DEQ-MPC:
Deep Equilibrium Model Predictive Control

Swaminathan Gurumurthy[*1], **Khai Nguyen**[1→2], **Arun Bishop**[1], **Zico Kolter**[1,3], **Zachary Manchester**[1]
[1]Carnegie Mellon University [2]Massachusetts Institute of Technology
[3]Bosch Center for Artificial Intelligence

https://sites.google.com/view/deq-mpc

Abstract: Incorporating task-specific priors within a policy or network architecture is crucial for enhancing safety and improving representation and generalization in robotic control problems. Differentiable Model Predictive Control (MPC) layers have proven effective for embedding these priors, such as constraints and cost functions, directly within the architecture, enabling end-to-end training. However, current methods often treat the solver and the neural network as separate, independent entities, leading to suboptimal integration. In this work, we propose a novel approach that co-develops the solver and architecture unifying the optimization solver and network inference problems. Specifically, we formulate this as a *joint fixed-point problem* over the coupled network outputs and necessary conditions of the optimization problem. We solve this problem in an iterative manner where we alternate between network forward passes and optimization iterations. Through extensive ablations in various robotic control tasks, we demonstrate that our approach yields richer representations and more stable training, while naturally accommodating warm starts, a key requirement for MPC.

Keywords: differentiable optimization, deep equilibrium model, model predictive control

1 Introduction

Incorporating task-specific priors within the policy training pipeline is often beneficial for robotic control problems. These priors give the system designer additional control and flexibility while designing the system and play a vital role in enhancing safety, improving representation, and boosting generalization. Previous approaches to policy learning have explored various methods to embed such priors, including reward/loss shaping [1], incorporating constrained optimization layers within the policy inference pipeline [2, 3], adding parallel/post-hoc safety checks/controllers [4], adversarial training [5], and domain randomization [6].

Differentiable Model Predictive Control (MPC) layers [2] have emerged as a promising approach [7, 8, 9] to embed such priors. This method integrates MPC as a differentiable layer within neural network architectures, embedding the constraints and cost functions directly into the network architecture. Importantly, they allow us to preserve the interpretability and safety guarantees associated with traditional MPC while providing a general framework applicable to a diverse range of robotic control problems. Moreover, it allows for test-time modifications of the MPC problem and facilitates online adaptation – a critical feature in dynamic environments.

However, differentiable MPC layers treats the optimization solver as a black box, overlooking its unique characteristics. This simplification, while convenient, overlooks the unique characteristics of MPC solvers. Unlike typical NN layers, MPC solvers are implicit, iterative, and prone to issues like ill-conditioning and discontinuities, potentially destabilizing training. Their structure also enables efficient warm-starting, often underutilized in current frameworks.

[*] Corresponding author: gauthamsindia95@gmail.com

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

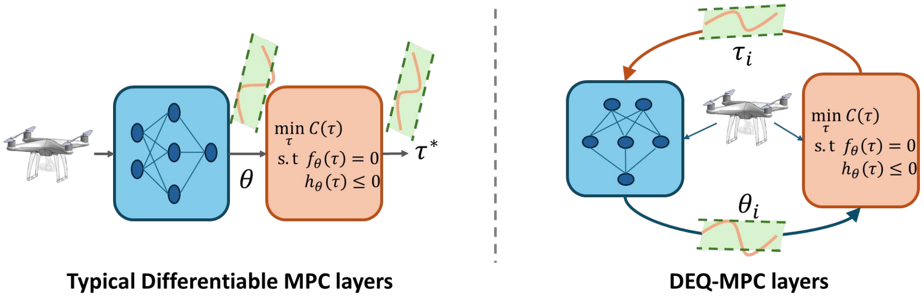


Figure 1: We propose DEQ-MPC layers (right) as a direct improvement over differentiable MPC layers (left). These layers offer increased representational power, smoother gradients, and are more amenable to warm-starting. DEQ-MPC layers formulate the network inference (θ) and trajectory optimization (τ) as a joint fixed-point problem, solving them in an alternating iterative manner, instead of the single-shot sequential inference used in differentiable MPC setups. This approach uses the network to adapt the solver inputs, θ_i, based on the current optimizer state, τ_i, enabling a richer feedback process. The specific example in the figure shows a trajectory tracking example, where the robot observations are fed to the system. The network predicts the waypoints θ_i (solver inputs). The solver solves the tracking problem to spit out solved trajectories τ_i to track the waypoints θ_i.

To address these limitations, we propose Deep Equilibrium Model Predictive Control (DEQ-MPC), a novel approach that unifies the optimization solver and the neural network architecture. Instead of treating the optimization layer as just another layer within the network, we formulate a joint inference and optimization problem as shown in figure 1, where we treat the network inference and the optimization problem as a unified system and jointly compute a fixed point over them. Thus, the network outputs can now depend on the solver iterates and vice-versa, thereby, allowing a tight coupling between the two. The fixed point is computed by alternating between the network forward pass (conditioned on the most recent optimizer iterate) and the optimization solver iterations (conditioned on the most recent network outputs) until the joint system reaches an equilibrium (hence the name DEQ-MPC, i.e, Deep Equilibrium Model Predictive Control).

This joint inference/optimization framework also allows us to explore several interesting aspects of the solver and architecture design. Specifically, for the optimization solver, we implement an augmented Lagrangian (AL) solver which works well with warm-starting and is robust at handling arbitrary non-linear constraints. This is important for the joint fixed point process as it allows us to change the optimization parameters (i.e, network outputs, θ_i) between successive optimization iterates. For the architecture, we experiment with parameterizing the network architecture itself as a Deep Equilibrium model (DEQ), a type of implicit neural network that computes the outputs/latents as a fixed point of a non-linear transformation. It can be seen as an infinite depth network which applies the same layer an infinite number of times eventually reaching a fixed point in the outputs/latents. This iterative fixed point finding procedure blends nicely with the equilibrium/fixed point finding nature of the overall system. We observe nicer stability properties when using a DEQ as the network architecture when going to more complicated settings.

This unified approach results in several key benefits: First, it enables richer representations by allowing the network to adapt its features/outputs depending on the solver state. Second, it allows us to naturally compute smoother gradients during training, facilitating more stable and efficient learning. Third, it inherently accommodates warm-starting, leveraging the recurrent nature of MPC to improve computational efficiency and solution quality. DEQ-MPC thus offers a more robust and flexible framework for integrating optimization-based control with deep learning.

The primary contributions of this work are as follows: (1) We introduce DEQ-MPC, a novel framework to integrate MPC layers within deep networks. (2) Through extensive ablations, we show that this unified approach results in richer representations, improved gradient flow, and enhanced suitability for warm-starting, compared to standard differentiable MPC methods. (3) We propose a training setup specifically for streaming MPC applications that leverages warm-starting across time steps. (4) We provide empirical evidence demonstrating the advantages of DEQ-MPC on trajectory prediction and tracking problems across various continuous control tasks that require strict constraint satisfaction both on simulation and hardware. While the paper focuses on MPC to ground our methods in concrete problems, we believe that the insights and techniques would generalize to a much broader class of constrained optimization layers and applications.

2 Related Work

Differentiable optimization layers were introduced as a means to embed convex optimization problems [10, 11] as differentiable units within deep networks. Recent works have extended the range of optimization and fixed-point problems that can be made differentiable [12, 13, 14, 15]. Since their introduction, they have been applied to a variety of robotics problems, such as state estimation [16], SLAM [17, 18], motion planning [19, 13] and control [2, 3] for applications such as autonomous driving [7, 20], navigation [8, 9], and manipulation [13]. We specifically look at differentiable MPC problems building off of work such as [2, 3, 21, 22, 23, 13].

However, incorporating MPC (and optimization) layers within deep networks often comes with its own set of challenges. The bi-level problem can often be very non-convex resulting in the local gradient direction being mis-aligned with the desired global update direction [13, 2]. The gradient landscape often has discontinuities resulting in undesirable gradient artifacts [24, 25]. Furthermore, the problem structure can also result in very high variance in gradients [26]. It's often challenging to incorporate warm-starting techniques as the problem parameters change with each problem instance [27] resulting in long inference and solve times. The network predicted constraint parameters can often be infeasible [28], resulting in undefined problem solutions or gradients. Some modelling assumptions in the optimization layer are often not faithful to the real data causing model mismatch problems [29]. Addressing these challenges is key for practical adoption of optimization layers.

