Warm-starting active-set solvers using graph neural networks

Ella J. Schmidtobreick ELLA-JOHANNA.SCHMIDTOBREICK@IT.UU.SE
Daniel Arnström DANIEL.ARNSTROM@IT.UU.SE
Paul Häusner PAUL.HAUSNER@IT.UU.SE
Jens Sjölund JENS.SJOLUND@IT.UU.SE
Department of Information Technology, Uppsala University, Sweden

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

Quadratic programming (QP) solvers are widely used in real-time control and optimization, but their computational cost often limits applicability in time-critical settings. To resolve this, we propose a learning-to-optimize approach using graph neural networks (GNNs) to predict active constraints in the dual active-set solver `DAQP`. Our method exploits the structural properties of QPs by representing them as bipartite graphs and learns to approximate the optimal active set for effectively warm-starting the solver. Across varying problem sizes, the GNN consistently reduces the number of solver iterations compared to cold-starting, while performance is comparable to a multilayer perceptron baseline. In contrast to the baseline, our GNN-based approach trained on varying problem sizes generalizes to unseen dimensions, demonstrating flexibility and scalability. These results highlight the potential of structure-aware learning to accelerate optimization in real-time applications such as model predictive control.

1. Introduction

Optimization problems with quadratic objectives and linear constraints, referred to as quadratic programs, form the foundation of numerous applications, including robotics (Kuindersma et al., 2014), control (Bartlett et al., 2002), and finance (Gondzio and Grothey, 2007; Mitra et al., 2007). Given their prevalence, efficient and reliable solution methods are critical for practical applications.

Convex QPs can be solved to global optimality with polynomial complexity (Nocedal and Wright, 1999). While interior point methods are highly effective in general, they cannot leverage approximate solutions from related instances, reducing their efficiency for sequential problems. In contrast, active-set methods naturally support this technique, known as warm starting, by iteratively refining a hypothesis of the active constraint set. Moreover, they are particularly well suited for small- and medium-scale problems (Nocedal and Wright, 1999).

Solving QPs efficiently is critical in for example real-time model predictive control (MPC) (Borrelli et al., 2017). These applications provide sequences of closely related optimization problems, making it ideal to exploit solutions from previous iterations. However, naively using the previous solution as a starting iterate can exacerbate worst-case solution time (Herceg et al., 2015). Recent works have explored ways of finding warm starts by using machine learning to adapt optimization solvers to problem classes characterized by prior instances (Chen et al., 2022b; Amos, 2023).

These learning-to-optimize approaches vary in how closely they adhere to conventional solvers. Some focus on tuning hyperparameters (Sambharya and Stellato, 2024; Doerks et al., 2025), others learn entire update steps as a black box (Andrychowicz et al., 2016), and hybrid methods combine learned updates with traditional steps to retain convergence guarantees (Banert et al., 2024).

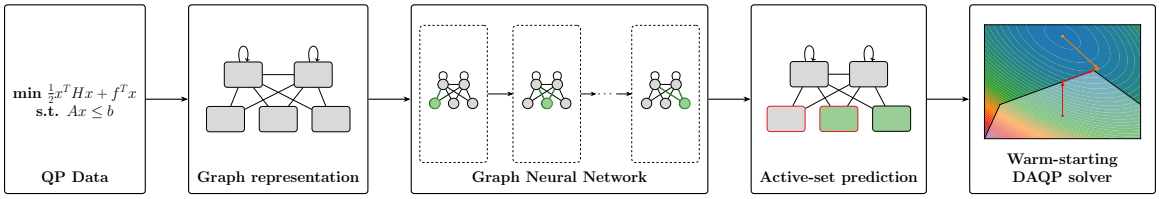


Figure 1: Method overview: The problem is formulated as a QP and represented as a graph, which serves as input to the Graph Neural Network for node-level prediction. The resulting prediction is then used to warm-start the `DAQP` solver.

Sambharya et al. (2023) employs an end-to-end approach by predicting initial iterates to warm-start the solver, which are subsequently used to obtain a candidate solution. Prior work has also explored learning penalty parameters for splitting-based QP solvers (Ichnowski et al., 2021; Saravanos et al., 2025), or predicting search directions for interior-point methods using LSTMs (Gao et al., 2024). Further hybrid approaches employ classification trees or k-NN classifiers (Klaučo et al., 2019), neural networks (Chen et al., 2022a) or transformers (Zinage et al., 2025) to predict active constraints. However, most machine-learning-based approaches for active-set methods ignore the structural properties inherent in optimization problems (Klaučo et al., 2019; Chen et al., 2022a; Zinage et al., 2025).

In contrast, we investigate how graph neural networks (GNN) accelerate the active-set method for solving QPs. This technique leverages the underlying problem structure using graph representations similar to Sjölund and Bånkestad (2022); Häusner et al. (2024, 2025). By predicting the active set, our proposed approach provides an improved initial guess compared to conventional initialization strategies, enabling the solver to converge in fewer iterations and thereby reducing computation time. The method overview is depicted in Figure 1. GNNs are particularly well-suited for this task since they inherently capture the structural properties of QPs. They are invariant to node permutations, corresponding to reordering constraints, and can exploit sparsity patterns effectively (Cappart et al., 2023). Previous work has transformed QPs into graph representations to study the existence of GNNs capable of mapping problems to optimal solutions (Chen et al., 2025), but lacks extensive experimental validation, a gap we address in this paper.

