Empowering Neural Networks with Control and Planning Abilities

Shuyuan Wang
School of Applied Science
University of British Columbia
wshuyuan@student.ubc.ca

Philip D. Loewen
Department of Mathematics
University of British Columbia
loew@math.ubc.ca

Michael G. Forbes
Honeywell Process Solutions
michael.forbes@honeywell.com

R. Bhushan Gopaluni
School of Applied Science
bhushan.gopaluni@ubc.ca

Abstract

Learning effective behaviors requires both adaptability and structured planning, traditionally split between model-free and model-based methods. Differentiable control combines the strengths of both, but iLQR, a powerful nonlinear controller, lacks differentiability, limiting its use in end-to-end learning. Differentiating through extended iterations introduces scalability challenges, further hindering its application. We propose a framework that enables iLQR to function as a trainable and differentiable module, either as or within a neural network, by using implicit differentiation to compute accurate gradients with constant backward cost. On behavior imitation tasks across standard benchmarks, our method achieves up to 128x speedup (minimum 21x) over automatic differentiation and improves learning efficiency by 10^6x compared to conventional neural policies. This framework equips neural networks with control and planning abilities, bridging control theory and behavioral learning.

1 Introduction

Human learning, such as riding a bicycle, is often accelerated by leveraging accumulated prior knowledge about the dynamics of the world. This ability allows adults to master new tasks with relatively few trials, underscoring the importance of prior knowledge in efficient learning [1]. Inspired by this principle, researchers have sought to incorporate structured control knowledge within machine learning models [2], particularly in reinforcement learning (RL) and imitation learning, to improve sample efficiency and performance.

Differentiable control has emerged as a powerful approach in these fields, enabling gradient-based optimization techniques to directly adjust policy parameters. By embedding control policies within a differentiable framework, this approach allows for end-to-end training, where both the control strategy and the model can be learned simultaneously, enhancing adaptability and precision.

The iterative Linear Quadratic Regulator (iLQR) [3] is widely used for trajectory optimization [4] due to its computational efficiency [5] and strong control performance [6]. However, making iLQR trainable as a neural network module is challenging. Naively differentiating through iLQR introduces scalability issues, as forward and backward passes are tightly coupled. The forward pass involves solving LQR optimization iteratively to find an optimal trajectory, while the backward pass propagates gradients through each iteration step. This coupling increases memory usage and slows down training, making it impractical for long-horizon or high-dimensional tasks.

NeurIPS 2024 Workshop on Behavioral Machine Learning.

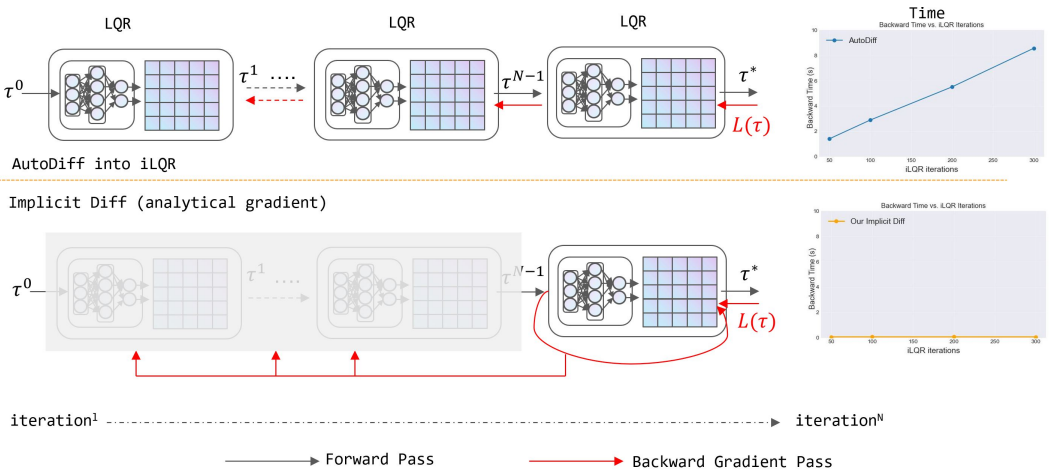


Figure 1: An overview of iLQR, and AutoDiff vs our proposed planner with implicit differentiation. As shown in the flowchart, autodiff must backpropagate through each layer of the LQR process, which leads to significantly increased memory usage to store intermediate gradients and computational load. In contrast, our proposed planner, using implicit differentiation, only needs to handle the final layer. This results in constant computational costs and memory usage, making our method much more efficient.