Our approach formulates network inference and optimization as a joint equilibrium problem, addressing representation, gradient smoothness, and warm-starting. This relates to prior work using joint inference/optimization for inverse problems [30] and SLAM/pose estimation [31, 17, 18], which focused on unconstrained non-linear least squares. We generalize this concept to constrained optimization, specifically MPC, and explicitly compare against standard differentiable optimization to highlight the benefits of the joint approach.

3 Background

3.1 Differentiable Model Predictive Control

Model Predictive Control (MPC) solves a finite-horizon optimal control problem at each time step :

$$\begin{aligned} \tau^*_{0:T} = \arg\min_{\tau_{0:T}} \quad & \sum_t C_{\theta,t}(\tau_t) \\ \text{subject to} \quad & x_0 = x_{\text{init}},\ x_{t+1} = f_\theta(\tau_t),\ h_\theta(\tau_t) \le 0,\ t = 0, \dots, T, \end{aligned} \tag{1}$$

where $\tau_t = (x_t, u_t)$, $C_{\theta,t}$ is the cost, f_θ the dynamics and h_θ ineq. constraints (e.g. safety constraints, joint limits, etc.). This problem is typically solved using non-linear optimization techniques.

Differentiable MPC computes gradients of the solution τ^* w.r.t. solver inputs θ using the implicit function theorem (IFT) on the Karush-Kuhn-Tucker (KKT) conditions of equation 1 [2]. Let $z^* = (\tau^*, \lambda^*, \nu^*)$ be the primal-dual solution to the KKT conditions $F(z, \theta) = 0$ of equation 1. The corresponding gradient can be computed as:

$$\frac{\partial z^*}{\partial \theta} = -\left(\frac{\partial F}{\partial z}\right)^{-1} \cdot \frac{\partial F}{\partial \theta}, \tag{2}$$

3.2 Deep Equilibrium Models

Deep Equilibrium Models [32] are a class of implicit deep learning models that compute the output as a solution to a fixed point problem. Specifically, given an input $x \in \mathcal{X}$, computing the forward pass in a DEQ model involves finding a fixed point $z \in \mathcal{Z}$, such that

$$z^\star = d_\phi(z^\star, x), \tag{3}$$

where, $d_\phi : \mathcal{Z} \times \mathcal{X} \to \mathcal{Z}$ is some parameterized layer conditioned on input x, $\mathcal{Z}$ denotes the hidden state or outputs of the network which we are computing the fixed point on, $\mathcal{X}$ denotes the input space, and ϕ denotes the parameters of the layer. Computing this fixed point (under proper stability conditions) corresponds to the "infinite depth" limit of repeatedly applying the function $z^+ := d_\phi(z, x)$ starting at some arbitrary initial value of z (typically 0).

4 Method

4.1 Problem Grounding through Trajectory Prediction and Tracking

We begin by grounding our discussion in a simple trajectory prediction and tracking example. This setting will also serve as the default configuration for most of our subsequent experiments.

Given system dynamics f and a dataset of optimal trajectories across different initial and environmental conditions, we learn a policy using imitation learning problem while respecting several constraints. We model this policy as consisting of two components. The first is a neural network NN_ϕ that predicts the waypoints to be tracked and other environment parameters $\theta_{0:T}$ for the next T time steps given the current state x_{init} and some observations o:

$$\theta_{0:T} = \text{NN}_\phi(x_{\text{init}}, o). \tag{4}$$

The second is an MPC solver that solves the trajectory tracking problem to compute dynamically feasible trajectories $\tau_{0:T}$ that track the waypoints while satisfying the required constraints:

$$\begin{aligned} \tau^*_{0:T} = \underset{\tau_{0:T}}{\arg\min} \quad & \sum_t \|x_t - \theta_t\|_Q^2 + \|u_t\|_R^2 \\ \text{subject to} \quad & x_0 = x_{\text{init}},\ x_{t+1} = f_\theta(\tau_t),\ h_\theta(\tau_t) \leq 0,\ t = 0, \ldots, T. \end{aligned} \tag{5}$$

In a standard differentiable-MPC setup these two components are executed sequentially, one after the other as shown in figure 1. The outputs of the system, $\tau^*_{0:T}$ are used to compute a loss, $\ell(\tau^*_{0:T})$, such as a supervised L1 loss with some expert trajectory demonstrations $\tau^{\text{exp}}_{0:T}$. The loss is then optimized using a stochastic gradient optimizer to learn the network parameters.

4.2 DEQ-MPC

4.2.1 The Inference Problem, Architecture and Solver

MPC solvers are implicit layers and hence iterative. Using a single θ estimate throughout the solver iterations is inefficient, especially for non-linear optimization problems. To address this, DEQ-MPC modifies the single-shot inference problem described in equations 4 and 5 into a joint inference/optimization problem over the network outputs and the optimizer iterates. This approach, illustrated in Figure 1, allows us to condition the network outputs (solver inputs, θ) on the optimizer state τ and vice versa. This is expressed as a single constrained optimization problem:

$$\tau^*_{0:T}, \theta^* = \underset{\tau_{0:T}, \theta}{\arg\min} \quad \sum_t C_{\theta,t}(\tau_t) \tag{6}$$

$$\text{subject to} \quad x_0 = x_{\text{init}},\ x_{t+1} = f_\theta,\ h_\theta(\tau_t) \leq 0, \tag{7}$$

$$\theta = \text{NN}_\phi(x_{\text{init}}, o, \tau_{0:T}),\ t = 0, \ldots, T, \tag{8}$$

where the last constraint expresses the neural network inference as an equality constraint. Typical non-linear optimization solvers [33, 34] or bi-level optimization solvers [35, 36, 37] struggle with this constraint due to the nastiness of the resulting constraint Jacobians. We propose to solve this problem using the alternating direction method of multipliers (ADMM) algorithm [38], alternating between (1) solving the MPC optimization problem (with fixed θ), equations 6 and 7 using the augmented Lagrangian (AL) method and (2) the constraint projection step, equation 8 (i.e, the neural net inference to compute θ with fixed τ). Specifically, we alternate between the following two operations for N iterations or until convergence,

$$\theta^i = \text{NN}_\phi(x_{\text{init}}, o, \tau^{i-1}), \tag{9}$$

$$\tau^i = \text{MPC-m}_{\theta^i}(x_{\text{init}}, \tau^{i-1}), \tag{10}$$

where MPC-m performs m solver iterations using the AL algorithm, with the most recent estimate θ^i from the network and warm-started using τ^{i-1} from the last MPC-m solve. The initial value τ^0 are initialized at x_{init} and zero controls across time steps. We refer to each alternating step as a DEQ-MPC-iteration, with the super-script, i, denoting the iteration count. This is illustrated in figure 1. This iterative inference/optimization approach enables the network to provide an initial coarse θ estimate and iteratively refine it based on the solver's progress.

Choice of N, m: Empirically, we find that updating the MPC inputs θ every two AL iterations ($m = 2$) is sufficient to obtain most of the gains. Furthermore, DEQ-MPC typically converges within $N = 6$ DEQ-MPC-iterations with $m = 2$ and thus we use these values for all our experiments. We discuss the considerations around the convergence of this alternating problem in section A.2.

Network architecture. We explore two architectural choices for NN_ϕ with distinct trade-offs:

(1) *DEQ-MPC-NN:* NN_ϕ is a standard feedforward network. While this is simple and often effective, it has limitations. The iterative nature of the DEQ-MPC framework can lead to instabilities when using a standard feedforward architecture, particularly in complex settings. Moreover, this architecture is somewhat computationally inefficient, as it doesn't leverage the similarity of computations across successive iterations – each iteration starts anew without reusing previous computational results.

(2) *DEQ-MPC-DEQ:* NN_ϕ itself is a DEQ network [32]. Specifically, the network inference step in equation 9 is itself computed via an inner fixed-point solve: $z_i^\star = d_\phi(z_i^\star, x_{\text{init}}, o, \tau_{i-1})$. followed by $\theta_i = g_\phi(z_i^\star)$. Note that this fixed point solve is distinct from the equilibrium computations in the DEQ-MPC-iterations discussed earlier and is simply computing the network inference (i.e constraint projections) from equation 9. Furthermore, we can also warm-start these inner fixed points across successive DEQ-MPC-iterations given they are likely to be similar, i.e, z_i can be conveniently initialized with z_{i-1} while computing the fixed points. This allows us to re-use the network computation from earlier iterations.