The main contributions of this paper are (i) the first application of GNNs to accelerate QP solving by learning the active set, and (ii) a comprehensive experimental evaluation demonstrating the practical benefits of this approach over multilayer perceptrons, including size generalization.

2. Background

In this paper, we use machine learning to accelerate solving convex quadratic programs (QPs) with linear inequality constraints,

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}x^T Hx + f^T x, \qquad \text{subject to} \quad Ax \leq b. \tag{1}$$

The objective function is defined by the positive definite matrix $H \in \mathbb{R}^{n\times n}$ and the vector $f \in \mathbb{R}^n$, while the feasible region is defined by $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Convexity ensures the global optimum can be found efficiently, with computational difficulty comparable to linear programs (Nocedal and Wright, 1999).

Many applications require solving a sequence of similar QPs, where only specific parameters change between iterations. We consider problems where the parameter $\eta \in \mathbb{R}^p$ only affects the linear term in the objective function f and the right-hand side of the constraints b, i.e.

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2} x^T H x + f(\eta)^T x, \qquad \text{subject to} \quad Ax \leq b(\eta). \tag{2}$$

Hence, the overall problem structure is fixed across problem instances of equal dimensions.

2.1. Active-set method

The active-set method is an iterative solution method for QPs aiming to identify the active set $\mathcal{A}^* \subseteq \{1, 2 \ldots, m\}$, consisting of all inequality constraints that hold with equality, at an optimal solution x^*. Knowing the active set simplifies the optimization problem by transforming an inequality-constraint problem into one with a reduced set of equality constraints, as stated in Lemma 3.1 in Arnström (2023). The KKT optimality conditions of this reduced problem form a linear system, which can be solved in a single iteration via matrix factorization techniques.

To identify the active set $\mathcal{A}^*$, a working set $\mathcal{W}$ serves as an approximation of the active set. It is updated iteratively by adding or removing one constraint per iteration until the true active set of an optimal solution x^* is found and the linear system can be solved in a single step. Solvers initialized with an empty working set $\mathcal{W}_0 = \emptyset$ are cold-started, while warm-started solvers begin with a non-empty working set $\mathcal{W}_0 \neq \emptyset$, which can significantly reduce the number of iterations needed to identify the active set and thereby the optimum (Otta et al., 2015; Arnström, 2023). While the improvement is most pronounced when the initial working set $\mathcal{W}_0$ closely matches the true active set, even partial overlap can noticeably reduce the iteration count. This motivates a learning-to-optimize approach: a perfect prediction is hard (essentially equivalent to solving the problem), but leveraging prior instances can yield a close approximation and practically significant acceleration.

2.2. Dual active-set solver

To benefit from warm-starting, we apply the dual active-set solver proposed by Arnström et al. (2022). Unlike primal solvers, which require a feasible initial iterate x_0, dual solvers are easier to warm-start, as dual non-negativity always allows $\lambda_0 = \mathbf{0}$ as a feasible starting point. The `DAQP` solver[1] follows Algorithm 1 (Appendix A) from Arnström et al. (2022) and leverages an LDL^T factorization for efficient updates. Since the working set changes by at most one constraint per iteration, the lower unit triangular matrix L and diagonal matrix D are updated via rank-one modifications, reducing complexity. Singularity is detected directly from the diagonal of D, and the computations rely on forward/backward substitution, avoiding explicit matrix inversion. The reuse of previous computations depends on where in the ordered representation of the working set $\mathcal{W}$ the modification occurs, changes near the end allow most of the factorization to be reused, whereas earlier changes require recomputing a larger portion during the substitution steps.

2.3. Graph Neural Networks

Graph neural networks (GNN) are neural network architectures operating on graphs. A graph is defined by the vertex set V and the directed edge set $E \subseteq V \times V$. Each vertex $s \in V$ is assigned a

1. https://github.com/darnstrom/daqp

feature vector $x_s \in \mathbb{R}^{d_v}$, while each directed edge $e_{ts} \in E$ from vertex t to s is associated with an edge feature vector $z_{ts} \in \mathbb{R}^{d_e}$. The model consists of multiple GNN layers, each updating vertex and edge features. In this paper, we follow the framework of message-passing GNNs (Battaglia et al., 2018). The update is performed by the permutation-invariant aggregation function $\bigoplus$ acting on the neighborhood of each vertex, and the learnable functions ϕ and ψ. The graph topology remains fixed throughout the forward pass, while vertex and edge features are updated at each layer.

The update of layer l begins with computing the edges features for the subsequent layer $l+1$ using the parametrized message function $\phi_{\theta_z^{(l)}}$

$$z_{ts}^{(l+1)} := \phi_{\theta_z^{(l)}}\left(z_{ts}^{(l)}, x_t^{(l)}, x_s^{(l)}\right), \tag{3}$$

where the output is referred to as message. In the next step, all incoming messages to a vertex s are aggregated by applying the aggregation function $\bigoplus$ over the neighborhood $\mathcal{N}_s = \{t \mid (t,s) \in E\}$,

$$m_s^{(l+1)} := \bigoplus_{t \in \mathcal{N}_s} z_{ts}^{(l+1)}. \tag{4}$$

Common choices for $\bigoplus$ include summation, maximization, and averaging. The outcome $m_s^{(l+1)}$ serves as input for the vertex feature update for layer $l+1$ by the parametrized function $\psi_{\theta_x^{(l)}}$

$$x_s^{(l+1)} := \psi_{\theta_x^{(l)}}\left(x_s^{(l)}, m_s^{(l+1)}\right). \tag{5}$$

The update functions ϕ and ψ are parametrized by learnable parameters θ (Bronstein et al., 2021). To apply this method to solve QPs, we next describe how they are represented as graphs.

