Learning Legged MPC with Smooth Neural Surrogates

Samuel A. Moore[1], Easop Lee[2], and Boyuan Chen[1,2,3]

***Abstract*—Reinforcement learning (RL) and model predictive control (MPC) offer complementary strengths for legged robotics; yet, combining them remains challenging due to incompatibilities between stiff contact dynamics, neural networks, and gradient-based optimization. These difficulties are amplified when dynamics are learned with neural networks: (1) stiff transitions may be inherited, (2) additional non-physical local nonsmoothness can arise, and (3) datasets can produce heavy-tailed model errors from rapid state changes. We address (1) and (2) by introducing the smooth neural surrogate, a neural network with tunable smoothness designed to provide more informative derivatives for trajectory optimization. To address (3), we train these smooth neural surrogates using a heavy-tailed likelihood that better matches empirical error distributions. Using these techniques, we improve the control cost of gradient-based MPC by $\sim$3-10$\times$ compared to standard neural dynamics, and achieve $\sim$2$\times$ lower control cost in $\sim$4$\times$ fewer training steps than with standard Gaussian likelihoods. Overall, gradient-based methods with smooth neural surrogates obtain $\sim$2-10$\times$ lower cost than sampling-based methods while scaling more favorably to high-dimensional optimization problems and large-scale reinforcement learning. Finally, we transfer learned dynamics and state estimator modules to a quadruped robot, synthesizing new behaviors on unseen terrains with MPC.**

***Index Terms*—Reinforcement Learning, Model Predictive Control, Legged Robots, Neural Networks, Contact Dynamics**

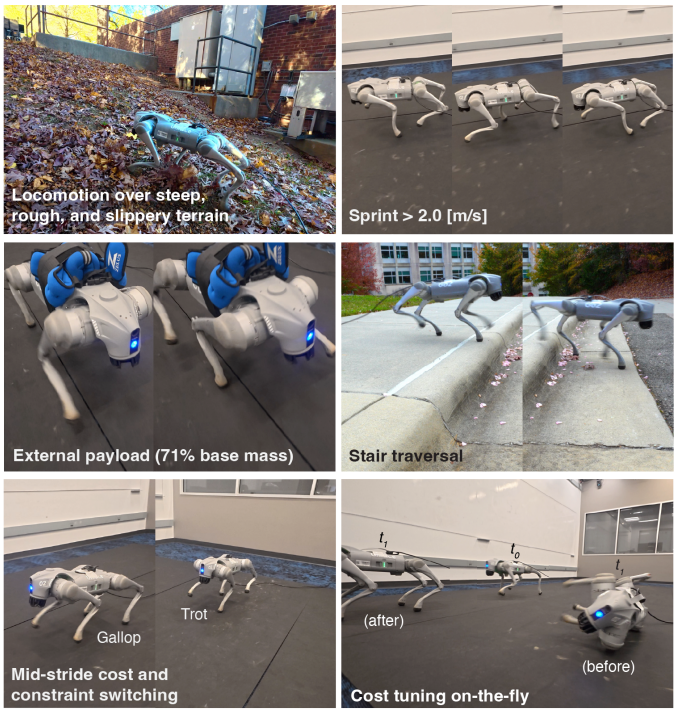

Fig. 1. **Learning model predictive control and state estimation for legged robots.** Stiff legged-robot dynamics, when modeled with standard neural networks, introduce nonsmooth behavior that destabilizes learning and gradient-based MPC. Motivated by smooth approximations of nonsmooth functions and by robust optimization, we introduce a lightweight neural architecture and a heavy-tailed likelihood that provides stable, informative gradients through contact events. Smooth neural surrogates learn terrain-varying dynamics and state estimation and generalize to new tasks at test time with real-time MPC. Together, they combine the representational flexibility of deep learning with the task-level adaptability of MPC, enabling reliable whole-body control across diverse behaviors and environments without additional training.

I. Introduction

REINFORCEMENT learning and model predictive control have seen widespread success across legged robotics. On one hand, learning offers unparalleled representational flexibility and versatility through techniques like domain randomization, which enables tasks like blind locomotion over challenging terrain [1], [2]. On the other hand, MPC can reshape behavior by adjusting costs and constraints online, making it appealing for rapid task switching on hardware [3], [4]. Despite their complementary strengths, RL and MPC remain difficult to use in tandem. Neural network policies are tightly coupled to specific tasks, reward functions, and training curricula; modifying behavior usually requires retraining or fine-tuning. Classical MPC, by contrast, can adapt behavior simply by changing the objective or constraints, but relies on analytic models that are inherently scenario-specific. As a result, many successful controllers either sacrifice some task-level flexibility (policy learning) or rely on hand-derived models (classical MPC) that can limit their ability to generalize beyond the design scenarios.

Learned MPC. in which a model for the dynamics is learned, offers a natural way to bridge this gap: in principle, it could combine the ability of deep learning to represent rich, abstract world models with the online adaptability of MPC. However, simply improving the prediction error of a learned model does not guarantee better control performance or a higher reward [5]. In fact, even exact analytic models can lead to poor planning if the optimization problem is not formulated to handle contact dynamics [6]–[8].

Our starting point is a simple but important observation: *the pathologies of gradient-based optimization under contact dynamics are compounded when the dynamics are modeled with standard neural networks.* We hypothesize that these failures are driven primarily by nonsmoothness, which manifests in three ways:

1) Neural network dynamics often inherit the very stiff transitions present in the ground-truth contact dynamics, making these models difficult to optimize through.
2) Neural networks frequently introduce additional local

[1]Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC, USA.
[2]Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA.
[3]Department of Computer Science, Duke University, Durham, NC, USA.