MPC-m solver. We implement an Augmented Lagrangian (AL) solver [39, 40] for MPC-m, chosen for its robustness to non-linear constraints (handled as penalties) and suitability for warm-starting. The penalty-based approach also allows us to use the unconverged iterations as smoothed/relaxed versions of the problem to handle discontinuities (more discussion in section 4.2.2). Our solver implementation is friendly with both CPU and GPU.

Specifically, for the general MPC problem in equation 1, we form the following Lagrangian

$$\mathcal{L}(\tau, \lambda, \eta, \mu) = \sum_t C_{\theta,t}(\tau_t) + \lambda^T h_\theta(\tau) + \eta^T k_\theta(\tau_t, x_{t+1}) + \frac{\mu}{2}\|h_\theta(\tau_t)^+\|_2^2 + \frac{\mu}{2}\|k_\theta(\tau_t, x_{t+1})\|_2^2, \tag{11}$$

where $h_\theta(\tau_t) \leq 0$ are the inequality constraints and $k_\theta(\tau_t, x_{t+1}) = 0$ are all the equality constraints, λ and η are the corresponding Lagrange multipliers and $\mu > 0$ the penalty parameter. $h_\theta(\tau_t)^+$ represents an element-wise clipping at zero $\max(0, h_\theta(\tau_t))$. The details of the AL method are described in algorithm 1 in the appendix.

Moreover, with MPC-m, we only perform m AL iterations per DEQ-MPC-iteration i, initializing all the AL variables (τ^i, λ^i, η^i, μ^i) at iteration i from the state at the end of iteration $i - 1$.

4.2.2 Loss and Gradients

Augmented Lagrangian gradients. Previous work [24, 25] has shown that computing gradients through optimization problems can be problematic due to inherent discontinuities in the landscape and have proposed various relaxations to mitigate this problem. We take inspiration from these approaches and propose a relaxation for use with our solver.

We compute input gradients ∇_θ for the AL solver by applying IFT (Eq. 2) on the Lagrangian :

$$\tau^* = -(\nabla_\tau^2 \mathcal{L})^{-1} \nabla_{\tau\theta}\mathcal{L} = -(Q + \mu A^T A + \mu G^T G)^{-1} \nabla_{\tau\theta}\mathcal{L}. \tag{12}$$

where, A and G are the constraint Jacobians of the equality and inequality constraints respectively. At convergence, the value of μ is very high. This results in the components of the gradient in the column space of the linearized active constraints getting squished to zero. Thus, when the constraints are non-linear/discontinuous, and the optimizer converges to some arbitrary active sets, the gradients computed using equation 12 are also arbitrary/meaningless. To address this, we compute losses on multiple intermediate iterates $\tau^i \forall i \in [1, N]$ from the DEQ-MPC iterations, not just the final one τ^N. Gradients computed through earlier iterates (with smaller effective μ) provide smoother signals, acting as a relaxation. Later iterates (with larger μ) provide more accurate gradients as the solver converges. This creates a natural curriculum during training. We discuss the details of gradient computation for the DEQ network in the appendix.

Losses. We supervise the policy outputs with the expert trajectories for the following T steps. We use an L1 loss over the output states against the corresponding ground truths for supervision. As discussed before, we compute losses on multiple intermediate iterates and backpropagate gradients through all of them. The resulting objective for a single instance is

$$\ell(x_{0:T}^{\text{exp}}, x_{0:T}^{1:I}) = \sum_{t=0:T} \sum_{j=1:I} \|x_t^{\text{exp}} - x_t^i\|_1, \tag{13}$$

where $x_{0:T}^{\text{exp}}$ are expert demonstrations and $x_{0:T}^{1:I}$ are the states output by the model across I iterations.

4.2.3 Warm-Starting and Streaming

Warm-starting. Warm-starting MPC by initializing the current solve with the previous time step's solution is vital for efficiency [41, 37, 42]. Standard MPC warm-starting involves shifting the previous solution $\hat{\tau}_{t-1:T+t-1}$ to initialize the solve for $\tau_{t:T+t}$. The AL method accommodates this very elegantly. We initialize τ with the shifted estimate $\tau_{t:T+t} = [\hat{\tau}_{t:T+t-1}, \hat{\tau}_{T+t-1}]$, where $\hat{\tau}_{T+t-1}$ is assumed to be a reasonable estimate for τ_{T+t}, reset the dual variables λ and η to zeros and set the initial value of $\rho = \rho_{\text{max}}/10^{N*m-i}$ where $(N*m-i)$ is the total number of AL iterations we expect to perform after warm-starting (see Appendix A.5 for additional explanations).

In standard differentiable-MPC setups, the network infers the MPC solver inputs θ afresh at each successive time step. These estimates can often be arbitrarily far from the previous estimates, thus requiring a significant number of AL iterations post warm-start. On the other hand, in DEQ-MPC, the network is conditioned on the previous optimizer iterate. This allows us to train the network to predict consistent θ estimates across time-steps by training it specifically for the streaming setting as described below.

Streaming training. We customize the training procedure to suit the warm-started streaming setup. Given a sampled ground truth trajectory $\tau_{0:T+L}^{\text{exp}}$, we break the inference problem into a two step process. First, we solve for $\tau_{0:T}$ given x_0^{exp} as usual without any warm-starting. Then, we successively solve L problems for $\tau_{t:T+t}$ for $t = 1 \ldots N$ with the iterates warm-started with solution from the previous solve, $\tau_{t-1:T+t-1}$. Then we simply compute losses on all the intermediate optimization iterates (from both steps) and supervise them using the corresponding ground truths as described in section 4.2.2. For all of our experiments we use $L = 2$.

5 Experiments

We demonstrate the effectiveness of our proposed modifications across a variety of simulated robotic control tasks. Additionally, we present ablation studies to highlight the specific advantages of DEQ-MPC regarding representation, training stability, and warm-starting. Finally, we validate our approach with hardware experiments on a Crazyflie drone.

Setup. We use the trajectory prediction and tracking problem (section 4.2.1) as our default experimental setting. For each task, we generate ground truth trajectories using expert policies; details on dataset generation and partitioning are in Appendix.Models are trained via supervised learning to predict the next T steps (T=5 unless specified) given the current state. We evaluate models using validation error (for general optimization layer effectiveness) and average returns over 200 receding-horizon rollouts (for MPC policy performance).

Variants/Baselines. Throughout the experiments, we compare our methods (DEQ-MPC-*) against their corresponding differentiable MPC counterparts (Diff-MPC-*):

DEQ-MPC-DEQ: Our method using a DEQ network architecture.

DEQ-MPC-NN: Our method using a standard feedforward network.

Diff-MPC-NN: Standard differentiable MPC with a feedforward network predicting θ in one shot, solved via MPC, with loss on the converged iterate via IFT.

Diff-MPC-DEQ: Same as Diff-MPC-NN but using a DEQ network architecture.

Network architecture. Given the sequential nature of the task, we use a temporal convolution-based architecture for both feedforward (NN) and DEQ networks. Details are in Appendix.

5.1 Comparison Results

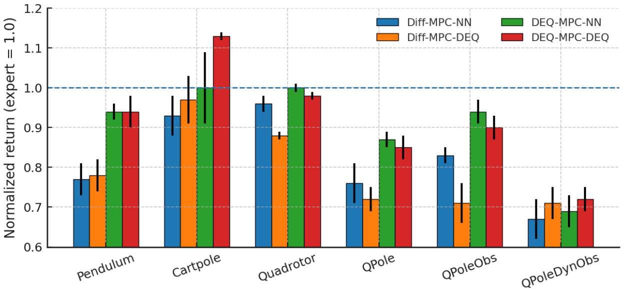


Figure 2: Performance comparison across various simulated environments, with values normalized against the expert return for each environment. Higher score is better.

We evaluate the methods on a series of underactuated continuous control tasks with constraints: *Pendulum*, *Cartpole*, *Quadrotor*, *QPole*, *QPoleObs*, and *QPoleDynObs*. QPole is a quadrotor with a pole hanging (task is to swing up the pole while reaching a goal). *QPoleObs* adds obstacles in the environment that need to be avoided (using obstacle avoidance constraints in MPC). *QPoleDynObs* makes these obstacles dynamic. Detailed descriptions of envs is provided in A.6.3.