Efficient differentiable controllers are essential in frameworks where neural networks are integrated with control systems, such as multi-modal learning [7, 8] and deep reinforcement learning [9, 10]. Differentiable controllers enhance sample efficiency and reduce computational time for online tuning, making them critical for large-scale learning systems. Analytical solutions, such as DiffMPC [11], have demonstrated significant improvements in computational efficiency, scalability, and generalization, inspiring their use in broader planning and control applications [12, 13].

This paper makes the following contributions

1. We develop an efficient method for analytical differentiation. We derive analytical trajectory derivatives for optimal control problems with tunable additive cost functions and constrained dynamics described by first-order difference equations, focusing on iLQR as the controller. Our analytical solution is exact, considering the entire iLQR graph. The method guarantees $O(1)$ computational complexity with respect to the number of iteration steps.
2. We demonstrate the effectiveness of our framework in imitation and system identification tasks using the inverted double pendulum and cartpole examples, showcasing superior sample efficiency and generalization compared to traditional neural network policies, as well as more accurate parameter estimations than the reference method.

Notation For a scalar function f with a vector input, ∇f denotes the gradient. Subscripts on ∇ indicate partial derivatives with respect to a subvector or the variable of interest. For tensor operations, we use $\frac{\partial(\cdot)}{\partial(\cdot)}$ for linearization (e.g., eq. (9) for Jacobians). To simplify some equations, we use $D_\theta X$ to represent $\frac{\partial X}{\partial \theta}$. Careful tracking of dependencies is crucial throughout.

2 Related Work

2.1 Differentiable Planning

Model-free policy search methods have achieved notable results in various domains by mapping observations directly to actions [14, 15, 16, 17]. However, these methods often lack interpretability, generalization, and sample efficiency [9, 18, 19, 20]. Differentiable planning combines classical planning with deep learning, enabling end-to-end training of models and policies, leveraging the strengths of both model-free and model-based approaches. Value Iteration Networks (VIN) [21] and its extensions [22, 23, 24, 25] have shown significant improvements, particularly in discrete spaces.

In continuous control, differentiable LQR approaches are prevalent, focusing on finite and infinite horizon LQR [11, 26, 12], constrained LQR [27], and PMP-based methods [28, 29, 30]. While PMP-based methods have slower convergence compared to iLQR [28], iLQR remains a robust numerical control technique [3, 31, 32], with approaches like [33] differentiating through iLQR to learn cost-shaping terms.

Many of these methods, however, involve unrolling optimization procedures, which increases memory, computation costs, and reduces stability [34, 35]. A key advancement is [11], which avoids full unrolling by differentiating through the last iLQR layer, though their analytical gradients are limited due to treating the fixed point as a constant, rather than a function of learning parameters.

2.2 Relations with Neural Networks

A neural network is essentially a function. By formulating iLQR as a differentiable module, iLQR can be embedded into auto-differentiation tools such as PyTorch for training. Thus, the iLQR module can be treated as a trainable network or embedded into a larger network structure for complex tasks, such as interacting with image processing networks for vision-based control (end-to-end learning and control). Previous works have demonstrated the advantages of such methodologies in terms of sample efficiency over black-box neural networks [21, 36, 37]. Our work serves as a foundational study on differentiable iLQR, and we believe our technique can be broadly utilized as a fundamental tool for complex tasks as illustrated by the references. Detailed applications of the technique are beyond the scope of this research.

3 Differentiable iLQR

3.1 End-to-end learning framework

In the differentiable control pipeline, the cost function g and system dynamics f are parameterized by θ, which is unknown and learnable through a performance metric L. We followed standard practice by leaving the θ-dependence implicit in the previous section, but at this point we must bring this into view by changing the notation to $f = f(x, u, \theta)$ and $g = g(x, u, \theta)$. Of course the derivatives shown in (9) and (10) are also functions of θ. For a given reference trajectory τ, the dynamics will generate three θ-dependent matrices we must consider:

$$A_t(\theta) = \frac{\partial f}{\partial x}, \quad B_t(\theta) = \frac{\partial f}{\partial u}, \quad \text{and} \quad \frac{\partial f}{\partial \theta}.$$