3. Method

The proposed GNN-based approach for solving QPs comprises three key components: the graph representation of the data, the learned mapping, and the final model architecture.

3.1. Graph representation

We represent the underlying QPs as bipartite graphs following Chen et al. (2025).

Definition 1 *The graph representation of a QP is defined by $G = (W \cup C,\ E_W \cup E_C)$, where*

- *the set $W = \{1, \dots, n\}$ represents the variable nodes. The i-th vertex represents the i-th variable of the QP and the assigned feature vector $x_s \in \mathbb{R}^w$ captures the corresponding component of f as well as the node type.*
- *the set $C = \{n+1, \dots, m\}$ represents the constraint nodes. The j-th vertex represents the j-th constraint of the QP and the assigned feature vector $x_s \in \mathbb{R}^c$ captures the corresponding component of b as well as the direction of the inequality sign and the node type.*
- *the edges contained in E_W and their feature vectors are given by the adjacency matrix H, which defines the quadratic term in the QP.*
- *the edges contained in E_C and their feature vectors are given by the adjacency matrix A and show the relations between the variable and constraint nodes.*

This is exemplified in Figure 2.

$$\begin{aligned}
\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}\begin{pmatrix}x_1 & x_2\end{pmatrix}\underbrace{\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}}_{=H}\begin{pmatrix}x_1\\ x_2\end{pmatrix} + \underbrace{\begin{pmatrix}-4 & -8\end{pmatrix}}_{=f}\begin{pmatrix}x_1\\ x_2\end{pmatrix} \\
\text{subject to} \quad & \begin{pmatrix} x_1 + x_2 \\ -x_1 + 2x_2 \\ \underbrace{-3x_1 + x_2}_{=A} \end{pmatrix} \leq \begin{pmatrix} 3 \\ 0 \\ \underbrace{10}_{=b} \end{pmatrix}.
\end{aligned}$$

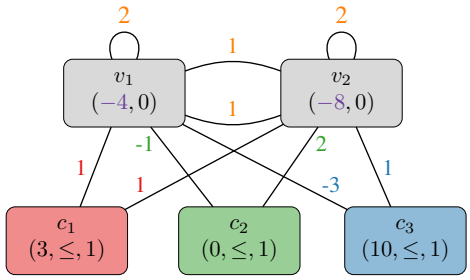


Figure 2: A QP can be represented as a bipartite graph having a set of variable nodes (gray) and a set of constraint nodes (colored), with edge features representing relationships between variables.

3.2. Learning problem

The key idea is to learn the parameters θ of a mapping p_θ from a problem instance represented by a graph G to the resulting active set $\mathcal{A}^*$

$$p_\theta : \mathcal{G} \to \mathcal{P}(\mathbb{N}_m), \quad p_\theta(G) \approx \mathcal{A}^*, \tag{6}$$

where $\mathcal{G}$ denotes the set of all graphs with m variable nodes and $\mathcal{P}$ the power set. This corresponds to a node classification task on the graph G, determining whether each constraint vertex is active or not. Finding this mapping is not trivial, since the optimal solution x^* must be known to determine the active set $\mathcal{A}^*$ (Zinage et al., 2025). The model is trained in a supervised fashion, where ground-truth active sets are obtained by solving problem instances to optimality using the cold-started `DAQP` solver, and the resulting predictions are used to warm-start the solver at inference time.

This approach has the potential to automate and improve on the non-trivial task of finding a good initial guess, while retaining the theoretical convergence guarantees of conventional solvers. Moreover, the graph representation implies permutation equivariance, such that the order of constraints does not affect the model. Finally, GNNs can operate on graphs of varying sizes, making them easily adaptable to different problem structures (Chen et al., 2025).

3.3. Model architecture

The only restriction our approach places on the GNN architecture is to operate on the bipartite graph representation in Definition 1. As a proof of concept, this paper uses Local Extrema Convolution (LEConv) layers (Ranjan et al., 2020), leaving the design of tailored layers to future work.

In the LEConv architecture, the edge feature update for layer $l+1$ is defined by the function $\phi_{\theta_z^{(l)}}$ in Equation (3) as

$$z_{ts}^{(l+1)} := z_{ts}^{(l)} \cdot \left(x_s^{(l)} W_2 - x_t^{(l)} W_3 \right), \tag{7}$$

where $z_{ts}^{(l)}$ represents the feature of the directed edge from node t to s in layer l, $x_s^{(l)}$ and $x_t^{(l)}$ are the corresponding vertex features, and W_j are learnable weight matrices. Sum aggregation as in Equation (4) yields the message for node s in layer $l+1$

$$m_s^{(l+1)} := \sum_{t\in\mathcal{N}_s} z_{ts}^{(l)} \left(x_s^{(l)} W_2 - x_t^{(l)} W_3 \right). \tag{8}$$

The vertex update function $\psi(\cdot)$, as introduced in Equation (5), then combines the node's previous feature with the aggregated message using an activation function $\sigma(\cdot)$:

$$x_s^{(l+1)} := \sigma\left(x_s^{(l)} W_1 + m_s^{(l+1)}\right) = \sigma\left(x_s^{(l)} W_1 + \sum_{t \in \mathcal{N}_s} z_{ts}^{(l)} \left(x_s^{(l)} W_2 - x_t^{(l)} W_3\right)\right). \tag{9}$$

The final model consists of three layers, an input layer, a hidden layer of width 128 and an output layer. All layers share the same architectural structure, presented above. LeakyRELU was chosen as the activation function with a negative slope of 0.1. A sigmoid activation function is applied to the output layer to interpret the outputs as probabilities of a node being active or not.