The policies are trained and executed in the streaming setting (section 4.2.3) with a single DEQ-MPC-iteration (DEQ-MPC variants)/two AL iterations (Diff-MPC variants) with warm-starting across environments, except in the QPoleObs env, where all methods needed two DEQ-MPC-iterations (DEQ-MPC)/four AL iterations (Diff-MPC). (Note that each DEQ-MPC iteration itself also does exactly two AL iterations with $m = 2$). Figure 2 shows the normalized returns obtained by each policy for each task averaged across policies trained with three dataset splits. The returns are normalized against the expert policy (1.00). We observe that the DEQ-MPC variants consistently perform better than the Diff-MPC counterparts across most environments. While DEQ-MPC-DEQ performs consistently well across all environments, we observed that DEQ-MPC-NN occasionally got unstable (e.g. resulting in its sub-par performance in the Cartpole balancing task).

5.2 Ablations

We explore three aspects: representation capabilities, training stability, and warm-startability, primarily using the QPole environment unless specified.

5.2.1 Representation Ablations

We demonstrate DEQ-MPC's enhanced representation capabilities. First, DEQ-MPC variants scale more effectively with dataset size and model capacity. Second, they show less performance degradation as constraint complexity increases. Additional representation ablations are in Appendix.

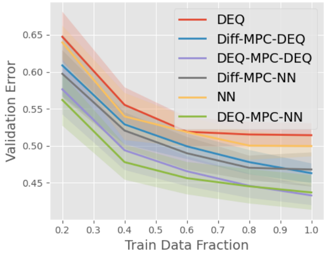


Figure 3: Generalization

Generalization. Figure 3 shows validation error versus training set size. DEQ-MPC models show benefits even with smaller datasets and continue improving with more data, unlike Diff-MPC variants which saturate. This suggests better representation power. Networks trained without MPC layers (DEQ, NN) also saturate, indicating benefits arise from solver-network interplay.

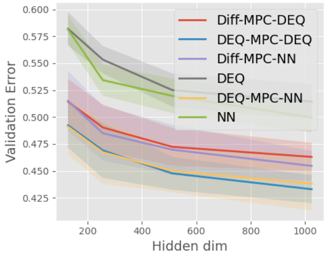


Figure 4: Network capacity

Network capacity. Figure 4 shows validation error versus network hidden state size (128 to 1024). We observe that the DEQ-MPC variants benefit more from the higher network capacity than the Diff-MPC variants which saturate beyond hidden size of 512. This shows that the DEQ-MPC variants are better able to utilize additional model capacity and thus are more amenable to scaling.

Constraint hardness. We add 40 obstacles to QPole with collision avoidance constraints ($\|x_d - x_o\|_2^2 \geq r^2$) added to MPC, where x_d = COM of drone and x_o = COM of obstacle. Figure 5 shows the returns obtained by different models as we vary the obstacle radius r from 0.20 to 0.50. DEQ-MPC's performance advantage persists as we add additional constraints and even increases as constraints become harder (larger r). (Note: These runs are without warm-starting to isolate representational effects).

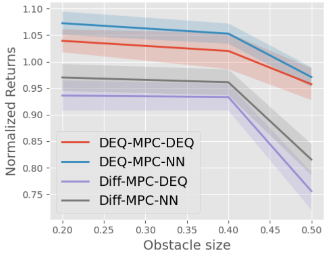


Figure 5: Constraints hardness

5.2.2 Training Stability

Gradient niceness. Figure 6 presents the validation errors during training for DEQ-MPC-DEQ (where we compute losses across multiple intermediate AL iterates and backpropagate) and Diff-MPC-DEQ (where gradients are computed only with the final AL iterate). Diff-MPC-DEQ shows significant instability when tight control limits are added (postfix 1), unlike DEQ-MPC-DEQ. Both are stable without these constraints (postfix 0). This highlights the benefit of using intermediate iterates for smoother gradients.

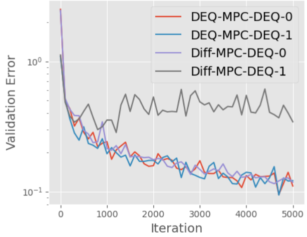


Figure 6: Gradient instability

MPC input sensitivity ∇_θ. Figure 7 shows validation errors as we vary velocity coefficients in the MPC cost Q (lower values → worse conditioning → higher problem sensitivity). We observe that this leads to more training instability in models. The validation errors plotted represent the 'best' performance of the model throughout training (typically just before the training became unstable). DEQ-MPC-DEQ remains stable for the largest range of values. Even DEQ-MPC-NN, although best performing with well conditioned Q, quickly gets very unstable as conditioning worsens.

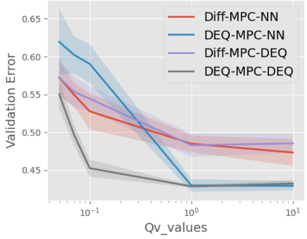


Figure 7: Cost parameter sensitivity

5.2.3 Warm-Starting Ablations

Figure 8 shows returns as we vary the number of DEQ-MPC/AL iterations allowed per step in the streaming/warm-started evaluation (section 4.2.3, $L = 2$). Note that, each DEQ-MPC iteration does exactly two AL iterations ($m = 2$). As the computational budget (iterations per step) decreases, the performance gap between DEQ-MPC and Diff-MPC widens significantly. DEQ-MPC's iterative nature makes it inherently better suited to leverage warm-starting effectively, given that the warm-starting required at each new timestep is very similar to the warm-starts done across DEQ-MPC iterations.

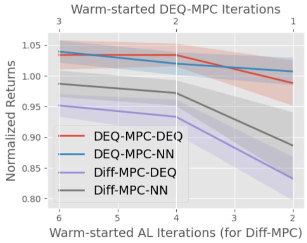


Figure 8: Warm-starting

5.3 Hardware Experiments

Table 1: Hardware results

Method	Average Return	Failure Rate (%)
Diff-MPC-NN	0.82 (±0.15)	33.33
Diff-MPC-DEQ	0.86 (±0.12)	33.33
DEQ-MPC-NN	**0.93 (±0.03)**	33.33
DEQ-MPC-DEQ	**0.93 (±0.04)**	**0.0**

We further validate our approach on the Crazyflie nanoquadrotor platform, deploying policies trained in simulation for navigation to the origin amidst 40 virtual static obstacles. The policies were evaluated over 3 trials on same robot with different initializations. Table 1 summarizes the hardware results, which corroborate simulation findings. DEQ-MPC variants achieved significantly higher returns with zero failures/collisions, outperforming the less reliable Diff-MPC policies. More details are in Appendix.

6 Discussions

Our experimental results highlight several key advantages of DEQ-MPC over differentiable MPC layers. The performance gap between DEQ-MPC variants and Diff-MPC becomes increasingly apparent as task complexity increases, whether through harder constraints, longer planning horizons, or increased problem sensitivity. A particularly promising aspect of DEQ-MPC is its favorable scaling behavior. Unlike Diff-MPC variants which show signs of performance saturation, DEQ-MPC models continue to improve with increasing dataset size and network capacity. This suggests potential for exploiting scaling laws in robotics applications. Furthermore, DEQ-MPC's effectiveness in warm-starting scenarios, requiring fewer augmented Lagrangian iterations while maintaining performance, offers significant practical advantages for real-world deployment. This advantage was also evident in our hardware experiments, where DEQ-MPC methods demonstrated superior reliability. Interestingly, there exist trade-offs even between the DEQ-MPC variants. While DEQ-MPC-NN performs slightly better on average in simulation, DEQ-MPC-DEQ remains stable across a wider range of conditions compared to DEQ-MPC-NN.

