LEARNING CONTEXT-AWARE ADAPTIVE SOLVERS TO ACCELERATE QUADRATIC PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Quadratic programming (QP) is an important sub-field of mathematical optimization. The alternating direction method of multipliers (ADMM) is a successful method to solve QP. Even though ADMM shows promising results in solving various types of QP, its convergence speed is known to be highly dependent on the step-size parameter ρ. Due to the absence of a general rule for setting ρ, it is often tuned manually or heuristically. In this paper, we propose CA-ADMM (Context-aware Adaptive ADMM)) which learns to adaptively adjust ρ to accelerate ADMM. CA-ADMM extracts the spatio-temporal context, which captures the dependency of the primal and dual variables of QP and their temporal evolution during the ADMM iterations. CA-ADMM chooses ρ based on the extracted context. Through extensive numerical experiments, we validated that CA-ADMM effectively generalizes to unseen QP problems with different sizes and classes (i.e., having different QP parameter structures). Furthermore, we verified that CA-ADMM could dynamically adjust ρ considering the stage of the optimization process to accelerate the convergence speed further.

1 INTRODUCTION

Among the optimization classes, quadratic program (QP) is widely used due to its mathematical tractability, e.g. convexity, in various fields such as portfolio optimization (Boyd et al., 2017; Cornuéjols et al., 2018; Boyd et al., 2014; Markowitz, 1952), machine learning (Kecman et al., 2001; Sha et al., 2002), control, (Buijs et al., 2002; Krupa et al., 2022; Bartlett et al., 2002), and communication applications (Luo & Yu, 2006; Hons, 2001). As the necessity to solve large optimization problems increases, it is becoming increasingly important to ensure the scalability of QP for achieving the solution of the large-sized problem accurately and quickly.

Among solution methods to QP, first-order methods (Frank & Wolfe, 1956) owe their popularity due to their superiority in efficiency over other solution methods, for example active set (Wolfe, 1959) and interior points methods (Nesterov & Nemirovskii, 1994). The alternating direction method of multipliers (ADMM) (Gabay & Mercier, 1976; Mathematique et al., 2004) is commonly used for returning high solution quality with relatively small computational expense (Stellato et al., 2020b). Even though ADMM shows satisfactory results in various applications, its convergence speed is highly dependent on both parameters of QP and user-given step-size ρ. In an attempt to resolve these issues, numerous studies have proposed heuristic (Boyd et al., 2011; He et al., 2000; Stellato et al., 2020a) or theory-driven (Ghadimi et al., 2014)) methods for deciding ρ. But a strategy for selecting the best performing ρ still needs to be found (Stellato et al., 2020b). Usually ρ is tuned in a case-dependent manner (Boyd et al., 2011; Stellato et al., 2020a; Ghadimi et al., 2014).

Instead of relying on hand-tuning ρ, a recent study (Ichnowski et al., 2021) utilizes reinforcement learning (RL) to learn a policy that adaptively adjusts ρ to accelerate the convergence of ADMM. They model the iterative procedure of ADMM as the Markov decision process (MDP) and apply the generic RL method to train the policy that maps the current ADMM solution states to a scalar value of ρ. This approach shows relative effectiveness over the heuristic method (e.g., OSQP (Stellato et al., 2020a)), but it has several limitations. It uses a scalar value of ρ that cannot adjust ρ differently depending on individual constraints. And, it only considers the current state without its history to determine ρ, and therefor cannot capture the non-stationary aspects of ADMM iterations. This method inspired us to model the iteration of ADMM as a non-stationary networked system. We

developed a more flexible and effective policy for adjusting ρ for all constraints simultaneously and considering the evolution of ADMM states.

In this study, we propose Context-aware Adaptive ADMM (CA-ADMM), an RL-based adaptive ADMM algorithm, to increase the convergence speed of ADMM. To overcome the mentioned limitations of other approaches, we model the iterative solution-finding process of ADMM as the MDP whose context is determined by QP structure (or parameters). We then utilize a graph recurrent neural network (GRNN) to extract (1) the relationship among the primal and dual variables of the QP problem, i.e., its spatial context and (2) the temporal evolutions of the primal and dual variables, i.e., its temporal context. The policy network then utilizes the extracted spatio-temporal context to adjust ρ.

From the extensive numerical experiments, we verified that CA-ADMM adaptively adjusts ρ in consideration of QP structures and the iterative stage of the ADMM to accelerate the convergence speed further. We evaluated CA-ADMM in various QP benchmark datasets and found it to be significantly more efficient than the heuristic and learning-based baselines in number of iterations until convergence. CA-ADMM shows remarkable generalization to the change of problem sizes and, more importantly, benchmark datasets. Through the ablation studies, we also confirmed that both spatial and temporal context extraction schemes are crucial to learning a generalizable ρ policy. The contributions of the proposed method are summarized below:

- **Spatial relationships:** We propose a heterogeneous graph representation of QP and ADMM state that captures spatial context and verifies its effect on the generalization of the learned policy.
- **Temporal relationships:** We propose to use a temporal context extraction scheme that captures the relationship of ADMM states over the iteration and verifies its effect on the generalization of the learned policy.
- **Performance/Generalization:** CA-ADMM outperforms state-of-the-art heuristics (i.e., OSQP) and learning-based baselines on the training QP problems and, more importantly, out-of-training QP problems, which include large size problems from a variety of domains.

2 Related Works

Methods for selecting ρ of ADMM. In the ADMM algorithm, step-size ρ plays a vital role in determining the convergence speed and accuracy. For some special cases of QP, there is a method to compute optimal ρ (Ghadimi et al., 2014). However, this method requires linear independence of the constraints, e.g., nullity of $\boldsymbol{A}$ is nonzero, which is difficult to apply in general QP problems. Thus, various heuristics have been proposed to choose ρ (Boyd et al., 2011; He et al., 2000; Stellato et al., 2020a). Typically, the adaptive methods that utilize state-dependent step-size ρ_t show a relatively faster convergence speed than non-adaptive methods. He et al. (2000); Boyd et al. (2011) suggest a rule for adjusting ρ_t depending on the ratio of residuals. OSQP (Stellato et al., 2020a) extends the heuristic rule by adjusting ρ with the values of the primal and dual optimization variables. Even though OSQP shows improved performance, designing such adaptive rules requires tremendous effort. Furthermore, the designed rule for a specific QP problem class is hard to generalize to different QP classes having different sizes, objectives and constraints. Recently, Ichnowski et al. (2021) employed RL to learn a policy for adaptively adjusting ρ depending on the states of ADMM iterations. This method outperforms other baselines, showing the potential that an effective rule for adjusting ρ can be learned without problem-specific knowledge using data. However, this method still does not sufficiently reflect the structural characteristics of the QP and the temporal evolution of ADMM iterations. Both limitations make capturing the proper problem context challenging, limiting its generalization capability to unseen problems of different sizes and with alternate objectives and constraints.

Graph neural network for optimization problems. An optimization problem comprises objective function, decision variables, and constraints. When the optimization variable is a vector, there is typically interaction among components in the decision vector with respect to an objective or constraints. Thus, to capture such interactions, many studies have proposed to use graph representation to model such interactions in optimization problems. Gasse et al. (2019) expresses mixed integer programming (MIP) using a bipartite graph consisting of two node types, decision variable nodes,

and constraint nodes. They express the relationships among variables and the relationships between decision variables and constraints using different kinds of edges whose edge values are associated with the optimization coefficients. Ding et al. (2019) extends the bipartite graph with a new node type, 'objective'. The objective node represents the linear objective term and is connected to the variable node, with the edge features and coefficient value of the variable component in the objective term. They used the tripartite graph as an input of GCN to predict the solution of MIP. Boyd et al. (2011) expresses the distributed optimization as a graph where the dependency between decision variables is represented as a graph edge. The algorithm applies the consensus updating rule only for the decision variables connected. Dai et al. (2016) expresses probabilistic inference problem, a type of optimization, as a probabilistic graph model where the dependency of random and observed variables are represented as edges between graph nodes. Then, this method applies graph neural network to infer the sufficient statistics of random variables given observation nodes. Ryu et al. (2020) expresses the cooperative game as a graph where each node represents an agent's action, and the relationships among agents (i.e., distance or affinity) are represented as edges. Then, they apply Graph Neural Network to extract each agent's essential information and map it to individual action. In this study, we employ graph representation to express the spatial structure of a QP instance.