The model is trained using the AdamW optimizer with default learning rates and a weighted binary cross-entropy loss, where class weights reflect the empirical class distribution. Early stopping based on validation loss is applied, terminating training if the loss does not improve by at least 0.001 over five consecutive epochs, with the best-performing parameters retained. The threshold for predicting active constraint is set by performing a grid search over $[0, 1]$ with a step size of 0.1.

4. Experiments

The experiments evaluate the model on synthetic data to assess its ability to predict the active set and compare it against a MLP. Subsequently, the GNN is applied to datasets with varying problem sizes and an inverted pendulum control problem. The results examine the predictive accuracy and the impact on solver efficiency. All experiments are conducted on a single NVIDIA Titan Xp GPU.

4.1. Synthetic data generation

The first experiments use synthetic data parameterized by $\eta \in \mathbb{R}^p$ as in Equation (2). For each problem dimension, the matrices H and A are generated once and kept constant across all instances, while only the parameter η varies. To ensure that the matrix in the objective function H is symmetric and positive definite, it is generated using the matrix $M \in \mathbb{R}^{n \times n}$ with $H = MM^T$. If required, sparsity patterns can be applied to the matrices. The right-hand side of the constraints and the linear term in the objective function are defined by affine transformations $f : \mathbb{R}^p \to \mathbb{R}^n$ and $b : \mathbb{R}^p \to \mathbb{R}^m$,

$$f(\eta) = \hat{f} + F\eta, \qquad b(\eta) = \hat{b} - AT\eta, \tag{10}$$

where $\hat{f} \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times p}$, $\hat{b} \in \mathbb{R}^m$ and the transformation matrix $T \in \mathbb{R}^{n \times p}$ ensures primal feasibility of $x = T\eta$. All variables are sampled from a standard normal distribution $\mathcal{N}(0, 1)$, with the exception of $\hat{b}$, which is drawn from a uniform distribution over $[0, 1)$ to ensure that the origin is feasible. This guarantees that the `DAQP` solver used in the implementation finds a feasible solution.

4.2. Active-set prediction using GNN

To demonstrate the ability of capturing the problem structure using GNNs, an experimental pipeline was implemented following the methods in Section 3. The synthetic data, generated as described in Section 4.1, was converted into graph representations according to Section 3.1 and used as input to the model detailed in Section 3.3.

The evaluation uses standard classification metrics computed only over constraint nodes, as these are the only relevant nodes in this node-level classification task. All metrics are reported as percentages and rounded to two decimal places. As a baseline for comparison, a naïve model that classifies all nodes as inactive, corresponding to cold-starting the solver, is given.

The dataset contains 5000 instances with $n = 10$ (variables) and $m = 40$ (constraints). Table 1 shows averages over five runs with different random seeds. The GNN achieves over 89% across all metrics on the test set, clearly outperforming the naïve baseline (Accuracy 82.96%), indicating its strong predictive capability. The accuracy is the highest metric due to the unbalanced dataset, as each graph contains more inactive than active nodes.

Table 1: Metrics of the GNN model trained on synthetic data with $n = 10$ (variables) and $m = 40$ (constraints).

Metric	Test
Accuracy (%)	95.43 ± 0.4
Precision (%)	89.53 ± 1.4
Recall (%)	89.74 ± 1.7
F1-score (%)	89.62 ± 0.9

4.3. Impact of problem structure on predictive performance

To assess whether incorporating problem structure through a graph neural network improves predictive performance, we compare it to a standard multilayer perceptron (MLP). For the MLP, all QP components are concatenated into a single input vector by flattening $H \in \mathbb{R}^{n\times n}$ and $A \in \mathbb{R}^{m\times n}$, and appending $f \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, yielding a vector of size $n^2+mn+n+m$. The MLP produces binary predictions for all $n + m$ elements, enabling direct comparison with the GNN, though only the m constraint predictions are of primary interest. For each problem size, 2000 QPs are generated with shared matrices H and A and sparsity is introduced via banded matrices M and A. For different problem sizes, new matrices are generated. It is important to note that an MLP has a fixed input size, such that a separate model needs to be trained for each problem size.

The comparison considers the number of iterations (Figure 3) and total solve time (Figure 4) for increasing problem sizes, while maintaining a fixed 1:4 ratio between variable and constraint nodes. All values represent the mean over five independent runs per problem size.

Figure 3 shows that warm-starting the `DAQP` solver with predictions from either model substantially reduces the iteration count compared to cold-starting. The iteration growth remains sublinear across all approaches.