7 Limitations and Future Work

Several important directions remain for future work. While our method is designed to be general, our current evaluation focuses primarily on trajectory tracking and prediction problems. Exploring the applicability of DEQ-MPC to a broader class of MPC and constrained optimization problems, both within and beyond robotics, would be valuable. Additionally, we have primarily only tested the models in the imitation learning setting in this paper. Their applicability to a broader class of reinforcement learning problems is less clear given the complicated architecture could result in higher variance during training. Investigating whether the representational richness of DEQ-MPC can be leveraged effectively beyond the imitation learning setup such as in reinforcement learning settings to directly learn constrained optimal policies could be a promising line of future work. Moreover, our current MPC implementation currently only handles relatively continuous dynamics and constraints. Dealing with more complicated dynamics such as contacts would require a more careful MPC implementation which we leave for future work. Finally, given the strong performance in constraint handling, exploring DEQ-MPC in safety-critical scenarios such as human-robot interaction settings would be an interesting direction for future research.

Acknowledgments

SG was supported by a grant from the Bosch Center for Artificial Intelligence. We would like to thank John Zhang, Jon Arrizabalaga, Giri Anantharaman and numerous others for various brainstorming sessions and feedback during various stages of the project. We would also like to thank the authors of Differentiable MPC [2] for open sourcing their code and making it easy to use. Lastly, we'd like to thank the CoRL and ICLR conference reviewers for providing valuable feedback throughout the reviewing process and helping us improve the presentation as well as fill in critical holes in the paper.

References

[1] A. Gupta, A. Pacchiano, Y. Zhai, S. Kakade, and S. Levine. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity. *Advances in Neural Information Processing Systems*, 35:15281–15295, 2022.

[2] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

[3] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato. Learning convex optimization control policies. In *Learning for Dynamics and Control*, pages 361–373. PMLR, 2020.

[4] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431, 2019. doi:10.23919/ECC.2019.8796030.

[5] L. Schott, J. Delas, H. Hajri, E. Gherbi, R. Yaich, N. Boulahia-Cuppens, F. Cuppens, and S. Lamprier. Robust deep reinforcement learning through adversarial attacks and training: A survey. *arXiv preprint arXiv:2403.00420*, 2024.

[6] X. Chen, J. Hu, C. Jin, L. Li, and L. Wang. Understanding domain randomization for sim-to-real transfer. *arXiv preprint arXiv:2110.03239*, 2021.

[7] J. Shrestha, S. Idoko, B. Sharma, and A. K. Singh. End-to-end learning of behavioural inputs for autonomous driving in dense traffic. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10020–10027. IEEE, 2023.

[8] X. Xiao, T. Zhang, K. Choromanski, E. Lee, A. Francis, J. Varley, S. Tu, S. Singh, P. Xu, F. Xia, et al. Learning model predictive controllers with real-time attention for real-world navigation. *arXiv preprint arXiv:2209.10780*, 2022.

[9] C. Diehl, T. Klosek, M. Krüger, N. Murzyn, and T. Bertram. On a connection between differential games, optimal control, and energy-based models for multi-agent interactions. *arXiv preprint arXiv:2308.16539*, 2023.

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

[11] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

[12] S. Gould, R. Hartley, and D. Campbell. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):3988–4004, 2021.

[13] B. Landry, Z. Manchester, and M. Pavone. A differentiable augmented lagrangian method for bilevel nonlinear optimization. *arXiv preprint arXiv:1902.03319*, 2019.

[14] W. Jin, Z. Wang, Z. Yang, and S. Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. *Advances in Neural Information Processing Systems*, 33: 7979–7992, 2020.

[15] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, et al. Theseus: A library for differentiable nonlinear optimization. *Advances in Neural Information Processing Systems*, 35:3801–3818, 2022.

[16] B. Yi, M. A. Lee, A. Kloss, R. Martín-Martín, and J. Bohg. Differentiable factor graph optimization for learning smoothers. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1339–1345. IEEE, 2021.

[17] Z. Teed, L. Lipson, and J. Deng. Deep patch visual odometry. *Advances in Neural Information Processing Systems*, 2023.

[18] L. Lipson, Z. Teed, A. Goyal, and J. Deng. Coupled iterative refinement for 6d multi-object pose estimation. 2022.

[19] M. Bhardwaj, B. Boots, and M. Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 10598–10604. IEEE, 2020.

[20] C. Diehl, T. Klosek, M. Krueger, N. Murzyn, T. Osterburg, and T. Bertram. Energy-based potential games for joint motion forecasting and control. In *Conference on Robot Learning*, pages 3112–3141. PMLR, 2023.

[21] A. Romero, Y. Song, and D. Scaramuzza. Actor-critic model predictive control. *arXiv preprint arXiv:2306.09852*, 2023.

[22] W. Wan, Y. Wang, Z. M. Erickson, and D. Held. Difftop: Differentiable trajectory optimization for deep reinforcement and imitation learning. *ArXiv*, abs/2402.05421, 2024. URL https://api.semanticscholar.org/CorpusID:267548058.

[23] T. A. Howell, K. Tracy, S. Le Cleac'h, and Z. Manchester. Calipso: A differentiable solver for trajectory optimization with conic and complementarity constraints. In *The International Symposium of Robotics Research*, pages 504–521. Springer, 2022.

[24] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

[25] R. Antonova, J. Yang, K. M. Jatavallabhula, and J. Bohg. Rethinking optimization with differentiable simulation from a global perspective. In *Conference on Robot Learning*, pages 276–286. PMLR, 2023.

[26] S. Gurumurthy, K. Ram, B. Chen, Z. Manchester, and Z. Kolter. From variance to veracity: Unbundling and mitigating gradient variance in differentiable bundle adjustment layers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27507–27516, 2024.

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

[28] P. L. Donti, D. Rolnick, and J. Z. Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=V1ZHVxJ6dSS.

[29] S. Gurumurthy, J. Z. Kolter, and Z. Manchester. Deep off-policy iterative learning control. In *Learning for Dynamics and Control Conference*, pages 639–652. PMLR, 2023.

[30] S. Gurumurthy, S. Bai, Z. Manchester, and J. Z. Kolter. Joint inference and input optimization in equilibrium networks. *Advances in Neural Information Processing Systems*, 34:16818–16832, 2021.

[31] Z. Teed and J. Deng. DROID-SLAM: Deep visual SLAM for monocular, stereo, and RGB-d cameras. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ZBfUo_dr4H.

[32] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

[33] L. T. Biegler and V. M. Zavala. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33 (3):575–582, 2009.

[34] P. E. Gill, W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[35] L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff. Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 9217–9224, 2022.

[36] P. Huang, M. Xu, F. Fang, and D. Zhao. Robust reinforcement learning as a stackelberg game via adaptively-regularized adversarial training. *arXiv preprint arXiv:2202.09514*, 2022.

[37] S. Le Cleac'h, T. A. Howell, S. Yang, C.-Y. Lee, J. Zhang, A. Bishop, M. Schwager, and Z. Manchester. Fast contact-implicit model predictive control. *IEEE Transactions on Robotics*, 2024.

[38] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, July 2011. ISSN 1935-8237, 1935-8245. doi:10.1561/2200000016. URL https://www.nowpublishers.com/article/Details/MAL-016. Publisher: Now Publishers, Inc.

[39] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[40] M. Toussaint. A novel augmented lagrangian approach for inequalities and convergent any-time non-central updates. *arXiv preprint arXiv:1412.4329*, 2014.

[41] T. A. Howell, B. E. Jackson, and Z. Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679. IEEE, 2019.

[42] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester. Tinympc: Model-predictive control on resource-constrained microcontrollers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2024.

[43] E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. *Advances in neural information processing systems*, 33:10718–10728, 2020.

[44] S. Bai, V. Koltun, and Z. Kolter. Stabilizing equilibrium models by jacobian regularization. In *Proceedings of the International Conference on Machine Learning*, pages 554–565, 2021.

[45] E. K. Ryu and S. Boyd. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1): 3–43, 2016.

[46] Z. Geng, X.-Y. Zhang, S. Bai, Y. Wang, and Z. Lin. On training implicit models. *Advances in Neural Information Processing Systems*, 34:24247–24260, 2021.

[47] S. W. Fung, H. Heaton, Q. Li, D. Mckenzie, S. J. Osher, and W. Yin. Jfb: Jacobian-free backpropagation for implicit networks. In *AAAI Conference on Artificial Intelligence*, 2021. URL https://api.semanticscholar.org/CorpusID:238198721.

[48] S. Gurumurthy, Z. Manchester, and J. Z. Kolter. Practical critic gradient based actor critic for on-policy reinforcement learning. In *5th Annual Learning for Dynamics & Control Conference*, 2023. URL https://openreview.net/forum?id=ddl_4qQKFmY.

