DILQR: DIFFERENTIABLE ITERATIVE LINEAR QUADRATIC REGULATOR

Anonymous authors
Paper under double-blind review

ABSTRACT

Differentiable control promises end-to-end differentiability and adaptability, effectively combining the advantages of both model-free and model-based control approaches. However, the iterative Linear Quadratic Regulator (iLQR), despite being a powerful nonlinear controller, still lacks differentiable capabilities. The scalability of differentiating through extended iterations and horizons poses significant challenges, hindering iLQR from being an effective differentiable controller. This paper introduces a framework that facilitates differentiation through iLQR, allowing it to serve as a trainable and differentiable module, either as or within a neural network. for control purposes. A novel aspect of this framework is the analytical solution that it provides for the gradient of an iLQR controller through implicit differentiation, which ensures a constant backward cost regardless of iteration, while producing an accurate gradient. We evaluate our framework on imitation tasks on famous control benchmarks. Our analytical method demonstrates superior computational performance, achieving up to **128x speedup** and a minimum of **21x speedup** compared to automatic differentiation. Our method also demonstrates superior learning performance ($\mathbf{10^6x}$) compared to traditional neural network policies and better model loss with differentiable controllers that lack exact analytical gradients. Furthermore, we integrate our module into a larger network with visual inputs to demonstrate the capacity of our method for high-dimensional, fully end-to-end tasks. Codes can be found on the project homepage `https://sites.google.com/view/dilqr/`.

1 INTRODUCTION

Differentiable control has emerged as a powerful approach in the fields of reinforcement learning (RL) and imitation learning, enabling significant improvements in sample efficiency and performance. By integrating control policies into a differentiable framework, researchers can leverage gradient-based optimization techniques to directly optimize policy parameters. This integration allows for end-to-end training, where both the control strategy and the underlying model can be learned simultaneously, enhancing the adaptability and precision of control systems.

As a numerical controller, the iterative Linear Quadratic Regulator (iLQR) Todorov et al. (2012) has been extensively adopted for trajectory optimization Spielberg et al. (2021); Choi et al. (2023); Zhao et al. (2020); Mastalli et al. (2020) due to its computational efficiency Tassa et al. (2014); Dean et al. (2020); Collins et al. (2021) and excellent control performance Dantec et al. (2022); Xie et al. (2017); Chen et al. (2017). To make iLQR trainable as a neural network module, naively differentiating through an iLQR controller may be a reasonable choice, but the scalability of differentiating through hundreds of iterations steps poses a significant challenge, as the forward and backward passes during training are coupled. The forward pass involves iteratively solving an LQR optimization problem to converge on the optimal trajectory. The backward pass computes gradients through backpropagation, and becomes increasingly complex as it needs to traverse through all the layers of the forward pass, which requires significant computational resources (time and memory), especially for tasks requiring long iterations and long horizons. This coupling not only increases memory usage, but also significantly slows down the training process, making it difficult to scale to larger problems.

Efficient differentiable controllers are especially valuable in systems involving neural networks, such as multi-modal frameworks Mao et al. (2023); Xu et al. (2024b); Xiao et al. (2022) and deep reinforcement learning Ye et al. (2021); van Hasselt et al. (2016), where an upstream neural network module is required. Developing differentiable controllers with efficient gradient propagation is crucial, as they greatly enhance sample efficiency and reduce computational time for online tuning.

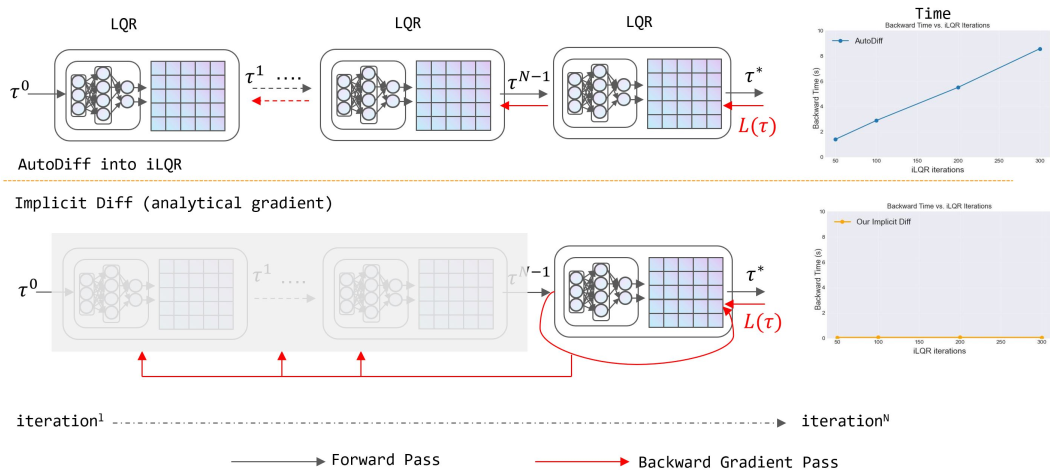


Figure 1: An overview of iLQR, and AutoDiff vs our proposed planner with implicit differentiation. As shown in the flowchart, autodiff must backpropagate through each layer of the LQR process, which leads to significantly increased memory usage to store intermediate gradients and computational load. 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.

Developing analytical solutions would greatly alleviate these challenges. DiffMPC Amos et al. (2018) pioneered the use of analytical gradients in LQR control, leading to significant improvements in computational efficiency and generalization of the learned controller. Its success has inspired extensions in various planning and control applications East et al. (2020); Romero et al. (2024); Karkus et al. (2023); Cheng et al. (2024); Soudbakhsh et al. (2023); Shrestha et al. (2023). Numerous studies have since shown that analytical gradients significantly improve learning performance, reducing computational costs, and improving scalability in complex, long-horizon tasks Jin et al. (2020); Xu et al. (2024a); Jin et al. (2021); Böttcher et al. (2022); Zhao et al. (2022).

In this paper, we introduce an innovative analytical framework that leverages implicit differentiation to handle iLQR at its fixed point. This approach effectively separates the forward and backward computations, maintaining a constant computational load during the backward pass, irrespective of the iteration numbers for iLQR. By doing so, our method significantly reduces computational time and the memory usage needed for training, thereby enhancing scalability and efficiency in handling non-convex control problems.