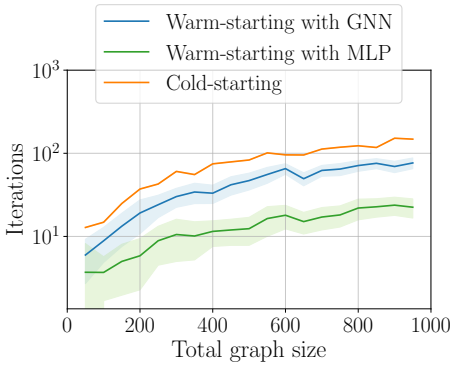
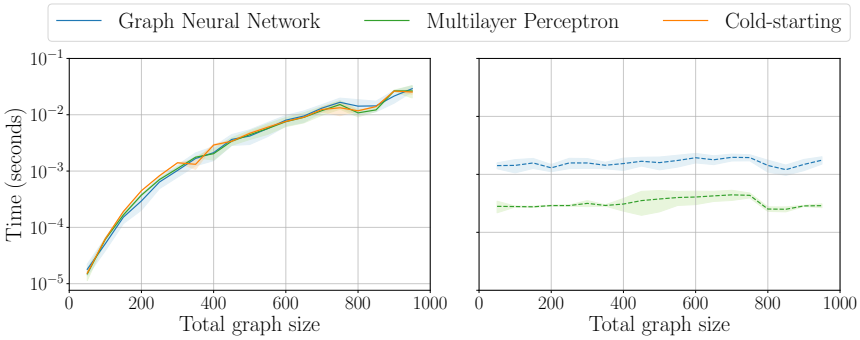


Figure 3: Comparison of iterations when warm-starting the `DAQP` solver with predictions from our graph neural network, a standard multilayer perceptron and cold-starting the solver without any learned prediction.

Figure 4 compares solve and prediction times across varying problem sizes. The solve time is measured using the built-in timing of the `DAQP` solver, combining initialization and computing time, ensuring consistent evaluation conditions. When a model predicts an active set, it is provided as input to the solver. For cold-starting, no such initialization is available, and the problem is solved without prior knowledge. The prediction time refers to the forward pass during testing, and all results are averaged over five independent runs. The solve times for all models scale similarly, approximately proportional to $x^{3/2}$ as



(a) Solve time (b) Prediction time

Figure 4: Comparison of (a) solve time and (b) prediction time for our graph neural network (blue), a standard multilayer perceptron (green), and the cold-started active set method (orange).

shown in Appendix B, with the GNN consistently reducing solve time compared to cold-starting, as further illustrated in the next section. Both models exhibit approximately constant prediction times as the problem size increases, with the MLP achieving predictions nearly an order of magnitude faster than the GNN due to its simpler architecture. This simplicity, however, comes at the cost of generalization. The weight-sharing and structure-aware design of the GNN is precisely what enables it to generalize across problem sizes, as shown in the next section.

4.4. Generalization across variable problem sizes

Although the solve time for the GNN and MLP is comparable, and the MLP achieves a slightly lower iteration count, its fixed input size limits its applicability to varying problem sizes. In contrast, GNNs naturally handle graphs of arbitrary size, making them well-suited for scenarios where the number of variables and constraints varies across instances. This flexibility enables generalization across heterogeneous problem structures, offering clear practical advantages.

To illustrate this, the GNN was trained on graphs of size $[60, 120, 180]$ and tested on graphs of size 300, with a single pair of matrices H and A shared across all problems of the same dimension.

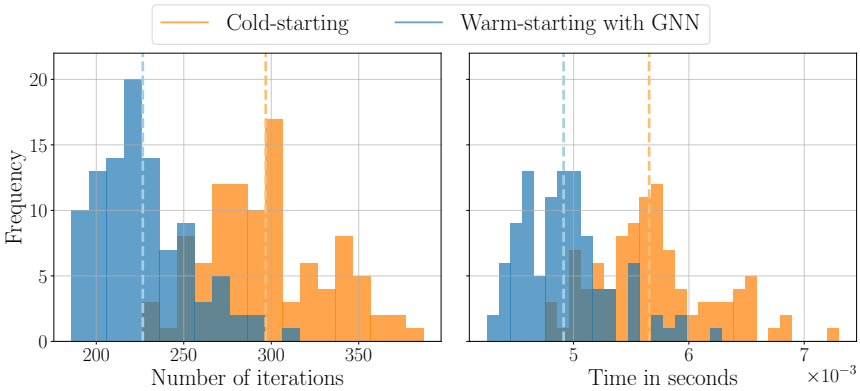


Figure 5: Comparison of iterations (left) and solve time (right) of cold-starting (orange) and warm-starting the `DAQP` solver using the GNN (blue) on problem instances with 100 variables and 200 constraints. The dashed line represents the mean.

As Figure 5 shows, the model still achieves substantial reductions in iterations and solve time, demonstrating that models trained on smaller problem sizes can generalize effectively to larger ones, reducing the need for retraining and providing practical efficiency gains.