Careful accounting for the θ-dependence at every level is required for accurate gradients. If the loss function L can be presented as an explicit function of the trajectory τ, the influence of θ on the observed L-values will be indirect. For the composite function $\theta \mapsto L(\tau(\theta))$, the chain rule gives

$$\nabla_\theta (L \circ \tau)(\theta) = \nabla_\tau L(\tau(\theta)) \frac{\partial \tau}{\partial \theta}. \tag{1}$$

The partial derivatives required to form $\nabla_\tau L$ are provided during the backward pass by automatic differentiation tools [38, 39]. The main challenge, however, is to solve $\frac{\partial \tau}{\partial \theta}$, i.e., *the derivative of the optimal trajectory with respect to the learnable parameters*. Next, we will analytically solve $\frac{\partial \tau}{\partial \theta}$ through implicit differentiating on the fixed point.

3.2 Fixed point differentiation

The sequence of trajectories produced by iLQR can be understood as an iterative rollout of LQR layers:

$$\tau^0 \xrightarrow{iLQR} \tau^1 \xrightarrow{iLQR} \tau^2 \xrightarrow{iLQR} \cdots \xrightarrow{iLQR} \tau^\star \xrightarrow{iLQR} \tau^\star \xrightarrow{iLQR} \cdots . \tag{2}$$

Each LQR layer includes the three steps noted above: linearizing the system, conducting the backward pass, and performing the forward pass. After a sufficient number of iterations (shown as N in above), the output $\tau^\star$ to an LQR step will be the same as the input, indicating that the LQR can no longer improve the trajectory it linearizes around. This trajectory $\tau^\star$ is called a fixed point for the iLQR. The fixed point of iLQR is usually reached because the line search operation in the forward pass ensures that the accumulated cost decreases with each iteration.

A fixed-point problem can be calculated by various methods, typically iterative in nature. As pointed out in [35], naively differentiating through such a scheme would require intensive memory usage [21, 23] and computational effort [34]. Instead, we propose use implicit differentiation directly on the defining identity. This gives direct access to the derivatives required by decoupling the forward (fixed-point iteration as the solver) and backward passes (differentiating through the solver).

Let us write $X = (x_1, \ldots, x_T)$ and $U = (u_1, \ldots, u_T)$ for the components of a trajectory $\tau = (x_1, u_1, x_2, u_2, \ldots, x_T, u_T)$, and abuse notation somewhat by identify τ with (X, U). At a fixed point (X^*, U^*) of the iLQR, we have the following:

$$X^* = F(X^*, U^*, \theta), \quad U^* = G(X^*, U^*, \theta) \tag{3}$$

where F and G summarize the operations that make up a single LQR layer.

In eq. (3), the solutions X^* and U^* depend on the parameter θ. By treating both X^* and U^* explicitly as functions of θ, we can interpret eq. (3) as an identity valid for all θ. Differentiating through this identity yields a new one:

$$\begin{aligned} D_\theta X^* &= \frac{\partial F}{\partial X} D_\theta X^* + \frac{\partial F}{\partial U} D_\theta U^* + \frac{\partial F}{\partial \theta} \\ D_\theta U^* &= \frac{\partial G}{\partial X} D_\theta X^* + \frac{\partial G}{\partial U} D_\theta U^* + \frac{\partial G}{\partial \theta}. \end{aligned} \tag{4}$$

Here $D_\theta X^*$ and $D_\theta U^*$ are the sensitivity matrices (Jacobians) that quantify the θ-dependence of the optimal trajectory; both depend on θ. The matrix-valued partial derivatives of F and G above are evaluated at $(X^*(\theta), U^*(\theta), \theta)$. Rearranging the identities above produces a system of linear equations in which these matrices provide the unknowns:

$$\begin{aligned} \left(I - \frac{\partial F}{\partial X}\right) D_\theta X^* - \frac{\partial F}{\partial U} D_\theta U^* &= \frac{\partial F}{\partial \theta} \\ -\frac{\partial G}{\partial X} D_\theta X^* + \left(I - \frac{\partial G}{\partial U}\right) D_\theta U^* &= \frac{\partial G}{\partial \theta}. \end{aligned} \tag{5}$$

The analytical solution for this system is given below.