[49] B. E. Jackson, J. H. Lee, K. Tracy, and Z. Manchester. Data-efficient model learning for control with jacobian-regularized dynamic-mode decomposition. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=ED0G14V3WeH.

[50] D. G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

[51] H. F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

A Appendix

A.1 Additional Ablations

We provide more ablation experiments to demonstrate the representational benefits of DEQ-MPC.

Representational hardness. We look at the effect of increasing horizon length as it serves as a good proxy for various metrics such as problem conditioning, dimensionality, practical utility etc. Specifically, figure 9 shows the validation errors obtained by each model after training as we vary the horizon length from $T = 3$ to $T = 12$. We observe that the gap between the validation errors of the iterative models and the non-iterative ones is preserved even as we increase the size of the problem. Further, we observe that the representational benefits of the DEQ network in DEQ-MPC-DEQ starts becoming more obvious in the longer horizon problems as the difference in validation error between DEQ-MPC-DEQ and DEQ-MPC-NN increases. This illustrates the effectiveness of the infinite depth in DEQs helping with capturing the longer context.

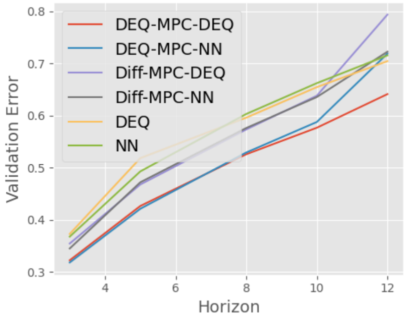


Figure 9: Time horizon ablations

Validation error with iteration count. Figure 10 shows the validation error across the Augmented Lagrangian iterations. As discussed earlier, the Diff-MPC variants here use the same predicted θ throughout iterations while the DEQ-MPC variants use ADMM and thus update the optimization inputs θ using the network inference every two AL iterations. Interestingly, the gap in validation error starts accruing from the early AL iterations itself. But gap gets pronounced after the fourth AL iteration as the Diff-MPC variants saturate while DEQ-MPC continues to improve thanks to the repeated updates to the problem inputs.

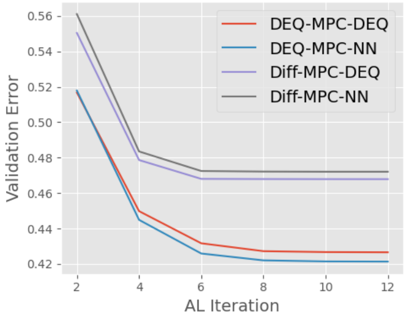
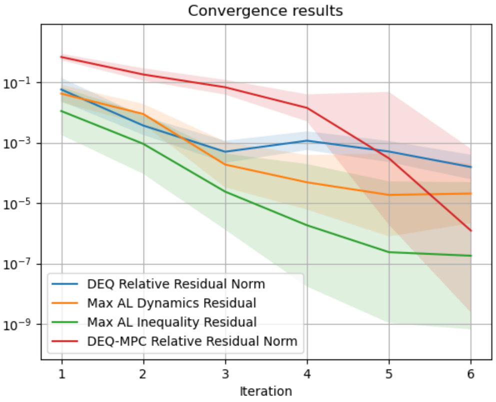


Figure 10: AL iteration ablations

A.2 Notes on Convergence



Our treatment of the joint system as a DEQ allows us to borrow results from [43][44] to ensure convergence of the fixed point iteration. Specifically, if we assume the joint Jacobian of the ADMM fixed point iteration is strongly monotone with smoothness parameter m and Lipschitz constant L, then by standard arguments (see e.g., Section 5.1 of [45]), the fixed point iteration with step size $\alpha < m/L^2$ will converge. However, going from the strong monotonicity assumption on the joint fixed point iterations to specific assumptions on the network or the optimization problem is less straightforward. But, in practice a wide suite of techniques have been used to ensure that such fixed points exist and can be found using relatively few fixed point iterations.

In fact, for all of our experiments, we that the problem converges within 6 ADMM iterations once trained. As an example, we show the convergence behavior of the joint optimization problem of the trained DEQ-MPC-DEQ model in the QuadPole-obstacle avoidance environment in Figure A.2. We find that the maximum constraint residuals for both equality (dynamics) and inequality (obstacle avoidance and control limits) constraints, as well as the relative fixed-point residuals of the DEQ network and the DEQ-MPC iterates converge to $< 1\text{e}{-3}$ tolerance within 6 DEQ-MPC iterations. We observe similar plots for the other systems we tested as well with the model often converging faster in the easier systems.

In fact, across all our experiments, we barely faced any issues with convergence of the optimization problem itself as long as we followed the best practices while designing the network architectures (used appropriate skip connections and normalization layers as is common in DEQ architectures [32, 44, 46, 31]) and the AL solver (using a merit function-based line search for monotonic convergence and appropriate solver initializations as described earlier). However, when dealing with more complicated (more ill-conditioned/non-convex) problems, it is conceivable that we might face convergence issues. In such cases, we might advise increasing the number of inner AL iterations as well as outer DEQ-MPC iterations to ensure convergence but we did not face those issues at least for the problems we tested.

A.3 Computational Cost and Inference Time

We'd like to stress that the inference times during the MPC rollouts are very similar across our models and the baselines given that all of them perform one network inference and one inner optimization loop (2 AL iterations). For instance, the total inference times for our DEQ-MPC-DEQ (86.6ms) and the Diff-MPC-DEQ baseline (88.8ms) are comparable on our machine. Similarly, the NN variants, DEQ-MPC-NN (71.2ms) and Diff-MPC-NN (61.6ms), also show similar timings. The solver clearly consumes the majority of the time (72ms-75ms for DEQ variants and 58-68 for NN variants). But the solver code is currently a PyTorch implementation and not optimized for speed. Production solver implementations can be made much faster but require significant systems effort which we leave for future work given that is not the focus of our work. However, even with our implementation, we're able to run the policies at > 10 Hz frequency as mentioned in our hardware experiments. The additional time consumed by the solver is often worth it in some applications given that it gives additional flexibility in handling distribution shifts, online problem specification changes and constraints especially in safety critical domains, etc.

A.4 DEQ Network Gradients

Computing gradients through the fixed point iteration in a DEQ model typically requires using the implicit function theorem equation 2, which involves computing a linear system solve. However, recent work [46, 47] has shown that the approximations of the gradient by simply assuming an identity Jacobian or differentiating through the last few iterations of the fixed point iteration using vanilla backpropagation is equally/more effective while being computationally cheaper. We adopt this approach. Specifically, we run the function a couple more times after computing the fixed point, and simply backpropagate through those last couple of iterations to compute the parameter gradients.

A.5 Augmented Lagrangian Algorithm

Specifically, given the general MPC problem in equation 1, we form the following Lagrangian

$$\mathcal{L}(\tau, \lambda, \eta, \mu) = \sum_t C_{\theta,t}(\tau_t) + \lambda^T h_\theta(\tau) + \eta^T k_\theta(\tau_t, x_{t+1}) + \frac{\mu}{2}\|h_\theta(\tau_t)^+\|_2^2 + \frac{\mu}{2}\|k_\theta(\tau_t, x_{t+1})\|_2^2, \quad (14)$$

where $h_\theta(\tau_t) \leq 0$ are the inequality constraints and $k_\theta(\tau_t, x_{t+1}) = 0$ are all the equality constraints (including the dynamics and initial state constraints), λ and η are the corresponding Lagrange multipliers and $\mu > 0$ is the penalty parameter. $h_\theta(\tau_t)^+$ represents an element-wise clipping at zero $\max(0, h_\theta(\tau_t))$. The augmented Lagrangian method then proceeds by alternating between updating the primal variables (τ), dual variables (λ, η) and the penalty parameter (μ) as shown in algorithm 1.

Warm starting notes. We discussed in the warm-starting section that we warm-start the primal variables τ but not the dual variables. We instead reset them to zeros. Dual variables are tricky to warm-start in the AL-MPC setup for two reasons : (1) the active sets often change across consecutive time-steps especially when the optimized state at t + 1 differs significantly from the next observation. This becomes prominent when deploying the policies in the real world due to the sim-2-real gap. (2) When warm-starting AL, the penalty ρ is often re-initialized to a smaller value which means that the relative scale of the multiplier updates $\lambda := \lambda + \rho * res$ are much smaller than the multiplier magnitudes making the initial updates ineffective. Thus, it's generally considered a safer practice to reset the multiplier values instead of warm-starting them.

