Computational Optimization and Applications
https://doi.org/10.1007/s10589-026-00850-9

Reinforcement learning for adaptive interior point methods in convex quadratic programming

Jeremy Bertoncini[1] · **Alberto De Marchi**[1] · **Matthias Gerdts**[1] · **Simon Gottschalk**[1]

Received: 30 August 2025 / Accepted: 19 October 2026
© The Author(s) 2026

Abstract
Quadratic programming is a workhorse of modern nonlinear optimization, control, and data science. Although regularized methods offer convergence guarantees under minimal assumptions on the problem data, they can exhibit the slow tail-convergence typical of first-order schemes, thus requiring many iterations to achieve high-accuracy solutions. Moreover, hyperparameter tuning significantly impacts the solver performance but how to find an appropriate parameter configuration remains an elusive research question. To address these issues, we explore how data-driven approaches can accelerate the solution process. Aiming at high-accuracy solutions, we focus on a regularized interior-point solver and carefully handle its two-loop flow and control parameters. We will show that reinforcement learning can make a significant contribution to facilitating the solver tuning and to speeding up the optimization process. Numerical experiments demonstrate that, after a lightweight training, the learned policy generalizes well to different problem classes with varying dimensions.

Keywords Quadratic programming · Reinforcement learning · Amortized optimization · Solver tuning

✉ Alberto De Marchi
alberto.demarchi@unibw.de

Jeremy Bertoncini
jeremy.bertoncini@unibw.de

Matthias Gerdts
matthias.gerdts@unibw.de

Simon Gottschalk
simon.gottschalk@unibw.de

[1] Department of Aerospace Engineering, Institute of Applied Mathematics and Scientific Computing, University of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

Published online: 29 October 2026

1 Introduction

Quadratic programming provides a fundamental optimization model with applications in finance, data analysis, robotics, process control, and operations research. A quadratic program (QP) with n variables, m equality constraints, and p inequality constraints can be written in the form

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}x^\top Qx + q^\top x \qquad \text{subject to} \quad Ax = b, \quad Gx \leq d \tag{1}$$

where $x \in \mathbb{R}^n$ is the optimization variable, $Q \in \mathbb{R}^{n\times n}$ is a symmetric positive semi-definite matrix that defines the quadratic cost, $q \in \mathbb{R}^n$ is the linear cost vector, matrix $A \in \mathbb{R}^{m\times n}$ with vector $b \in \mathbb{R}^m$ define the linear equality constraints, and matrix $G \in \mathbb{R}^{p\times n}$ with vector $d \in \mathbb{R}^p$ represent the linear inequality constraints.

Methods for the numerical solution of QPs have been extensively studied. These differ in how they balance the number of iterations and the computational cost for each iteration. First-order methods, such as OSQP [34] based on ADMM [6], build upon simple steps but typically require many iterations, so several acceleration schemes have been proposed [9, 16, 21]. In contrast, Newton-type methods such as interior point (IP) [17], active set [14] and semismooth Newton [24] usually need fewer, yet more demanding, iterations to reach accurate solutions. Recently, proximal techniques [26] and the augmented Lagrangian (AL) framework [28] have seen a resurgent interest due to their inherent regularization properties, resulting in the development of solvers such as QPDO [10], QPALM [20] and ProxQP [3]. On this vein, the integration of proximal regularization with the IP strategy led to IP-PMM [27], PS-IPM [8] and PIQP [33]. Thanks to proximal regularization, these solvers can cope with matrix Q merely positive semi-definite and with matrices A, G having linearly dependent rows.

Although convergence guarantees can be established under minimal requirements on the problem data and algorithmic parameters, there are often numerous hyperparameters that must be tuned to speed up convergence, and few clues on how to adjust them. For example, the barrier parameter in interior-point methods must vanish; since there is no clear indication about its optimal rate of decay, heuristics are typically adopted [33]. In proximally regularized methods, the stepsize parameters not only affect the convergence rate but they also impact on the stability of linear algebra operations; cf. [8, §4]. Some theoretical works seek optimal values for the stepsize parameters, but they rely on solving auxiliary problems that are harder than the original QP itself [16, 18]. Another offline tuning approach is that of OPAL [2], which takes an algorithm as input and returns a recommendation on parameter values that maximize some user-defined performance metric. Alternatively, online heuristics introduce "feedback" by monitoring progress and adapting the parameters along the optimization process; see [5, §2], [34, §5.2], [20, §5.3] for some common instances.

After the seminal work [22] on "learning to optimize" for automated algorithm design, machine learning has been increasingly adopted to accelerate, warm-start, and tune optimization methods [1, 7]. For example, neural networks are used to warm-start QP solvers and fixed-point algorithms alike [30, 31].

Despite the flexibility of learning-based methodologies, considerable attention was devoted to accelerating *first-order* optimization methods, driven by their scalability and applications in machine learning. Exemplarily, the architecture proposed in [30] consists of a neural network mapping from the QP problem data to warm-starts, learned across families of QPs. Dedicated to proximal methods for image processing, the tuning-free approach of [36] formulates the online hyperparameter selection as a sequential decision-making problem and implements a policy network for automated parameter search. Closer to our focus here is RLQP [21], which couples the OSQP solver with a reinforcement learning (RL) agent: by learning a policy to adapt the internal parameters of the ADMM algorithm, RLQP can outperform OSQP (with its fine-tuned heuristic update rules) on a variety of problems.

In contrast, the combination of learning techniques and *second-order*, or Newton-type, methods remains relatively unexplored. We investigate the coupling of a regularized IP solver for QPs with a policy for online hyperparameter tuning, parametrized with a simple neural network and learned by RL. Figure 1 illustrates the algorithmic framework and the feedback interconnection between QP solver and RL agent. Following the approach of RLQP [21], we learn a policy π_ζ (e.g., a neural network parameterized by ζ) that maps states σ to actions α with the goal of maximizing an accumulated reward $\mathcal{R}$. Compared to the single-loop structure of OSQP, the architecture arising from an IP scheme (the *QP Environment*) can be represented with two nested loops, where the inner loop corresponds to the approximate solution of a subproblem (by executing some iterations of Newton method). While ubiquitous in second-order methods and with well established convergence theory, the two-loop structure does not fit the RL framework developed for RLQP, because of endogenous parameter sequences controlling the solution process (and affecting the MDP formalization). We aim to fill this gap by exploring a machine learning framework suitable for accelerating the tuning and solution process of QP solvers beyond first-order methods.

Contribution We formulate a RL framework to learn a policy that adaptively replaces the hand-tuned or heuristic hyperparameter-update rules typically used in regularized interior-point solvers, without modifying the underlying optimization algorithm, so that the base QP solver retaining provable convergence guarantees. The definitions of state, action and reward are intended (i) to yield a policy that is invariant to problem size, scaling and permutations, and (ii) to capture the two-loop structure typical of second-order methods.

Experimental results on random and benchmark QPs demonstrate that the learned policy outperforms the base solver and is competitive with state-of-the-art solvers. Moreover, it generalizes well to previously unseen problem sizes, scales, and classes, where tedious hand-tuning is typically required—an empirical observation rather than a theoretical guarantee of the method. Especially on ill-conditioned problems, we observe that the RL-enhanced solver automatically adapts its behavior, outperforming other solvers in terms of robustness. In addition, these capabilities are achieved after lightweight training of a small neural network on a standard laptop, in contrast to the substantially higher computational cost associated with training RLQP [21].

Outline Convex quadratic programming is introduced in Sect. 2 with a stabilized IP numerical scheme and a discussion on the challenges and opportunities of hyperparameter tuning. An overview of reinforcement learning is given in Sect. 3, with the

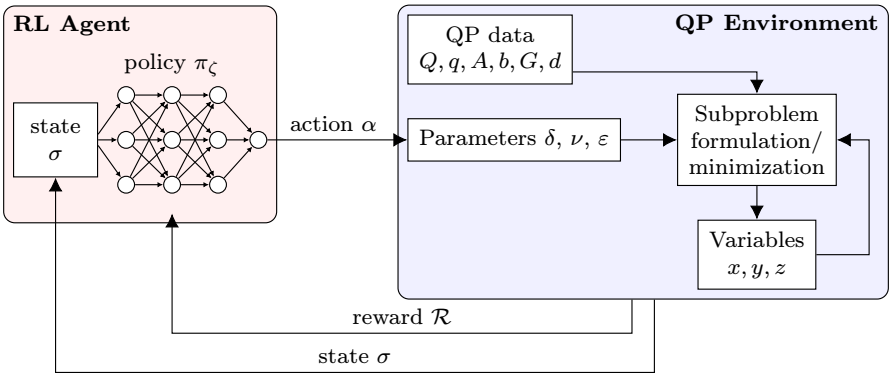


Fig. 1 A QP solver is combined with a reinforcement learning (RL) agent that evaluates a policy to adapt the internal parameters of the QP solver. Based on its observation of the state of the environment, the RL action aims at improving the solver's performance. The RL policy is parameterized by a small artificial neural network, the state σ summarizes the internal state of the QP solver (at the beginning of each inner loop), the action α affects the parameters δ, ν, ε of the solver, and the reward $\mathcal{R}$ promotes faster convergence to a solution of the QP. Illustration inspired by [21, Fig. 1]

tools needed to properly interact with the QP solver. Training procedures and numerical results for three evaluation settings are then presented in Sect. 4. We conclude with some final remarks in Sect. 5.

Throughout this article we use various common acronyms, introduced in the table below. Our numerical experiments compare a QP solver named `QProxIP`, developed by the authors based on the methodology described in Sect. 2, and its RL–enhanced extension `RL-QProxIP`.