Proposition 1. *The sensitivity matrices in eq.* (5) *are given by*

$$\begin{aligned} D_\theta X^* &= M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta - G_\theta)) \\ D_\theta U^* &= (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta), \end{aligned} \tag{6}$$

where we denote $M = (I - F_X)^{-1}$ *and* $K = I - G_U$*, and use the condensed notation*

$$F_X = \frac{\partial F}{\partial X}, \quad F_U = \frac{\partial F}{\partial U}, \quad F_\theta = \frac{\partial F}{\partial \theta}, \quad G_X = \frac{\partial G}{\partial X}, \quad G_U = \frac{\partial G}{\partial U}, \quad G_\theta = \frac{\partial G}{\partial \theta}. \tag{7}$$

Proof. Please check Appendix for detailed proof. □

To be completely explicit, suppose a parameter θ is given. Then eq. (3) defines a fixed point τ^* in terms of this particular θ, and this τ^* provides the evaluation point $(X^*(\theta), U^*(\theta), \theta)$ for all the Jacobian matrices involving F and G in Equations (4) to (6).

4 Experiments

Following prior work [11, 28, 27, 40], we evaluate our method on two control benchmarks: **CartPole** and **Inverted Pendulum**. All experiments were conducted on an AMD 3700X CPU (3.6GHz), 16GB RAM, and RTX3080 GPU with 10GB VRAM using PyTorch [38].

4.1 Computational Performance

Figure 2 highlights the computational advantages of our method over AutoDiff. For 10 horizons and 300 iterations, AutoDiff takes 8.57s, while our method requires only 0.067s, yielding a **128x speedup**. Even with fewer iterations (50), AutoDiff takes 1.41s, compared to 0.067s for our method—a **21x speedup**. These results demonstrate that our method scales efficiently, maintaining near-constant time regardless of iteration count, unlike AutoDiff.

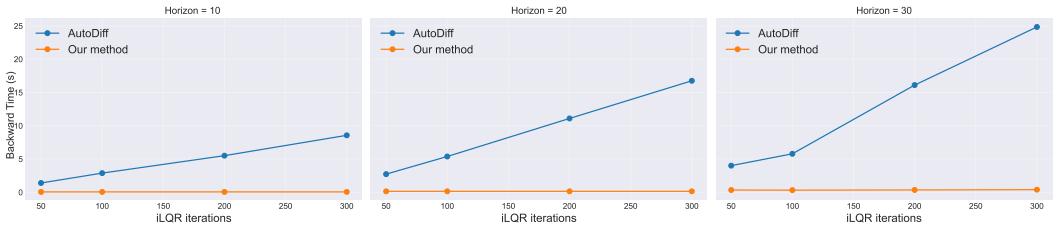
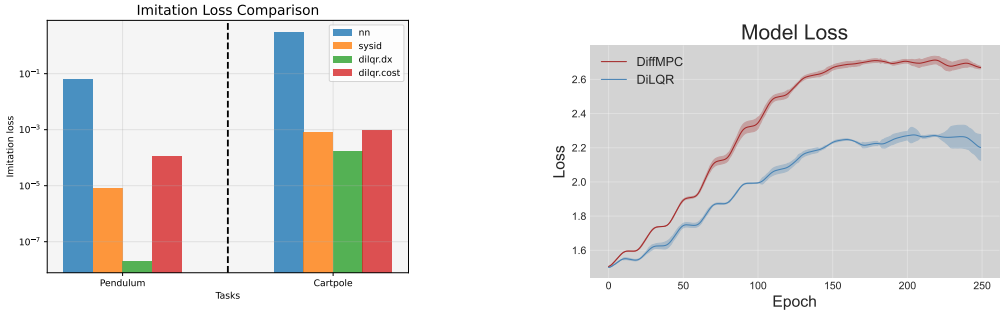


Figure 2: Backward computation time comparison between AutoDiff and our method under different iLQR iterations and horizons. AutoDiff scales linearly, while our method maintains constant computation time. Experiments conducted with batch size 20 on the pendulum domain.



(a) (b)

Figure 3: (a) Imitation loss comparison across methods. (b) Cost function parameter estimation between our approach and DiffMPC on CartPole.

4.2 Imitation Learning

We compare our approach against: 1. **NN**: An LSTM-based model predicting actions from state x. 2. **SysId**: Assumes known cost and learns system dynamics via next-state prediction. 3. **DiffMPC**: Uses analytical gradients for trajectory optimization.

We evaluate two variants of our approach: - **diLQR.dx**: Learns dynamics with known cost by optimizing imitation loss. - **diLQR.cost**: Learns cost with known dynamics by optimizing imitation loss. Details can be found in the Appendix.