4.5. Effect of predictive model on solver efficiency

To evaluate the GNN's impact on the solver performance under realistic conditions, we apply our method to the control problem of an inverted pendulum on a moving cart, generated using the `lmpc` package[2]. The package produces model predictive control (MPC) data for linear systems, resulting in problems of the form

$$\begin{aligned} \underset{u_0,\dots,u_{N_c-1}}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{N_p-1}\left((Cz_k - r)^T Q(Cz_k - r) + u_k^T R u_k + \Delta u_k^T R_r \Delta u_k\right) \\ \text{subject to} \quad & z_{k+1} = Fz_k + Gu_k, \qquad k = 0,\dots,N_p-1 \\ & z_0 = \hat{z} \\ & \underline{b} \le A_z z_k + A_u u_k \le \overline{b}, \qquad k = 0,\dots,N_p-1, \end{aligned} \tag{11}$$

where z_k and u_k are the system state and control action, respectively, at time step k. The current state $\hat{z}$ and a set point r make this a parametric optimization problem. This problem can be condensed (see Chapter 2 in (Arnström, 2023)) to yield QPs of the form (1). Here, the decision variable x contains the control actions $u_0,\dots,u_{N_c-1}$, and the linear term f in the objective function and the right-hand side b of the constraints are affine functions of $\hat{z}$ and r, while H and A remain fixed.

Two datasets with different control horizons are compared, both use a prediction horizon of $N_p = 50$. The first dataset has a control horizon of $N_c = 5$, resulting in 206 constraints, while the second uses $N_c = 50$ with $m = 296$ constraints. Note that the higher number of constraints in the MPC setting, reduces the fraction of active constraints by an order of magnitude compared to the synthetic data. The GNN is trained on the generated data, results are averaged over five independent runs, and a regularization term is added to encourage sparsity in the predicted active sets.

As in previous experiments, we compare our GNN-based warm-starting against a cold-started solver. While conventional warm-starting, is effective when consecutive parameters are similar, this does not always hold in MPC. In reference tracking, for instance, setpoint changes can vastly change parameters relative to the previous problem, causing conventional warm-starting to even perform worse than cold-starting (Herceg et al., 2015). Hence, our method complements the conventional warm-starting MPC solvers, yielding speedups even in scenarios where the latter fails.

Specifically for the inverted pendulum, the control u is a force applied to the cart, and the state is $z = (p, \dot{p}, \phi, \dot{\phi})$, where p is the cart position and ϕ is the pendulum angle. We have the actuation constraint $|u| \le 1$ and the state constraints $|p| \le 10$ and $|\phi| \le \frac{\pi}{4}$. The specific input parameters used in this implementation can be found in the repository[3].

Table 2: Iteration reduction on dataset with $N_c = 50$.

Percentiles	10%	25%	50 %	75%	90%
Iterations cold-start	3	4	13	55.5	94
Iterations warm-start	3	4	19	47.6	78.7

2. https://github.com/darnstrom/lmpc
3. https://github.com/ellaschmidtobreick/acc_daqp

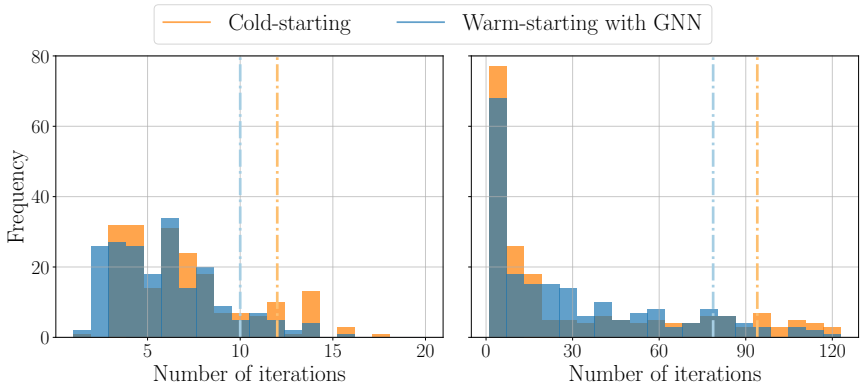


Figure 6: Comparison of iterations of cold-starting (orange) and warm-starting the `DAQP` solver using the GNN (blue) on problem instances with control horizon $N_c = 5$ (left) and $N_c = 50$ (right). The prediction horizon is $N_p = 50$ in both cases and the constraints are 206 and 296 respectively. The dash-dotted line refers to the 90% percentile.

Figure 6 shows the change in solver iterations. Warm-starting the QP solver with the active set predicted by the GNN reduces the number of iterations, with an average reduction of 1.0 iterations on the first dataset and 1.6 iterations on the second dataset. The dotted line in the figure indicates the 90% percentile, showing greater reduction for problems requiring more iterations when cold-started.

This trend is further confirmed in Table 2, which breaks down the iteration reduction across different percentile thresholds for the data set with control horizon $N_c = 50$: the harder the problem for the cold-started solver, the greater the reduction in number of iterations, while problems that converge quickly see minimal benefit. These iteration savings translate directly into reduced solve times (Appendix C), consistent with the findings of earlier experiments.

5. Conclusion

This work presented a learning-to-optimize approach using GNNs to predict the active set in dual active-set solvers for quadratic programs. By representing QPs as bipartite graphs, the model captures the problems' structural properties and can learn effective initial active sets for warm-starting. The experiments demonstrate consistent reductions in solver iterations and solve time, with the largest gains for problems requiring many iterations when cold-started. In contrast to the MLP baseline, our approach naturally handles and scales well with increasing problem size. The GNN exhibits strong size generalization, effectively transferring to larger graphs even when trained only on smaller instances, highlighting its flexibility and practical applicability.

These results indicate that incorporating learned structure-aware predictions can significantly accelerate optimization in sequential or real-time settings. Future work could extend this to other problem classes and further investigate model generalization from smaller to larger problem instances to reduce training cost. Furthermore, a multi-step approach, first predicting the number of active constraints and then identifying which are active, could improve prediction accuracy and enable faster and more adaptive control in real-world systems.