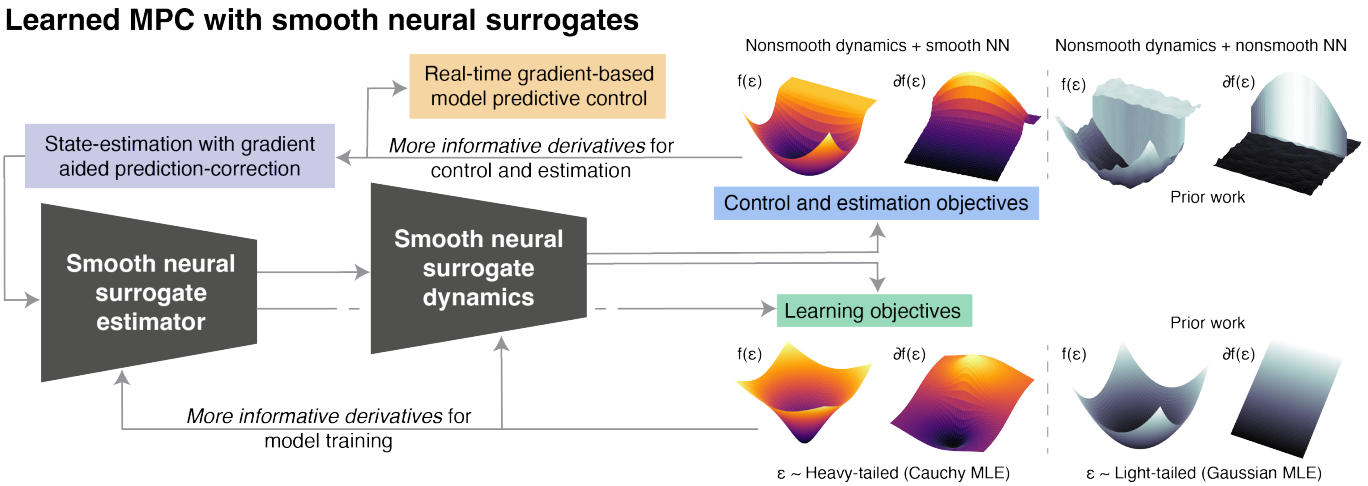

Fig. 2. **Our framework for transferable model-based reinforcement learning for legged robots.** The backbone of our method consists of dynamics and estimator modules with an architecture we introduce called the *smooth neural surrogate*. The smooth neural surrogate provides informative derivatives for planning and state estimation, even in the presence of nonsmooth dynamics and out-of-distribution data. We train both modules via Cauchy maximum-likelihood estimation (MLE) and show that heavy-tailed errors are common in legged-robot dynamics. Finally, we introduce a gray-box model predictive controller and model-based state-estimation strategy that exploit the smooth derivatives provided by our learned dynamics.

nonsmoothness and, consequently, local optima that do not exist in the underlying physical dynamics.

3) Training datasets for these stiff dynamics necessarily contain rapid changes in states, which can destabilize learning.

A promising direction for addressing (1) and (2) is to explicitly control neural network smoothness. Smooth architectures and Lipschitz-constrained networks can yield better gradients for test time optimization, improved generalization, and robustness to perturbations [9]–[13]. Yet, these ideas have seen limited use in control and robotics [14], [15], especially for learning stiff legged robot dynamics. Addressing 3) requires checking the assumptions behind the loss function and the implied likelihood model. In practice, Gaussian error assumptions are common [5], [16]–[22], but they are often unverified and inappropriate for datasets that introduce heavy-tailed errors. Cauchy noise models are known to perform significantly better when the errors are impulsive in nature [23].

In this paper, we study these pathologies through full-order, single-shooting MPC for legged robots, with both the dynamics and the state estimator learned from scratch. Single shooting is the simplest form of trajectory optimization: an open-loop control sequence $u(t)$ is parameterized over a planning horizon, the dynamics are rolled out forward in time, and an optimizer iteratively updates $u(t)$ to minimize a cost. In principle, this technique can be directly paired with any general-purpose gradient-based optimizer [24]. In practice, however, gradient-based single shooting is notorious for poorly conditioned gradients, sensitivity to local minima, and nonsmoothness. [7], [25]. Consequently, sampling-based variants have become a popular choice for single shooting when local optima or nonsmoothness are a concern [4], [5], [17], [22], [26]–[28]. The simplicity and generality of both gray-box (gradient-based) and black-box (sampling-based) single shooting make them pragmatic choices for neural MPC. They preserve the full representational flexibility of deep learning and impose no assumptions on the dynamics model (rigid-body, Markov, black-box, etc.) or on the structure of the optimization problem (stage-wise costs, constraints, and so on).

This context raises two specific motivating questions: (1) Can smooth neural dynamics and robust loss functions improve training and make gradient-based single shooting highly effective for learning legged MPC? and (2) if so, what benefits do these design choices provide relative to ignoring derivatives and using zeroth-order MPC with standard neural functions?

We answer these questions by extending smooth neural architectures for use as dynamics models in online gradient-based MPC through contact. We introduce the smooth neural surrogate (SNS): a multilayer perceptron (MLP) with learned layer-wise Lipschitz constants and a global "smoothness budget" that constrains first- and/or second-order behavior (Section V). We couple this with robust maximum-likelihood estimation (MLE) that better matches the empirical data distribution and improves training stability and prediction error (Section VI). We pair these dynamics with a predictor–corrector style state estimator and train both SNS modules under domain randomization (Section VI). We then deploy the learned models in a single-shooting framework (Sections VI-VII), and show that, with appropriately smooth dynamics, gradient-based shooting can outperform strong sampling-based baselines in terms of scalability, convergence, and computational efficiency (Section VII). Finally, we transfer these learned modules to hardware and synthesize new behaviors on unseen terrains without additional training (Fig. 1, Section VIII). Overall, our framework yields models that are expressive enough to capture variable-terrain legged robot dynamics, yet smooth enough to provide informative gradients for real-time single-shooting MPC through contact.