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 propose a forward method for differentiating linearized dynamics with respect to nonlinear dynamics parameters, achieving speeds dozens of times faster than auto-differentiation tools such as `torch.autograd.jacobian`. Furthermore, we exploit the sparsity of the tensor expressions to compute some tensor derivatives that scale linearly with trajectory length.
3. We demonstrate the effectiveness of our framework in imitation and system identification tasks using the inverted pendulum and cartpole examples, showcasing superior sample efficiency and generalization compared to traditional neural network policies. Finally, we integrate our differentiable iLQR into a large network for end-to-end learning and control from pixels, demonstrating the extensibility and multimodal capabilities of our method.

Notation For a scalar-valued function f with a vector input, ∇f is the usual gradient. The subscripts in the symbol ∇ indicate partial derivatives involving a subvector of the full input, or serve to emphasize the variable of interest. For a more general operation mapping tensors to tensors, we write $\frac{\partial(\cdot)}{\partial(\cdot)}$ for the appropriate linearization. See, for example, eq. (2), where Jacobian matrices are constructed. To improve the readability of some equations, we sometimes use the notation $D_\theta X$ as a synonym for $\frac{\partial X}{\partial \theta}$. Careful tracking of the dependencies involved is essential at every stage.

2 Related Work on Differentiable Planning

Pure model-free techniques for policy search have demonstrated promising results in many domains by learning reactive policies that directly map observations to actions Haarnoja et al. (2018); Sutton & Barto (2018); Schulman et al. (2017); Fujimoto et al. (2018). However, due to the black box nature of these policies, model-free methods suffer from a lack of interpretability, poor generalization, and high sample complexity Ye et al. (2021); Yu (2018); Bacon et al. (2017); Deisenroth & Rasmussen (2011). Differentiable planning integrates classical planning algorithms with modern deep learning techniques, enabling end-to-end training of models and policies, thereby combining the complementary advantages of model-free and model-based methods. Value Iteration Network (VIN) Tamar et al. (2016) is a representative work that performs value iteration using convolution on lattice grids and has been extended further Niu et al. (2018); Lee et al. (2018); Chaplot et al. (2021); Schleich et al. (2019). These works have demonstrated significant performance improvements on various tasks.

However, these works primarily focus on discrete action and state spaces. In the field of continuous control, most efforts have focused on differentiable LQR, including differentiating through finite horizon LQR Amos et al. (2018); Shrestha et al. (2023), infinite horizon East et al. (2020); Brewer (1977), and constrained LQR Xu et al. (2024a). References (Jin et al., 2020; 2021; Böttcher et al., 2022) propose frameworks that can differentiate through Pontryagin's Maximum Principle (PMP) conditions. However, the convergence speed of PMP-based methods is slower than that of iLQR Jin et al. (2020), due to the 1.5 order convergence rate of iLQR. More importantly, these methods and Xu et al. (2024a) assume a broad range of forward pass solutions and do not align the gradient in the backward pass with forward solution.

For iLQR, which is a powerful numerical control technique Todorov et al. (2012); Li & Todorov (2004); Zhu et al. (2023), Tamar et al. (2017) differentiates through an iterative LQR (iLQR) solver to learn a cost-shaping term offline. Other methods based on numerical control techniques include Okada et al. (2017); Pereira et al. (2018), which provide methods to differentiate through path integral optimal control, and Srinivas et al. (2018), which shows how to embed differentiable planning (unrolled gradient descent over actions) within a goal-directed policy.

However, all of these methods require differentiation through planning procedures by explicitly unrolling the optimization algorithm itself, introducing drawbacks such as increased memory and computational costs and reduced computational stability Zhao et al. (2022); Bai et al. (2019). DiffMPC Amos et al. (2018) is a representative work in the field of differentiable MPC. Significant progress has been made in the efficient differentiable LQR with box constraints by Amos et al. (2018). To differentiate iLQR, Amos et al. (2018) proposes a methodology that differentiates through the last layer of iLQR to avoid unrolling of the entire iLQR graph. However, Amos et al. (2018) treats the input to the last layer of LQR as a constant, rather than a function of the learning parameters. Using implicit differentiation, we develop a framework that provides exact analytical solutions for iLQR gradients, improving the gradient computation presented in Amos et al. (2018). Our approach not only addresses scalability issues, but also improves learning performance.

3 Background

The Iterative Linear Quadratic Regulator (iLQR) addresses the following control problem:

$$\min_{x_{1:T}, u_{1:T}} \sum_{t=1}^{T} g_t(x_t, u_t) \;\text{ s.t. }\; x_{t+1} = f_t(x_t, u_t),\; x_1 = x_{init}; \quad \underline{u} \leq u \leq \bar{u}. \tag{1}$$

At each iteration step, it linearizes the dynamics and makes a quadratic approximation of the cost function to produce a finite-time Linear Quadratic Regulator (LQR) problem. Solving this auxiliary problem produces updates for the original trajectory. Here are some details.

3.1 The Approximate Problem

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

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

and form a quadratic approximation of the cost function using

$$c_t^\top = [c_{t,x}, c_{t,u}] = \left[\frac{\partial g_t}{\partial x}\bigg|_{\tau_t^i}, \frac{\partial g_t}{\partial u}\bigg|_{\tau_t^i} \right], \quad C_t = \begin{bmatrix} C_{t,xx} & C_{t,xu} \\ C_{t,ux} & C_{t,uu} \end{bmatrix}, \qquad t = 1, 2, \dots, T, \tag{3}$$

where

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

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

$$\min_{\delta\tau_{1:T}} \sum_{t=0}^{T} \frac{1}{2} \delta\tau_t{}^\top C_t \delta\tau_t + c_t^\top \delta\tau_t \quad \text{s.t.} \quad \delta x_{t+1} = D_t \delta\tau_t,\ \delta x_1 = 0; \quad \underline{u} \le u \le \bar{u}. \tag{4}$$

3.2 The Trajectory Update

Problem (4) can be solved by the two-pass method detailed in Tassa et al. (2014). First a backward pass is conducted, using the Riccati-Mayne method Mayne et al. (2000) 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 (4). We use the controls in $\delta\tau^*$ directly, but discard the states in favor of an update based on the original dynamics, setting

$$u_t^{i+1} = u_t^i + \delta u_t^\star, \qquad x_{t+1}^{i+1} = f(x_t^{i+1}, u_t^{i+1}). \tag{5}$$

