet matches OSQP closed-loop tracking accuracy. We evaluate TQPNet on a quadrotor closed-loop tracking task following [45], with state $x \in \mathbb{R}^{13}$, control $u \in \mathbb{R}^4$, and condensed-MPC dimensions n=100, m=300 at horizon N_H=25 (Δt=71 ms). The mpQP parameter $p \in \mathbb{R}^{17}$ comprises the current state (13 dims) and a four-component Fourier basis $[\sin(\omega t), \cos(\omega t), \sin(2\omega t), \cos(2\omega t)]$ encoding the time-varying lemniscate reference. The closed loop is simulated for T=600 steps with i.i.d. process noise (σ_{pos}=3 mm shared across methods for fair comparison). We compare TQPNet at $K \in \{10, 20, 30\}$ refinement steps against OSQP through its raw C API (`prob.update` + `solve`, warm-started) at three tolerance levels.

Table 5 reports tracking RMSE and per-step latency. All TQPNet variants attain RMSE in 0.0297–0.0302 m, within $\sim 9\%$ of OSQP's 0.0273 m across the three tolerance levels and improving over the cold-start ADMM(ReLU) baseline (0.0306 m); the closed-loop trajectories (Fig. 10) overlap visually because all methods track the reference within the noise floor. TQPNet additionally bounds worst-case execution at fixed K: its per-step latency stays within $\sim 2.7\times$ of its median (GPU launch jitter), whereas OSQP's iteration count varies up to 1500 at ε=10^{-8} and its worst-case latency reaches 10–25× the median. For hard real-time deployment where worst-case rather than mean latency is binding, this matters. The asymptotic batch advantage of the proposed architecture is realized when many independent QPs must be solved in parallel, as demonstrated by the quadrupedal locomotion experiment in Subsection 3.3.

Table 5: Single-UAV lemniscate tracking (N_H=25, n=100, m=300, T=600). OSQP solved via raw C API with warm-started `prob.update` between steps. RMSE is comparable across methods. Latency and iteration counts report per-step median / maximum: OSQP exhibits worst-case latency 10–25× its median, whereas TQPNet at fixed K has bounded worst-case execution.

Method	RMSE (m)	Latency med/max (ms)	Iter med/max
OSQP (ε=10^{-6})	0.0273	0.73 / 7.03	50 / 450
OSQP (ε=10^{-7})	0.0273	0.75 / 13.48	50 / 950
OSQP (ε=10^{-8})	0.0273	0.75 / 21.10	50 / 1500
ADMM(ReLU) (K=50)	0.0306	5.84 / 13.05	50 / 50
TQPNet (K=10)	0.0302	1.67 / 4.39	10 / 10
TQPNet (K=20)	0.0301	2.78 / 6.81	20 / 20
TQPNet (K=30)	0.0297	3.90 / 10.66	30 / 30



(a) Closed-loop trajectories.

(b) Tracking error.

Figure 10: Single-UAV lemniscate tracking (T=600), showing the best TQPNet variant (K=30) alongside cold-start ADMM(ReLU) and OSQP (ε=10^{-8}). All methods overlap at trajectory resolution and tracking errors are within the same magnitude; full method comparison in Table 5.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction are supported by the theoretical derivations, residual analysis, and numerical experiments provided in the paper. The proposed ADMM-ReLU formulation and low-dimensional representation are derived in Section 2, while Section 3 validates the proposed method. The scope of the method is explicitly discussed in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses several limitations of the proposed approach, including the lack of a rigorous convergence analysis characterizing when the proposed low-dimensional ADMM is faster than the standard full-dimensional ADMM. The paper also notes that infinitely many COPE decompositions may exist for a given matrix, and identifying the decomposition that yields the best convergence performance remains an open problem. These limitations and future directions are discussed in the corresponding discussion/limitations section of the paper.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All theoretical results are stated together with their corresponding assumptions, and complete proofs are provided in the main paper or appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides sufficient details to reproduce the main experimental results, including the problem formulations, algorithmic procedures, network architectures, hyperparameter settings, and evaluation metrics used in the experiments. The experimental setup and implementation details are described in the paper to support reproducibility. The source code for all experiments is publicly available at `https://github.com/SOLARIS-JHU/TQPNet`.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The source code, problem generators, and scripts reproducing every figure and table are publicly available at `https://github.com/SOLARIS-JHU/TQPNet`, together with installation instructions and a figure-by-figure reproduction guide. The paper additionally documents the problem formulations, algorithmic procedures, network architectures, hyperparameter settings, and evaluation protocols.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the experimental settings necessary to understand and reproduce the reported results, including the training and test problem generation procedures, network architectures, optimizer settings, hyperparameters, evaluation metrics, and implementation details used in the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports statistical information over multiple randomly generated problem instances, including median performance together with percentile bands across experiments. The experimental results are aggregated over multiple trials to reduce the effect of randomness and provide statistically meaningful comparisons.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the hardware used (a single NVIDIA RTX 3080) in Section 3 and Appendix C.3, and reports training and inference times for the experiments (e.g., Fig. 5(b) and Table 2).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: The research conforms to the NeurIPS Code of Ethics. The paper focuses on methodological and theoretical developments in optimization and machine learning and does not involve human subjects, sensitive data, or high-risk applications.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: The paper focuses on the technical and theoretical contributions and does not include a specific discussion of broader positive or negative societal impacts. While the targeted application areas imply potential positive impact, this is not explicitly elaborated, nor are potential negative societal consequences discussed.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not involve the release of high-risk models, scraped datasets, or systems with foreseeable misuse concerns. The work focuses on methodological developments in optimization and learning-based solvers.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper properly cites the existing libraries and software assets used in this work, including optimization and machine learning frameworks. All external resources are used in accordance with their corresponding licenses and terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper documents the proposed methods, architectures, and experimental protocols in sufficient detail to support understanding and reproducibility. The implementation code is released at `https://github.com/SOLARIS-JHU/TQPNet` under the Apache-2.0 license, with a README describing installation and the reproduction of every figure and table.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This work does not involve crowdsourcing, human-subject experiments, surveys, or any form of participant-based data collection.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This work does not involve human subjects or participant-based research; therefore, IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: Large language models are not part of the proposed methodology, experiments, or scientific contributions of this work. Any use of LLMs was limited to writing or language editing assistance and does not affect the originality or technical content of the research.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.