ML accelerated scientific computing. ML models are employed to predict the results of complex computations, e.g., the solution of ODE and PDE, and the solution of optimization problems, and trained by supervised learning. As these methods tend to be sample inefficient, other approaches learn operators that expedite computation speeds for ODE (Poli et al., 2020; Berto et al., 2021), fixed point iterations (Bai et al., 2021; Park et al., 2021b), and matrix decomposition (Donon et al., 2020). Poli et al. (2020) uses ML to predict the residuals between the accurate and inaccurate ODE solvers and use the predicted residual to expedite ODE solving. Bai et al. (2021); Park et al. (2021b) trains a network to learn alternate fixed point iterators, trained to give the same solution while requiring less iterations to converge. However, the training objective all the listed methods is minimizing the network prediction with the ground truth solution, meaning that those methods are at risk to find invalid solutions. CA-ADMM learns to provide the proper ρ that minimize ADMM iterations, rather than directly predicting the solution of QP. This approach, i.e., choose ρ for ADMM, still guarantees that the found solution remains equal to the original one.

3 Background

Quadratic programming (QP) A quadratic program (QP) associated with N variables and M constraints is an optimization problem having the form of the following:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^\top \boldsymbol{P} x + \boldsymbol{q}^\top x \\ \text{subject to} \quad & \boldsymbol{l} \leq \boldsymbol{A} x \leq \boldsymbol{u}, \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^N$ is the decision variable, $\boldsymbol{P} \in \mathbb{S}_+^N$ is the positive semi-definite cost matrix, $\boldsymbol{q} \in \mathbb{R}^N$ is the linear cost term, $\boldsymbol{A} \in \mathbb{R}^{M \times N}$ is a $M \times N$ matrix that describes the constraints, and $\boldsymbol{l}$ and $\boldsymbol{u}$ are their lower and upper bounds. QP generally has no closed-form solution except some special cases, for example $\boldsymbol{A} = \boldsymbol{0}$. Thus, QP is generally solved via some iterative methods.

First-order QP solver Out of the iterative methods to solve QP, the alternating direction method of multipliers (ADMM) has attracted considerable interest due to its simplicity and suitability for various large-scale problems including statistics, machine learning, and control applications. (Boyd et al., 2011). ADMM solves a given QP($\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{l}, \boldsymbol{A}, \boldsymbol{u}$) through the following iterative scheme. At each step t, ADMM solves the following system of equations:

$$\begin{bmatrix} P + \sigma I & A^T \\ A & \text{diag}(\rho)^{-1} \end{bmatrix} \begin{bmatrix} x_{t+1} \\ \nu_{t+1} \end{bmatrix} = \begin{bmatrix} \sigma x_t - q \\ z_t - \text{diag}(\rho)^{-1} y_t \end{bmatrix}, \tag{2}$$

where $\sigma > 0$ is the regularization parameter that assures the unique solution of the linear system, $\rho \in \mathbb{R}^M$ is the step-size parameter, and $\text{diag}(\rho) \in \mathbb{R}^{M \times M}$ is a diagonal matrix with elements ρ. By solving Eq. (2), ADMM finds x_{t+1} and ν_{t+1}. Then, it updates y_t and z_t with the following

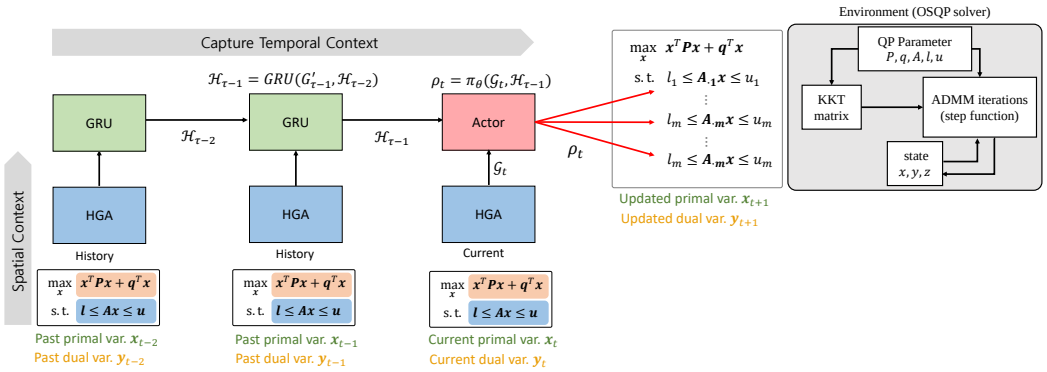

Figure 1: **An overview of CA-ADMM**. CA-ADMM uses the spatio-temporal context, which is extracted by GRNN from the ADMM state observations, to compute ρ for ADMM algorithm.

equations:

$$\tilde{z}_{t+1} \leftarrow z_t + \text{diag}(\rho)^{-1}(\nu_{t+1} - y_t) \tag{3}$$

$$z_{t+1} \leftarrow \Pi(\tilde{z}_t + \text{diag}(\rho)^{-1} y_t) \tag{4}$$

$$y_{t+1} \leftarrow x_t + \text{diag}(\rho)(\tilde{z}_{t+1} - z_{t+1}), \tag{5}$$

where Π is the projection onto the hyper box $[\boldsymbol{l}, \boldsymbol{u}]$. ADMM proceeds the iteration until the primal $r_t^{\text{primal}} = \boldsymbol{A}x_t - z_t \in \mathbb{R}^M$ and dual residuals $r_t^{\text{dual}} = \boldsymbol{P}x_t + \boldsymbol{q} + \boldsymbol{A}^\top y_t \in \mathbb{R}^N$ are sufficiently small. For instance, the termination criteria are as follows:

$$||r_t^{\text{primal}}||_\infty \leq \epsilon^{\text{primal}}, ||r_t^{\text{dual}}||_\infty \leq \epsilon^{\text{dual}}, \tag{6}$$

where ϵ^{primal} and ϵ^{dual} are sufficiently small positive numbers.

4 METHOD

In this section, we present Context-aware Adaptive ADMM (CA-ADMM) that automatically adjusts ρ parameters considering the spatial and temporal contexts of the ADMM to accelerate the convergence of ADMM. We first introduce the contextual MDP formulation of ADMM iterations for solving QP problems and discuss two mechanisms devised to extract the spatial and temporal context from the ADMM iterations.

4.1 CONTEXTUAL MDP FORMULATION

As shown in Eqs. (2) to (5), ADMM solves QP by iteratively updating the intermediate variables x_t,y_t, and z_t. Also, the updating rule is parameterized with QP variables $\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{l}, \boldsymbol{A}, \boldsymbol{u}$.

We aim to learn a QP solver that can solve general QP instances whose structures are different from QPs used for training the solver. As shown in Eqs. (2) to (5), ADMM solves QP by iteratively updating the intermediate variables x_t, y_t, and z_t. Also, the updating rule is parameterized with QP variables $\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{l}, \boldsymbol{A}, \boldsymbol{u}$. In this perspective, we can consider the iteration of ADMM for each QP as a different MDP whose dynamics can be characterized by the structure of QP and the ADMM's updating rule. Based on this perspective, our objective, learning a QP solver that can solve general QP instances, is equivalent to learning an adaptive policy (ADMM operator) that can solve a contextual MDP (QP instance with different parameters).