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

4 Differentiable iLQR

4.1 End-to-end learning framework

In the learning problem of interest here, the cost functions g_t and system dynamics f_t involve structured uncertainty parameterized by a vector variable θ. For example, in a drone, θ could represent physical parameters like mass or propeller length, while in a humanoid robot, it might refer to limb lengths or joint masses; additionally, θ can include reference trajectories for robot tracking, which help parametrize the cost function for control. Suppressing θ in the notation is typical when θ has a fixed value, but now we face the challenge of choosing θ to optimize some scalar criterion. This requires changing the notation to $f_t = f_t(x, u, \theta)$ and $g_t = g_t(x, u, \theta)$. As such, the derivatives shown in (2) and (3) must also be considered as functions of θ. So, along a given reference trajectory τ, the dynamics in (1) will generate three θ-dependent matrices we must consider:

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

The same is true for the coefficients in the quadratic approximation to the loss function in the original problem. Careful accounting for the θ-dependence at every level is required for accurate gradients.

Suppose the loss function L to be minimized by "learning" θ is expressed entirely in terms of the trajectory τ. Then the influence of θ on the observed L-values will be indirect, and we will need the chain rule to express the gradient of the composite function $\theta \mapsto L(\tau(\theta))$:

$$\nabla_\theta (L \circ \tau)(\theta) = \nabla_\tau L(\tau(\theta)) \frac{\partial \tau}{\partial \theta}. \tag{6}$$

In practical implementations, the partial derivatives required to form $\nabla_\tau L$ are provided during the backward pass by automatic differentiation tools Paszke et al. (2019); Abadi et al. (2015). The main challenge, however, is to determine $\frac{\partial \tau}{\partial \theta}$, i.e., *the derivative of the optimal trajectory with respect to the learnable parameters*. This is the focus of the next section.

4.2 Fixed point differentiation

For a particular choice of θ, we can consider the sequence of trajectories produced by iLQR:

$$\tau^0 \xrightarrow{iLQR} \tau^1 \xrightarrow{iLQR} \tau^2 \xrightarrow{iLQR} \cdots \xrightarrow{iLQR} \tau^\star \xrightarrow{iLQR} \tau^\star \xrightarrow{iLQR} \cdots . \tag{7}$$

Each iteration includes the three steps noted above: linearizing the system, conducting the backward pass, and performing the forward pass. Iterations proceed until the output $\tau^\star$ from an iLQR step is indistinguishable from the input, indicating that the process can no longer improve the input trajectory. This trajectory $\tau^\star$ is called a fixed point for the iLQR. We expect the value of θ to influence the fixed point produced above.

In general, an operator's fixed point can be calculated by various methods, typically iterative in nature. As pointed out in Bai et al. (2019), naively differentiating through such a scheme would require intensive memory usage Tamar et al. (2016); Lee et al. (2018) and computational effort Zhao et al. (2022). Instead, we propose to 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 identifying τ with (X, U). At a fixed point $(X^\star, U^\star)$ of the iLQR process for parameter θ, we have the following:

$$X^\star = F(X^\star, U^\star, \theta), \quad U^\star = G(X^\star, U^\star, \theta) \tag{8}$$

where F and G summarize the operations that define a single iteration in the iLQR algorithm. (Thus eq. (8) formalizes the graphical summary in eq. (7).)

In eq. (8), the solutions $X^\star$ and $U^\star$ depend on the parameter θ. By treating both $X^\star$ and $U^\star$ explicitly as functions of θ, we can interpret eq. (8) as an identity valid for all θ. Differentiating through this identity yields a new one:

$$\begin{aligned} \nabla_\theta X^\star &= \frac{\partial F}{\partial X}\nabla_\theta X^\star + \frac{\partial F}{\partial U}\nabla_\theta U^\star + \frac{\partial F}{\partial \theta}, \\ \nabla_\theta U^\star &= \frac{\partial G}{\partial X}\nabla_\theta X^\star + \frac{\partial G}{\partial U}\nabla_\theta U^\star + \frac{\partial G}{\partial \theta}. \end{aligned} \tag{9}$$

Here, the matrix-valued partial derivatives of F and G above are evaluated at $(X^\star(\theta), U^\star(\theta), \theta)$. Likewise, $D_\theta X^\star$ and $D_\theta U^\star$ are the Jacobians (sensitivity matrices) that quantify the θ-dependence of the optimal trajectory; both depend on θ. Rearranging eq. (9) produces a system of linear equations in which these two matrices provide the unknowns:

$$\begin{aligned} \left(I - \frac{\partial F}{\partial X}\right)\nabla_\theta X^\star - \frac{\partial F}{\partial U}\nabla_\theta U^\star &= \frac{\partial F}{\partial \theta}, \\ -\frac{\partial G}{\partial X}\nabla_\theta X^\star + \left(I - \frac{\partial G}{\partial U}\right)\nabla_\theta U^\star &= \frac{\partial G}{\partial \theta}. \end{aligned} \tag{10}$$

The analytical solution for this system is given below.

Proposition 1. *The Jacobians in eq.* (10) *are given by*

$$\begin{aligned} \nabla_\theta X^\star &= M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta - G_\theta)) \\ \nabla_\theta U^\star &= (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta), \end{aligned} \tag{11}$$

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

$$F_X = \frac{\partial F}{\partial X}, \quad F_U = \frac{\partial F}{\partial U}, \quad F_\theta = \frac{\partial F}{\partial \theta}, \quad G_X = \frac{\partial G}{\partial X}, \quad G_U = \frac{\partial G}{\partial U}, \quad G_\theta = \frac{\partial G}{\partial \theta}. \tag{12}$$

See the Appendix.

To be completely explicit, suppose a parameter θ is given. Then eq. (8) defines a fixed point $\tau^\star$ in terms of this particular θ, and this $\tau^\star$ provides the evaluation point $(X^\star(\theta), U^\star(\theta), \theta)$ for all the Jacobian matrices involving F and G in Equations (9) to (11).