LP/QP	Linear/Quadratic problem	RL	Reinforcement learning
IP	Interior point	MDP	Markov decision process
AL	Augmented Lagrangian	PPO	Proximal policy optimization
PPA	Proximal point algorithm	`QProxIP`	Base QP solver
ADMM	Alternating direction method of multipliers	`RL-QProxIP`	RL-enhanced QP solver

2 Solving convex QPs

In view of IP methods, the QP problem (1) can be equivalently formulated using an auxiliary variable $s \in \mathbb{R}^p$ as

$$\underset{x\in\mathbb{R}^n, s\in\mathbb{R}^p}{\text{minimize}} \quad \frac{1}{2}x^\top Qx + q^\top x \qquad \text{subject to} \quad Ax = b, \quad Gx - d + s = 0, \quad s \geq 0. \tag{2}$$

It is a standard result that, for convex QPs, the following necessary conditions are also sufficient for optimality: there exists Lagrange multipliers $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^p$ such

that

$$0 = Qx + q + A^\top y + G^\top z, \quad 0 = \min\{s, z\}, \quad 0 = Ax - b, \quad \text{and} \quad 0 = Gx - d + s, \tag{3}$$

where the second equality coincides with the complementarity condition $0 \leq s \perp z \geq 0$. Therefore, a valid criterion for returning a triplet (x, y, z) as approximately optimal, with tolerance $\epsilon \geq 0$, is to check that the conditions

$$\begin{aligned} r_{\text{prim}}(x, y, z) &:= \left\| \begin{matrix} Ax - b \\ \min\{d - Gx, z\} \end{matrix} \right\|_\infty \leq \epsilon \text{ and} \\ r_{\text{dual}}(x, y, z) &:= \left\| Qx + q + A^\top y + G^\top z \right\|_\infty \leq \epsilon \end{aligned} \tag{4}$$

are satisfied. The primal r_{prim} and dual r_{dual} residual functions indicate how far a triplet (x, y, z) is from being a primal-dual solution to (1), in view of (3). As such, these metrics will play a crucial role in monitoring and quantifying the performance of a solution process.

2.1 Regularized interior point schemes

Moving towards numerical schemes, we notice that, from an operator perspective, (3) is equivalent to a monotone inclusion. Adopting the (inexact) proximal point algorithm (PPA) [13, 29], different reformulations of (3) lead to different regularized schemes, whose subproblems correspond to the (approximate) evaluation of the proximal operator. This construction naturally gives rise to two-loop algorithms: the outer for PPA and the inner for the subproblem solution. Among others, QPDO [10] and PS-IPM [8] have been developed in this spirit, so that proximally regularized subproblems can be (efficiently and reliably) solved with Newton-type methods.

Here we outline a regularized IP scheme that, reminiscent of QPDO and PS-IPM, treats equality and inequality constraints respectively with primal-dual augmented Lagrangian and barrier strategies. For further regularization, a Tikhonov-like proximal term is included to handle also non-strictly convex QPs, as in [3, 4, 34].

Primal IP Beginning from the classical augmented Lagrangian function for (2), for any given primal-dual estimate $\theta = (\theta_x, \theta_y, \theta_z) \in \mathbb{R}^{n+m+p}$, regularization parameter $\delta = (\delta_x, \delta_y, \delta_z) \in \mathbb{R}^3_{++}$ and barrier parameter $\nu > 0$, we define the merit function $\mathcal{L}_\delta(\cdot, \cdot; \theta, \nu) \colon \mathbb{R}^n \times \mathbb{R}^p_{++} \to \mathbb{R}$ as

$$\begin{aligned} \mathcal{L}_\delta(x, s; \theta, \nu) := \frac{1}{2} x^\top Q x + q^\top x + \theta_y^\top (Ax - b) + \theta_z^\top (Gx - d + s) \\ + \frac{\delta_x}{2} \|x - \theta_x\|^2 + \frac{1}{2\delta_y} \|Ax - b\|^2 + \frac{1}{2\delta_z} \|Gx - d + s\|^2 - \nu \sum_{i=1}^{p} \ln s_i. \end{aligned} \tag{5}$$

As usual in barrier methods, the positivity constraint $s \geq 0$ in (2) is incorporated in a smooth way by replacing its indicator with a (logarithmic) barrier function: the smaller

the barrier parameter ν, the better the approximation. Note that, for any fixed θ, δ and ν, $\mathcal{L}_\delta(\cdot, \cdot; \theta, \nu)$ is strictly convex under mere convexity of (1). Hence, it always admits a unique minimizer (x, s) that necessarily satisfies $s > 0$. Effectively, the minimization of $\mathcal{L}_\delta(\cdot, \cdot; \theta, \nu)$ constitutes a suitable subproblem for tackling (1), since its minimizer is closely related to the proximal point of an operator associated to (3); see [8, 28].

In practice, by minimizing $\mathcal{L}_{\delta^k}(\cdot, \cdot; \theta^k, \nu_k)$ one can obtain an improved primal estimate (x^k, s^k) and classical multiplier updates from AL methods yield a dual estimate (y^k, z^k) as

$$y^k \leftarrow \theta_y^k + \frac{Ax^k - b}{\delta_{y,k}} \quad \text{and} \quad z^k \leftarrow \theta_z^k + \frac{Gx^k - d + s^k}{\delta_{z,k}}. \tag{6}$$

After updating the incumbent estimate θ^{k+1} and the hyperparameters δ^{k+1} and ν_{k+1}, typically based on (x^k, s^k, y^k, z^k), one moves on to the next subproblem.

Primal-dual IP A widespread technique is that of *primal-dual* interior point methods, where the subproblem is modified to involve both primal and dual variables, so that improved values for (y, z) are directly obtained from the inner minimization procedure. One way to generate a suitable subproblem for this task is to adopt the primal-dual augmented Lagrangian function [15]. We define $\mathcal{M}_\delta(\cdot, \cdot, \cdot, \cdot; \theta, \nu) \colon \mathbb{R}^n \times \mathbb{R}^p_{++} \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ as

$$\begin{aligned} \mathcal{M}_\delta(x, s, y, z; \theta, \nu) := \mathcal{L}_\delta(x, s; \theta, \nu) &+ \frac{1}{2\delta_y}\|Ax - b + \delta_y(\theta_y - y)\|^2 \\ &+ \frac{1}{2\delta_z}\|Gx - d + s + \delta_z(\theta_z - z)\|^2 \end{aligned} \tag{7}$$

where the additional terms guarantee that, when minimizing $\mathcal{M}_{\delta^k}(\cdot, \cdot, \cdot, \cdot; \theta^k, \nu_k)$ exactly, the optimal values for (y, z) coincide with the classical update (6) that would be performed within a purely primal scheme. Similarly to its primal counterpart, the primal-dual augmented Lagrangian function $\mathcal{M}$ depends on the regularization parameters δ and barrier parameter ν.

Algorithm 2.1: Regularized interior point method for convex QPs.

Data: primal-dual guess (x, y, z), tolerance ϵ
set $\theta^0 \leftarrow (x, y, z)$
for $k = 0, 1, 2 \ldots$ **do**
  choose $\delta^k, \nu_k, \varepsilon_k > 0$ `// hyperparameters update`
  define $\mathcal{M}_k := (x, s, y, z) \mapsto \mathcal{M}_{\delta^k}(x, s, y, z; \theta^k, \nu_k)$
  find (x^k, s^k, y^k, z^k) such that $\left\|\nabla \mathcal{M}_k(x^k, s^k, y^k, z^k)\right\| \le \varepsilon_k$ and $s^k > 0$ `// subsolve`
  if $r_{\text{prim}}(x^k, y^k, z^k) \le \epsilon$ **and** $r_{\text{dual}}(x^k, y^k, z^k) \le \epsilon$ **then** `// termination check`
    return (x^k, y^k, z^k)
  set $\theta^{k+1} \leftarrow (x^k, y^k, z^k)$ `// estimates update`

The resulting primal-dual scheme is outlined in Algorithm 2.1, where Step 5 poses the most computational requirements and will be discussed in the following Sect. 2.2. The termination conditions (4) are checked at Step 6, before updating the incumbent primal-dual estimate θ^{k+1} at Step 8. Of all steps, the choice of hyperparameters δ^k, ν_k and ε_k at Step 3 has the least guidelines or specifications dictated by the theory. Their tuning and influence on the solution process are discussed in Sect. 2.3 below.

Convergence of Algorithm 2.1 is guaranteed insofar as the regularization parameters δ^k remain positive and bounded, the barrier parameter ν_k vanishes from above, and the inner tolerance ε_k is summable. The technical requirements and mild conditions for convergence are detailed, e.g., in [13]. The convergence properties of Algorithm 2.1 can be analyzed with the lens of inexact proximal point algorithms; we refer the interested reader to [3, 4, 8, 10, 11, 20, 24, 27] for recent results established by exploiting this connection for regularized QP solvers.