Assuming we have a QP($\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{A}, \boldsymbol{l}, \boldsymbol{u}$), where $\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{A}, \boldsymbol{l}, \boldsymbol{u}$ are the parameters of the QP problem, and ADMM(ϕ), where ϕ is the parameters of the ADMM algorithm. In our study, ϕ is ρ, but it can be any configurable parameter that affects the solution updating procedure. We then define the contextual MDP $\mathcal{M} = (\mathbb{X}, \mathbb{U}, T_\rho^c, R, \gamma)$, where $\mathbb{X} = \{(x_t, y_t, z_t)\}_{t=1,2,\ldots}$ is the set of states, $\mathbb{U} = \{\rho_t\}_{t=1,2,\ldots}$ is the set of actions, $T_\rho^c(x, x') = P(x_{t+1} = x' | x_t = x, \rho_t = \rho, c)$ is the transition function whose behaviour is contextualized with the context $c = \{\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{A}, \boldsymbol{l}, \boldsymbol{u}, \phi\}$, R is the reward

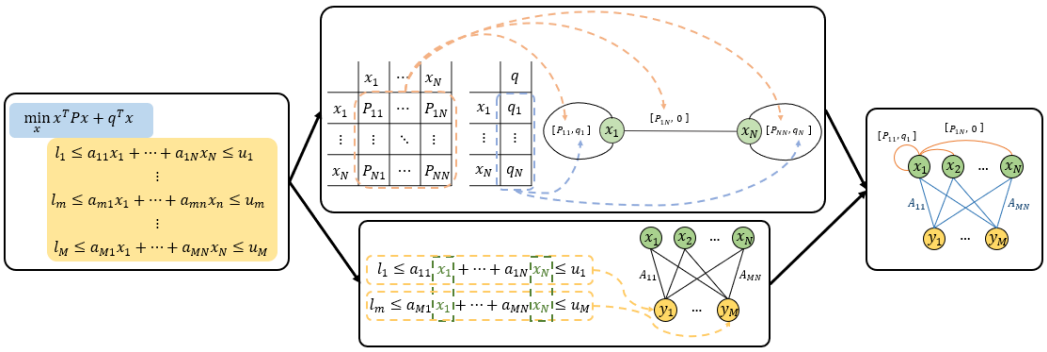

Figure 2: **Heterograph representation of QP**.

function that returns 0 when x_t is the terminal state (i.e., the QP problem is solved) and -1 otherwise, and $\gamma \in [0, 1)$ is a discount factor. We set an MDP transition made at every 10 ADMM steps to balance computational cost and acceleration performance. That is, we observe MDP states at every 10 ADMM steps and change ρ_t; thus, during the ten steps, the same actions ρ_t are used for ADMM. As shown in the definition of T_ρ^c, the context c alters the solution updating procedure of ADMM (i.e., $T_\rho^c \neq T_\rho^{c'}$ to the given state x). Therefore, it is crucial to effectively process the context information c in deriving the policy $\pi_\theta(x_t)$ to accelerate the ADMM iteration.

One possible approach to accommodate c is manually designing a feature that serves as a (minimal) sufficient statistic that separates T_ρ^c from the other $T_\rho^{c'}$. However, designing such a feature can involve enormous efforts; thus, it derogates the applicability of learned solvers to the new solver classes. Alternatively, we propose to learn a context encoder that extracts the more useful context information from not only the structure of a target QP but also the solution updating history of the ADMM so that we can use context-dependent adaptive policy for adjusting ρ. To successfully extract the context information from the states, we consider the relationships among the primal and dual variables (i.e., the spatial relationships) and the evolution of primal/dual variables with ADMM iteration (i.e., temporal relationship). To consider such spatio-temporal correlations, we propose an encoder network that is parameterized via a GNN followed by RNN. Fig. 1 visualizes the overall network architecture.

4.2 Extracting spatial context information via heterogeneous graph neural network

Heterogeneous graph representation of QP. The primal x_t and dual y_t variables of QP are related via the QP parameters $(\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{A}, \boldsymbol{l}, \boldsymbol{u})$. As shown in Fig. 2, their relationship can be modeled as a heterogeneous graph where the nodes represent the primal and dual variables, and the edges represent the relationship among these variables. As the roles of primal and dual variables in solving QP are distinctive, it necessitates the graph representation that reflects the different variable type information. Regarding that, we represent the relationships of the variables at t-th MDP step with a directed heterogeneous graph $\mathcal{G}_t = (\mathbb{V}_t, \mathbb{E}_t)$. As the graph construction is only associated with the current t, we omit the step-index t for notational brevity.

The node set $\mathbb{V}$ consist of the primal $\mathbb{V}_{\text{primal}}$ and dual node sets $\mathbb{V}_{\text{dual}}$. The n-th primal and m-th dual nodes have node feature s_n^{primal} and s_m^{dual} defined as:

$$s_n^{\text{primal}} = [\underbrace{\log_{10} r_n^{\text{dual}}, \log_{10} ||r^{\text{dual}}||_\infty, \mathbb{1}_{r_n^{\text{dual}} < \epsilon^{\text{dual}}}}_{\text{Encoding ADMM state}}], \tag{7}$$

$$s_m^{\text{dual}} = [\underbrace{\log_{10} r_m^{\text{primal}}, \log_{10} ||r^{\text{primal}}||_\infty, \mathbb{1}_{r_m^{\text{primal}} < \epsilon^{\text{primal}}}, y_m, \rho_m, \min(z_m - l_m, u_m - z_m)}_{\text{Encoding ADMM state}} \underbrace{\mathbb{1}_{\text{equality}}, \mathbb{1}_{\text{inequality}}}_{\text{Encoding QP problem}}], \tag{8}$$

where r_m^{primal} and r_n^{dual} denotes the m-th and n-th element of primal and dual residuals, respectively, $||\cdot||_\infty$ denotes the infinity norm, $\mathbb{1}_{r_m^{\text{primal}} < \epsilon^{\text{primal}}}$ and $\mathbb{1}_{r_n^{\text{dual}} < \epsilon^{\text{dual}}}$ are the indicators whether the m-th primal and n-th dual residual is smaller than ϵ^{primal} and ϵ^{dual}, respectively. $\mathbb{1}_{\text{equality}}$ and $\mathbb{1}_{\text{inequality}}$ are the indicators whether the m-th constraint is equality and inequality, respectively.

The primal and dual node features are design to capture ADMM state information and the QP structure. However, those node features do not contain information of $\boldsymbol{P}$, $\boldsymbol{q}$, or $\boldsymbol{A}$. We encode such QP problem structure in the edges. The edge set $\mathbb{E}$ consist of

- The primal-to-primal edge set $\mathbb{E}^{\text{p2p}}$ is defined as $\{e_{ij}^{\text{p2p}} | \boldsymbol{P}_{ij} \neq 0 \, \forall (i,j) \in [\![1,N]\!] \times [\![1,N]\!]\}$ where $[\![1,N]\!] = \{1,2,\ldots,N\}$. i.e., the edge from the i th primal node to j th primal node exists when the corresponding $\boldsymbol{P}$ is not zero. The edge e_{ij}^{p2p} has the feature s_{ij}^{p2p} as $[\boldsymbol{P}_{ij}, \boldsymbol{q}_i]$.
- The primal-to-dual edge set $\mathbb{E}^{\text{p2d}}$ is defined as $\{e_{ij} | \boldsymbol{A}_{ji} \neq 0 \, \forall (i,j) \in [\![1,N]\!] \times [\![1,M]\!]\}$. The edge e_{ij}^{p2d} has the feature s_{ij}^{p2d} as $[\boldsymbol{A}_{ji}]$.
- The dual-to-primal edge set $\mathbb{E}^{\text{d2p}}$ is defined as $\{e_{ij} | \boldsymbol{A}_{ij} \neq 0 \, \forall (i,j) \in [\![1,M]\!] \times [\![1,N]\!]\}$. The edge e_{ij}^{d2p} has the feature s_{ij}^{d2p} as $[\boldsymbol{A}_{ij}]$.

Graph preprocessing for unique QP representation. QP problems have the same optimal decision variables for scale transformation of the objective function and constraints. However, such transformations alter the QP graph representation. To make graph representation invariant to the scale transform, we preprocess QP problems by dividing the elements of $\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{a}, \boldsymbol{A}, \boldsymbol{u}$ with problem-dependent constants. The details of the preprocessing steps are given in Appendix A.

Grpah embedding with Hetero GNN. The heterogeneous graph $\mathcal{G}$ models the various type of information among the primal and dual variables. To take into account such type information for the node and edge embedding, we propose heterogeneous graph attention (HGA), a variant of type-aware graph attention (Park et al., 2021a), to embed $\mathcal{G}$. HGA employs separate node and edge encoders for each node and edge type. For each edge type, it computes edge embeddings, applies the attention mechanism to compute the weight factors, and then aggregates the resulting edge embeddings via weighted summation. Then, HGA sums the different-typed aggregated edge embeddings to form a type-independent edge embedding. Lastly, HGA updates node embeddings by using the type-independent edge embeddings as an input to node encoders. We provide more details of HGA in Appendix B.