Imitation Loss: In Figure 3a, our method (diLQR.dx) outperforms NN and SysId, achieving improvements of 10^6 and 10^4 on the respective tasks. In diLQR.cost mode, our method performs comparably to SysId, despite relying solely on action data, which contains less information than state-based models.

Model Loss: Figure 3b shows the model loss (MSE($\theta - \hat{\theta}$)) when estimating cost parameters. Our method achieves an **18% reduction in model loss** compared to DiffMPC, demonstrating superior accuracy in recovering the true parameters.

5 Conclusions

We introduced **DiLQR**, an efficient framework that empowers neural networks with control and planning abilities through implicit differentiation. Our analytical solution eliminates the overhead of iterative unrolling, achieving *O(1)* computational complexity in the backward pass for scalable and real-time learning.

Experiments show that DiLQR outperforms existing methods in both runtime and learning efficiency, making it a powerful tool for behavioral learning tasks. By integrating control algorithms into neural networks, DiLQR enables adaptive, structured decision-making, paving the way for more efficient learning in complex environments.

References

[1] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017.

[2] Shuyuan Wang, Jingliang Duan, Nathan P Lawrence, Philip D Loewen, Michael G Forbes, R Bhushan Gopaluni, and Lixian Zhang. Guiding reinforcement learning with incomplete system dynamics. *arXiv preprint arXiv:2410.16821*, 2024.

[3] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[4] Nathan A Spielberg, Matthew Brown, and J Christian Gerdes. Neural network model predictive motion control applied to automated driving with unknown friction. *IEEE Transactions on Control Systems Technology*, 30(5):1934–1945, 2021.

[5] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[6] Ewen Dantec, Maximilien Naveau, Pierre Fernbach, Nahuel Villa, Guilhem Saurel, Olivier Stasse, Michel Taix, and Nicolas Mansard. Whole-body model predictive control for biped locomotion on a torque-controlled humanoid robot. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 638–644. IEEE, 2022.

[7] Jiageng Mao, Yuxi Qian, Junjie Ye, Hang Zhao, and Yue Wang. Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415*, 2023.

[8] Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee K Wong, Zhenguo Li, and Hengshuang Zhao. Drivegpt4: Interpretable end-to-end autonomous driving via large language model. *IEEE Robotics and Automation Letters*, 2024.

[9] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering Atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

[10] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the 30th AAAI conference on artificial intelligence*, volume 30, pages 2094–2100, 2016.

[11] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable MPC for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

[12] Sebastian East, Marco Gallieri, Jonathan Masci, Jan Koutnik, and Mark Cannon. Infinite-horizon differentiable model predictive control. *Proceedings of ICLR 2020*, 2020.

[13] Angel Romero, Yunlong Song, and Davide Scaramuzza. Actor-critic model predictive control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14777–14784. IEEE, 2024.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[17] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596, 2018.

[18] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.

[19] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), 2017.

[20] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[21] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[22] Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa Smith, and Jelena Kovačević. Generalized value iteration networks: Life beyond lattices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[23] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. In *International Conference on Machine Learning*, pages 2947–2955. PMLR, 2018.

[24] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. Differentiable spatial planning using transformers. In *International conference on machine learning*, pages 1484–1495. PMLR, 2021.

[25] Daniel Schleich, Tobias Klamt, and Sven Behnke. Value iteration networks on multiple levels of abstraction. *arXiv preprint arXiv:1905.11068*, 2019.

[26] Jatan Shrestha, Simon Idoko, Basant Sharma, and Arun Kumar 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.

[27] Ming Xu, Timothy L Molloy, and Stephen Gould. Revisiting implicit differentiation for learning problems in optimal control. *Advances in Neural Information Processing Systems*, 36, 2024.

[28] Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. *Advances in Neural Information Processing Systems*, 33:7979–7992, 2020.

[29] Wanxin Jin, Shaoshuai Mou, and George J Pappas. Safe pontryagin differentiable programming. *Advances in Neural Information Processing Systems*, 34:16034–16050, 2021.

[30] Lucas Böttcher, Nino Antulov-Fantulin, and Thomas Asikis. AI Pontryagin or how artificial neural networks learn to control dynamical systems. *Nature communications*, 13(1):333, 2022.

[31] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *First International Conference on Informatics in Control, Automation and Robotics*, volume 2, pages 222–229. SciTePress, 2004.