2.2 Inner minimization with Newton

The minimization procedure adopted to solve the subproblem at Step 5 of Algorithm 2.1 is typically based on Newton's method with a backtracking line search procedure for globalization. Methods of this kind for convex unconstrained minimization have been extensively studied and provide strong convergence guarantees. However, a direct application of Newton method to minimize $\mathcal{M}_k$ is usually deemed impractical and avoided, since the Hessian matrix $\nabla^2\mathcal{M}_k$ tends to be dense, but, owing to convexity of $\mathcal{M}_k$, it is sufficient to find a zero of $\nabla\mathcal{M}_k$. Using the linear transformation T_{δ^k} as in [10, §3.1], we can obtain an equivalent root-finding subproblem that retains structure and sparsity of the original problem (1):

$$0 = T_{\delta^k}\nabla\mathcal{M}_k(x,s,y,z) = \begin{pmatrix} Qx+q+A^\top y+G^\top z+\delta_{x,k}(x-\theta_x^k) \\ -\nu_k s^{-1}+z \\ Ax-b+\delta_{y,k}(\theta_y^k-y) \\ Gx-d+s+\delta_{z,k}(\theta_z^k-z) \end{pmatrix}, \tag{8}$$

where the vector-inverse in the second block-row is intended componentwise. Scaling the second block by $s > 0$, the classical primal-dual IP subproblem is recovered with additional proximal regularization terms, as in [27, §2], [8, §3]. Then, at any point (x,s,y,z), a Newton direction $(\Delta x,\Delta s,\Delta y,\Delta z) \in \mathbb{R}^{n+p+m+p}$ can be obtained by solving the following system of linear equations:

$$\begin{bmatrix} Q+\delta_{x,k}\mathbb{I}_n & 0 & A^\top & G^\top \\ 0 & \operatorname{diag}(z) & 0 & \operatorname{diag}(s) \\ A & 0 & -\delta_{y,k}\mathbb{I}_m & 0 \\ G & \mathbb{I}_p & 0 & -\delta_{z,k}\mathbb{I}_p \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta y \\ \Delta z \end{pmatrix}$$
$$= -\begin{pmatrix} Qx+q+A^\top y+G^\top z+\delta_{x,k}(x-\theta_x^k) \\ -\nu_k+s\odot z \\ Ax-b+\delta_{y,k}(\theta_y^k-y) \\ Gx-d+s+\delta_{z,k}(\theta_z^k-z) \end{pmatrix}, \tag{9}$$

for which there exist several numerical techniques. In (9), $\mathbb{I}_n$ denotes the identity matrix of size n, diag(s) builds a square diagonal matrix with diagonal s, and $\odot$ indicate the Hadamard product. As our focus is on the combination of the optimization routine in Algorithm 2.1 with a learning-based hyperparameter policy, the (sparse) linear system (9) is solved directly through LU decomposition with pivoting, for numerical stability, without any additional structure exploitation. It is worth mentioning here that the regularization parameters δ are introduced as penalty weights in the definition (5) of the augmented Lagrangian, with the goal of discouraging constraint violations, but they also play an important role in mitigating the ill-conditioning of matrices arising from the IP scheme; cf. [8, §4].

2.3 Hyperparameters

Although convergence guarantees are retained under minimal assumptions, the practical behavior and performance of Algorithm 2.1 depend on the sequences of hyperparameters $\{\delta^k\}$, $\{\nu_k\}$ and $\{\varepsilon_k\}$. While parameters ν and ε should become sufficiently small (relative to a user-specified tolerance), there is no clear way to choose the weights δ; in fact, convergence is guaranteed even if they are kept constant.

As depicted in Fig. 1, all hyperparameters will be subject to the action of the external RL agent and updated according to its policy. In particular, at Step 3 of Algorithm 2.1, the regularization parameters δ, the barrier parameter ν, and the inner tolerance ε are all updated through the output α of the trained RL policy. Our approach is to define the update rule as

$$\begin{aligned} \delta_{x,k+1} &= \max\{\delta_{\min}, \alpha_{x,k}\delta_{x,k}\}, \quad & \delta_{y,k+1} &= \max\{\delta_{\min}, \alpha_{y,k}\delta_{y,k}\}, \\ \delta_{z,k+1} &= \max\{\delta_{\min}, \alpha_{z,k}\delta_{z,k}\}, & & \\ \nu_{k+1} &= \max\{\nu_{\min}, \alpha_{\nu,k}\nu_k\}, & \varepsilon_{k+1} &= \max\{\varepsilon_{\min}, \alpha_{\varepsilon,k}\varepsilon_k\}, \end{aligned} \tag{10}$$

starting from some $\delta^0 = (\delta_{x,0}, \delta_{y,0}, \delta_{z,0}) \in \mathbb{R}^3_{++}$, $\nu_0 > 0$ and $\varepsilon_0 > 0$. For notational convenience, the single actions $\alpha_{x,k}$, $\alpha_{y,k}$, $\alpha_{z,k}$, $\alpha_{\nu,k}$, $\alpha_{\varepsilon,k} \in (0, 1)$ are collected in a vector α^k. The design choice in (10) of a saturated linear decrease is common; it follows the update rules in QPALM [20], ProxQP [3] and PIQP [33], to name a few. The small thresholds $\delta_{\min}, \nu_{\min}, \varepsilon_{\min} > 0$ safeguard the numerical linear algebra when addressing the linear system (9).

To fix ideas, we let the RL action take the form

$$\alpha^k = \alpha\left(\bar{r}_{\text{prim}}, \bar{r}_{\text{dual}}, \nu_k, \varepsilon_k\right), \tag{11}$$

where $\bar{r}_{\text{prim}}, \bar{r}_{\text{dual}}$ denote (normalized) residuals at (x^k, y^k, z^k), and the mapping α is learned by training a neural network π_ζ (that is, optimizing its weight and biases ζ). The choice of defining this mapping with residuals $\bar{r}_{\text{prim}}, \bar{r}_{\text{dual}} \in \mathbb{R}_+$ as inputs, as opposed to primal-dual decision variables $(x, y, z) \in \mathbb{R}^{n+m+p}$, is intended to make the mapping α invariant to the problem dimensions (n, m, p); more details will be given in Sect. 3.1.

We conclude this overview of the QP solver by noting that various features (for numerical stability and practical performance) are implemented in both the base and the RL-enhanced versions of the solver. These standard techniques include presolving and Ruiz-based preconditioning [34], nonmonotone line search [19], boundary control [27], centrality correction [33].

3 Learning-based parameter adaptation

In the following, we aim at finding a good strategy to choose the parameters during the optimization. So far, the choice of these parameters is up to the operators, and thus, the optimization highly depends on their experiences. Reinforcement learning [35] makes it possible to automate this choice and avoid manual hyperparameter tuning. Therefore, reinforcement learning approaches benefit from their ability to train a policy by interacting with the environment in a trial-and-error manner.

Compared to previous work [21] on the RL-acceleration of an ADMM-based QP solver, we focus on the regularized interior point approach discussed earlier. Thus, the goal is to automate the choice of hyperparameters δ^k, ν_k, ε_k at each iteration k, by acting through α^k according to (10). Thereby, the choice should be made with respect to scale-invariant properties of the problems. In our case we choose the primal and dual residuals, the barrier parameter and the inner tolerance as in (11). This design choice is supported by the numerical experiments presented in Sect. 4. Nevertheless, the residuals naturally give a succinct representation of the optimization status, while the barrier parameter reflects the current relaxation of the problem and the inner tolerance determines the effort required to force convergence. Both directly influence the synchronization between the proximal regularization and the performance of the Newton solves.

Figure 1 shows the embedding of the RL policy into the optimizer which is summarized in Algorithm 2.1. In an inner loop, the augmented Lagrangian function $\mathcal{M}_\delta(\cdot,\cdot,\cdot,\cdot;\theta,\nu)$ is minimized for fixed $\theta = \theta^k$, barrier parameter $\nu = \nu_k$ and weights $\delta = \delta_k$, up to a tolerance ε_k. Once this has been achieved, the estimates θ^k are updated for the next outer loop iteration. In addition, the hyperparameters δ^k, ν_k, and ε_k are adapted by the factors α^k determined by the RL policy. Thus, the RL policy acts at Step 3 in Algorithm 2.1, updating all hyperparameters. This procedure is repeated until the primal and dual residuals are smaller than a tolerance $\epsilon > 0$. During training, the policy to be learned is improved step by step by interacting with the described framework, which in RL context is also called "environment", while trying to solve some training QPs.

3.1 State and reward

We continue with the mathematical background, which is needed for RL. It is based on the Markov decision process (MDP) [12], which is represented by the quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. The state space $\mathcal{S}$ comprises all possible configurations of the environment, while the action space $\mathcal{A}$ represents all actions which allow the policy to

influence the environment. The probability transition $\mathcal{P}$ describes the probability of reaching a state under the condition that the previous state and action are given. Finally, the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ rates the state transition and enables to compare and improve solutions. While the state and action space as well as the reward function need to be defined by the operator, the probability transition does not need to be given explicitly. It is compensated by the interaction of the policy with the environment. In the following, we will define this framework for our present task.

State One reason to use RL in this framework is that it is not clear what is the best choice for the factors α^k and what their choice depends on. Thus, it is natural to choose a state, which best represents the optimization process. We are convinced that the primal and dual residuals of the QPs are good representatives of the solver status and decisive factors for choosing α^k. Thus, we make them part of the state definition and interpret the upcoming numerical results as indication supporting our choice.

First, the normalized residuals $\bar{r}_{\text{prim}}$ and $\bar{r}_{\text{dual}}$ defined by

$$\bar{r}_{\text{prim}}(x, y, z) := \frac{r_{\text{prim}}(x, y, z)}{\max\{\|A\|_\infty, \|b\|_\infty, \|G\|_\infty, \|d\|_\infty\}} \tag{12a}$$

$$\bar{r}_{\text{dual}}(x, y, z) := \frac{r_{\text{dual}}(x, y, z)}{\max\{\|Q\|_\infty, \|q\|_\infty, \|A\|_\infty, \|G\|_\infty\}} \tag{12b}$$

are intended to make the state definition invariant to problem scaling. Then, since the (normalized) residuals and hyperparameters can span several orders of magnitude during the solution process, causing scaling issues while training the RL policy π_ζ, we encode the environment state σ with a logarithmic scale:

$$\sigma(\bar{r}_{\text{prim}}, \bar{r}_{\text{dual}}, \nu, \varepsilon) := \begin{pmatrix} \sigma_{\text{prim}} \\ \sigma_{\text{dual}} \\ \sigma_\nu \\ \sigma_\varepsilon \end{pmatrix} := \begin{pmatrix} -\log\left(\bar{r}_{\text{prim}} + 10^{-9}\right) \\ -\log\left(\bar{r}_{\text{dual}} + 10^{-9}\right) \\ -\log\left(\nu + 10^{-17}\right) \\ -\log\left(\varepsilon + 10^{-9}\right) \end{pmatrix}, \tag{13}$$

where the small shifts are included to prevent numerical instabilities during the training process.