4.3 Extracting temporal context via RNN

The previous section discusses extracting context by considering the spatial correlation among the variables within the MDP step. We then discuss extracting the temporal correlations. As verified from numerous optimization literature, deciding ρ based on the multi-stage information (e.g., momentum methods) helps increase the stability and solution quality of solvers. Inspired by that, we consider the time evolution of the ADMM states to extract temporal context. To accomplish that we use an RNN to extract the temporal context from the ADMM state observations. At the tth MDP step, we consider l historical state observations to extract the temporal context. Specifically, we first apply HGA to $\mathcal{G}_{t-l}, \ldots, \mathcal{G}_{t-1}$ and then apply GRU (Chung et al., 2014) as follows:

$$\mathcal{G}'_{t-l+\delta} = \text{HGA}(\mathcal{G}_{t-l+\delta}) \qquad \text{for } \delta = 0, 1, 2, \cdots, l-1 \tag{9}$$

$$\mathcal{H}_{t-l+\delta} = \text{GRU}(\mathbb{V}'_{t-l+\delta}, \mathcal{H}_{t-l+\delta-1}) \qquad \text{for } \delta = 1, 2, \cdots, l-1 \tag{10}$$

where $\mathcal{G}'_{t-l+\delta}$ is updated graph at $t-l+\delta$, $\mathbb{V}'_{t-l+\delta}$ is the set of updated nodes of $\mathcal{G}'_{t-l+\delta}$, and $\mathcal{H}_{t-l+\delta}$ are the hidden embeddings. We define the node-wise context $\mathcal{C}_t$ at time t as the last hidden embedding $\mathcal{H}_{t-1}$.

4.4 Adjusting ρ_t using extracted context

We parameterize the policy $\pi_\theta(\mathcal{G}_t, \mathcal{C}_t)$ via the HGA followed by an MLP. Since action ρ_t is not defined at t, we exclude ρ_t from the node features of $\mathcal{G}_t$ and instead node-wise concatenate $\mathcal{C}_t$

Table 1: **In-domain results (in # iterations)**. Smaller is better (↓). Best in **bold**. We measure the average and standard deviation of ADMM iterations for each QP benchmark. All QP problems are generated to have $10 \leq N \leq 15$.

	Random QP	EqQP	Porfolio	SVM	Huber	Control	Lasso
OSQP	160.99 ± 92.21	25.52 ± 0.73	185.18 ± 28.73	183.79 ± 40.88	54.38 ± 5.37	30.38 ± 24.98	106.17 ± 45.70
RLQP	64.66 ± 38.78	20.27 ± 0.49	28.45 ± 5.52	34.14 ± 3.84	25.16 ± 0.66	26.78 ± 15.84	25.61 ± 0.68
CA-ADMM	**24.22** ± 4.46	**19.61** ± 0.61	**20.12** ± 0.47	**25.05** ± 2.30	**20.32** ± 0.69	**17.52** ± 3.88	**19.08** ± 1.01

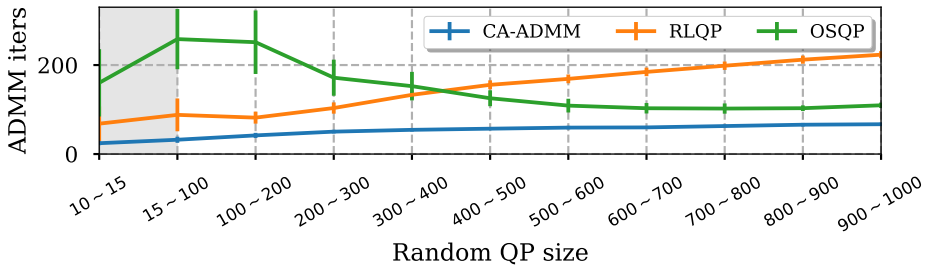

Figure 3: **QP size generalization results**. Smaller is better (↓). We plot the average and standard deviation of the iterations as the bold lines and error bars, respectively. The gray area (▢) indicates the training QP sizes.

to the node features. For the action-value network $Q_\phi(\mathcal{G}_t, \rho_t, \mathcal{C}_t)$, we use $\mathcal{G}_t$, ρ_t, and $\mathcal{C}_t$ as inputs. $Q_\phi(\mathcal{G}_t, \rho_t, \mathcal{C}_t)$ has a similar network architecture as $\pi_\theta(\mathcal{G}_t, \mathcal{C}_t)$. We train $\pi_\theta(\mathcal{G}_t, \mathcal{C}_t)$ and $Q_\phi(\mathcal{G}_t, \rho_t, \mathcal{C}_t)$ on the QP problems of size 10∼15 (i.e., $N \in [10, 15]$) with DDPG (Lillicrap et al., 2015) algorithm. Appendix C details the used network architecture and training procedure further.

Comparison to RLQP. Our closest baseline RLQP (Ichnowski et al., 2021) is also trained using RL. At step t, it utilizes the state $x = (\{s_m^{\text{dual}}\}_{m=1,\ldots,M})$, with

$$\begin{aligned} s_m^{\text{dual}} = [&\log_{10} ||r^{\text{primal}}||_\infty, \log_{10} ||r^{\text{dual}}||_\infty], y_m, \rho_m, \\ &\min(z_m - l_m, u_m - z_m), z_m - (Ax)_m]. \end{aligned} \tag{11}$$

It then uses an shared MLP $\pi_\theta(s_n^{\text{dual}})$ to compute ρ_n. We observe that the state representation and policy network are insufficient to capture the contextual information for differentiating MDPs. The state representation does not include information of $\boldsymbol{P}$ and $\boldsymbol{q}$, and the policy network does not explicitly consider the relations among the dual variables. However, as shown in Eq. (2), ADMM iteration results different (x_t, ν_t) with changes in $\boldsymbol{P}$, $\boldsymbol{q}$ and $\boldsymbol{A}$, meaning that T_ρ^c also changes. Therefore, such state representations and network architectures may result in an inability to capture contextual changes among QP problems. Additionally, using single state observation as the input for π_θ may also hinder the possibility of capturing temporal context.

5 Experiments

We evaluate CA-ADMM in various canonical QP problems such as Random QP, EqQP, Porfolio, SVM, Huber, Control, and Lasso. We use RLQP, another learned ADMM solver (Ichnowski et al., 2021), and OSQP, a heuristic-based solver (Stellato et al., 2020a), as baselines. Training and testing problems are generated according to Appendix D.

In-domain results. It is often the case that a similar form of optimization is repeatedly solved with only parameter changes. So, we train the solver using a particular class of QP instances and apply the trained solver to solve the same class of QP problems but with different parameters. We use the number of ADMM iterations until it's convergence as a performance metric for quantifying the solver speed. Table 1 compares the averages and standard deviations of the ADMM iterations required by different baseline models in solving the QP benchmark problem sets. As shown in Table 1, CA-ADMM exhibits 2 ∼ 8x acceleration compared to OSQP, the one of the best heuristic

Table 2: **Cross-domain results. zero-shot transfer (`Random QP + EqQP`) $\rightarrow$ `X`, where `X` $\in$ {`Random QP`, `EqQP`, `Porfolio`, `SVM`, `Huber`, `Control`, `Lasso`}**: Smaller is better (↓). Best in **bold**. We measure the average and standard deviation of ADMM iterations for each QP benchmark. The gray colored cell (▒) denotes the training problems. Green colored values are the case where the transferred model outperforms the in-domain model. All QP problems are generated to have $40 \leq N + M \leq 70$.

	`Random QP`	`EqQP`	`Porfolio`	`SVM`	`Huber`	`Control`	`Lasso`
`OSQP`	160.99 ± 92.21	25.52 ± 0.73	185.18 ± 28.73	183.79 ± 40.88	54.38 ± 5.37	30.38 ± 24.98	106.17 ± 45.70
`RLQP` (transferred)	65.28 ± 57.75	28.48 ± 0.70	31.80 ± 7.97	37.96 ± 10.16	28.70 ± 0.67	74.60 ± 49.71	29.18 ± 29.18
`RLQP` (in-domain)	64.66 ± 38.78	20.27 ± 0.49	28.45 ± 5.52	34.14 ± 3.84	25.16 ± 0.66	26.78 ± 15.84	25.61 ± 0.68
`CA-ADMM` (transferred)	**28.52** ± 7.13	**19.38** ± 0.64	**25.38** ± 7.03	**31.26** ± 13.44	**19.56** ± 0.92	**27.16** ± 4.97	**23.84** ± 2.10
`CA-ADMM` (in-domain)	24.22 ± 4.46	19.61 ± 0.61	20.12 ± 0.47	25.05 ± 2.30	20.32 ± 0.69	17.52 ± 3.88	19.08 ± 1.01

Table 3: **Ablation study results.** Best in **bold**. We measure the average and standard deviation of ADMM iterations.