Algorithm 1 Augmented Lagrangian Solver for MPC-m

```
Require: Initialize $\tau^0$, $\lambda^0$, $\eta^0$, $\mu^0$ (warm-started using previous DEQ-MPC-iteration, parameters), $\gamma > 1$
Set $j = 0$
repeat
    Primal update: Solve the unconstrained minimization problem using the Gauss-Newton method
        $$\tau^{j+1} = \arg\min_{\tau} \mathcal{L}(\tau, \lambda^j, \eta^j, \mu^j)$$
    Dual update: Update the Lagrange multipliers
        $$\lambda^{j+1} = \max(\lambda^j + \mu^j h_\theta(\tau^{j+1}), 0)$$
        $$\eta^{j+1} = \eta^j + \mu^j k_\theta(\tau^{j+1})$$
    Penalty update: Update the penalty parameter
        $$\mu^{j+1} = \gamma \mu^j$$
    $j = j + 1$
until Stopping criterion is met (or $j = m$ iterations)
return Final solution $\tau^m$, $\lambda^m$, $\eta^m$, $\mu^m$
```

A.6 Experimental Setup and Environment Details

This section provides further details on the experimental methodology, including dataset generation, partitioning, model training objectives, evaluation procedures, and comprehensive descriptions of the continuous control environments used in our study.

A.6.1 Dataset Generation, Partitioning, and Training

Trajectory generation. For each experimental task, we first generated a dataset of ground truth trajectories. These trajectories represent near-optimal solutions to the respective control problems. They were generated using 'expert' policies trained via CGAC [48], a state-of-the-art on-policy reinforcement learning algorithm. This ensures that the supervised learning models are trained on high-quality, dynamically feasible motion data relevant to the task.

Data partitioning. The generated trajectory data for each environment was partitioned into a training set, comprising 90% of the trajectories, and a validation set, comprising the remaining 10%. This partitioning was used for both model training and hyperparameter tuning.

Supervised learning objective. The core task for the models evaluated in this paper is trajectory prediction (for agent and optionally for obstacles when dealing with dynamic obstacles). Models are trained via supervised learning to predict the next T steps of a trajectory $(s_{t+1}, ..., s_{t+T})$ given the current state s_t as input. Unless otherwise specified in the experiments, the default prediction horizon was set to $T = 5$ steps for all environments. This setup corresponds to the problem formulation discussed in Section section 4.2.1 of the main paper.

A.6.2 Evaluation Metrics

We employed two distinct methods to evaluate the performance of the compared models:

1. **Validation error:** To assess the models' general capability as optimization layers within differentiable pipelines (e.g., for use in control or system identification tasks), we measured their prediction performance on the held-out validation set. Lower validation error indicates better generalization and prediction accuracy.
2. **MPC policy performance (average returns):** To evaluate the models' suitability specifically for the control setting, we integrated them as feedback policies within the simulation environments or real system. Using a receding horizon control approach, the policy derived from the model selects actions at each step. The performance was quantified by the average cumulative reward (average return) achieved over 200 independent rollouts (episodes) for each task. Higher average returns indicate better closed-loop control performance.

A.6.3 Environment Descriptions

We evaluate the proposed methods on a series of underactuated continuous control tasks with state and control constraints. These environments are commonly used benchmarks in control and reinforcement learning research.

Pendulum:

- **Task:** The standard pendulum swing-up task, where the goal is to swing an underactuated pendulum to an upright position and maintain it there.
- **Dynamics:** State dimension $n = 2$ (angle, angular velocity), control dimension $m = 1$ (torque).
- **Constraints:** Control input (torque) is limited to ± 5 units. These are modeled as inequality constraints ($|u| \leq 5$) in the MPC formulation.
- **Dataset Size:** 300 trajectories.

Cartpole:

- **Task:** The standard cartpole swing-up task. The objective is to swing the pole attached to a cart via an unactuated joint to an upright position and keep it balanced while controlling the cart's horizontal force.
- **Dynamics:** State dimension $n = 4$ (cart position, cart velocity, pole angle, pole angular velocity), control dimension $m = 1$ (horizontal force on the cart).
- **Constraints:** Control input (force) is limited to ± 100 units. Modeled as inequality constraints ($|u| \leq 100$) in the MPC formulation.
- **Dataset Size:** 300 trajectories.

Quadrotor:

- **Task:** Navigate a quadrotor drone from a randomly initialized position and orientation to the origin $(0, 0, 0)$ in 3D space. The model is based on the dynamics described in [49].
- **Dynamics:** State dimension $n = 12$ (position, orientation (quaternion), linear velocity, angular velocity), control dimension $m = 4$ (motor thrusts).
- **Constraints:** The thrust for each of the four motors is constrained to the range $[11.5, 18.3]$ units. These are formulated as inequality constraints ($11.5 \leq u_i \leq 18.3$ for $i = 1, \ldots, 4$) in the MPC problem.
- **Dataset Size:** 2000 trajectories.

QPole (Quadrotor-Pole):

- **Task:** This environment increases the complexity of the *Quadrotor* task by attaching a free-rotating pole to the center of mass (COM) of the quadrotor. The goal is to guide the quadrotor to the origin while simultaneously ensuring the attached pole is swung up to an upright position. This task is highly dynamic and challenging due to the coupled dynamics.

- **Dynamics:** State dimension $n = 14$ (Quadrotor state + pole angles and angular velocities), control dimension $m = 4$ (motor thrusts).
- **Constraints:** Same control limits as the *Quadrotor* environment ($u_i \in [11.5, 18.3]$).
- **Dataset Size:** 2000 trajectories.

QPoleObs (Quadrotor-Pole with Obstacles):

- **Task:** Builds upon the *QPole* task by introducing 40 static spherical obstacles into the environment. The quadrotor must reach the origin with the pole swung up, while also avoiding collisions with all obstacles.
- **Dynamics:** State dimension $n = 14$, control dimension $m = 4$.
- **Constraints:** Includes the motor thrust constraints ($u_i \in [11.5, 18.3]$) and collision avoidance constraints. Collision avoidance is enforced by requiring the quadrotor's COM position x_d to maintain a minimum distance r from the center of each obstacle $x_{o,i}$. These are modeled as inequality constraints in the MPC layer: $(\|x_d - x_{o,i}\|_2^2 \geq r^2)$ for all $i = 1, \ldots, 40$. By default, the minimum safe radius is $r = 0.5$ units, unless specified otherwise.
- **Dataset Size:** 2000 trajectories.

QPoleDynObs (Quadrotor-Pole with Adversarial Dynamic Obstacles):

- **Task:** Further increases the complexity of the *QPoleObs* task by making the 40 obstacles dynamic (i.e., they move over time according to predefined or reactive trajectories). The objective remains to guide the quadrotor to the origin with the pole swung up, while dynamically avoiding collisions with the moving obstacles.
- **Dynamics:** State dimension $n = 14$, control dimension $m = 4$. Obstacle positions $x_{o,i}(t)$ are now time-varying and adversarial. The obstacles adversarially move towards the linearly projected position of the quadrotor in 5 time steps.
- **Constraints:** Includes the motor thrust constraints ($u_i \in [11.5, 18.3]$). The collision avoidance constraints are now time-dependent, requiring $(\|x_d(t) - x_{o,i}(t)\|_2^2 \geq r^2)$ for all obstacles i at all times t during the trajectory planned by the MPC. The default minimum safe radius is $r = 0.5$ units.
- **Dataset Size:** 2000 trajectories.

A.7 Hardware Experiments

To further validate our approach, we conducted experiments on physical quadrotor hardware. We deployed policies, learned through imitation of a pre-trained Reinforcement Learning (RL) expert, onto a Crazyflie 2.1 micro-quadrotor. The task involved navigating to the origin amidst 40 virtual static obstacles. Figure 11 shows the CrazyFly flying in the real-world assuming the virtual obstacles exist and the traversed trajectory visualized in the sim overlayed with the virtual obstacles.