Action The action of our task is the choice of α^k in each iteration step k. Since α^k contains factors that are used to sequentially decrease the hyperparameters δ^k, ν_k, and ε_k, according to (10), it must be positive and smaller than one.

Reward The reward $\mathcal{R}$ is constructed as a combination of a continuous term and a single success incentive, namely $\mathcal{R} := \mathcal{R}_{\text{base}} + \mathcal{R}_{\text{bonus}}$ where

$$\mathcal{R}_{\text{base}} := -\frac{1}{2}\left[\log(\bar{r}_{\text{prim}} + 10^{-9}) + \log(\bar{r}_{\text{dual}} + 10^{-9})\right] - \frac{N_{\text{in}}}{N_{\text{in}}^{\max}},$$

$$\mathcal{R}_{\text{bonus}} := \left[1 + 4\left(1 - \frac{N_{\text{out}}}{N_{\text{out}}^{\max}}\right)\right] \iota(\bar{r}_{\text{prim}} \le \epsilon)\iota(\bar{r}_{\text{dual}} \le \epsilon),$$

and the characteristic function ι is defined as $\iota(\texttt{cond}) = 1$ if condition `cond` is true, $\iota(\texttt{cond}) = 0$ otherwise. The base reward $\mathcal{R}_{\text{base}}$ encourages convergence by reduction

of residuals while penalizing an excessive number of inner Newton iterations N_{in}. The bonus term $\mathcal{R}_{\text{bonus}}$ is activated only upon satisfaction of the termination criterion at Step 6 of Algorithm 2.1. When satisfied, it provides a bounded incentive depending on the number of outer iterations required N_{out}. The resulting formulation remains intentionally simple conforming with the MDP framework.

Note that N_{in} and N_{out} are not explicitly part of the state σ. However, under the hypothesis that the state σ provides a good representation of the current status of the optimizer, then one can expect similar values of N_{in} and N_{out} for similar states σ. This observation leads us to consider N_{in} and N_{out} as (indirectly) dependent on σ, while relying on the fact that any deviation can be seen as variance of our transition probability.

Neural network After depicting the MDP, all that remains is to define the policy in order to have a RL framework. We parametrize our policy π_ζ by means of a feedforward neural network, whose weights and biases form the parameters ζ. We determined the size of the network by a grid search and decided on a net with two hidden layers with 25 neurons in each layer. The size of the input layer (four) and output layer (three) are given by the dimension of the state and action space. For each layer, the commonly used hyperbolic tangent is chosen as the activation function, because of its zero centering and smoothness.

Based on these definitions, RL aims to maximize the objective with respect to the weights and biases of the neural network:

$$\min_{\zeta} \quad \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i \mathcal{R}(\sigma_i, \sigma_{i+1})\right]$$

with discount factor $\gamma \in (0, 1)$ and $\sigma_i \in \mathcal{S}$ for $i \in \mathbb{N}$. Note that the expected value, which appears in the objective function, is defined over all trajectories $(\sigma_0, \alpha_0, \sigma_1, \alpha_1, \dots)$ for $\alpha_i \in \mathcal{A}$.

3.2 Training procedures

Based on the MDP, which was defined in the previous subsection, we run the RL approach. We chose Proximal Policy Optimization (PPO) [32], as it is considered robust and stable and is capable of handling continuous action spaces. It is an actor-critic RL approach, which means that the policy π_ζ and the value function V_Φ with parameters Φ are trained simultaneously. Note that in our case, both the policy and the value function share the majority of the neural network, which was described before.

For an update step in the training, the policy is used for various training problems within the above introduced framework (see Fig. 1). At the same time, the states and actions are stored. In order to update the policy, PPO forces that the fraction of the updated policy over the previous policy increases for well-rated and decreases for poorly rated behavior in the stored data. In this way, good behavior is reinforced. The rating of the behavior is given by the general advantages estimation and the policy update is clipped, which avoids big update steps, for a more robust improvement.

Table 1 PPO hyperparameters used in training

Learning rate schedule	$(0, 5 \cdot 10^{-4})$ $(500\text{k}, 3 \cdot 10^{-4})$ $(1\text{M}, 10^{-4})$	Entropy coeff. schedule	$(0, 5 \cdot 10^{-3})$ $(1\text{M}, 2 \cdot 10^{-3})$
Batch size	8192	γ	0.99 (default)
λ	1.0 (default)	ϵ_{clip}	0.2 (default)
Gradient clip	25	KL coeff.	0.05
KL target	0.01	Network layers	[25, 25] (tanh)
System configuration	Kubuntu 22.04.5 LTS, Intel Core i9-12950HX		
Num. CPUs (local worker)	20	Num. CPUs per worker	1

Afterwards, the general advantages estimation is updated with respect to the generated data and its corresponding rewards.

Our technical implementation is based on the RL library `RLlib` [23]. The choice of hyperparameters for the described PPO approach can be seen in Table 1. Thereby, the learning rate schedule is tailored to the magnitude of the reward and decreases for the final stages of the training. The gradient clip and the Kullback–Leibler parameters (KL coefficient, KL target), which together with the clipping parameter ϵ_{clip} limit the policy updates, are chosen empirically in order to avoid an instable training. The choice of the entropy schedule leads together with a high number of samples used for an update step (batch size) to an exploratory, but stable training. Typical default values have been selected for other PPO parameters, which we have confirmed by the robust training behavior. These include, for instance, the constant λ depicting how strongly future rewards influence the update of the general advantages estimation.

For the training and numerical experiments reported in this work, the QP solver is configured as summarized in Table 2. The tolerance ϵ in the termination conditions (3) is set to 10^{-6}. The minimum values of δ, ν, ε, and the step-size parameter α define the operational bounds of the algorithm, i.e., its standard numerical regime.

The inner-loop tolerance ε_k is bounded below by $\epsilon/10$, since the inner subproblems do not need to converge beyond this level to guarantee convergence of the outer iterations. The parameter $\nu_{\min}$ is chosen close to machine precision to handle near-degenerate cases with almost linear dependence in the constraints, where a very small barrier and a fine relaxation of feasibility are required to obtain accurate solutions. The regularization lower bound $\delta_{\min} = 10^{-12}$ is applied uniformly to AL methods and proximal terms. These weights push Newton-type methods toward their numerical limits, while improving performance by strongly penalizing the merit function terms (5), yielding KKT systems (9) that remain very close to those of the original problem. The initialization scales $\delta_{x,0}$, $\delta_{y,0}$, and $\delta_{z,0}$ are tuned manually based on the baseline `QProxIP`. The initial inner tolerance ε_0 is selected such that it is smaller than the initial residual norm, preventing premature satisfaction of the first outer iteration. The bounds $\alpha_{\min}$ and $\alpha_{\max}$ are set strictly within $(0, 1]$ to ensure monotone decrease. Finally, the iteration limits $N_{\text{in}}^{\max}$ and $N_{\text{out}}^{\max}$ constrain the algorithmic budget for each problem instance.

Table 2 Default solver setup

ϵ	$\delta_{\min}$	$\nu_{\min}$	$\varepsilon_{\min}$	$\delta_{x,0}$	$\delta_{y,0}$	$\delta_{z,0}$
10^{-6}	10^{-12}	10^{-16}	$0.1 \cdot \epsilon$	1	10	10
ν_0	ε_0	$\alpha_{\min}$	$\alpha_{\max}$	$N_{\text{in}}^{\max}$	$N_{\text{out}}^{\max}$	
1	$0.1 \cdot \min\{r_{\text{prim}}(\theta^0), r_{\text{dual}}(\theta^0)\}$	0.05	0.95	25	25	

3.3 Training problem set

We generate difficult random QP problems to train the RL policy π_ζ illustrated in Fig. 1. Each problem of the form (1) is defined by matrices Q, A, G and vectors q, b, d, with properties such as convexity, sparsity, rank, and conditioning carefully controlled to produce meaningful and diverse problem batches. [1]

Ill-conditioning QP problems become numerically challenging when the linear system (9) is ill-conditioned, as this directly affects the accuracy of the linear solver in each Newton step. When the condition number is high, small numerical errors are amplified during factorization, making the Newton directions inaccurate and slowing down convergence, or breaking it altogether. In our training set, the rank and condition number of matrix Q are targeted to make difficult random QPs, as follows.

Q matrix generator Matrix Q is constructed as a positive semidefinite matrix with prescribed rank and spectrum. An orthogonal matrix $V \in \mathbb{R}^{n\times n}$ is obtained via QR factorization of a Gaussian matrix $N \in \mathbb{R}^{n\times n}$ with, $N = VR$, where $R \in \mathbb{R}^{n\times n}$ is upper triangular. Furthermore, let $\Lambda \in \mathbb{R}^{n\times n}$ denote a diagonal matrix containing the eigenvalues of Q, whose construction is specified below. The Hessian is then defined as $Q = \mathcal{S}_\rho\left(V\Lambda V^\top\right) \in \mathbb{R}^{n\times n}$, where $\mathcal{S}_\rho$ denotes an operator that enforces symmetry and a density level ρ (so that Q has approximately ρn^2 nonzero components).

A rank $h \le n$ is sampled, and the nonzero eigenvalues are partitioned into $n_L = \lfloor h/2 \rfloor$ large and $n_S = h - n_L$ small values. For measuring the spectral properties of the cost matrix, we consider the effective condition number

$$\kappa_e(Q) := \frac{\lambda_{\max}(Q)}{\lambda_{\min,+}(Q)},$$

where $\lambda_{\min,+}$ extracts the smallest positive eigenvalue of a matrix. Given a target effective condition number $\kappa_e^{\text{target}} \ge 1$, we choose $1 < c_0 \ll \kappa_e^{\text{target}}$, draw the eigenvalues as

$$\lambda_i \sim \mathcal{U}(1, c_0), \quad i = 1, \ldots, n_S, \qquad \lambda_j \sim \mathcal{U}(0.1, 1) \cdot \kappa_e^{\text{target}}, \quad j = n_S + 1, \ldots, h,$$

and assemble the diagonal matrix $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_h, 0, \ldots, 0)$. This spectral construction induces a spread between the smallest positive and largest eigenvalues, yielding a controlled ill-conditioning. This results in a sparse positive semidefinite matrix with $\text{rank}(Q) \le h < n$ and effective condition number $\kappa_e(Q) \approx \kappa_e^{\text{target}}$.