II. System Overview

Fig. 2 provides a high-level overview of our approach to learned, gradient-based MPC for legged robots under domain randomization and partial observability. An SNS-MLP estimator is trained to refine prior state estimates using corrections derived from the learned dynamics. The SNS-MLP dynamics, in turn, predicts future states using inputs from the estimator. Both modules are trained with heavy-tailed (Cauchy) maximum-likelihood estimation , which improves robustness to outliers and nonsmooth data. The estimator's smoothness acts purely as a regularizer, whereas the dynamics' smoothness is designed specifically to stabilize gradient-based single-shooting through contact.

III. Contributions

A summary of our primary contributions are:

1) We introduce the smooth neural surrogate MLP: a network with tunable first- and second-order smoothness (Section V). We show that this architecture provides more informative derivatives for legged MPC through contact compared to standard neural baselines (Section VI).
2) We show that the fitted dynamics residuals in model-based reinforcement learning for legged robots can be heavy-tailed, and we detail the advantages of heavy-tailed likelihood estimation for robust learning (Section VI).
3) We demonstrate that, with appropriately smooth neural dynamics, gradient-based single-shooting can outperform strong sampling-based methods, even though the underlying dynamics are nonsmooth. (Section VII).
4) We learn full-order legged MPC and state estimation under domain randomization and deploy them on a real quadruped robot; synthesizing new behaviors in unseen environments without additional training (Section VIII).

IV. Related Work

A. Model-Based Control of Hybrid Systems

Contact dynamics are hybrid, with smooth modes separated by discrete jumps that can be non-differentiable or have stiff gradients. Broadly, existing approaches address this either by avoiding gradients or by smoothing the underlying dynamics to recover informative derivatives.

Sampling-based single-shooting methods exemplify the gradient-free strategy. They have become increasingly popular due to their ease of implementation and GPU parallelization, and recent advances have enabled deployment on quadrupeds with full-order rigid-body models [26], [29]. Much of this progress comes from reducing the dimensionality of the decision variables through spline-based action parameterizations [4], [26], [29]. Nevertheless, sampling suffers from the curse of dimensionality, heuristic convergence, and high computational costs. From a numerical optimization perspective, well-conditioned gradients could offer faster convergence, lower computational costs, and better scalability [24], [30], [31].

Other gradient-based approaches avoid differentiating through contact by introducing explicit structure to the optimization problem. Multi-phase methods require predefined contact modes [7], [32], while direct trajectory optimization through contact optimizes over contact forces that satisfy complementarity constraints [6], [7]. These techniques assume rigid-body dynamics and depend on accurate contact modeling, which can be difficult to obtain in practice and is not strictly necessary for effective planning or when using neural networks [28]. When various components are swapped with neural networks [33], these methods become complementary to ours.

Our work is more closely aligned with methods that smooth contact dynamics to enable gradient-based planning. Many differentiable-simulation approaches achieve this by relaxing complementarity conditions or regularizing contact forces [8], [34]–[38]. Such smoothing has been effective, but typically relies on analytic, scenario-specific dynamics. More general smoothing strategies, such as randomized smoothing [39], [40], can, in principle, be applied to arbitrary models, including neural networks; however, they incur substantial computational costs and can yield noisy derivatives due to their stochastic nature [35].

Smoothing has similarly been shown to benefit policy learning via differentiable simulation by regularizing contact forces and truncating the planning horizon [41], [42]. Many of the methods discussed here can directly benefit from replacing analytical or learned components with SNSs for well behaved first- and second-order derivatives, making them complementary rather than competing approaches.

B. Neural Dynamics

Many works learn dynamics models but rely on tools borrowed from model-free RL, such as policy gradients, value functions, to enable planning and control [18], [43], [44]. While effective, these approaches reduce the task-level flexibility that online trajectory optimization provides. Their learned components could also benefit from smoother approximators, making them compatible with the type of architectural considerations we introduce.

In learned MPC, it is common to pair black-box neural dynamics with sampling-based optimizers to avoid difficulties associated with gradients or nonconvexity [5], [17], [22], [27], [45]. Other methods learn to plan with diffusion [44], but they still require model-free components, like PPO demonstrations for quadruped robots [46]. Other approaches introduce structure into the dynamics through contact-aware or physics-informed priors [20], [47]–[49], or simplify learning via reduced-order representations and fixed low-level controllers [21], [50]. Hybrid approaches blending simulation with learned modules [16], [33], [51] are also used. Our contribution is compatible with most of these formulations: smooth neural surrogates and heavy-tailed MLE, can replace standard neural networks and Gaussian MLE components without altering the surrounding optimization problems or physics-informed priors.

C. Smooth Neural Networks and Robust Optimization

Outside of robotics, many works have shown that bounding a network's Lipschitz constant can improve adversarial robustness and correlates with better generalization [9], [13], [52], [53]. However, many neural smoothing techniques are either computationally expensive or overly restrictive [9], [10], [12]. Strict 1-Lipschitz methods based on weight orthonormalization [11] can severely under-express dynamics and behave pathologically during optimization, while spectral-normalization approaches are approximate, sensitive to hyperparameters, and incur nontrivial overhead [13], [52]. Liu et al. propose the Lipschitz-MLP to address several challenges with smooth neural functions using weight normalization [9]. Our SNS-MLP is a direct extension of their approach with reparameterization to improve convergence properties and scalability with moderate-to-large sized networks under Lipschitz constraints.

In the context of trajectory optimization, only early variants of Lipschitz-constrained networks have been evaluated primarily in benign settings with smooth, low-dimensional dynamics (e.g., a 2D chemical process model) [15]. Their effectiveness for high-dimensional, stiff legged-robot dynamics and online trajectory optimization, therefore, remains largely untested. Although analogous Lipschitz-bounding techniques exist for more complex architectures [54], it is still unclear whether even simple models, like MLPs, meaningfully influence planning performance in trajectory optimization through contact. Moreover, practical strategies for Lipschitz-based weight normalization remain underexplored in more expressive network architectures. For these reasons, we restrict our study to standard MLPs, using them as a controlled setting to isolate the effects of smoothness and leaving broader architectural investigations to future work.