Acknowledgments

This work was funded by the Swedish Research Council, grant number 2024-04130, and supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

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

Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Daniel Arnström. *Real-Time Certified MPC : Reliable Active-Set QP Solvers*. PhD thesis, Linköping University, 2023.

Daniel Arnström, Alberto Bemporad, and Daniel Axehill. A Dual Active-Set Solver for Embedded Quadratic Programming Using Recursive LDL^{t} Updates. *IEEE Transactions on Automatic Control*, 67(8):4362–4369, 2022.

Sebastian Banert, Jevgenija Rudzusika, Ozan Öktem, and Jonas Adler. Accelerated forward-backward optimization using deep learning. *SIAM Journal on Optimization*, 34(2):1236–1263, 2024.

Roscoe A. Bartlett, Lorenz T. Biegler, Johan Backstrom, and Vipin Gopal. Quadratic programming algorithms for large-scale model predictive control. *Journal of Process Control*, 12(7):775–795, 2002.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv:2104.13478*, 2021.

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial Optimization and Reasoning with Graph Neural Networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

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

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

Ziang Chen, Xiaohan Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. Expressive power of graph neural networks for (mixed-integer) quadratic programs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *ICML'25*, pages 7880–7911. JMLR.org, 2025.

Henri Doerks, Paul Häusner, Daniel Hernández Escobar, and Jens Sjölund. Learning to accelerate distributed ADMM using graph neural networks. *arXiv:2509.05288*, 2025.

Xi Gao, Jinxin Xiong, Akang Wang, Qihong Duan, Jiang Xue, and Qingjiang Shi. IPM-LSTM: A learning-based interior point method for solving nonlinear programs. *Advances in Neural Information Processing Systems*, 37:122891–122916, 2024.

Jacek Gondzio and Andreas Grothey. Parallel interior-point solver for structured quadratic programs: Application to financial planning problems. *Ann Oper Res*, 152(1):319–339, 2007.

Martin Herceg, CN Jones, and M Morari. Dominant speed factors of active set methods for fast MPC. *Optimal Control Applications and Methods*, 36(5):608–627, 2015.

Paul Häusner, Ozan Öktem, and Jens Sjölund. Neural incomplete factorization: learning preconditioners for the conjugate gradient method. *Transactions on Machine Learning Research*, 2024.

Paul Häusner, Aleix Nieto Juscafresa, and Jens Sjölund. Learning incomplete factorization preconditioners for GMRES. In *Proceedings of the 6th Northern Lights Deep Learning Conference (NLDL)*, pages 85–99. PMLR, 2025.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating Quadratic Optimization with Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 21043–21055. Curran Associates, Inc., 2021.

Martin Klaučo, Martin Kalúz, and Michal Kvasnica. Machine learning-based warm starting of active set methods in embedded model predictive control. *Engineering Applications of Artificial Intelligence*, 77:1–8, 2019.

Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2589–2594, 2014.

Gautam Mitra, Frank Ellison, and Alan Scowcroft. Quadratic programming for portfolio planning: Insights into algorithmic and computational issues. *J Asset Manag*, 8(3):200–214, 2007.

Jorge Nocedal and Stephen J. Wright, editors. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, 1999.

Pavel Otta, Ondrej Santin, and Vladimir Havlena. Measured-state driven warm-start strategy for linear MPC. In *2015 European Control Conference (ECC)*, pages 3132–3136, 2015.

Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5470–5477, 2020.

Rajiv Sambharya and Bartolomeo Stellato. Learning algorithm hyperparameters for fast parametric convex optimization. *arXiv:2411.15717*, 2024.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-End Learning to Warm-Start for Real-Time Quadratic Optimization. In *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, pages 220–234. PMLR, 2023.

Augustinos D Saravanos, Hunter Kuperman, Alex Oshin, Arshiya Taj Abdul, Vincent Pacelli, and Evangelos Theodorou. Deep distributed optimization for large-scale quadratic programming. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jens Sjölund and Maria Bånkestad. Graph-based Neural Acceleration for Nonnegative Matrix Factorization. *arXiv:2202.00264*, 2022.

Vrushabh Zinage, Ahmed Khalil, and Efstathios Bakolas. Transformermpc: Accelerating model predictive control via transformers. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9221–9227, 2025.

Appendix A. Algorithm of the DAQP solver

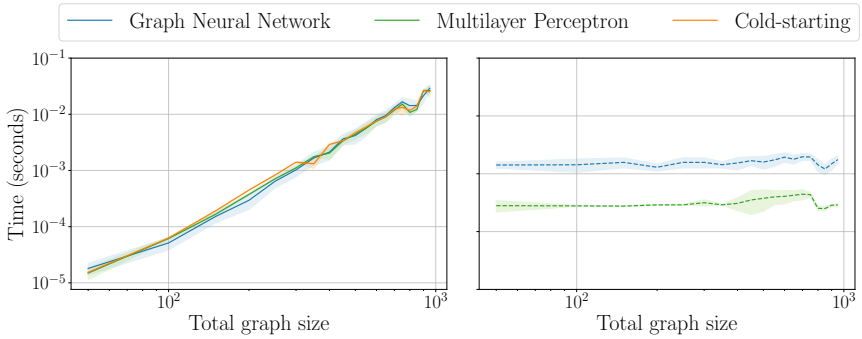
Algorithm 1 Dual active-set method for solving a dual QP (Arnström et al., 2022)