	Components			Problem size (N)								
	Graph repr.	HGA	GRU	100 ∼ 200	200 ∼ 300	300 ∼ 400	400 ∼ 500	500 ∼ 600	600 ∼ 700	700 ∼ 800	800 ∼ 900	900 ∼ 1000
`RLQP`	✗	✗	✗	81.81 ± 13.45	103.41 ± 12.66	133.102 ± 10.12	155.55 ± 10.10	168.96 ± 9.82	184.60 ± 9.64	198.50 ± 9.41	212.10 ± 9.14	223.30 ± 7.92
`HGA`	✓	✓	✗	**38.61** ± 4.47	**47.06** ± 3.53	**51.51** ± 3.05	**55.59** ± 2.47	**58.37** ± 2.27	61.08 ± 2.59	64.46 ± 2.55	65.94 ± 2.18	68.94 ± 2.76
`CA-ADMM`	✓	✓	✓	41.92 ± 5.69	50.35 ± 4.08	54.41 ± 3.81	57.00 ± 3.15	59.43 ± 3.95	**59.88** ± 3.10	**62.92** ± 3.31	**65.92** ± 4.00	**67.00** ± 3.27

method. We also observed that `CA-ADMM` consistently outperforms `RLQP` with a generally smaller standard deviations.

We evaluate the size-transferability and scalability of CA-ADMM by examining whether the models trained using smaller QP problems still perform well on the large QPs. We trained CA-ADMM using small-sized Random QP instances having $10 \sim 15$ variables and applied the trained solve to solve large-scaled Random QP instances having up to 1000 variables. Fig. 3 shows how the number of ADMM iterations varies with the size of QP. As shown in the figure, `CA-ADMM` requires the least iterations for all sizes of test random QP instances despite having been trained on the smallest setting of $10 \sim 15$. The gap between our method and `RLQP` becomes larger as the QP size increases. It is interesting to note that `OSQP` requires relatively few ADMM iterations when the problem size becomes larger. It is possibly because the parameters of `OSQP` are tuned to solve the medium-sized QPs. We conclude that CA-ADMM learn how to effectively adjust ADMM parameters such that the method can effectively solve even large-sized QP instances.

Cross-domain results. We hypothesize that our learned QP solvers can transfer to new domains more easily than other baselines. To test this hypothesis, we train `CA-ADMM` and `RLQP` on `Random QP` and `EqQP` datasets, which do not assume specific QP parameter values or constraints. The trained models are evaluated on different QP domains without additional training, i.e., which is equivalent to a zero-shot transfer setting. As shown in Table 2, `CA-ADMM` consistently outperforms `RLQP` in absolute ADMM steps and shows less performance degradation when transferring to new domains. We also observed that for `EqQP` and `Huber` `CA-ADMM` exhibits better performance than in-domain cases. The experiment results indicate that the context encoder plays a crucial role in transferring to different problem classes.

6 Ablation Study

Effect of spatial context extraction methods. To understand the contribution of spatial and temporal context extraction schemes to the performance of our networks, we conduct an ablation study. The variants are `HGA` that uses spatial context extraction schemes and `RLQP` that does not use a spatial context extracting scheme. As shown in Table 3, the models using the spatial context (i.e., `CA-ADMM`

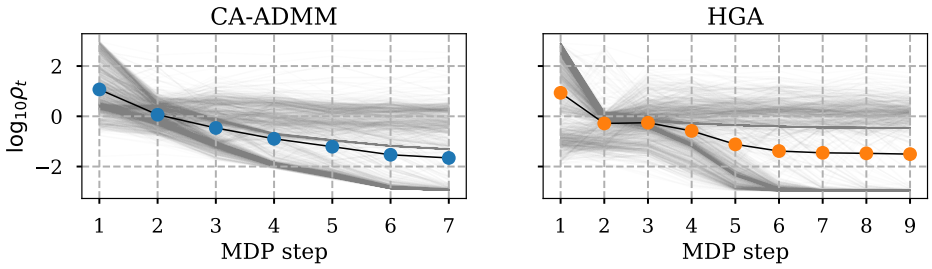

Figure 4: **Example of** $\log_{10}\rho_t$ **over MDP steps on a QP of** $N = 2,000$ **and** $M = 6000$. The gray lines visualize the time evolution of $\log_{10}\rho_t$. The black lines and markers visualize the averages of $\log_{10}\rho_t$ over the dual variables. `CA-ADMM` terminates at the 6-th MDP (i.e., solving QP in 60 ADMM steps). `HGA` takes additional two MDP steps to solve the given QP.

and `HGA`) outperform the model without the spatial context extraction. The results indicate that the spatial context significantly contributes to higher performance.

Effect of temporal context extraction methods. To further investigate the effects of temporal context, we evaluate `CA-ADMM`, `HGA` that exclude GRU (i.e., temporal context extraction scheme) from our model. We also consider `RLQP`. Table 3 shows that for $N \leq 600$ `HGA` produces better performance than `CA-ADMM`, while for $N \geq 600$ the opposite is true. We conclude that the temporal-extraction mechanism is crucial to generalizing to larger QP problems.

Qualitative analysis. Additionally, we visualize the ρ_t of `CA-ADMM` and `HGA` during solving a QP problem size of 2,000. As shown in Fig. 4, both of the models gradually decrease ρ_t over the course of optimization. This behavior is similar to the well-known step-size scheduling heuristics implemented in OSQP. We also observed that ρ_t suggested by `CA-ADMM` generally decreases over the optimization steps. On the other hand, `HGA` shows the plateau on scheduling ρ_t. We observed similar patterns from the different QP instances with different sizes.

7 Conclusion

To enhance the convergence property of ADMM, we introduced CA-ADMM, a learning-based policy that adaptively adjusts the step-size parameter ρ. We employ a spatio-temporal context encoder that extracts the spatial context, i.e., the correlation among the primal and dual variables, and temporal context, i.e., the temporal evolution of the variables, from the ADMM states. The extracted context contains information about the given QP and optimization procedures, which are used as input to the policy for adjusting ρ for the given QP. We empirically demonstrate that CA-ADMM significantly outperforms the heuristics and learning-based baselines and that CA-ADMM generalizes to the different QP class benchmarks and large-size QPs more effectively.

References

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Neural deep equilibrium solvers. In *International Conference on Learning Representations*, 2021.

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. ISSN 0959-1524. doi: https://doi.org/10.1016/S0959-1524(02)00002-1. URL https://www.sciencedirect.com/science/article/pii/S0959152402000021.

Federico Berto, Stefano Massaroli, Michael Poli, and Jinkyoo Park. Neural solvers for fast and accurate numerical optimal control. In *International Conference on Learning Representations*, 2021.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011.

Stephen Boyd, Mark T. Mueller, Brendan O'Donoghue, and Yang Wang. Performance bounds and suboptimal policies for multi–period investment. *Found. Trends Optim.*, 1(1):1–72, jan 2014. ISSN 2167-3888. doi: 10.1561/2400000001. URL https://doi.org/10.1561/2400000001.

Stephen Boyd, Enzo Busseti, Steven Diamond, Ronald N. Kahn, Kwangmoo Koh, Peter Nystrup, and Jan Speth. Multi-period trading via convex optimization, 2017. URL https://arxiv.org/abs/1705.00109.

J. Buijs, J. Ludlage, W. Van Brempt, and B. De Moor. Quadratic programming in model predictive control for large scale systems. *IFAC Proceedings Volumes*, 35(1):301–306, 2002. ISSN 1474-6670. doi: https://doi.org/10.3182/20020721-6-ES-1901.00300. URL https://www.sciencedirect.com/science/article/pii/S1474667015387218. 15th IFAC World Congress.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL https://arxiv.org/abs/1412.3555.

Gérard Cornuéjols, Javier Peña, and Reha Tütüncü. *Optimization Methods in Finance*. Cambridge University Press, 2 edition, 2018. doi: 10.1017/9781107297340.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. *CoRR*, abs/1603.05629, 2016. URL http://arxiv.org/abs/1603.05629.

Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction, 2019. URL https://arxiv.org/abs/1906.09575.

Balthazar Donon, Zhengying Liu, Wenzhuo Liu, Isabelle Guyon, Antoine Marot, and Marc Schoenauer. Deep statistical solvers. *Advances in Neural Information Processing Systems*, 33:7910–7921, 2020.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956. doi: https://doi.org/10.1002/nav.3800030109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030109.

Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976. ISSN 0898-1221. doi: https://doi.org/10.1016/0898-1221(76)90003-1. URL https://www.sciencedirect.com/science/article/pii/0898122176900031.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *CoRR*, abs/1906.01629, 2019. URL http://arxiv.org/abs/1906.01629.

Euhanna Ghadimi, André Teixeira, Iman Shames, and Mikael Johansson. Optimal parameter selection for the alternating direction method of multipliers (admm): Quadratic problems. *IEEE Transactions on Automatic Control*, 09 2014. doi: 10.1109/TAC.2014.2354892.

Bingsheng He, Hai Yang, and Shengli Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and Applications*, 106:337–356, 2000.

Erik Hons. *Constrained quadratic optimization: Theory and application for wireless communication systems*. PhD thesis, University of Waterloo, 2001.

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. *CoRR*, abs/2107.10847, 2021. URL https://arxiv.org/abs/2107.10847.

V. Kecman, T. Arthanari, and I. Hadzic. Lp and qp based learning from empirical data. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 4, pp. 2451–2455 vol.4, 2001. doi: 10.1109/IJCNN.2001.938751.

Pablo Krupa, Ignacio Alvarado, Daniel Limon, and Teodoro Alamo. Implementation of model predictive control for tracking in embedded systems using a sparse extended ADMM algorithm. *IEEE Transactions on Control Systems Technology*, 30(4):1798–1805, jul 2022. doi: 10.1109/tcst.2021.3128824. URL https://doi.org/10.1109%2Ftcst.2021.3128824.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015. URL https://arxiv.org/abs/1509.02971.

Zhi-Quan Luo and Wei Yu. An introduction to convex optimization for communications and signal processing. *IEEE Journal on Selected Areas in Communications*, 24(8):1426–1438, 2006. doi: 10.1109/JSAC.2006.879347.

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

Modelisation Mathematique, Analyse Man, and Eliane Ecache. Mathematical modelling and numerical analysis esaim: M2an modelisation mathematique et analyse numerique m2an, vol. 36, n. 04 2004.

Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994. doi: 10.1137/1.9781611970791. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611970791.

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021a.

Junyoung Park, Jinhyun Choo, and Jinkyoo Park. Convergent graph solvers. In *International Conference on Learning Representations*, 2021b.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. *Advances in Neural Information Processing Systems*, 33:21105–21117, 2020.

Heechang Ryu, Hayong Shin, and Jinkyoo Park. Multi-agent actor-critic with hierarchical graph attention network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7236–7243, Apr. 2020. doi: 10.1609/aaai.v34i05.6214. URL https://ojs.aaai.org/index.php/AAAI/article/view/6214.

Fei Sha, Lawrence Saul, and Daniel Lee. Multiplicative updates for nonnegative quadratic programming in support vector machines. In S. Becker, S. Thrun, and K. Obermayer (eds.), *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002. URL https://proceedings.neurips.cc/paper/2002/file/dc16622ddc767e6bc1200fe5df2fbdfb-Paper.pdf.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, feb 2020a. doi: 10.1007/s12532-020-00179-2. URL https://doi.org/10.1007%2Fs12532-020-00179-2.

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

Philip Wolfe. The simplex method for quadratic programming. *Econometrica*, 27:170, 1959.

A GRAPH PREPROCESSING

In this section, we provide the details of the graph preprocessing scheme that imposes the scale invariant to the heterograph representation. The preprocessing step is done by scaling the objective function, and the constraints. Algorithm 1 and 2 explain the preprocessing procedures.

Algorithm 1: Objective scaling

Input: Quadratic cost matrix $\boldsymbol{P} \in \mathbb{S}_{+}^{N}$, linear cost terms $q \in \mathbb{R}^{N}$
Output: Scaled quadratic cost matrix $\boldsymbol{P}' \in \mathbb{S}_{+}^{N}$, scale linear cost terms $q' \in \mathbb{R}^{N}$

for $n = 1, 2, ...$ **do**
  $p^* = \max(\max_n(|\boldsymbol{P}_{nn}|), 2\max_{(i,j),i\neq j}|\boldsymbol{P}_{ij}|)$ `// Compute scaler`
  $\boldsymbol{P}' = \boldsymbol{P}/p^*$
  $\boldsymbol{q}' = \boldsymbol{q}/p^*$
end

Algorithm 2: Constraints scaling

Input: constraint matrix $\boldsymbol{A} \in \mathbb{R}^{M\times N}$, lower bounds $\boldsymbol{l} \in \mathbb{R}^{M}$, upper bound $\boldsymbol{u} \in \mathbb{R}^{M}$.
Output: Scaled constraint matrix $\boldsymbol{A}' \in \mathbb{R}^{M\times N}$, Scaled lower bounds $\boldsymbol{l}' \in \mathbb{R}^{M}$,
Scaled upper bound $\boldsymbol{u}' \in \mathbb{R}^{M}$.

for $m = 1, 2, ...$ **do**
  $a_m^* = \max_n(|\boldsymbol{A}_{mn}|)$ `// Compute scaler for each row`
  $\boldsymbol{A}'_m = \boldsymbol{A}_m/a_m^*$
  $\boldsymbol{l}'_m = \boldsymbol{l}_m/a_m^*$
  $\boldsymbol{u}'_m = \boldsymbol{u}_m/a_m^*$
end

B DETAILS OF HGA

In this section, we provide the details of the HGA layer.

HGA is composed of edge update, edge aggregation, node update and hetero node aggregation to extract the informative context. This is done by considering each edge type and computing the updated node, edge embedding.

We denote source and destination nodes with the index i and j and the edge type with k. h_i and h_j denote the source and destination node embedding and h_{ij}^k denotes the embedding of the k type edge between the node i and j.

Edge update HGA computes the edge embedding h'_{ij} and the corresponding attention logit z_{ij} as:

$$h'_{ij} = \text{HGA}_E^k([h_i, h_j, h_{ij}^k]) \tag{A.1}$$

$$z_{ij} = \text{HGA}_A^k([h_i, h_j, h_{ij}^k]), \tag{A.2}$$

where HGA_E^k and HGA_A^k are the hetero edge update function and the hetero attention function for edge type k, respectively. Both functions are parameterized with Multilayer Perceptron (MLP).

Edge aggregation HGA aggregates the typed edges as follows:

$$\alpha_{ij} = \frac{\exp(z_{ij}^k)}{\sum_{i\in N_k(j)}\exp(z_{ij}^k)} \tag{A.3}$$

where $N_k(j) = \{v_i | \text{type of } e_{ij} = k, v_j \in \mathcal{N}(i)\ \forall i\}$ is the set of all neighborhood of v_j, which type of edge from the element of the set to v_j is k.
Second, HGA aggregates the messages and produces the type k message m_j^k for node v_j.

$$m_j^k = \sum_{i\in N_k(j)} \alpha_{ij} h'_{ij} \tag{A.4}$$

Table A.1: **The MLP sturcture of functions in HGA's first layer**

	input dimensions	hidden dimensions	output dimensions	hidden activations	output activation
$\text{HGA}_E^{\text{p2p}}$	8				
$\text{HGA}_E^{\text{p2d}}$	12				
$\text{HGA}_E^{\text{d2p}}$	12				
$\text{HGA}_A^{\text{p2p}}$	8				
$\text{HGA}_A^{\text{p2d}}$	12	$[64, 16]$	8	LeakyReLU	
$\text{HGA}_A^{\text{d2p}}$	12				
$\text{HGA}_V^{\text{p2p}}$	11				
$\text{HGA}_V^{\text{p2d}}$	16				
$\text{HGA}_V^{\text{d2p}}$	11				
HGA_N	8	$[64, 32]$			

Edge type-aware node update The aggregated message with edge type k, m_j^k, is used to compute the updated node embedding with edge type k, $\mathfrak{h}_j^{k\prime}$, as:

$$\mathfrak{h}_j^k = \text{HGA}_V^k([h_j, m_j^k]) \tag{A.5}$$

where HGA_V^k is the hetero node update function, which is composed of MLP.