A large body of work in robust optimization also highlights that Gaussian losses such as mean squared error are often ill-suited for real-world regression because they implicitly assume light-tailed noise and finite variance—conditions that can be violated in practice [23], [55]. When data contain heavy-tailed residuals, or impulsive disturbances, Gaussian losses overweight large deviations and can drive estimators away from the true solution, motivating the use of robust alternatives (e.g., Cauchy, Huber, or other heavy-tailed likelihoods) that bound the influence of outliers and remain stable under non-Gaussian noise [23], [55]–[57]. However, Gaussian error models are widely used when learning dynamics for model-based RL, even when datasets are generated from impulsive contact dynamics [5], [17], [18], [28].

V. Smooth Neural Representations

A. Preliminaries

Gradient-based optimization. Many general-purpose optimization algorithms rely on *Lipschitz continuity* as a key ingredient in local convergence guarantees and, in practice, its importance to algorithm performance is well known [24]. A function f is c-Lipschitz if small input changes produce proportionally bounded output changes:

$$\|f(a_0) - f(a_1)\|_p \;\le\; c\,\|a_0 - a_1\|_p, \tag{1}$$

where $\|\cdot\|_p$ is the p-norm and $c \ge 0$ is the *Lipschitz constant*. Algorithms requiring some form of Lipschitz continuity include gradient descent and its stochastic or adaptive variants, as well as Newton, Gauss-Newton, and quasi-Newton methods [24], [58], [59].

Smooth neural networks. There is no general formula for directly measuring or shaping the Lipschitz constant of an arbitrary analytic function. However, smooth surrogates based on convolution can make a discontinuous function continuously differentiable, providing some qualitative control over smoothness but at the cost of requiring many additional samples at each evaluation point [39], [40].

Neural network architectures, by contrast, follow fixed analytic templates while remaining capable of approximating arbitrary functions, making quantitative assessments of smoothness more tractable. Thus, for neural networks, a more direct and efficient strategy to quantify and shape smoothness is to use the *Lipschitz upper bound*. For a fully connected network, or MLP, with L layers, the Lipschitz upper bound is given by

$$c_{\text{MLP}} \;\lesssim\; \prod_{\ell=1}^{L} \left\|W^{(\ell)}\right\|_p, \tag{2}$$

with proportionality determined by the activation functions; for 1-Lipschitz activations (e.g., softplus, tanh), the relation becomes exact. Although this bound is conservative, it provides meaningful quantitative insight into the global smoothness of the network and removes the need for post hoc randomized smoothing. Compared to the naive minimization of (2), weight normalization strategies have proven more fruitful [9].

B. Lipschitz-Based Weight Normalization

We base our approach to on Liu et al. [9], who introduce a weight-normalization layer paired with learned layerwise scalars c_ℓ that parameterize the ∞-norm of each weight matrix:

$$\begin{aligned} \hat{W}_{ij}^{(\ell)} &= \text{normalization}\left(W_{ij}^{(\ell)},\, c_\ell\right), \qquad c_\ell > 0, \\ &= \min\left(1,\, \frac{c_\ell}{\sum_k |W_{ik}^{(\ell)}|}\right) W_{ij}^{(\ell)}. \end{aligned} \tag{3}$$

Here, c_ℓ serves as the learned Lipschitz constant of the ℓ-th layer (ignoring the activation). Liu et al. enforce $c_\ell > 0$ using the parameterization $c_\ell = \text{softplus}(\theta_c^{(\ell)})$, where $\theta_c^{(\ell)}$ is the true trainable variable. The training objective is then augmented with a global Lipschitz regularization term via the upper bound $\prod_\ell c_\ell$.

C. Smooth Neural Surrogates

In practice, the learned layerwise Lipschitz constants c_ℓ are often large at initialization, causing their product to grow exponentially with network depth. As we show in our experiments, this can lead to slow convergence or even collapse when the model is pressured to satisfy strict smoothness constraints. We hypothesize that this issue stems from the parameterization

TABLE I
SMOOTH NEURAL SURROGATES QUANTITIES

Term	Definition
c_ℓ	Learned Lipschitz constant of layer ℓ
$c_{\text{MLP}} \lesssim C = \prod_\ell c_\ell$	Approx. Lipschitz upper bound
$S = \sum_\ell c_\ell \prod_{j<\ell} c_j$	Propagated curvature terms
$d_{\text{MLP}} \lesssim CS$	Approx. Lipschitz upper bound of the Jacobian
c_{ub}	Sensitivity budget
d_{ub}	Curvature budget