[32] James Zhu, J Joe Payne, and Aaron M Johnson. Convergent ilqr for safe trajectory planning and control of legged robots. *arXiv preprint arXiv:2304.00346*, 2023.

[33] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic MPC improvement. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 336–343. IEEE, 2017.

[34] Linfeng Zhao, Huazhe Xu, and Lawson LS Wong. Scaling up and stabilizing differentiable planning with implicit differentiation. *arXiv preprint arXiv:2210.13542*, 2022.

[35] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

[36] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

[37] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*, 2019.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[39] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[40] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28, 2015.

[41] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

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

A Appendix / supplemental material

A.1 ILQR Background

The Iterative Linear Quadratic Regulator (iLQR) is a trajectory optimization algorithm. At each iteration step, it linearizes the dynamics and makes quadratic approximation of the cost function, and then solves the problem using a finite-time Linear Quadratic Regulator (LQR). For a non-linear control problem defined by

$$J^\star = \min_{x_{1:T}, u_{1:T}} \sum_{t=1}^{T} g(x_t, u_t) \text{ s.t. } x_{t+1} = f(x_t, u_t), \quad \underline{u} \le u \le \bar{u}, \quad x_1 = x_{init}, \tag{8}$$

the iLQR algorithm optimizes the trajectory by repeatedly proceeding the following steps:

A.2 The Approximate Problem

Iteration i begins with the trajectory $\tau^i = \{\tau_1^i, \ldots, \tau_T^i\}$, where $\tau_t^i = \{x_t^i, u_t^i\}$. We linearize the dynamics by defining

$$D_t = [A_t, B_t] = \left[\left.\frac{\partial f_t}{\partial x}\right|_{\tau_t^i}, \left.\frac{\partial f}{\partial u}\right|_{\tau_t^i} \right], \quad d_t = f_t(x_t^i, u_t^i) - D_t \begin{bmatrix} x_t^i \\ u_t^i \end{bmatrix}, \qquad t = 1, 2, \ldots, T, \tag{9}$$

and form a quadratic approximation of the cost function using

$$c_t^\top = [c_{t,x}, c_{t,u}] = \left[\left.\frac{\partial g_t}{\partial x}\right|_{\tau_t^i}, \left.\frac{\partial g_t}{\partial u}\right|_{\tau_t^i} \right], \quad C_t = \begin{bmatrix} C_{t,xx} & C_{t,xu} \\ C_{t,ux} & C_{t,uu} \end{bmatrix}, \qquad t = 1, 2, \ldots, T, \tag{10}$$

where

$$C_{t,xx} = \left.\frac{\partial^2 g_t}{\partial x^2}\right|_{\tau_t^i}, \quad C_{t,uu} = \left.\frac{\partial^2 g_t}{\partial u^2}\right|_{\tau_t^i}, \quad C_{t,xu} = C_{t,ux}^\top = \left.\frac{\partial^2 g_t}{\partial u \partial x}\right|_{\tau_t^i}.$$

These elements lead to an approximate problem whose unknowns are $\delta\tau_t = \tau_t - \tau_t^i$:

$$\min_{\delta\tau_{1:T}} \sum_{t=0}^{T} \frac{1}{2} \delta\tau_t{}^\top C_t \delta\tau_t + c_t^\top \delta\tau_t \quad \text{s.t.} \quad \delta x_{t+1} = D_t \delta\tau_t, \; \delta x_1 = 0; \quad \underline{u} \le u \le \bar{u}. \tag{11}$$

A.3 The Trajectory Update

Problem (11) can be solved by the two-pass method detailed in [5]. First a backward pass is conducted, using the Riccati-Mayne method [41] to obtain a quadratic value function and a projected-Newton method to optimize the actions under box constraints. Then a forward pass uses the linear control gains K_t, k_t obtained in the backward pass to roll out a new trajectory. Let $\delta\tau^\star$ denote the minimizing trajectory in (11). We use the controls in $\delta\tau^*$ directly, but discard the states in favor of an update based on the original dynamics, setting

$$u_t^{i+1} = u_t^i + \delta u_t^\star, \qquad x_{t+1}^{i+1} = f(x_t^{i+1}, u_t^{i+1}). \tag{12}$$

With these choices, defining $\tau_t^{i+1} = \{x_t^{i+1}, u_t^{i+1}\}$ provides a feasible trajectory for (8) that can serve as the starting point for another iteration.