The above three steps (edge update, edge aggregation, edge type-aware node update) are performed separately on each edge type to preserve the characteristics of the edge type and extract meaningful embeddings.

Hetero Node update HGA aggregates the updated hetero node feature $\mathfrak{h}_j^k$ to produce the updated node embedding h'_j as follows.
First, HGA computes the attention logit d_j^k corresponding to $\mathfrak{h}_j^k$ as:

$$d_j^k = \text{HGA}_N(\mathfrak{h}_j^k) \tag{A.6}$$

where HGA_N is the hetero node update function, which is composed of MLP.
Second, HGA computes attention β_j^k using the softmax function with d_j^k:

$$\beta_j^k = \frac{\exp(d_j^k)}{\sum_{k \in E_{dst}(j)} \exp(d_j^k)} \tag{A.7}$$

where $E_{dst}(j)$ is the set of edge types that has a destination node at j.
Finally, HGA aggregates the per-type the updated messages to compute updated node feature h'_j for v_j as:

$$h'_j = \sum_{k \in E_{dst}(j)} \beta_j^k \mathfrak{h}_j^k \tag{A.8}$$

C Details of network architecture and training

In this section, we provide the architecture of pocliy network π_θ and action-value network Q_ϕ.

Policy netowrk π_θ architecture. As explained in Section 4.4, the policy network is consist HGA and MLP. We parameterize π_θ as follows:

$$\rho_t = \text{MLP}_\theta\Big(\text{HGA}_\theta\big(\text{CONCAT}(\mathcal{G}_t, \mathcal{C}_t)\big)\Big), \tag{A.9}$$

Table A.2: **The MLP sturcture of functions in HGA's n th layer($n \geq 2$)**

	input dimensions	hidden dimensions	output dimensions	hidden activations	output activation
$\text{HGA}_E^{\text{p2p}}$	8	$[64, 16]$	8	LeakyReLU	
$\text{HGA}_E^{\text{p2d}}$					
$\text{HGA}_E^{\text{d2p}}$					
$\text{HGA}_A^{\text{p2p}}$					
$\text{HGA}_A^{\text{p2d}}$					
$\text{HGA}_A^{\text{d2p}}$					
$\text{HGA}_V^{\text{p2p}}$					
$\text{HGA}_V^{\text{p2d}}$					
$\text{HGA}_V^{\text{d2p}}$					
HGA_N		$[64, 32]$			

where MLP_θ is two-layered MLP with the hidden dimension $[64, 32]$, the LeakyReLU hidden activation, the ExpTanh (Eq. (A.10)) last activation, and HGA_θ is a HGA layer.

$$\text{ExpTanh}(x) = (\tanh(x) + 1) \times 3 + (-3) \tag{A.10}$$

Q network Q_π architecture. As mentioned in Section 4.4, Q_π has similar architecture to π_θ. We parameterize Q_π as follows:

$$\rho_t = \text{MLP}_\phi\Big(\text{READOUT}\Big(\text{HGA}_\phi\big(\text{CONCAT}(\mathcal{G}_t, \mathcal{C}_t, \rho_t)\big)\Big)\Big), \tag{A.11}$$

where MLP_ϕ is two-layered MLP with the hidden dimension $[64, 32]$, the LeakyReLU hidden activation, the Identity last activation, HGA_ϕ is a HGA layer, and READOUT is the weighted sum, min, and amx readout function that summarizes node embeddings into a single vector.

Training detail. We train all models with mini-batches of size 128 for 5,000 epochs by using Adam with the fixed learning rate. For π_θ, we set learning rate as 10^{-4} and, for Q_ϕ, we set learning rate as 10^{-3}. We set history length l as 3.

D Details of Problem Generation

In this section, we provide the generation scheme of problem classes in OSQP, which is used to train and test our model. Every problem class is based on OSQP (Stellato et al., 2020a), but we modified some settings, such as the percentage of nonzero elements in the matrix or matrix size, to match graph sizes among the problem classes.

D.1 Random QP

$$\begin{aligned} \min_x \quad & \frac{1}{2} x^\top \boldsymbol{P} x + \boldsymbol{q}^\top x \\ \text{subject to} \quad & \boldsymbol{l} \leq \boldsymbol{A} x \leq \boldsymbol{u}, \end{aligned} \tag{A.12}$$

Problem instances We set a random positive semidefinite matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ by using matrix multiplication on random matrix $\boldsymbol{M}$ and its transpose, whose element $\boldsymbol{M}_{ij} \sim \mathcal{N}(0, 1)$ has only 15% being nonzero elements. We generate the constraint matrix $x \in \mathbb{R}^{m \times n}$ with $\boldsymbol{A}_{ij} \sim \mathcal{N}(0, 1)$ with only 45% nonzero elements. We chose upper bound $\boldsymbol{u}$ with $\boldsymbol{u}_i \sim (Ax)_i + \mathcal{N}(0, 1)$, where x is randomly chosen vector, and the upper bound $\boldsymbol{l}_i$ is same as $-\infty$.

D.2 EQUALITY CONSTRAINED QP

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^\top \boldsymbol{P} x + \boldsymbol{q}^\top x \\ \text{subject to} \quad & \boldsymbol{l} = \boldsymbol{A} x = \boldsymbol{u}, \end{aligned} \tag{A.13}$$

Problem instances We set a random positive semidefinite matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ by using matrix multiplication on random matrix $\boldsymbol{M}$ and its transpose, whose element $\boldsymbol{M}_{ij} \sim \mathcal{N}(0,1)$ has only 30% of being nonzero elements. We generate the constraint matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $\boldsymbol{A}_{ij} \sim \mathcal{N}(0,1)$ with only 75% nonzero elements. We chose upper bound $\boldsymbol{u}$ with $\boldsymbol{u}_i \sim (Ax)_i$, where x is randomly chosen vector, and the upper bound $\boldsymbol{l}_i \in \boldsymbol{l}$ is same as $\boldsymbol{u}_i$.

D.3 PORTFOLIO OPTIMIZATION

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^\top D x + \frac{1}{2} \boldsymbol{y}^\top \boldsymbol{y} - \frac{1}{2\gamma} \mu^\top x \\ \text{subject to} \quad & \boldsymbol{y} = \boldsymbol{F}^\top x, \\ & \mathbf{1}^\top x = 1, \\ & x \geq 0, \end{aligned} \tag{A.14}$$

Problem instances We set the factor loading matrix $\boldsymbol{F} \in \mathbb{R}^{n \times m}$, whose element $\boldsymbol{F}_{ij} \sim \mathcal{N}(0,1)$ as 90% nonzero elements. We generate a diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$, whose element $\boldsymbol{D}_{ii} \sim \mathcal{C}(0,1) \times \sqrt{m}$ is the asset-specific risk. We chose a random vector $\mu \in \mathbb{R}^n$, whose element $\mu_i \sim \mathcal{N}(0,1)$ and is the expected returns.

D.4 SUPPORT VECTOR MACHINE (SVM)

$$\begin{aligned} \min_{x} \quad & x^\top x + \lambda \mathbf{1}^\top \boldsymbol{t} \\ \text{subject to} \quad & \boldsymbol{t} \geq \mathbf{diag}(\boldsymbol{b}) \boldsymbol{A} x + \mathbf{1}, \\ & \boldsymbol{t} \geq 0, \end{aligned} \tag{A.15}$$

Problem instances We set the matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, which has elements $\boldsymbol{A}_{ij}$ as follows:

$$A_{ij} = \begin{cases} \mathcal{N}(\frac{1}{n}, \frac{1}{n}), & i \leq \frac{m}{2} \\ \mathcal{N}(-\frac{1}{n}, \frac{1}{n}), & \text{otherwise} \end{cases}$$

We generate the random vector $\boldsymbol{b} \in \mathbb{R}^m$ ahead as follows:

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

we choose the scalar λ as $\frac{1}{2}$.