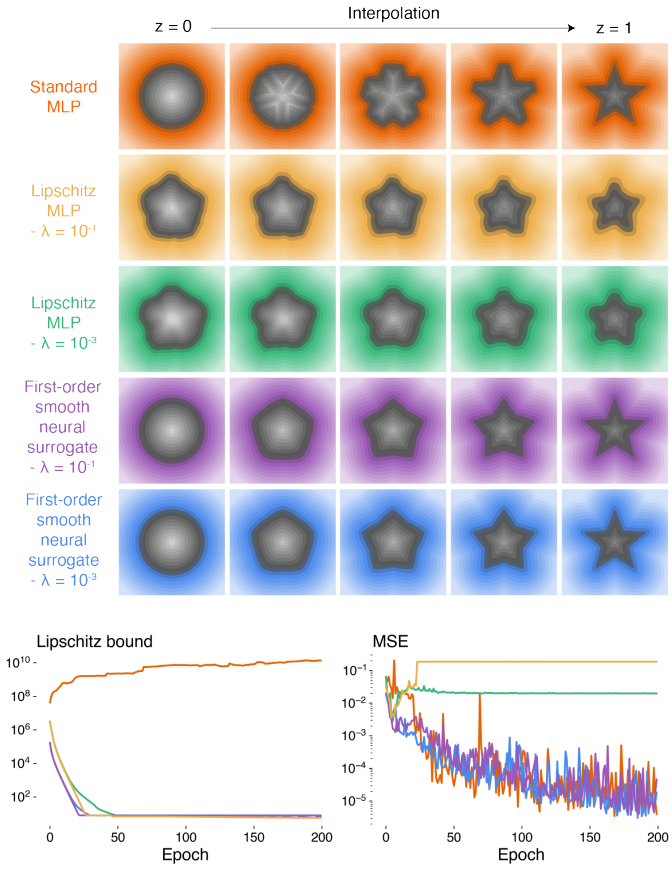

Fig. 3. **Smooth neural surrogates converge under strict Lipschitz constraints and enable zero-shot generalization. Top:** Lowest-MSE models for the 2-D shape interpolation task. Only the SNS converges under tight Lipschitz budgets; the Lipschitz MLP [9] collapses. **Middle:** Evolution of Lipschitz upper bounds and training MSE.

of c_ℓ. To mitigate it, we introduce an exponential parameterization $c_\ell = \exp(\theta_c^{(\ell)})$, which modifies the regularization objective to satisfy

$$\prod_\ell \exp(\theta_c^{(\ell)}) = \exp\Big(\sum_\ell \theta_c^{(\ell)}\Big) = \frac{\partial \exp\big(\sum_\ell \theta_c^{(\ell)}\big)}{\partial \theta_c^{(\ell)}}. \quad (4)$$

This parameterization is equivalent to learning $\theta_c^{(\ell)} = \log(c_\ell)$, and ensures that the regularization term remains well-scaled and that its gradient is distributed evenly across layers, regardless of network depth or width. As a result, the optimization does not saturate or collapse when enforcing smoothness, even for very smooth or very nonsmooth networks.

Lipschitz regularization penalizes only first-order nonsmoothness, which is not always sufficient for smoothing. For example, both ReLU and softplus are 1-Lipschitz, yet only softplus has bounded curvature. Motivated by this distinction, we additionally bound the second derivative of the learned network. We define the total Lipschitz bound $C := \prod_{\ell=1}^{L} c_\ell$ and the propagated terms $S := \sum_{\ell=1}^{L} c_\ell \prod_{j<\ell} c_j$ which account for curvature. With this notation, the Lipschitz constant of the Jacobian, d_{MLP}, of an L-layer MLP satisfies

$$d_{\text{MLP}} \lesssim C\,S. \quad (5)$$

We provide proof of this relation in Appendix A.

Smooth neural surrogate objectives. We define the k-th order SNS objective for $k \in \{1, 2\}$ as

$$\mathcal{L}_k = \mathcal{L}_{\text{main}} + \lambda \max\left(1, \frac{C\,S^{k-1}}{B_k}\right), \quad (6)$$

where B_k is the corresponding smoothness budget. Rescaling the numerator by B_k (rather than enforcing $\max(C\,S^{k-1}, B_k)$ directly) ensures that the regularization coefficient λ remains independent of the smoothness scale. For a first-order SNS ($k = 1$) and $B_1 = c_{\text{ub}}$ is the sensitivity budget. Likewise, for a second-order SNS ($k = 2$) and $B_2 = d_{\text{ub}}$ is the curvature budget. Clearly, the second-order objective also bounds first-order behavior, but one could also jointly enforce specific constraints $c_{\text{MLP}} \le c_{\text{ub}}$ and $d_{\text{MLP}} \le d_{\text{ub}}$ by using both forms of Eq. (6). As a guideline, a practitioner may prefer curvature suppression ($k = 2$) in settings where reliable second-order derivatives are needed, for example, Newton-type optimization, physics-informed machine learning, and differentiable optimization or simulation. Table I provides an overview of the SNS quantities for convenience.

D. Experiments and Examples

We evaluate the proposed SNS-MLP through a set of illustrative experiments and examples intended to build intuition for the behavior of smooth neural surrogates and to examine their optimization properties. Because our method is a direct extension of Liu et al., who already provide extensive empirical comparisons against alternative smooth neural parameterizations and regularization strategies, we do not aim to re-establish state-of-the-art performance here. Instead, we restrict our baselines to their Lipschitz MLP and a standard MLP. For the Lipschitz MLP baseline, we train using the SNS objective with $k = 1$ to enable a controlled comparison and to isolate the effect of the exponential parameterization of the layerwise Lipschitz constants.

These experiments are designed to help answer the following questions: (1) What do smooth neural surrogates of simple functions look like, and how do they differ from standard MLP approximations? and (2) How does the proposed parameterization of the layerwise constants c_ℓ influence convergence behavior during training? All experiments are implemented in JAX, with additional implementation details provided in Appendix B.

Shape interpolation. A key advantage of smooth neural surrogates is their ability to learn globally smooth representations of unknown functions. We evaluate this property on a 2-D shape interpolation task, where the model must interpolate between two shapes given samples only at $z = 0$ and $z = 1$.

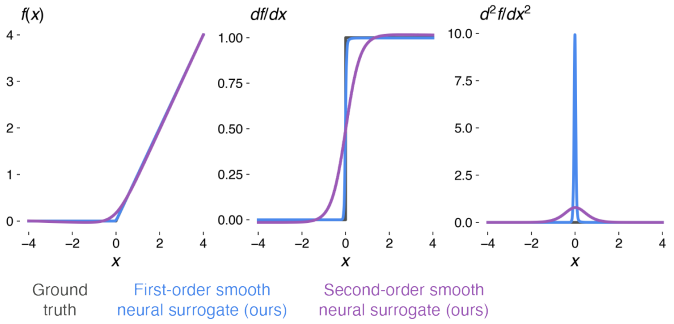

Fig. 4. **First- and second-order smooth neural surrogates for the ReLU function.** First-order surrogates eliminate large jumps but can form cusp-like shapes if trained for longer periods. Second-order surrogates seek to bound curvature throughout training, yielding smooth transitions similar to analytical smooth surrogates such as softplus or mish.