4.3 OBTAINING EACH TERM

The functions F and G whose Jacobian appear in eq. (12) 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

$$\begin{aligned} H_X &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial X} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial X} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial X} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial X}, \\ H_U &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial U} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial U} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial U} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial U}, \\ H_\theta &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial \theta} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial \theta} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial \theta} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial \theta}. \end{aligned} \tag{13}$$

In each term on the right, the first matrix factor (e.g., $\partial H/\partial D$) expresses the sensitivity of the optimal LQR trajectory with respect to the corresponding named ingredient of the formulation in eq. (4). Efficient methods for calculating these terms are known: see Amos et al. (2018); Amos & Kolter (2017). The second factor in each term of (13) can be computed using automatic differentiation. The next subsections talk about how to calculate these terms efficiently.

4.4 PARALLELIZATION

Amos et al. (2018) proposes method that directly calculates $\frac{\partial L}{\partial D}$, $\frac{\partial L}{\partial d}$, $\frac{\partial L}{\partial C}$, and $\frac{\partial L}{\partial c}$ with a complexity of only $O(T)$. We adopt these results in our framework. To facilitate parallelization, we construct batches of binary loss functions. Specifically, to compute $\frac{\partial H_{i,j}}{\partial D}$, we set the $L_{i,j}$ element in L to 1, while all other elements are set to 0, and then calculate $\frac{\partial L}{\partial D}$. Although this approach introduces more computations, the computations can be fully parallelized since each operation is completely independent. As a result, the calculation of $\frac{\partial H}{\partial D}$ can be parallelized efficiently. The same method also applies to $\frac{\partial H}{\partial d}$, $\frac{\partial H}{\partial C}$, and $\frac{\partial H}{\partial c}$.

4.5 EXPLORING THE SPARSITY

Some care is required when coding the calculations for which eq. (13) provides the models. With $X = (x_1, \ldots, x_T)$ as above, and the corresponding $D = (D_1, \ldots, D_T)$, the quantity $\frac{\partial D}{\partial X}$ suggests a huge structure involving T^2 submatrices of the general form $\frac{\partial D_t}{\partial x_{t'}}$. However, the definitions in eq. (2) show that any such submatrix in which $t' \neq t$ will be zero. Thus the quantity $\frac{\partial D}{\partial X}$ shown above never appears explicitly in our implementation. Instead, we work directly with the information-bearing blocks $\frac{\partial D_t}{\partial x_t}$, $1 \leq t \leq T$.

4.6 FORWARD ALGORITHM

It can be costly to evaluate matrices like $\frac{\partial D}{\partial \theta}$. In Pytorch, for example, such tools such as `torch.autograd.jacobian` rely on backpropagation, which means that gradient information from one time step is not reused for the next time step. However, the derivation above makes it clear that knowing $\frac{\partial D_{t-1}}{\partial \theta}$ allows for a direct calculation of $\frac{\partial D_t}{\partial \theta}$.

We now propose an efficient **forward approach** that uses available information efficiently to accelerate later steps. We refer to $\frac{\partial D_t}{\partial \theta}$ from (13) as $\nabla_\theta D_t$ here for clarity and to distinguish it from other gradient notations, a convention we apply similarly to other gradients such as $\frac{\partial d_t}{\partial \theta}$.

Given a trajectory satisfying $x_{t+1} = f_t(x_t, u_t, \theta)$, the matrices D_t and d_t defined in eq. (2) are functions of x_t, u_t, and θ. For time step t, we will have

$$\nabla_\theta D_t = \frac{\partial D_t}{\partial \theta} + \left[\frac{\partial D_t}{\partial x_t} + \frac{\partial D_t}{\partial u_t}\frac{\partial u_t}{\partial x_t}\right]\nabla_\theta x_t \tag{14}$$

with

$$\nabla_\theta x_t = \frac{\partial x_t}{\partial \theta} + \left[\frac{\partial x_t}{\partial x_{t-1}} + \frac{\partial x_t}{\partial u_{t-1}}\frac{\partial u_{t-1}}{\partial x_{t-1}}\right]\nabla_\theta x_{t-1}, \tag{15}$$

where $\frac{\partial D_t}{\partial \theta}$, $\frac{\partial D_t}{\partial x_t}$, $\frac{\partial D_t}{\partial u_t}$ and $\frac{\partial x_t}{\partial \theta}$, $\frac{\partial x_t}{\partial x_{t-1}}$, $\frac{\partial x_t}{\partial u_{t-1}}$ are analytically calculated in first so that on each time step we only need to instantly plug in the corresponding parameter values to obtain the numerical gradients. $\frac{\partial u_t}{\partial x_t}$ and $\frac{\partial u_{t-1}}{\partial x_{t-1}}$ are the linear control gain solved from FT-LQR. $\nabla_\theta x_{t-1}$ is the stored information from time step $t-1$ and reused here, and $\nabla_\theta x_t$ is prepared for the next time step $t+1$. Finally

$$\nabla_\theta d_t = \nabla_\theta x_{t+1} - \frac{\partial D_t}{\partial \theta}\begin{bmatrix} x_t \\ u_t \end{bmatrix} - D_t \begin{bmatrix} I \\ \frac{\partial u_t}{\partial x_t} \end{bmatrix} \nabla_\theta x_t, \quad \nabla_{x_t} d_t = -\frac{\partial D_t}{\partial x_t}\begin{bmatrix} x_t \\ u_t \end{bmatrix}, \quad \nabla_{u_t} d_t = -\frac{\partial D_t}{\partial u_t}\begin{bmatrix} x_t \\ u_t \end{bmatrix}. \tag{16}$$

The calculation of $\nabla_\theta C_t$ and $\nabla_\theta c_t$ is similar.