Require: $M, d, v, R^{-1}, \mathcal{W}_0, \lambda_0$
Ensure: $x^*, \lambda^*, \mathcal{A}^*$

while true **do**
  if $M_\mathcal{W} M_\mathcal{W}^T$ is nonsingular **then**
    Solve $M_\mathcal{W} M_\mathcal{W}^T \lambda^*_\mathcal{W} = -d_\mathcal{W}$
    if $\lambda^* \geq 0$ **then**
      $\mu_{\overline{\mathcal{W}}} \leftarrow M_{\overline{\mathcal{W}}} M_\mathcal{W}^T \lambda^*_\mathcal{W} + d_{\overline{\mathcal{W}}}$
      $\lambda \leftarrow \lambda^*$ // λ^* dual feasible
      if $\mu \geq 0$ **then**
        break // x^* primal feasible
      else
        $j \leftarrow \arg\min_{i \in \overline{\mathcal{W}}} [\mu]_i$ // x^* not primal feasible
        $\mathcal{W} \leftarrow \mathcal{W} \cup \{j\}$
      end if
    else
      $p \leftarrow \lambda^* - \lambda$ // λ^* not dual feasible
      $\mathcal{B} \leftarrow \{i \in \mathcal{W} \mid [\lambda^*]_i < 0\}$
      $(\lambda, \mathcal{W}) \leftarrow$ FIXCOMPONENT$(\lambda, \mathcal{W}, \mathcal{B}, p)$
    end if
  else
    Solve $M_\mathcal{W} M_\mathcal{W}^T p_\mathcal{W} = 0$
    $\mathcal{B} \leftarrow \{i \in \mathcal{W} \mid [p]_i < 0\}$
    $(\lambda, \mathcal{W}) \leftarrow$ FIXCOMPONENT$(\lambda, \mathcal{W}, \mathcal{B}, p)$
  end if
end while
$x^* \leftarrow -R^{-1}(M_\mathcal{W}^T \lambda^*_\mathcal{W} + v)$
return $(x^*, \lambda^*, \mathcal{W})$

procedure FIXCOMPONENT$(\lambda, \mathcal{W}, \mathcal{B}, p)$
  $j \leftarrow \arg\min_{i \in \mathcal{B}} \left(-\frac{[\lambda]_i}{[p]_i} \right)$
  $\mathcal{W} \leftarrow \mathcal{W} \setminus \{j\}$
  $\lambda \leftarrow \lambda - \frac{[\lambda]_j}{[p]_j} p$
return $(\lambda, \mathcal{W})$

Appendix B. Scaling results on log-log axes



(a) Solve time (b) Prediction time

Figure 7: Comparison of solve time (a) and prediction time (b) for our graph neural network (blue), a standard multilayer perceptron (green), and the cold-started active set method (orange), shown on logarithmic axes. For the largest graph sizes, the solve time with GNN warm-starting is about five times shorter than the other two, with a negligible prediction time.

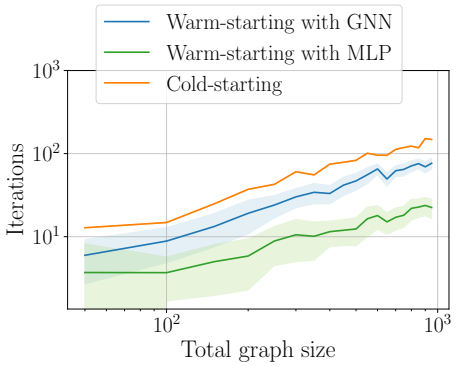


Figure 8: Comparison of iterations when warm-starting the `DAQP` solver with predictions from our graph neural network, a standard multilayer perceptron and cold-starting the solver without any learned prediction, shown on logarithmic axes.

Appendix C. MPC solve time results

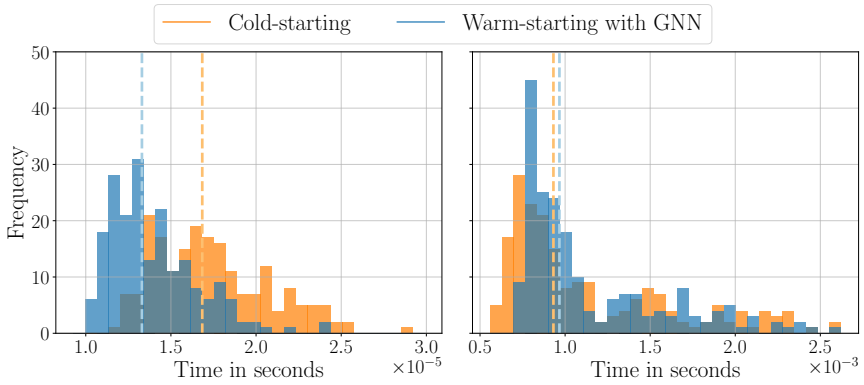


Figure 9: Comparison of solve time of cold-starting (orange) and warm-starting the DAQP solver using the GNN (blue) on problem instances with control horizon 5 (left) and 50 (right). The prediction horizon is 50 in both cases and the constraints are 206 and 296 respectively. The dash-dotted line refers to the 90% percentile.