D.5 HUBER FITTING

$$\begin{aligned} \min_{x} \quad & \boldsymbol{u}^\top \boldsymbol{u} + 2M \mathbf{1}^\top (\boldsymbol{r} + \boldsymbol{s}) \\ \text{subject to} \quad & \boldsymbol{A} x - \boldsymbol{b} \boldsymbol{u} = \boldsymbol{r} - \boldsymbol{s}, \\ & \boldsymbol{r} \geq 0, \\ & \boldsymbol{s} \geq 0, \end{aligned} \tag{A.16}$$

Problem instances We set the matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with element $\boldsymbol{A}_{ij} \sim \mathcal{N}(0,1)$ to have 50% nonzero elements. To generate the vector $\boldsymbol{b} \in \mathbb{R}^m$, we choose the random vector $\boldsymbol{v} \in \mathbb{R}^n$ ahead as follows:

$$v_i = \begin{cases} \mathcal{N}(0, \frac{1}{4}), & \text{with probability } p = 0.95 \\ \mathcal{U}[0, 10], & \text{otherwise} \end{cases}$$

Then, we set the vector $\boldsymbol{b} = A(v + \epsilon)$, where $\epsilon_i \sim \mathcal{N}(0, \frac{1}{n})$.

Table A.3: **Range of n, m which uses for each problem class generation**

	Random QP	EqQP	Porfolio	SVM	Huber	Control	Lasso	En_Random QP
n	$10 \sim 15$	$30 \sim 40$	$50 \sim 60$	$5 \sim 6$	$5 \sim 6$	$2 \sim 6$	$5 \sim 6$	$20 \sim 30$
m	$10 \sim 15$	$15 \sim 20$	$5 \sim 6$	$50 \sim 60$	$50 \sim 60$	$1 \sim 3$	$50 \sim 60$	$45 \sim 70$
Relation m & n	$m = 3 \times n$	$m = \lfloor \frac{n}{2} \rfloor$	$m = 10 \times n$	$m = \lfloor \frac{n}{10} \rfloor$	$m = 10 \times n$	$m = \lfloor \frac{n}{2} \rfloor$	$m = 10 \times n$	$m = \lfloor \frac{7n}{3} \rfloor$

D.6 Optimal Control

$$\begin{aligned} \min_{x} \quad & x_T^\top \boldsymbol{Q}_T x_T + \sum_{t=0}^{T-1} x_t^\top Q x_t + \boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t \\ \text{subject to} \quad & x_{t+1} = \boldsymbol{A} x_t + \boldsymbol{B} \boldsymbol{u}_t, \\ & x_t \in \mathcal{X}, \boldsymbol{u}_t \in \mathcal{U}, \\ & x_0 = x_{\text{init}}, \end{aligned} \tag{A.17}$$

Problem instances We set the dynamic $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ as $\boldsymbol{A} = \boldsymbol{I} + \Delta$, where $\Delta_{ij} \sim \mathcal{N}(0, 0.01)$. We generate the matrix $\boldsymbol{B} \in \mathbb{R}^{n \times m}$, whose element $\boldsymbol{B}_{ij} \sim \mathcal{N}(0, 1)$. We choose state cost mQ as $\text{diag}(\boldsymbol{q})$, where $q_i \sim \mathcal{U}(0, 10)$ with 30% zeros elements in vq. We generate the input cost $\boldsymbol{R}$ as $0.1\boldsymbol{I}$. We set time T as 5.

D.7 Lasso

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} \boldsymbol{y}^\top \boldsymbol{y} + \gamma \boldsymbol{1}^\top \boldsymbol{t} \\ \text{subject to} \quad & \boldsymbol{y} = \boldsymbol{A} x - \boldsymbol{b}, \\ & -\boldsymbol{t} \leq x \leq \boldsymbol{t}, \end{aligned} \tag{A.18}$$

Problem instances We set the matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, whose element $\boldsymbol{A}_{ij} \sim \mathcal{N}(0, 1)$ with 90% nonzero elements. To generate the vector $\boldsymbol{b} \in \mathbb{R}^m$, we set the sparse vector $\boldsymbol{v} \in \mathbb{R}^n$ ahead as follows:

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

Then, we chose the vector $\boldsymbol{b} = Av + \epsilon$ where $\epsilon_i \sim \mathcal{N}(0, 1)$. We set the weighting parameter γ as $\frac{1}{5} ||\boldsymbol{A}^\top b||_\infty$.

For the seven problem types mentioned above, the range of variables n and m can be referred to in the Table A.3. The relationship between n and m is also described to facilitate understanding.

D.8 Entire Random QP

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^\top \boldsymbol{P} x + \boldsymbol{q}^\top x \\ \text{subject to} \quad & \boldsymbol{l} \leq \boldsymbol{A} x \leq \boldsymbol{u}, \\ & \boldsymbol{B} x = \boldsymbol{b}, \end{aligned} \tag{A.19}$$

Problem instances We set a random positive semidefinite matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ by using matrix multiplication on random matrix $\boldsymbol{M}$ and its transpose, whose element $\boldsymbol{M}_{ij} \sim \mathcal{N}(0, 1)$ has only 15% being nonzero elements. We generate the inequality constraint matrix $\boldsymbol{A} \in \mathbb{R}^{m_1 \times n}$ with $\boldsymbol{A}_{ij} \sim \mathcal{N}(0, 1)$ with only 60% nonzero elements. We chose upper bound $\boldsymbol{u} \in \mathbb{R}^{m_1}$ with $\boldsymbol{u}_i \sim (Ax)_i + \mathcal{N}(0, 1)$, where x is a randomly chosen vector and $\boldsymbol{l}_i$ as $-\infty$. We generate the equality constraint matrix $\boldsymbol{B} \in \mathbb{R}^{m_2 \times n}$ with $\boldsymbol{B}_{ij} \sim \mathcal{N}(0, 1)$ with only 60% nonzero elements. We set constant $\boldsymbol{b} \in \mathbb{R}^{m_2}$, with $\boldsymbol{b}_i = (Bx)_i$, where x is a randomly chosen vector at the generating process of vector $\boldsymbol{u}$. We set m_1, and m_2 as $\lfloor \frac{m}{7} \rfloor$ and $\lfloor \frac{6m}{7} \rfloor$, respectively.

Table A.4: **QP size generalization results in table format**. Smaller is better (↓). Best in **bold**. We measure the average and standard deviation of the iterations for each QP sizes.

	Problem size (N)								
	$100 \sim 200$	$200 \sim 300$	$300 \sim 400$	$400 \sim 500$	$500 \sim 600$	$600 \sim 700$	$700 \sim 800$	$800 \sim 900$	$900 \sim 1000$
CA-ADMM	**41.92** ± 5.69	**50.35** ± 4.08	**54.08** ± 3.81	**57.00** ± 3.15	**59.43** ± 3.95	**59.88** ± 3.10	**62.92** ± 3.31	**65.92** ± 4.00	**67.00** ± 3.27
RLQP	81.81 ± 13.45	103.41 ± 12.66	133.102 ± 10.12	155.55 ± 10.10	168.96 ± 9.82	184.60 ± 9.64	198.50 ± 9.41	212.10 ± 9.14	223.30 ± 7.92
OSQP	251.35 ± 71.36	171.59 ± 40.56	152.57 ± 32.08	125.39 ± 17.85	108.84 ± 15.44	102.92 ± 11.73	102.24 ± 11.47	103.02 ± 6.36	109.56 ± 6.04

E DETAIL OF THE EXPERIMENT RESULTS

E.1 DETAILS OF THE IN-DOMAIN EXPERIMENT

In this section, we provide the problem generation, and additional experimental results for the In-domain experiments.

Training data generation For the training set generation, the problems for each class are generated randomly with instance generating rules in (Appendix D). Each problem's size of QP is described in Table A.3.

Evaluation data generation Table 1's evaluation data generated in the same way as training data generation. To construct empty intersection between train and test set, each set is generated from a different random seed. To verify the QP size generalization test (Fig. 3, Table A.4), problems are generated randomly for the problem class **Random QP** with dimension of x in $[100, 1000]$.

Additional results We provide the following Table A.4 to show the numerical results difficult to visualize in Fig. 3.

E.2 DETAILS OF THE CROSS-DOMAIN EXPERIMENT

We provide the problem generation for the cross-domain experiments.

Training data generation We create a new problem class named **Entire Random QP** that contains both the randomly generated inequality and equality constraints as there is no problem class in OSQP. The form of the QP problem class and instance generating rule are described in Appendix D.8. With the new problem class, we generate the training set with size in Table A.3.

Evaluation data generation For the training set generation, the problems for each class are generated randomly with instance generating rules in (Appendix D). the size of each problem is described in Table A.3.