A.4 Proof of proposition 1

Proposition 2. *Define* $F_\theta := \frac{\partial F}{\partial \theta}$, $F_U := \frac{\partial F}{\partial U}$, $F_X := \frac{\partial F}{\partial X}$, $G_\theta := \frac{\partial G}{\partial \theta}$, $G_U := \frac{\partial G}{\partial U}$, $G_X := \frac{\partial G}{\partial X}$. *Define* $M := (I - F_X)^{-1}$, *and* $K := I - G_U$. *The analytical form of the gradients* $\frac{dX}{d\theta}$ *and* $\frac{dU}{d\theta}$ *are given as follows:*

$$\begin{aligned} \frac{dX}{d\theta} &= M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta - G_\theta)) \\ \frac{dU}{d\theta} &= (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta) \end{aligned} \tag{13}$$

Proof. With the new notations, equations can be rewritten as:

$$\begin{aligned}(I - F_X)\frac{dX^*}{d\theta} - F_U\frac{dU^*}{d\theta} &= F_\theta \\ -G_X\frac{dX^*}{d\theta} + (I - G_U)\frac{dU^*}{d\theta} &= G_\theta\end{aligned} \tag{14}$$

Focusing on the first equation, $\frac{dX}{d\theta}$ can be represented with $\frac{dU}{d\theta}$:

$$\begin{aligned}\frac{dX}{d\theta} &= (I - F_X)^{-1}(F_\theta + F_U\frac{dU}{d\theta}) \\ &= M(F_\theta + F_U\frac{dU}{d\theta})\end{aligned} \tag{15}$$

Then, substituting 15 into the second equation of 14 to obtain an equation with respect to only $\frac{dU}{d\theta}$:

$$-G_X(M(F_\theta + F_U\frac{dU}{d\theta})) + (I - G_U)\frac{dU^*}{d\theta} = G_\theta \tag{16}$$

Solving equation 16 will give the solution to $\frac{dU^*}{d\theta}$:

$$\frac{dU}{d\theta} = (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta) \tag{17}$$

Substituting 17 into 15, the solution to $\frac{dX}{d\theta}$ can be obtained:

$$\frac{dX}{d\theta} = M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta)) \tag{18}$$

This completes the proof. □

A.5 Obtaining each term in (14)

The functions F and G whose sensitivity matrices appear in eq. (7) are defined by rather complicated arg min operations. The Chain-Rule pattern below, which we can apply to either $H = F$ or $H = G$, suggests that

$$\begin{aligned}H_X &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial X} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial X} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial X} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial X}, \\ H_U &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial U} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial U} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial U} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial U}, \\ H_\theta &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial \theta} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial \theta} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial \theta} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial \theta}.\end{aligned} \tag{19}$$

In each term on the right, the first matrix factor (e.g., $\partial H/\partial D$) expresses the sensitivity of the optimal LQR trajectory to the corresponding named ingredient of the formulation in eq. (11). Efficient methods for calculating these terms are known: see [11, 42]. The second factor in each term of (19) can be computed using automatic differentiation.

A.6 Experiments Details

We refer the methods in DiffMPC as mpc.dx: Assumes the cost of the controller is known and approximates the parameters of the dynamics by directly optimizing the imitation loss; mpc.cost: Assumes the dynamics of the controller are known and approximates the cost by directly optimizing the imitation loss. For all settings involving learning the dynamics (mpc.dx, mpc.cost. iLQR.dx, and iLQR.cost.dx), a parameterized version of the true dynamics is used. In the pendulum domain, the parameters are the masses of the arm, length of the arm, and gravity; and in the cartpole domain, the parameters are the cart's mass, pole's mass, gravity, and length. For cost learning in mpc.cost, iLQR.cost and mpc,cost.dx, we parameterized the controller's cost as the weighted distance to a goal state $C(\tau) = \|w_g(\tau - \tau_g)\|$. As indicated in [11], simultaneously learning the weights w_g and goal state τ_g was unstable. Thus, we alternated learning w_g and τ_g independently every 10 epochs.

Training and Evaluation We collected a dataset of trajectories from an expert controller and varied the number of trajectories our models were trained on. The NN setting was optimized with Adam with a learning rate of 10^{-4}, and all other settings were optimized with RMSprop with a learning rate of 10^{-2} and a decay term of 0.5.