Simulation and policy imitation. The expert RL policy was initially trained in simulation to navigate a quadrotor from an arbitrarily initialized position to the origin in an environment with 40 obstacles of radius 10cm. We then used this expert policy to generate demonstration trajectories for imitating the DEQ-MPC and Diff-MPC variants, as described in the main paper. The virtual obstacles in the hardware experiments were represented by fixed spheres of radius=10cm, identical to those in the simulation.

Experimental setup. The DEQ-MPC or Diff-MPC model ran offboard on a ground station PC. State estimation for the Crazyflie was provided by an OptiTrack motion capture system operating at 100Hz. We used the most recent state estimate from the OptiTrack system directly, without additional filtering, to assess robustness to raw sensor data.

Control loop. The policies (DEQ-MPC or Diff-MPC) were executed in a receding horizon fashion. The system receives state estimates from the Mocap system, executes the policy on the PC, and

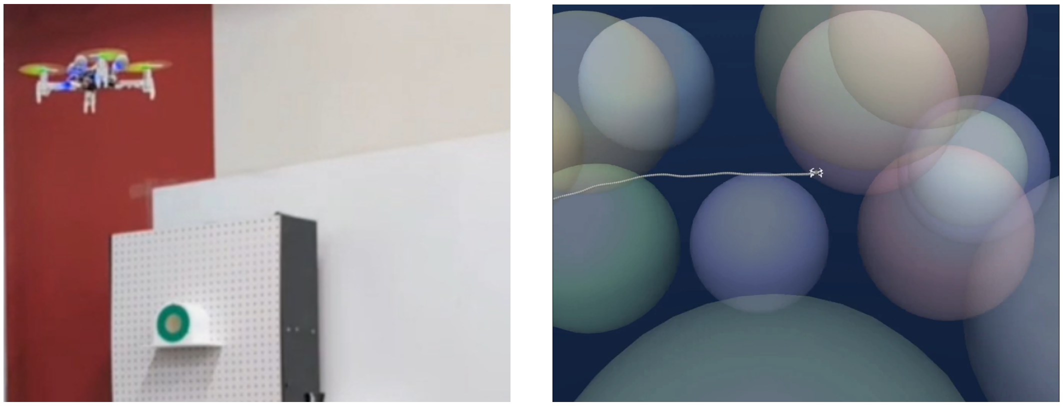


Figure 11: **Left :** The CrazyFlie flying with the DEQ-MPC-DEQ policy in the real-world assuming the virtual obstacles exist. **Right :** The CrazyFlie trajectory from the real world overlayed in the simulator with virtual obstacles.

sends the resulting next state commands to the Crazyflie's onboard tracking controller at a frequency of 10 Hz. The policy is executed (at 10Hz) in a receding horizon fashion with warm-starting from the solution at the previous time step. The prediction horizon for the policies is $T = 5$ steps as with most other sim experiments. Onboard Crazyflie tracks the specified desired state with a lower level Mellinger controller.

Evaluation protocol. Each policy variant was evaluated over 3 trials. For consistency across policy comparisons, these trials commenced from the same 3 distinct, randomly chosen initial positions of the Crazyflie. A trial was considered a ”failure” or ”collision” if the Crazyflie physically crashed or if its estimated position intersected with any of the virtual obstacles during the run. The returns for each run were computed by aggregating the rewards until the first instance of a failure or collision or until max time steps. Each policy was executed for a maximum of 200 time-steps (at a 10 Hz control rate). The average return and failure rates are reported in Table 1 in the main paper.

Observations and discussion of results. As detailed in Table 1, the hardware experiments corroborated our simulation findings. The DEQ-MPC variants (DEQ-MPC-NN and DEQ-MPC-DEQ) achieved higher average returns compared to the Diff-MPC variants. Notably, DEQ-MPC-DEQ achieved a zero failure rate, successfully completing all trials without collisions or crashes. While DEQ-MPC-NN also showed strong performance, it experienced a collision on one of the runs. We observed that the OptiTrack MoCap measurements, while generally accurate, occasionally exhibited noise and transient fluctuations. Qualitatively, the DEQ-MPC based policies, particularly DEQ-MPC-DEQ, demonstrated greater robustness to these real-world imperfections. This robustness manifested in smoother flight paths and a better ability to maintain stability and track the desired trajectory despite state estimation noise, ultimately contributing to their superior performance in terms of both higher average returns and lower (or zero) failure rates. The Diff-MPC policies appeared more susceptible to these disturbances, leading to less reliable performance and a higher incidence of failures.

A.8 Network Architecture Details

The trajectory prediction and tracking problem is inherently sequential, as the network takes the current system state as input and predicts the future T states to be tracked (and for dynamic obstacles the future T states of the obstacles to be avoided). Given this sequential nature, we employ a temporal convolution-based architecture for both the DEQ and the feedforward network used in our experiments.

Inputs. For the DEQ-MPC variants, we have 2 inputs: (x_0, the initial state and $x^i_{1:T}$, the current state estimates from the optimizer). For the Diff-MPC variants, we only get x_0 as input. But we repeat it T times and concatenate it ($[x_0] \times T$ to obtain a temporal input that can be fed to the temporal convolutional network described below. Further, for experiments with dynamic obstacles, we also provide the x^o estimate of the obstacles at the current step and predict their positions for the next T steps.

Feedforward network. We use a Temporal Convolution Network (TCN) architecture. We first compute input embeddings for the trajectory by computing a node embedding at each time-step with a node encoder (Linear-LayerNorm-ReLU). We then concatenate the node embedding of x_0 to all time-steps and a corresponding time embedding to indicate their respective time-steps. These are then passed through a series of four temporal convolution residual blocks (Conv1D-GroupNorm-ReLU) before computing the output with a final temporal Conv1D layer. The output at each time step represents the $\delta x_t = x_t - x_0$.

For experiments with dynamic obstacles, we use a separate obstacle TCN, that predicts the future trajectories of the obstacles. The obstacle TCN takes the current obstacle and agent state as input and predicts $\delta x^o_t = x^o_t - x^o_0$. Further, the graph embeddings from the obstacle GCN are also concatenated to the input embeddings for the agent TCN above.

More details are available in the code attached with the supplementary.

DEQ network. We again use a Temporal Convolution Network architecture. We have three separate blocks here, namely, input injection layer I, fixed point layer d and output layer g. We compute the forward pass by first computing the fixed point on the latents:

$$z^* = d_\phi(z^*, I_\phi(x_0, \hat{x}_{1:T})) \tag{15}$$

and then compute the outputs using $g_\phi(z^*)$. The input injection layer is similar to the feedforward network. We compute a node embedding at each time-step with a node encoder (Linear-LayerNorm-ReLU) and then concatenate the node embedding of x_0 and a corresponding time embedding to all time-step node embeddings. This sequence of concatenated node embeddings are then passed through a TCN block (Conv1D-GroupNorm-ReLU) to get the final input embeddings that are fed to the fixed point layer.

The fixed point layer: The input embeddings are passed through a TCN block (Conv1D-GroupNorm-ReLU) and added to a temporally arranged latent variable z. The resulting embeddings are passed through another TCN block (Conv1D-GroupNorm-ReLU) with a residual connection, to obtain the output z. These operations combined represent d_ϕ. We compute the fixed point of this layer using a standard Anderson acceleration fixed point solver [50, 51] to get the resulting z^*.

The output layer $g_\phi(z^*)$ is again a TCN block (Conv1D-GroupNorm-ReLU-Conv1D) that computes the computes the output $\delta x_t = x_t - x_0$.

For experiments with dynamic obstacles, we again use another TCN to predict future obstacle trajectories. The key difference here is that the obstacle TCN also takes the current obstacle trajectory prediction and agent trajectory prediction estimates as input to predict the equilibrium updates for future time steps using a similar setup to the agent TCN DEQ described above. Again, the hidden states from the node embeddings of the obstacle TCN at each time step are pooled and also given back as input to the agent TCN at the respective time steps.

More details are available in the code attached with the supplementary.

Default hyperparameters We use a hidden size of 256 for the Pendulum, 512 for Cartpole, 512 for Quadrotor, 1024 for QPole, QPoleObs and QPoleObsDyn for the agent TCN and unless otherwise specified. During training, we use a batch size of 200 for all environments. The hidden size for the obstacle TCN is one fourth the agent TCN hidden dim.