Algorithm 1 Forward Algorithm

Input: $\frac{\partial D_t}{\partial \theta}$, $\frac{\partial D_t}{\partial x_t}$, $\frac{\partial D_t}{\partial u_t}$ and $\frac{\partial x_t}{\partial \theta}$, D_t
Initialize variables $\nabla_\theta x_0 = 0$
for time step $t = 1, 2, \ldots, T$ **do**
    obtain $\nabla_\theta x_t$ through (15)
    obtain $\nabla_\theta D_t$ with $\nabla_\theta x_t$ and (14), and obtain $\nabla_\theta d_t$ with $\nabla_\theta x_t$ and (16)
end for
return $\nabla_\theta D$, $\nabla_\theta d$

4.7 Methodological Comparison and Discussion

Differences between our method and DiffMPC Amos et al. (2018) DiffMPC treats input X^* and U^* as constant and uses auto-differentiation to obtain $\frac{\partial D}{\partial \theta}$, and finally use the chain rule to obtain the derivative of the optimal trajectory. We improve DiffMPC by further considering the input X^* and U^* as a function of θ, that is, $X^*(\theta)$ and $U^*(\theta)$, and leverage implicit differentiation on the fixed-point to ***solve*** the exact analytical gradient, improving the accuracy of the gradient. The box in 27 illustrates the differences between the two approaches

$$A^i(\tau^i, \theta) = \left.\frac{\partial f(x, u, \theta)}{\partial x}\right|_{\tau^i}, \quad \nabla_\theta A^i = \frac{\partial A^i}{\partial \theta} + \boxed{\frac{\partial A^i}{\partial \tau^i}\frac{\partial \tau^i}{\partial \theta}}. \tag{17}$$

5 Experiments

We follow the examples and experimental setups from previous works Amos et al. (2018); Jin et al. (2020); Xu et al. (2024a); Watter et al. (2015) and conduct experiments on two well-known control benchmarks: CartPole and Inverted Pendulum. The experiments demonstrate our method's computational performance (at most **128x speedup**) and superior learning performance (**10^6** improvement). All experiments were carried out on a platform with an AMD 3700X 3.6GHz CPU, 16GB RAM, and an RTX3080 GPU with 10GB VRAM. The experiments are implemented with Pytorch Paszke et al. (2019).

5.1 Computational Performance

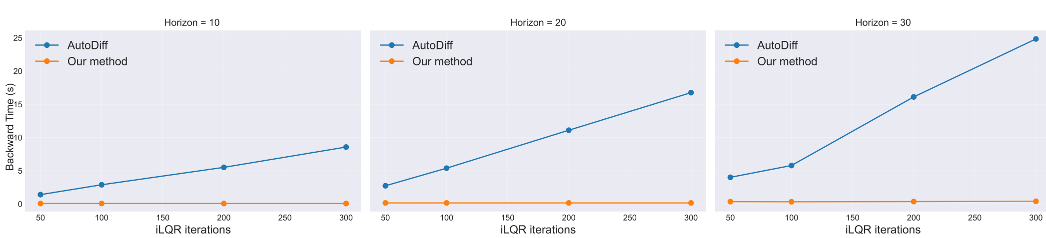
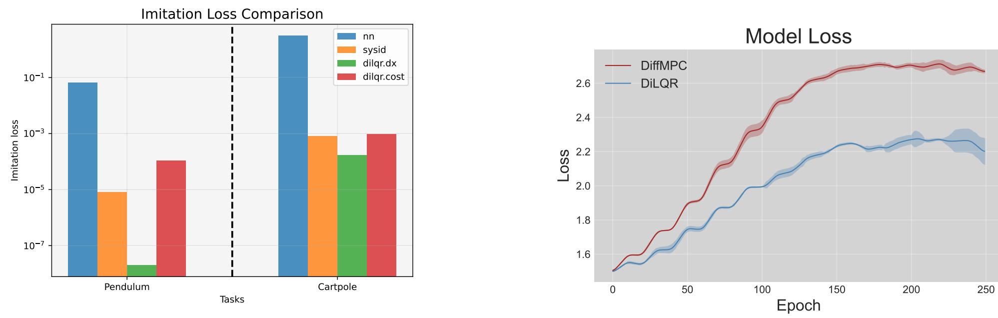


Figure 2: Backward computation time comparison between AutoDiff and our proposed method across different iLQR iterations and LQR horizons. AutoDiff's computation time scales linearly with the number of iterations, while our method maintains constant computation time. The experiments are conducted under pendulum domain, with batch size 20.

The performance of our differentiable iLQR solver is shown in Figure 2. We compare it to the naive approach, where the gradients are computed by differentiating through the entire unrolled chain of



(a) (b)

Figure 3: (a) Learning results on the pendulum and cartpole. We select the best validation loss observed during the training run and report the corresponding test loss. Every data point is averaged over five trials. (b) Comparison of cost function parameter estimation between our method and DiffMPC under the cartpole and cost learning domain.

iLQR. The results of the experiments clearly demonstrate the significant computational advantage of our method over AutoDiff across all configurations.

Backward pass efficiency: For example, for a horizon of 10 and 300 iterations, AutoDiff takes 8.57 seconds compared to just 0.067 seconds with our method, resulting in a **128x speedup**. Even in the case with the smallest improvement—horizon of 10 and 50 iterations, AutoDiff takes 1.41 seconds, while our method remains 0.067 seconds, still delivering a **21x speedup**. These results highlight the clear scalability and efficiency of our method, maintaining a near-constant computation time as the number of iLQR iterations increases, while AutoDiff's time grows significantly with longer horizons and more iterations.

5.2 Imitation Learning

Imitation learning recovers the cost and dynamics of a controller through ***only actions***. Similarly to Amos et al. (2018), we compare our approach with **Neural Network (NN)**: An LSTM-based approach that takes the state x as input and predicts the nominal action sequence, directly optimizing the imitation loss directly; **SysId**: Assumes that the cost of the controller is known and approximates the parameters of the dynamics by optimizing the next-state transitions; and DiffMPC Amos et al. (2018). We evaluated two variations of our method: diLQR.dx: Assumes that the cost of the controller is known and approximates the parameters of the dynamics by directly optimizing the imitation loss; diLQR.cost: Assumes that the dynamics of the controller are known and approximates the cost by directly optimizing the imitation loss. For more experimental details, please refer to the Appendix.

Imitation Loss: In Figure 3a, we compare our method with NN and Sysid using imitation loss. Notably, our method performs the best in the dx mode across both tasks, achieving a performance improvement of orders of magnitude—10^6 and 10^4—over the NN. In the dcost mode, our method is also dozens of times stronger than the NN but slightly weaker than Sysid. This is because Sysid directly leverages a system model with state estimates, while imitation learning relies solely on action data, which contains less information. The fact that our method achieves comparable results to Sysid in this mode demonstrates its effectiveness.