The standard MLP fails to produce coherent or continuous interpolation in z (Fig. 3). Although prior work reports successful interpolation with Lipschitz MLPs, it collapses under a tight sensitivity budget ($c_{ub} = 8$), even when the regularization weight is reduced by two orders of magnitude. In contrast, the SNS converges in both smoothness and MSE and consistently reconstructs valid intermediate shapes in a zero-shot manner.

1-D nonsmooth functions. We further study smoothness constraints on a controlled task: learning the ReLU function. This isolates how first- and second-order smooth surrogates regulate slope and curvature. As shown in Fig. 4, first-order surrogates reduce the discontinuity near the kink but can form cusp-like transitions, while second-order surrogates bound curvature and produce smooth transitions similar to analytical surrogates (e.g., softplus, mish). Second-order constraints introduce more bias but provide explicit curvature control.

We next approximate a 1-D piecewise function (Appendix B) that mimics a hopping robot's cost landscape [26]. In Fig. 5, the standard MLP fits the data with little bias but yields uninformative gradients, and the Lipschitz MLP again collapses under the constraint. First- and second-order SNSs converge to stable, slightly biased representations that smooth the discontinuities and nonconvexities; the second-order model adds further bias but maintains a lower curvature bound.

Smoothing particle–mass contact dynamics. Finally, we evaluate the surrogates on a physical system with stiff contact dynamics: a particle-mass model impacting the ground (Appendix B). Networks are trained to predict next-step position and velocity. Standard MLPs accurately capture zeroth-order behavior but yield erratic, stiff first- and second-order derivatives (Fig. 6). Notably, these learned derivatives can be slightly smoother than the ground truth in some instances, consistent with prior observations [20], but yield additional local non-smoothness. Both SNS variants model the dynamics while maintaining stable, well-behaved derivatives. Their zeroth-order predictions show minor bias near impact, but the derivatives remain nicely bounded and smooth. Interestingly, the first-order surrogate achieves low curvature despite having no explicit curvature regularization.

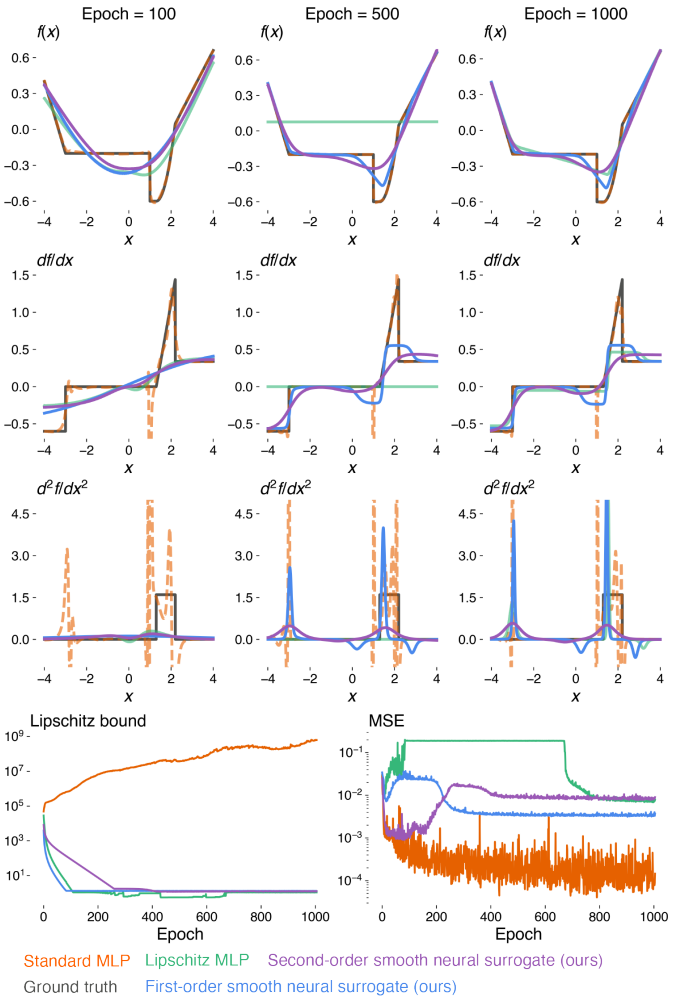

Fig. 5. **Smooth neural surrogates provide informative derivatives throughout training. Top:** Learned surrogates for a nonsmooth piecewise function. Standard MLPs develop extremely stiff gradients unsuitable for test time optimization, and Lipschitz MLPs briefly collapse under smoothness constraints. Smooth neural surrogates yield informative derivatives for downstream optimization throughout training. **Bottom:** Evolution of Lipschitz bounds and MSE during training.

VI. Learning Dynamics and State Estimation for Legged MPC Under Domain Randomization

In this section, we discuss our approach and experiments for learning legged MPC under domain randomization with smooth neural surrogates. The full training algorithm is given in Algorithm 1 and various implementation details are provided in the sections that follow and Appendix C.