[1] Implementation details and statistical analyses are available at github.com/JrmBertoncini/DegeneratedQPgenerator.

Constraints linear dependencies Numerical challenges arise for Newton-type solvers when dealing with linearly dependent constraints in (1), because, with $\delta_y = \delta_z = 0$, linear systems (9) become singular. In practice, we generate m equality and p inequality constraints and then a fraction of constraints, $\frac{m}{2}$ and $\frac{p}{2}$, is duplicated and thus linearly dependent, increasing the total number of constraints to $\frac{3m}{2}$ and $\frac{3p}{2}$. In addition, we consider nearly linearly dependent constraints: some duplicated constraints are perturbed by a very small quantity. These constraints remain effectively active while introducing near-degeneracy, which is known to be challenging for numerical solvers.

Initial guess: residual control and resampling It is essential that the training dataset covers a broad range of the state space to ensure that Algorithm 2.1 encounters diverse scenarios during convergence. Coverage can be improved by strategically selecting initial points while keeping generation costs low. If initial points are consistently chosen near feasibility and stationarity, the residuals $(r_{\text{primal}}, r_{\text{dual}})$ will exhibit limited variation, biasing the training distribution toward nearly optimal states. To avoid this, candidate initial solutions (x, y, z) are selected such that the initial residuals satisfy $\log r_{\text{prim}}^{\text{target}}, \log r_{\text{dual}}^{\text{target}} \sim \mathcal{U}(-2, 2)$ and are found by solving a least-squares problem.

The variable $x \in \mathbb{R}^n$ is randomly generated, and the vectors b and d are constructed, aiming at the target primal residual, as

$$b = Ax - r_{\text{prim}}^{\text{target}} \mathbf{1}_m \quad \text{and} \quad d = Gx - r_{\text{prim}}^{\text{target}} \mathbf{1}_p + \boldsymbol{\mu}, \tag{14}$$

where $\boldsymbol{\mu} \in \mathbb{R}^p$ is a random positive value vector ensuring the inequality direction, and $\mathbf{1}_m \in \mathbb{R}^m$ denotes the vector of ones of dimension m. Then, the target dual residual $r_{\text{dual}}^{\text{target}}$ can be obtained by solving the least-squares problem

$$\arg\min_{y,z} \left(\left\| Qx + q + A^\top y + G^\top z \right\|_2 - r_{\text{dual}}^{\text{target}} \right)^2 \quad \text{subject to} \quad z \geq 0 \tag{15}$$

with respect to the multipliers $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^p$. In practice, solving the constrained problem (15) is computationally expensive for large-scale dataset generation. Therefore, the unconstrained counterpart is solved and the inequality multipliers z are then projected onto the nonnegative orthant.

To make the training even lighter, each problem instance is reused with different initial guesses. These additional initializations are sampled randomly around the targeted residual configurations, allowing multiple convergence trajectories to be generated from a single QP instance.

Problem statistics The training and validation datasets are summarized in Table 3, which reports the ranges and distributions of the problem dimensions, the controlled effective condition number, and the sparsity pattern of Q. These settings are used to generate random QPs for both training and validation. The training dataset is created with problem sizes ranging from $n_{\min} = 5$ to $n_{\max} = 50$, in order to generate relatively small and fast to construct QPs.

We introduce in Table 3 several regimes of random problems; Standard, LP-like, High-p, and Equality-heavy, each designed to stress the solver in different ways. The Standard regime provides a balanced mix of equality and inequality constraints,

Table 3 Problem generation setup for training and validation QPs

Setup	n, with $n_{\min} = 5,\ n_{\max} = 50$	
Training	$n \sim \mathcal{U}(n_{\min}, n_{\max})$	
Validation	$n \sim \mathcal{U}(n_{\min}, n_{\max})$	
Scaled $10\times$	$n \sim \mathcal{U}(10 \cdot n_{\min}, 10 \cdot n_{\max})$	
Density (Q, A, G)	$\rho \sim \mathcal{U}(0.15, 0.45)$	
Condition number	$\log \kappa_e^{\text{target}} \sim \mathcal{U}([1, 4] \cup [12, 22])$	
Regime	m	p
Standard	$\frac{n}{\mathcal{U}(1,3)}$	$n \cdot \mathcal{U}(1, 3)$
LP-like	$\{0, 1\}$	$n \cdot \mathcal{U}(1, 4)$
High-p	$\{0, 1, 2\}$	$n \cdot \mathcal{U}(3, 33)$
Equality-heavy	$n \cdot \mathcal{U}(0.4, 0.9)$	0

serving as a baseline with moderate conditioning. The LP-like regime, characterized by a nearly vanishing Hessian ($Q \approx 0$) and few or no equality constraints, leads to almost degenerate linear systems typical of linear programs. The High-p regime introduces a large number of inequality constraints, increasing the size of the active set and the Equality-heavy regime leads to large saddle-point systems that (before regularization) are inherently indefinite.

Figure 2 presents the actual problem sizes and sparsity statistics for the generated datasets, including the number of variables, constraints, and the distribution of nonzeros per row in the constraint matrices. For reference, the figure also includes a comparison with the problem sizes and sparsity patterns from the Maros–Mészáros benchmark set. Importantly, these problem-dependent quantities are not provided to the RL controller, which only observes the state vector σ composed of convergence and barrier-related information. This deliberate design choice keeps the policy lightweight and problem-agnostic by relying exclusively on solver convergence indicators such as residual scalars.

The training random problems exhibit a broad range of nonzero values in Q and $[A^\top G^\top]$, which are further scaled by a factor of $100\times$ in the scaled $10\times$ validation set. In terms of sparsity, these generated problems maintain a consistent pattern across the dataset. In contrast, the Maros–Mészáros benchmark test set displays a wider diversity in both the number of nonzeros and the sparsity patterns, making it challenging to solve as a batch using a single solver configuration.

The distribution of effective condition numbers across the training set is shown in Fig. 3 (left panel). While a target distribution is prescribed during problem generation, the resulting effective condition numbers differ due to the sparsification operator. Nevertheless, this discrepancy introduces additional variability, which is desirable as it increases the diversity of the training set. The right panel of Fig. 3 illustrates the distribution of initial primal and dual residuals across the considered problem classes. The targeted residual values remain within the prescribed range $[10^{-2}, 10^{2}]$, following a log-uniform sampling strategy. In contrast, the resampled problems tend to exhibit

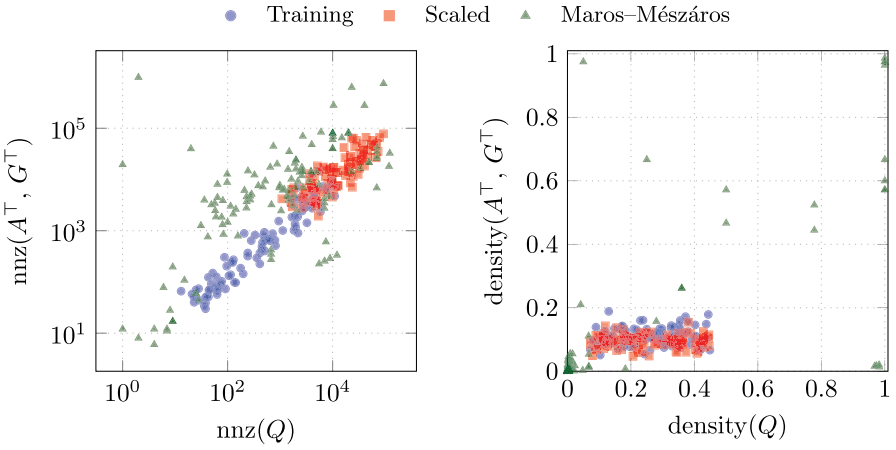


Fig. 2 Distribution of problem size (in terms of number of nonzero entries) and density for randomized training, validation, and scale 10× problems (sample of 10^3 instances) and for Maros-Mészáros problems (100 smallest instances)

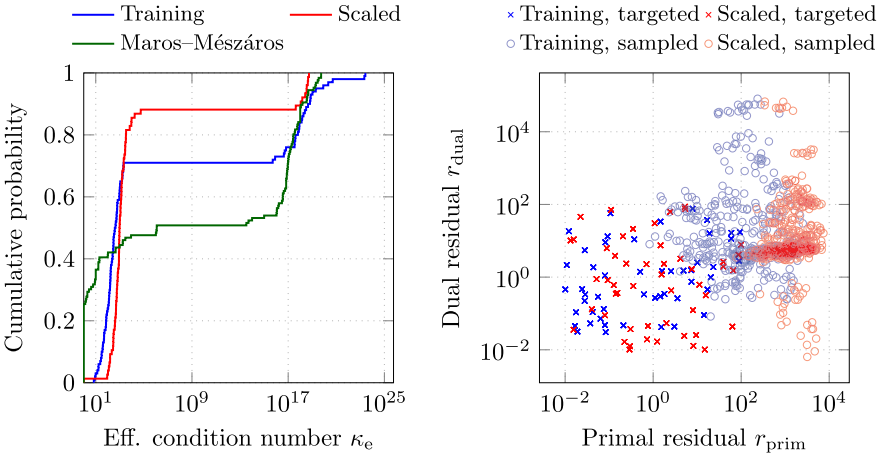


Fig. 3 Left: Empirical cumulative distribution functions (ECDFs) of the effective condition number κ_e for the training set, scaled validation set, and Maros–Mészáros benchmark problems. Right: Distributions of the initial primal and dual residuals over a sample of 10^3 problem instances

larger residuals overall, with even higher values observed for the scaled validation batch.