Model Loss: In Figure 3b, we compare the model error learned from our approach to that of DiffMPC. Model loss is defined as the MSE$(\theta - \hat{\theta})$, where θ represents the parameters of the cost function. Since learning in the cost mode is particularly challenging, we chose it as the case to demonstrate model loss. In the dcost mode, our approach recovers more accurate model parameters than DiffMPC, reducing model loss by 18%, indicating an improvement over our analytical results.

5.3 Visual control

We next explore a more complex, high-dimensional task: controlling an inverted pendulum system using images as input.

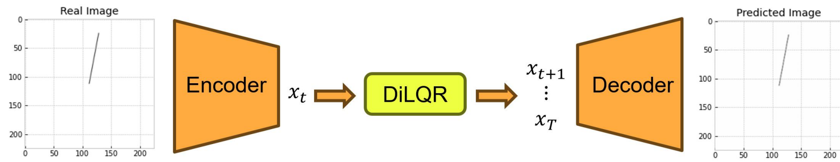


Figure 4: Diagram of the end-to-end control architecture. The encoder maps the compressed set of four input frames to the physical state variables (e.g., position, velocity). The differentiable iLQR then steps the state forward using the encoder's parameters. The decoder takes the predicted state and generates a future frame to match the true future observation.

In this task, the state of the pendulum is visualized by a rendered line starting from the center of the image, with the angle representing the position of the pendulum. The objective is to swing up the underactuated pendulum from its downward resting position and balance it. The network architecture consists of a mirrored encoder-decoder structure, each with five convolutional or transposed convolutional layers, respectively. For further architectural details, please refer to the Appendix. To capture the velocity information, we stack four compressed images as input channels. An example of these observations and reconstructions is provided in Figure 4.

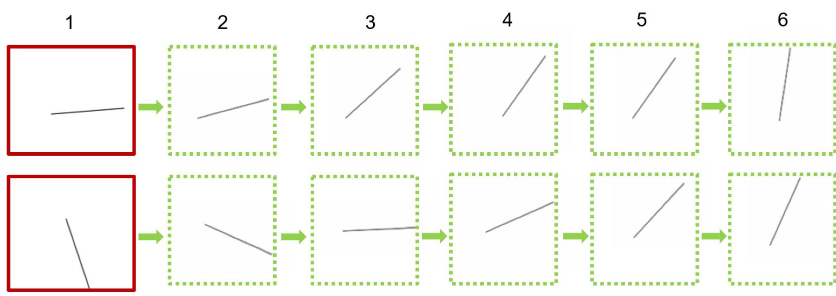


Figure 5: Imagined trajectory in the pendulum domain. The first image (red) represents the real input, while the following images are "dreamed up" by our model based on the initial image.

Our modular approach handles the coordination between the controller and decoder seamlessly. Figure 5 shows sample images drawn from the task depicting a trajectory generated by our system. In this scenario, the system is given just one real image and, with the help of DiLQR, it can output a sequence of predicted images, which closely approximate the actual trajectory of the pendulum.

6 Discussion

In this paper, we focus on the theoretical aspects of differentiable control methods. While our experiments are based on simpler control tasks, the advantages of our approach promise to extend to more complex, real-world applications. Many prior works Amos et al. (2018); Watter et al. (2015); Xu et al. (2024a); Jin et al. (2020) also rely on such toy examples to demonstrate foundational concepts.

One promising direction is embedding our differentiable controller into reinforcement learning (RL) frameworks. For instance, it could be integrated into a policy network and trained using an actor-critic approach, enabling more efficient policy updates. With its ability to propagate gradients through the control process, our method could enhance RL's performance, potentially achieving state-of-the-art results in more advanced tasks.

7 Conclusions

In this work, we introduced DiLQR, an efficient framework for differentiating through iLQR using implicit differentiation. By providing an analytical solution, our method eliminates the overhead of iterative unrolling and achieves O(1) computational complexity in the backward pass, significantly improving scalability. Experiments demonstrate that DiLQR outperforms existing methods in both runtime and learning performance, making it a promising approach for real-time control applications.

REFERENCES

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. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

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

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.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), 2017. doi: 10.1609/aaai.v31i1.10916.

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

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.

J Brewer. The derivative of the riccati matrix with respect to a matrix. *IEEE Transactions on Automatic Control*, 22(6):980–983, 1977.

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

Jianyu Chen, Wei Zhan, and Masayoshi Tomizuka. Constrained iterative lqr for on-road autonomous driving motion planning. In *2017 IEEE 20th International conference on intelligent transportation systems (ITSC)*, pp. 1–7. IEEE, 2017.

Sheng Cheng, Minkyung Kim, Lin Song, Chengyu Yang, Yiquan Jin, Shenlong Wang, and Naira Hovakimyan. Difftune: Auto-tuning through auto-differentiation. *IEEE Transactions on Robotics*, 2024.

Suyoung Choi, Gwanghyeon Ji, Jeongsoo Park, Hyeongjun Kim, Juhyeok Mun, Jeong Hyun Lee, and Jemin Hwangbo. Learning quadrupedal locomotion on deformable terrain. *Science Robotics*, 8(74):eade2256, 2023.

Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021.

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)*, pp. 638–644. IEEE, 2022.

Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. On the sample complexity of the linear quadratic regulator. *Foundations of Computational Mathematics*, 20(4):633–679, 2020.

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)*, pp. 465–472, 2011.

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

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

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*, pp. 1861–1870. PMLR, 2018.

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.

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

Peter Karkus, Boris Ivanovic, Shie Mannor, and Marco Pavone. Diffstack: A differentiable and modular control stack for autonomous vehicles. In *Conference on robot learning*, pp. 2170–2180. PMLR, 2023.

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

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, pp. 222–229. SciTePress, 2004.

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

Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2536–2542. IEEE, 2020.

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.

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.

Masashi Okada, Luca Rigazio, and Takenobu Aoshima. Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*, 2017.

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.

Marcus Pereira, David D Fan, Gabriel Nakajima An, and Evangelos Theodorou. MPC-inspired neural network policies for sequential decision making. *arXiv preprint arXiv:1802.05803*, 2018.

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

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

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

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)*, pp. 10020–10027. IEEE, 2023.