A. Preliminaries

Learned MPC. Given states x and actions u, a typical formulation [5], [17] for learned dynamics is

$$x_{t+1} = x_t + d\hat{\mu}_\theta(x_t, u_t)dt + \varepsilon, \qquad \varepsilon \sim \mathbb{P}, \tag{7}$$

where $\mathbb{P}$ denotes an unspecified residual distribution. It is common to train the dynamics with one-step or multi-step rollout gaussian losses. It is also standard for the dynamics to depend on a history of states and actions $(x_{t-H:t}, u_{t-H:t})$ to

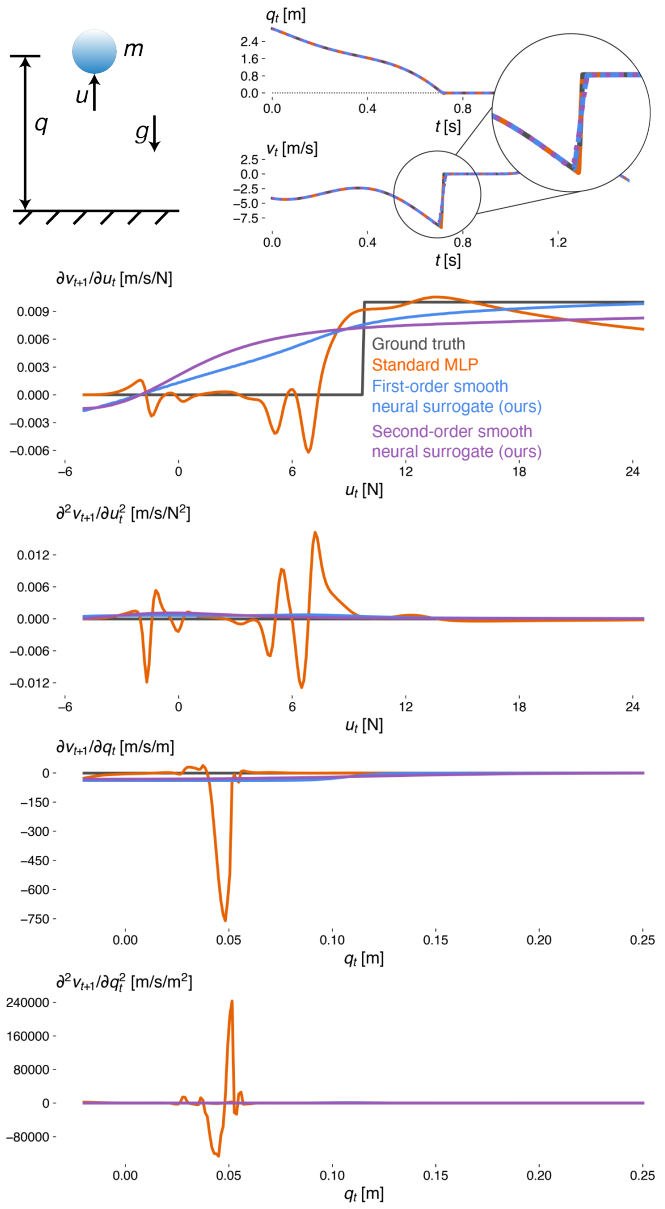

Fig. 6. **Smooth neural surrogates learn contact dynamics while preventing exploding gradients. Top:** Learned dynamics for a particle–mass impact model. **Bottom:** First- and second-order derivatives of the learned models when the mass is sitting on the surface (∂u_t) or traveling at -5 meters per second (∂q_t). Standard MLPs exhibit erratic, stiff gradients, whereas the SNS-MLPs' gradients are stable and attenuated.

improve accuracy or compensate for partial observability [16], [22], [60].

As noted earlier, we formulate the MPC problem as a single-shooting optimization over an action sequence $u_{0:T-1}$ with horizon T. The controller minimizes a cost function ℓ under inequality constraints g_i, using differentiable rollouts (forward simulation) of the learned dynamics, where $x_{1:T} = \texttt{rollout}_\theta(u_{0:T-1})$. We note that equality constraints can always be expressed as paired inequalities. The resulting optimization problem is

$$\begin{aligned} \min_{u_{0:T-1}} \quad & \ell\big(\texttt{rollout}_\theta(u_{0:T-1}),\ u_{0:T-1}\big) \\ \text{s.t.} \quad & 0 \geq g_i\big(\texttt{rollout}_\theta(u_{0:T-1}),\ u_{0:T-1}\big). \end{aligned} \tag{8}$$

Algorithm 1 Learning legged MPC and state estimation with smooth neural surrogates

Require: Replay buffer $\mathcal{D}$; smoothness budgets; horizon T; history length H; episode length T_{episode}; domain randomization, command, and state reset distributions; actuator net A_ψ

Initialize SNS dynamics model $\hat{\mu}_\theta$, SNS estimator $\bar{\mu}_\phi$, and simulator
Compute median and MAD over initial $\mathcal{D}$ to set dispersion $\hat{\Sigma}_\mathcal{D}$
while training not converged **do**
  for each environment instance **do**
    Randomize domain (dynamics, terrain, kinematics, etc.)
    Randomize agent commands and robot state x_0
    Reset episode buffer
    for $t = 0$ to T_{episode} **do**
      if $t > H$ **then**
        Estimate current state history $\bar{X}_{t|t}$ via $\bar{\mu}_\phi$
        Solve MPC (Eq. (8)) via our generalized Gauss-Newton solver (Section VII) and $\hat{\mu}_\theta$
        Set u_t to first element in optimal action trajectory
      else
        Set u_t to nominal action (Appendix C)
      end if
      Step simulator via u_t, and A_ψ (Appendix C) for x_{t+1}, y_{t+1}
      Store (x_t, u_t, y_t) into replay buffer $\mathcal{D}$
    end for
  end for
  for each training iteration **do**
    Sample trajectory batch $\{x_t, u_t, y_t\}_{t=0}^{T+H} \sim \mathcal{D}$
    Update θ via Algorithm 2
    Update ϕ via Algorithm 3
  end for
end while
Deploy MPC with learned dynamics and estimation in new environments with new costs and constraints

We compare two methods for solving (8): a state-of-the-art sampling-based method, DIAL-MPC [26], and a gradient-based generalized Gauss-Newton method (GGN-MPC) enabled by our SNS dynamics, described in Section VII. The optimization is vectorized across environments using JAX's `vmap(·)`.