4 Numerical investigations

We report the outcomes of the training process and corresponding validation, followed by an evaluation on larger randomly generated problems to assess scalability. [2] Finally, we benchmark the approach on the Maros-Mészáros test set to examine its zero-shot generalization on challenging real-world problem instances. The analysis is based on performance and data profiles, consistently comparing the RL-enhanced solver with the typical fixed-α strategy. The robustness and acceleration of the different methods will be discussed in the analysis.

Profiles For P the set of problems and S the set of solvers, let $t_{s,p}$ denote the (wall-clock) runtime required by solver $s \in S$ to solve instance $p \in P$ (for a given tolerance $\epsilon > 0$). We will monitor the computational performance of different solver settings and graphically summarize our results by means of two profiles. When solver $s \in S$ fails to solve $p \in P$, we set $t_{s,p} = \infty$.

- A *data profile* is the graph of the cumulative distribution function $f_s \colon [0, \infty) \to [0, 1]$ of the runtime, namely $f_s(t) := \frac{|\{p \in P \mid t_{s,p} \leq t\}|}{|P|}$, where $|P|$ denotes the cardinality of set P. As such, each data profile reports the fraction of problems $f_s(t)$ that can be solved by s with a computational budget t, and therefore it is independent of the other solvers.
- A *performance profile* is the graph of the cumulative distribution function of the performance ratio $r_{s,p} := \frac{t_{s,p}}{\min_{s' \in S} t_{s',p}}$. As such, it displays the fraction of problems solved within a time factor compared to the fastest solver of each problem. Therefore, the performance profile of a solver also depends on the performance of all other solvers in S.

4.1 Training results

The overall training spans under 8 h and includes 800k randomly generated QP problems. The training progress can be observed in Fig. 4, where the (blue) noisy curves show the raw per-episode values whereas the bold (red) line is a moving-window average. As metrics to monitor the training, we observe the value function loss, the policy loss, the reward and the entropy in each episode. The loss value tends to show the training behavior of the policy and the value function. The reward represents the objective of the process and therefore it needs to be maximized. The entropy is monitored, since it represents the exploration and uncertainty of the policy during the training.

In Fig. 4a it can be observed how the loss of the critic evolves. In first 600k steps a significant reduction of the loss can be noticed, which indicates that the approximation of the value function improves. In the last 200k steps, the PPO algorithm fine-tunes this approximation. In addition to the loss function, Fig. 4b shows the policy loss, which is an indicator of the improvement of the policy. The mostly negative values show that the policy improves, since advantageous actions, which means that they have a positive advantage function approximation, are reinforced, and disadvantageous actions are weakened.

[2] The software developed for these numerical investigations is available at github.com/JrmBertoncini/RL-QProxIP.

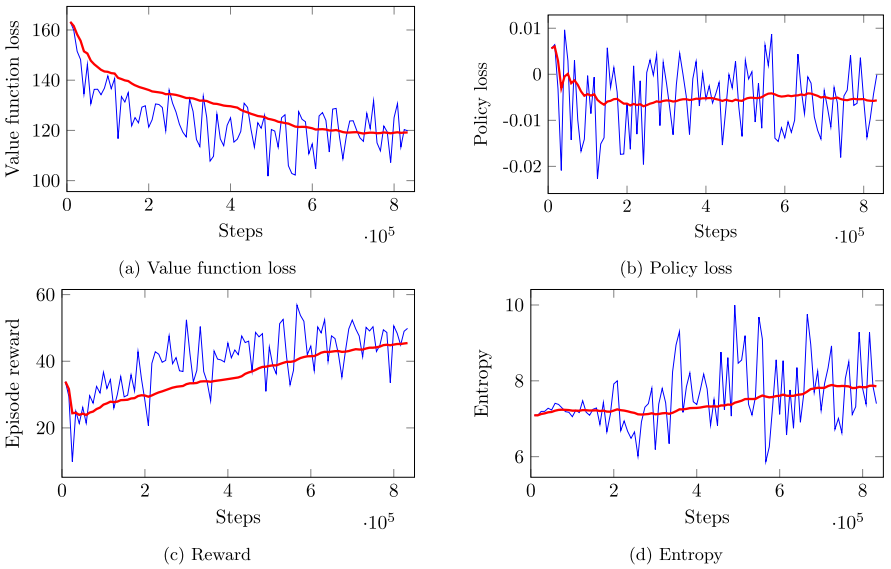


Fig. 4 Evolution of relevant metrics during the training procedure: raw per-episode values (blue) and moving-window average (thick red)

Figure 4c shows the evolution of the episode reward throughout training. After an initial exploration dip at ≈25k steps, the policy recovers and improves monotonically. The variance of the raw signal reflects the probabilistic behavior of the environment while the smoothed trend shows no plateau within the reported horizon.

Furthermore, we plot the entropy during training in Fig. 4d. Entropy can be used to determine how uncertain or exploratory the training is. We see that the entropy increases first, which is a typical behavior, since exploration is needed at the beginning of the training. Afterwards, the entropy stagnate to a certain level. Although this behavior is common for RL algorithms in general, it is noticeable that the final level of the entropy is relatively high. This means that the final policy is relatively uncertain and is not close to a deterministic policy. The investigation of whether an enhanced state space or even longer training phases can make the policy more certain is left for future works. In any case, the plots indicate a successful training, and the influence of the RL policy on the QP solver is discussed in the remainder of this manuscript.

4.2 Validation

After the training, the performance of the method is evaluated on a batch of 100 problems having the same statistical properties as the problems used for the training, (see Table 3). Figure 5 reports the performance and data profiles on the validation set, which is drawn from the same distribution as the training batch and consists of small-scale QPs with $n \in [5, 47]$ (median $n = 17$).

At performance ratio $r = 1$, `RL-QProxIP` solves $\approx$ 50% of problems as the best solver, against $\approx$ 35% for the baseline `QProxIP`, a relative gain of $\approx$ 30% in

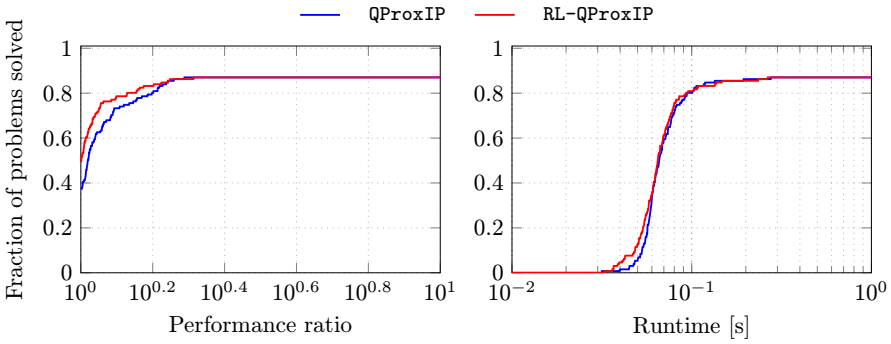


Fig. 5 Performance (left) and data (right) profiles of a validation test-set

problems solved at parity. Both curves saturate at the same ceiling of 87.0% for $r \geq 2$, confirming that the RL policy does not recover any previously unsolvable problem but consistently reaches the solution faster.

The data profile shows that all solvable problems are handled well under one second, with both solvers saturating at $t = 0.5\,\text{s}$. The bulk of the solve time is concentrated around 100 ms (median runtime $\approx$ 65 ms for both solvers. At $t = 50\,\text{ms}$, `RL-QProxIP` has already solved $\approx$ 12% of problems compared to $\approx$ 5% for the baseline. The inference overhead of the neural policy (measured in the running time) recorded amounts to $\approx$ 1 ms per problem, which is negligible relative to the solve time.

On problems of this scale, the iteration counts are already low leaving little room for large absolute savings. A consistent gain of over 30 percentage points in problems solved at parity therefore represents a meaningful improvement: the RL policy systematically selects parameters that reduce the number of innerloop steps even on small, ill-conditioned instances where a hand-tuned baseline already performs well.

Figure 6 shows a 3D voxel-based state-to-action mapping of the learned policy defined over the observation space. Each voxel aggregates all samples falling into the corresponding region of the discretized state space, and the displayed value corresponds to the average of the associated action component over these samples. The marker size is proportional to the number of samples per voxel. Since ε represents the inner-loop tolerance, its value decreases as the solver progresses.

During convergence, the residuals decrease from large to small values, while ε decreases. The resulting trajectories cluster around a hyperplane, as illustrated in Fig. 6. Along these convergence paths, the learned policy exhibits a clear structure across all action components. In particular, α_x, α_y, and α_z remain within approximately $[0.05, 0.3]$, while α_v lies in $[0.2, 0.4]$, and α_ε is consistently higher, in the range $[0.7, 0.95]$. This separation suggests differentiated roles of the control parameters across solver regimes.