Damoon Soudbakhsh, Anuradha M Annaswamy, Yan Wang, Steven L Brunton, Joseph Gaudio, Heather Hussain, Draguna Vrabie, Jan Drgona, and Dimitar Filev. Data-driven control: Theory and applications. In *2023 American Control Conference (ACC)*, pp. 1922–1939. IEEE, 2023.

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.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International conference on machine learning*, pp. 4732–4741. PMLR, 2018.

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

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

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)*, pp. 336–343. IEEE, 2017.

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

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*, pp. 5026–5033. IEEE, 2012.

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, pp. 2094–2100, 2016. doi: 10.1609/aaai.v30i1.10295.

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.

Xuesu Xiao, Tingnan Zhang, Krzysztof Choromanski, Edward Lee, Anthony Francis, Jake Varley, Stephen Tu, Sumeet Singh, Peng Xu, Fei Xia, et al. Learning model predictive controllers with real-time attention for real-world navigation. *arXiv preprint arXiv:2209.10780*, 2022.

Zhaoming Xie, C Karen Liu, and Kris Hauser. Differential dynamic programming with nonlinear constraints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 695–702. IEEE, 2017.

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

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*, 2024b.

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.

Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pp. 5739–5743, 2018.

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

Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pp. 737–744. IEEE, 2020.

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.

A APPENDIX / SUPPLEMENTAL MATERIAL

A.1 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:*

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

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

$$\begin{aligned} (I - F_X)\frac{dX^*}{d\theta} - F_U \frac{dU^*}{d\theta} &= F_\theta \\ -G_X \frac{dX^*}{d\theta} + (I - G_U)\frac{dU^*}{d\theta} &= G_\theta \end{aligned} \tag{19}$$

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

$$\begin{aligned} \frac{dX}{d\theta} &= (I - F_X)^{-1}(F_\theta + F_U \frac{dU}{d\theta}) \\ &= M(F_\theta + F_U \frac{dU}{d\theta}) \end{aligned} \tag{20}$$

Then, substituting 20 into the second equation of 19 to obtain an equation with respect to only $\frac{dU}{d\theta}$:

$$-G_X(M(F_\theta + F_U \frac{dU}{d\theta})) + (I - G_U)\frac{dU^*}{d\theta} = G_\theta \tag{21}$$

Solving equation 21 will give the solution to $\frac{dU^*}{d\theta}$:

$$\frac{dU}{d\theta} = (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta) \tag{22}$$

Substituting 22 into 20, the solution to $\frac{dX}{d\theta}$ can be obtained:

$$\frac{dX}{d\theta} = M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta)) \tag{23}$$

This completes the proof. □

A.2 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 Amos et al. (2018), 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.

A.3 Detailed Network Architecture

Encoder The encoder is a neural network designed to encode input image sequences into low-dimensional state representations. It is implemented as a subclass of `torch.nn.Module`, and consists of five convolutional layers and a regression layer:

- **Convolutional layers**: Each layer applies 2D convolutions, followed by batch normalization, ReLU activations, and max pooling. These operations progressively reduce the spatial dimensions of the input image.
- **Regression layer**: After the final convolutional layer, the output is flattened and passed through three fully connected layers, mapping the extracted features to the desired output dimension, which represents the system state.

The forward pass takes an input tensor of shape `[batch, 12, 224, 224]` (representing four stacked RGB images) and processes it through the convolutional layers. The output is a state vector of shape `[batch, out_dim]`.

Decoder The decoder mirrors the structure of the encoder and is also a subclass of `torch.nn.Module`. It reconstructs images from the low-dimensional state vector. The decoder consists of five transposed convolutional layers followed by a regression layer:

- **Transposed convolutional layers**: These layers progressively upsample the input, applying batch normalization and ReLU activations after each layer to restore the spatial dimensions.
- **Regression layer**: This layer, consisting of three fully connected layers, transforms the low-dimensional input vector into a form suitable for the initial transposed convolution.

The forward pass takes a state vector of shape `[batch, 3]` as input, upscales it through the transposed convolution layers, and outputs a reconstructed image tensor of shape `[batch, 3, 224, 224]`. A Sigmoid activation is applied to ensure the pixel values remain within the range `[0, 1]`.

A.4 Advantages of fixed-point method

A.4.1 Analytically discussion

In this section, we discuss how our method differs from non-fixed-point method (e.g. Amos et al. (2018)) and why our gradient is the accurate one. Given nonlinear dynamics $f_\theta(x,u)$, in the ith iteration, iLQR linearizes $f_\theta(x,u)$ around the trajectory τ^{i-1}

$$A^i_\theta = \frac{\partial f_\theta(x,u)}{\partial x}\bigg|_{\tau^{i-1}}, B^i_\theta = \frac{\partial f_\theta(x,u)}{\partial u}\bigg|_{\tau^{i-1}}. \tag{24}$$

Without loss of generality, the following discussion focuses on the backpropagation through A.

The goal of differentiable iLQR is to calculate $\frac{\nabla L}{\nabla \theta}$. The non-fixed-point method naturally uses chain rule

$$\frac{\nabla L}{\nabla \theta} = \frac{\partial L}{\partial \tau^i} \underbrace{\frac{\partial \tau^i}{\partial A^i_\theta} \frac{\nabla A^i_\theta}{\nabla \theta}}_{\frac{\partial \tau^i}{\partial \theta}}. \tag{25}$$

(25) is mathematically correct, but it is impractical. A^i_θ is not only a parameterized function of θ, but also a function of τ^{i-1}_θ, and τ^{i-1}_θ is also a function of θ, since it is the output of previous layers. Consequently,

$$\frac{\nabla A^i_\theta}{\nabla \theta} = \frac{\partial A^i_\theta}{\partial \theta} + \frac{\partial A^i_\theta}{\partial \tau^{i-1}} \frac{\partial \tau^{i-1}}{\partial \theta}. \tag{26}$$

For the final layer that outputs fixed-point, (26) would be written as

$$\frac{\nabla A^i_\theta}{\nabla \theta} = \frac{\partial A^i_\theta}{\partial \theta} + \boxed{\frac{\partial A^i_\theta}{\partial \tau^i} \frac{\partial \tau^i}{\partial \theta}} \tag{27}$$

which would drive us back to $\frac{\partial \tau^i}{\partial \theta}$, the thing we indeed want to derive. What non-fixed-point method does is treating τ^i as a constant, in the following way

$$A_\theta^i = \frac{\partial f_\theta(x,u)}{\partial x}\bigg|_{\tau^i}. \tag{28}$$

When taking gradient, they only consider $\frac{\partial A_\theta^i}{\partial \theta}$ that explicitly appears in the matrix. In a word, non-fixed-point method treats A as $A(\theta, \tau)$, while our method treats it as $A(\theta, \tau(\theta))$. Our main argument is that the accurate gradient is supposed to be ***'solved'***, instead of ***'multiplied'*** through the chain rule.