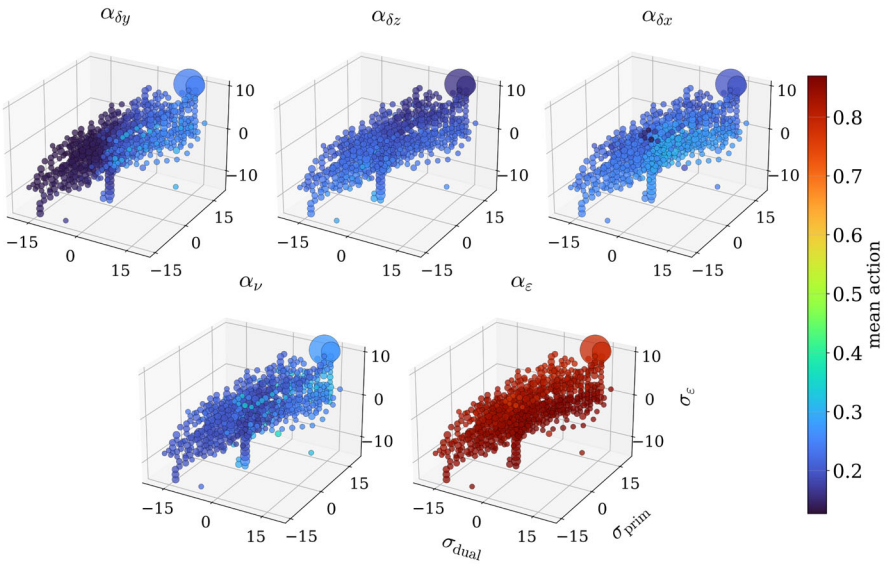


Fig. 6 Mappings α as a function of σ_{prim}, σ_{dual}, and σ_ε evaluated on the 100 smallest Maros–Mészáros problems. The dependency on ν is not shown

4.3 Problem dimensions scaling

When evaluating the RL-policy on the validation set, the observed improvements over the base solver `QProxIP` (with fixed α factors, $\alpha_x = 0.15$, $\alpha_x = 0.2$ and $\alpha_x = 0.2$) remain modest. This is largely due to the relatively small size of the validation problems, which, although ill-conditioned, non-strictly convex, and subject to redundant constraints, still exhibit limited complexity. However, the choice of random problem sizes during training (see Fig. 2 for a comparison of problem properties) was primarily motivated by the need to keep the computational training cost low enough to complete on a personal computer within a single day; see the system configuration in Table 1. A question then arises: How does the policy π_ζ, trained at low cost on small QPs, perform on larger problem instances?

Figure 7 presents the results for a batch of problems whose dimensions and sparsity are scaled by a factor of 10 relative to the validation set. To assess the performance of `QProxIP` and `RL-QProxIP`, we compare it against several widely used quadratic programming solvers representing distinct algorithmic paradigms. `PIQP` is a modern primal-dual interior-point solver designed specifically for quadratic programs. It employs a Mehrotra predictor-corrector strategy to achieve high numerical robustness and efficiency, particularly for high termination tolerance, see [33]. The `OSQP` solver implements ADMM for convex quadratic programs. It is well suited for large, sparse problems, offering predictable performance and simple warm-start capabilities, though it may require more iterations to reach high-accuracy solutions, see [34]. `ProxQP` is a proximal-based quadratic programming solver leveraging iterative refinement and proximal regularization. It provides robust convergence behavior

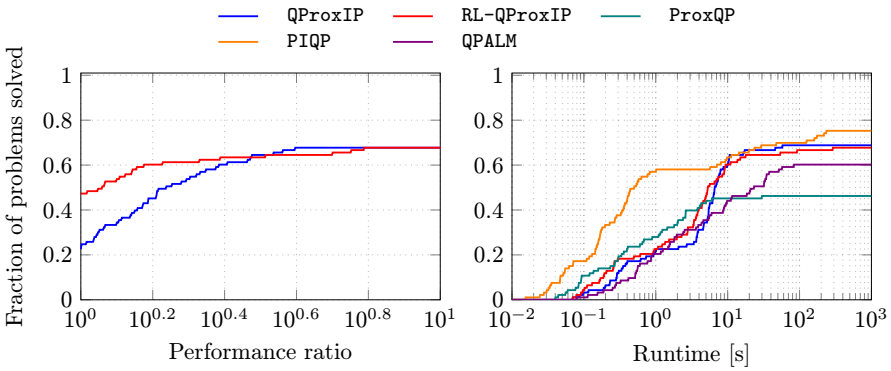


Fig. 7 Scaled 10× set: Performance (left) and data (right) profiles comparing benchmarked solvers on a 10× scaled validation set

and predictable numerical performance, particularly suitable for embedded or control-oriented applications [3].

We note that OSQP is not included in the profiles because it did not converge to the required tolerance for any of the test problems. This is consistent with its design as a first-order ADMM-based solver, which typically struggles to achieve high accuracy. We further emphasize that no preconditioner is used for any of the solvers.

4.4 Maros–Mészáros problems

Training was carried out on randomly generated problem instances, but an important question is how well the method generalizes to established benchmarks. To this end, we evaluate on the Maros–Mészáros test set [25], a widely used benchmark collection of challenging quadratic programs. This suite contains many large-scale and ill-conditioned problems and is considered a standard for assessing the robustness and efficiency of convex QP solvers. For evaluation, we consider the problem in a zero-shot generalization setting and focus on the 100 smallest instances (out of 138), where problem size is determined by folder size. A comparison of these instances with the randomly generated problems used for training is shown in Fig. 2. As before, the Maros–Mészáros set is used in its raw form, meaning that inequality constraints with "infinite" bounds (represented as values of 10^{20} in OSQP) are retained in the problem data (1). To accommodate the increased difficulty of these benchmarks, both the maximum number of inner and outer iterations are increased to 50.

Figure 8 reports the performance and data profiles for QProxIP and RL-QProxIP. RL-QProxIP is the fastest solver on about 70% of instances (performance ratio $r = 1$), while QProxIP requires $r \approx 3.5$ to reach the same fraction. The curves cross at $r \approx 8.8$: QProxIP solves 100% of instances while RL-QProxIP saturates at $\approx 92\%$. The curves meet at $r \approx 8.8$, with about 85% of instances solved; while QProxIP eventually solves about 96% of instances, rlqpsolver saturates at about 85%. In absolute runtime, RL-QProxIP solves 50% of instances within $t \approx 44$ ms, roughly 2.3× faster than QProxIP.

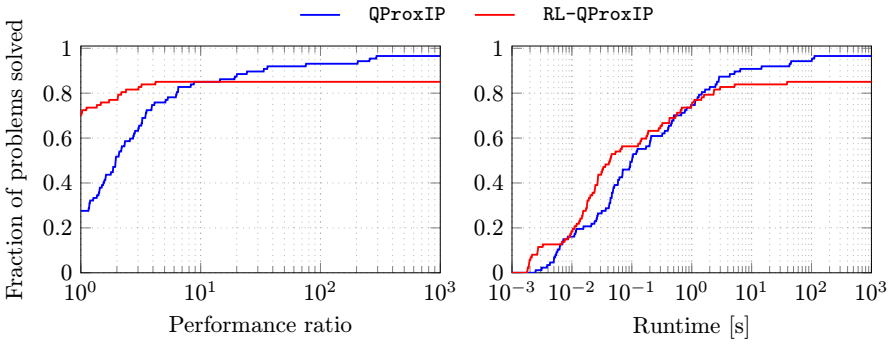


Fig. 8 Performance (left) and data (right) profiles on the Maros–Mészáros problems

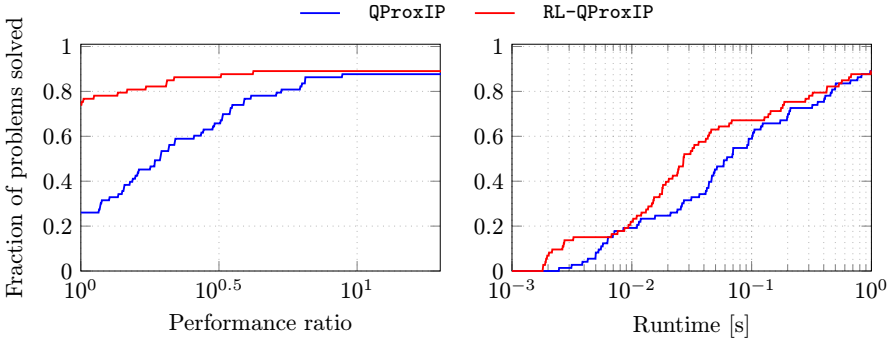


Fig. 9 Performance (left) and data (right) profiles on the subset of 73 Maros–Mészáros problems solved by at least one of the two solvers (`QProxIP` or `RL-QProxIP`) within the one-second time budget

To further analyze the effect of the RL policy, we restrict the benchmark to the subset of problems for which `RL-QProxIP` terminates within one second. This threshold is approximately 10^3 times longer than the time required to solve the small random problems during the training phase. Thus, it provides a genuine out-of-distribution evaluation.

Figure 9 presents the performance and data profiles on the subset of 73 Maros–Mészáros problems that at least one of the two solvers (`QProxIP` or `RL-QProxIP`) solves within a one-second time budget. We exclude only the instances that neither solver can handle under the one second time limit. The performance profile indicates that `RL-QProxIP` is the fastest solver on 74% of the problems (performance ratio $r = 1$), compared to 26% for `QProxIP`. In absolute terms, `RL-QProxIP` solved 66 of the 73 instances, while `QProxIP` solved 65. These results demonstrate `RL-QProxIP` generalizes effectively beyond its training distribution, delivering consistent speedups even on significantly larger and more challenging problems.

Table 4 illustrates a subset of problems where this advantage is most pronounced, with speedups reaching almost 300× in the most extreme cases. Interestingly, PRIMALCx and QSHIP0xx are very sparse with far more variables than constraints, while DUALC8 is dense with more constraints than variables: this suggests that the learned

Table 4 Comparison on problems where the RL-enhanced solver `RL-QProxIP` is at least 10× faster than the base solver `QProxIP`

Problem	`QProxIP` [s]	`RL-QProxIP` [s]	Speedup
PRIMALC8	6.433	0.0219	294
PRIMALC2	5.214	0.0203	257
PRIMALC5	2.005	0.0098	205
PRIMALC1	1.146	0.0151	76
DUALC8	0.9782	0.0271	36
QSHIP08S	101.1	2.999	34
QSHIP04S	1.666	0.0683	24
QSHIP04L	2.867	0.1436	20
QETAMACR	45.43	2.371	19
CVXQP2_M	14.71	1.013	15

policy is most valuable on such structural edge cases, which depart from the regime for which the baseline's fixed schedule was tuned.