A.4.2 A CONCRETE EXAMPLE

Consider a non-quadratic two-step optimal control problem defined by the following objective function:

$$\begin{aligned} J &= \sum_{k=0}^{0} \theta(x_k)^4 + (u_k)^2 + \theta(x_T)^4 \\ \text{s.t.} \quad x_{k+1} &= Ax_k + Bu_k \\ x_0 &= [1,1]^\top \end{aligned} \tag{29}$$

where the matrices are specified as:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{30}$$

and $(x_k)^4$ denotes the sum of the element-wise fourth powers of vector x_k. The parameter θ is a scalar and is a learnable coefficient influencing the cost function.

Given the initial state x_0 and considering u_0 as the sole control variable, the system evolves through the states:

$$x_0 = [1,1]^\top, \quad x_1 = [2, u_0 + 1]^\top. \tag{31}$$

The optimization problem then reduces to minimizing the following equivalent function:

$$\min J = \min u_0^2 + \theta(u_0+1)^4. \tag{32}$$

For the sake of simplicity, we will drop the subscript from u_0 and refer to it as u. The derivative of (32) with respect to u, necessary to find the optimal control u^*, is given by:

$$2u + 4\theta(u+1)^3 = 0. \tag{33}$$

Solving this equation yields the analytical solution for u^*:

$$u^* = \frac{\sqrt[3]{9\theta^2 + \sqrt{3}\sqrt{27\theta^4 + 2\theta^3}}}{6^{2/3}\theta} - \frac{1}{\sqrt[3]{6}\sqrt[3]{9\theta^2 + \sqrt{3}\sqrt{27\theta^4 + 2\theta^3}}} - 1. \tag{34}$$

Given that the problem is convex and the solution is unique, iLQR algorithms would converge to the optimal solution (34). The resulting optimal trajectory (u^*, x_1^*) is refered to as a fixed-point in the iLQR context.

In the linear approximation of the cost function for state $x_1 = [x_1^0, x_1^1]$, the expansion around the fixed-point state $x_1^* = [x_1^{0*}, x_1^{1*}]$ is:

$$\begin{aligned} (x_1)^4 \approx\ & \theta(x_1^0)^4|_{x_1^{0*}} + c_1^0(x_1^0 - x_1^{0*}) + C_1^0(x_1^0 - x_1^{0*})^2 \\ & + \theta(x_1^1)^4|_{x_1^{1*}} + c_1^1(x_1^1 - x_1^{1*}) + C_1^1(x_1^1 - x_1^{1*})^2 \end{aligned} \tag{35}$$

with constants:

$$c_1^0 = 4\theta(x_1^0)^3|_{x_1^{0*}}, \quad C_1^0 = 12\theta(x_1^0)^2|_{x_1^{0*}}, c_1^1 = 4\theta(x_1^1)^3|_{x_1^{1*}}, \quad C_1^1 = 12\theta(x_1^1)^2|_{x_1^{1*}}. \tag{36}$$

One of the core part in differentiable iLQR is to derive the gradient of cost functions c, C and dynamics A, B with respect to the learnable parameters. In our case, this turns to sensitivity analysis $\frac{\nabla c}{\nabla \theta}$ and $\frac{\nabla C}{\nabla \theta}$. Without loss of generality, we particularly study about $\frac{\nabla c_1^1}{\nabla \theta}$.

In non-fixed-point method, the following formulation is used

$$\frac{\nabla c_1^1}{\nabla \theta} = 4(x_1^{1*})^3. \tag{37}$$

The formulation is straightforward, however, it ignores the relation between c_1^1 and u^*. Even though u^* is a fixed-point for the iLQR, a different u^* would still result in a different c_1^1. Consequently, we argue that the correct formulation suppose to be

$$\frac{\nabla c_1^1}{\nabla \theta} = 4(x_1^{1*})^3 + \theta \frac{\nabla 4(x_1^{1*})^3}{\nabla x_1^{1*}} \frac{\nabla x_1^{1*}}{\nabla u} \frac{\nabla u}{\nabla \theta} = 4(x_1^{1*})^3 + \theta \frac{\nabla 4(x_1^{1*})^3}{\nabla x_1^{1*}} \frac{\nabla u}{\nabla \theta}. \tag{38}$$

In order to obtain $\frac{\nabla u}{\nabla \theta}$ on u^*, take difference for (33)

$$\begin{aligned} 2du + 4d\theta(u+1)^3 + 12\theta(u+1)^2 du = 0 \\ \rightarrow \frac{\nabla u}{\nabla \theta} = -\frac{2(u^*+1)^3}{6\theta(u^*+1)^2+1}. \end{aligned} \tag{39}$$

Plugin it to (38), we will have

$$\frac{\nabla c_1^1}{\nabla \theta} = 4(x_1^{1*})^3 - \frac{24\theta(u^*+1)^5}{6\theta(u^*+1)^2+1}. \tag{40}$$

To illustrate how huge difference the correction term can give, we use finite difference method as a baseline to calculate $\frac{\nabla c_1^1}{\nabla \theta}$, and plot the values of the gradients with θ.

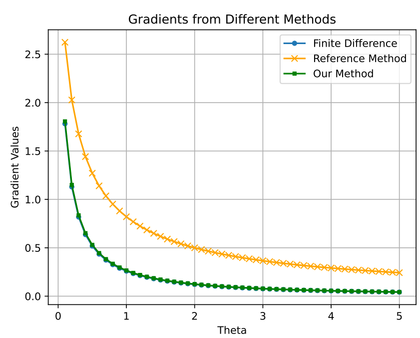


Figure 6: Comparison of gradients from different methods. Our method (squares) and the finite difference method (circles) produce nearly identical curves, but the two lines can still be distinguished by their different markers.