Beyond the specific instances in Table 4, the performance profiles reported throughout Sect. 4 point to a broader pattern: `RL-QProxIP` consistently adapts better than the fixed-α baseline across a wide range of problem sizes, scales, and structures, even when the magnitude of the improvement is modest. On the validation and scaled 10× sets (Figs. 5 and 7), `RL-QProxIP` is more often faster than `QProxIP`. On the full Maros–Mészáros set (Fig. 8), this gap widens, though `RL-QProxIP` saturates slightly lower, indicating that its consistent speed advantage comes with a marginal trade-off in robustness on a handful of instances. Overall, it is this reliability that we consider the more informative outcome, rather than the size of the peak speedups.

5 Final remarks and outlook

We presented a method based on reinforcement learning to speed up and harden the convergence of a regularized interior-point solver for convex quadratic programming. We used RL to learn a policy aimed at adapting the solver's hyperparameters to reduce the number of iterations needed to obtain a solution. For the RL training, we generated a framework to design small random QPs, which serves as good training problems. Then the parameterized RL policy, which is constructed as an artificial neural network, is trained on these QPs. In experiments, the results suggest that a single policy can improve convergence rates for a broad class of problems. In particular, the policy exhibits an input–output behavior that is interpretable, based on the underlying QP method, and it can successfully be used for problems with different sizes, scales and structures. Compared to previous approaches, like RLQP [21, §6], our lightweight training achieves similar validation performance with a fraction of time and memory (7 h on a laptop rather than days on a high-end computer). For QPs that are solved in only a few iterations and thus require limited hyperparameter adaptation, a learned

policy may provide only marginal additional benefit. In these cases, the inference overhead of evaluating the RL policy can partially offset the gains from faster convergence. Nevertheless, already for problems with sub-second solve times, `RL-QProxIP` is competitive with state-of-the-art solvers while achieving speed-ups at a comparable level of robustness.

These findings should be read against a few limitations of the present study. Training was restricted to small random QPs (Table 3) to keep the computational cost of policy learning tractable on a single laptop; validation was carried out on a single interior-point solver scheme; and the entropy of the trained policy plateaus at a relatively high level (Fig. 4d), indicating that it has not converged to a near-deterministic behavior. Addressing these aspects motivates the extensions outlined next.

Future work will investigate the policy performance to other QP problem classes, such as large-scale sparse problems arising in power systems, optimal control problems showing particular three-banded structures or financial optimization with highly volatile market data. Moreover, for even more adaptive RL approaches, the policy could continue learning and improving during solver deployment. Moreover, we will assess whether the presented RL policy, trained within a primal–dual interior-point framework, transfers to other second-order optimization schemes such as primal interior-point methods. More broadly, this raises the question of generalization across Newton-type algorithms, such as between augmented Lagrangian and interior-point methods. Such an analysis would help clarify to what extent the learned RL strategy captures algorithm-agnostic structure of second-order optimization dynamics, as opposed to being specific to a particular solver architecture.

Funding Open Access funding enabled and organized by Projekt DEAL. This research is funded by dtec.bw—Digitalization and Technology Research Center of the Bundeswehr [MissionLab, SeRANIS]. dtec.bw is funded by the European Union—NextGenerationEU.

Data availability The source code to reproduce the findings of this study is available online at github.com/JrmBertoncini/RL-QProxIP. The random QP generator can be found at https://github.com/JrmBertoncini/DegeneratedQPgenerator.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Amos, B.: Tutorial on amortized optimization (2023)
2. Audet, C., Dang, K.-C., Orban, D.: Optimization of algorithms with OPAL. Math. Program. Comput. **6**(3), 233–254 (2014)
3. Bambade, A., Schramm, F., El-Kazdadi, S., Caron, S., Taylor, A., Carpentier, J.: ProxQP: an efficient and versatile quadratic programming solver for real-time robotics applications and beyond. IEEE Transactions on Robotics, pp. 1–19 (2025)
4. Bemporad, A.: A numerically stable solver for positive semidefinite quadratic programs based on nonnegative least squares. IEEE Trans. Autom. Control **63**(2), 525–531 (2018)
5. Birgin, E.G., Martínez, J.M.: Augmented Lagrangian method with nonmonotone penalty parameters for constrained optimization. Comput. Optim. Appl. **51**(3), 941–965 (2012)
6. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends® Mach. Learn. **3**(1), 1–122 (2011)
7. Castera, C., Ochs, P.: From learning to optimize to learning optimization algorithms. In Li, Y., Mandt, S., Agrawal, S., Khan, E (eds.), Proceedings of the 28th International Conference on Artificial Intelligence and Statistics, volume 258 of Proceedings of Machine Learning Research, pp. 1792–1800 (2025)
8. Cipolla, S., Gondzio, J.: Proximal stabilized interior point methods and low-frequency-update preconditioning techniques. J. Optim. Theory Appl. **197**(3), 1061–1103 (2023)
9. d'Aspremont, A., Scieur, D., Taylor, A.: Acceleration methods. Found. Trends® Optim. **5**(1–2), 1–245 (2021)
10. De Marchi, A.: On a primal-dual Newton proximal method for convex quadratic programs. Comput. Optim. Appl. **81**(2), 369–395 (2022)
11. De Marchi, A.: Regularized interior point methods for constrained optimization and control. IFAC-PapersOnLine **56**(2), 1247–1252 (2023)
12. Feinberg, E.A., Shwartz, A. (eds.): Handbook of Markov Decision Processes: Methods and Applications. International Series in Operations Research & Management Science, vol. 40. Springer, Boston (2002)
13. Luque, F.J.: Asymptotic convergence analysis of the proximal point algorithm. SIAM J. Control. Optim. **22**(2), 277–293 (1984)
14. Ferreau, H.J., Kirches, C., Potschka, A., Bock, H.G., Diehl, M.: qpOASES: a parametric active-set algorithm for quadratic programming. Math. Program. Comput. **6**(4), 327–363 (2014)
15. Gill, P.E., Robinson, D.P.: A primal-dual augmented Lagrangian. Comput. Optim. Appl. **51**(1), 1–25 (2012)
16. Giselsson, P., Boyd, S.: Linear convergence and metric selection for Douglas-Rachford splitting and ADMM. IEEE Trans. Autom. Control **62**(2), 532–544 (2017)
17. Gondzio, J.: Interior point methods 25 years later. Eur. J. Oper. Res. **218**(3), 587–601 (2012)
18. Goujaud, B., Moucer, C., Glineur, F., Hendrickx, J.M., Taylor, A.B., Dieuleveut, A.: PEPit: computer-assisted worst-case analyses of first-order optimization methods in python. Math. Program. Comput. **16**(3), 337–367 (2024)
19. Grippo, L., Lampariello, F., Lucidi, S.: A nonmonotone line search technique for Newton's method. SIAM J. Numer. Anal. **23**(4), 707–716 (1986)
20. Hermans, B., Themelis, A., Patrinos, P.: QPALM: a proximal augmented Lagrangian method for nonconvex quadratic programs. Math. Program. Comput. **14**(3), 497–541 (2022)
21. Ichnowski, J., Jain, P., Stellato, B., Banjac, G., Luo, M., Borrelli, F., Gonzalez, J.E., Stoica, I., Goldberg, K.: Accelerating quadratic optimization with reinforcement learning. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P.S., Wortman Vaughan, J. (eds.), Advances in Neural Information Processing Systems, vol. 34, pp. 21043–21055 (2021)
22. Li, K., Malik, J.: Learning to optimize. In: International Conference on Learning Representations (2017)
23. Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J.E., Jordan, M.I., Stoica, I.: RLlib: Abstractions for distributed reinforcement learning. In: Dy, J., Krause, A (eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 3053–3062 (2018)
24. Liao-McPherson, D., Kolmanovsky, I.: FBstab: a proximally stabilized semismooth algorithm for convex quadratic programming. Automatica **113**, 108801 (2020)

25. Maros, I., Mészáros, C.: A repository of convex quadratic programming problems. Optim. Methods Softw. **11**(1–4), 671–681 (1999)
26. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends® Optim. **1**(3), 127–239 (2014)
27. Pougkakiotis, S., Gondzio, J.: An interior point-proximal method of multipliers for convex quadratic programming. Comput. Optim. Appl. **78**(2), 307–351 (2021)
28. Rockafellar, R.T.: Augmented Lagrangians and applications of the proximal point algorithm in convex programming. Math. Oper. Res. **1**(2), 97–116 (1976)
29. Rockafellar, R.T.: Monotone operators and the proximal point algorithm. SIAM J. Control. Optim. **14**(5), 877–898 (1976)
30. Sambharya, R., Hall, G., Amos, B., Stellato, B.: End-to-end learning to warm-start for real-time quadratic optimization. In: Matni, N., Morari, M., Pappas, G.J. (eds.), Proceedings of the 5th Annual Learning for Dynamics and Control Conference, volume 211 of Proceedings of Machine Learning Research, pp. 220–234 (2023)
31. Sambharya, R., Hall, G., Amos, B., Stellato, B.: Learning to warm-start fixed-point optimization algorithms. J. Mach. Learn. Res. **25**(166), 1–46 (2024)
32. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017)
33. Schwan, R., Jiang, Y., Kuhn, D., Jones, C.N.: PIQP: A proximal interior-point quadratic programming solver. In: 2023 62nd IEEE Conference on Decision and Control (CDC), pp. 1088–1093 (2023)
34. Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S.: OSQP: an operator splitting solver for quadratic programs. Math. Program. Comput. **12**(4), 637–672 (2020)
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. The MIT Press, USA (2018)
36. Wei, K., Aviles-Rivero, A., Liang, J., Ying, F., Huang, H., Schönlieb, C.-B.: TFPnP: tuning-free plug-and-play proximal algorithms with applications to inverse imaging problems. J. Mach. Learn. Res. **23**(16), 1–48 (2022)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.