Learned state estimation. Since legged robots operate under partial observability, we need state estimation to in order to use the learned dynamics for MPC. Classical estimators for legged robots, such as Kalman or particle-filter variants, often rely on accurate contact information, prior gait knowledge, or rigid-body assumptions, and they can become unreliable on varied terrain [61]–[63]. Although learned state estimation is increasingly common, most concurrent learning for estimation and control in legged robots appears in model-free RL [63], [64]. Other approaches to neural state estimation take inspiration directly from classical estimators and exploit model predictions [65]. Both learned dynamics and estimation from state and action histories have rigorous theoretical foundations in Takens's Embedding Theorem [66], and the technique is known more broadly as time-delay embedding [67].

B. Legged Robot Dynamics and Heavy-Tails

The Cauchy distribution is the canonical example of a "pathological" distribution. Unlike the Gaussian, the Cauchy distribution has no finite mean or variance and samples from this distribution are described as jumps or impulses, not unlike transitions in hybrid dynamics. In the n-dimensional case, the

TABLE II
1D CAUCHY AND GAUSSIAN RESIDUAL MODELS

Model	Assumption	Gradient	μ	σ
Gaussian	Light tails	$\propto \varepsilon$	Mean	STD
Cauchy	Heavy tails	$\propto \varepsilon/(1+\varepsilon^2)$	Median	MAD

probability density function is a special case of the Student's t-density and is given by:

$$\mathcal{C}(x;\mu,\Sigma) = \frac{\Gamma\left(\frac{n+1}{2}\right)\left[1+(x-\mu)^\top\Sigma^{-1}(x-\mu)\right]^{-\frac{n+1}{2}}}{\Gamma\left(\frac{1}{2}\right)\pi^{n/2}|\Sigma|^{1/2}}. \tag{9}$$

The parameter μ defines the *location*, while Σ acts as a *dispersion* term controlling scale along each direction. In the isotropic case $\Sigma = \sigma^2 I$, the location coincides with the median, and the dispersion parameter σ is proportional to the median absolute deviation (MAD) from the median. If we assume $\varepsilon \sim \mathcal{C}(x;\mu,\Sigma)$, we want to minimize any loss proportional to the *Cauchy negative log-likelihood* during training:

$$\begin{aligned}-\log\mathcal{C}(x;\mu,\Sigma) = \tfrac{n+1}{2}\log(1+(x-\mu)^\top\Sigma^{-1}(x-\mu)) \\ + \tfrac{1}{2}\log|\Sigma| + \text{const.}\end{aligned} \tag{10}$$

Variants of the Cauchy likelihood automatically rescale gradients when compared to Gaussian objectives as shown in Table II; which compares the 1D likelihood models for convenience. For small errors, both behave quadratically. For large errors, the Cauchy gradient saturates, preventing instability and reducing the impact of outliers. For cases where the impulse like dynamics are easy to predict, like our particle-mass example, the losses will likely perform similarly due to this comparability under small ε.

C. Partially Observable Dynamics and State Estimation

Learned dynamics. We categorize all quantities into measurements y_t (e.g., linear acceleration), state variables x_t used in control (e.g., velocity), and unmeasured environmental factors z_t (e.g., friction). The effect of z_t is captured implicitly through time-delay embedding. Let

$$X_t := x_{t-H:t}, \qquad U_t := u_{t-H:t}, \tag{11}$$

denote fixed-length state and action histories. The dynamics modify (7) as

$$\begin{aligned}x_{t+1} &= x_t + d\hat{\mu}_\theta(X_t,U_t)\,dt + \varepsilon, \qquad \varepsilon\sim\mathcal{C}(0,\hat{\Sigma}) \\ &\approx \hat{\mu}_\theta(X_t,U_t)\end{aligned} \tag{12}$$

Learned estimator (predictor–corrector). Because the dynamics depend on histories, the estimator must also maintain a history estimate defined as:

$$\bar{X}_{t|t-1} := \bar{x}_{t-H:t|t-1}, \qquad \bar{X}_{t|t} := \bar{x}_{t-H:t|t}, \tag{13}$$

where $\bar{X}_{t|t-1}$ is the prior (predicted) history and $\bar{X}_{t|t}$ is the posterior (corrected) history.

Prediction step. The estimator rolls its previous posterior one step forward using the learned dynamics:

$$\bar{X}_{t|t-1} = \mathrm{roll}\big(\bar{X}_{t-1|t-1}, \hat{\mu}_\theta(\bar{X}_{t-1|t-1},U_{t-1})\big), \tag{14}$$

Algorithm 2 Dynamics training step

Require: Trajectory batch $\{x_t,u_t,y_t\}_{t=0}^{T+H} \sim \mathcal{D}$, initial state estimate covariance $\Sigma_{\bar{X}}$, measurement noise, dynamics smoothness budget $\hat{c}_{\text{ub}}$, horizon T, history length H, SNS dynamics $\hat{\mu}_\theta$, SNS estimator $\bar{\mu}_\zeta$, discount factor γ, learning rate α_0, loss weights λ_0, λ_1, λ_2, and λ_3

Perturb initial state history estimate $\bar{X}_{t|t} \sim \mathcal{N}(X_t, \Sigma_{\bar{X}})$
for $t = H$ to $T+H$ **do**
  Predict x_{t+1} via $\hat{\mu}_\theta$ with X_t, U_t for $\mathcal{L}_{\text{step}}$ (Eq. (20))
  if $t > H$ **then**
    Predict x_{t+1} via $\hat{\mu}_\theta$ with $\hat{X}_t$, U_t for $\mathcal{L}_{\text{rollout}}$ (Eq. (21))
    Predict x_{t+1} via $\hat{\mu}_\theta$ with $\texttt{stop\_grad}(\bar{X}_{t|t})$, U_t for $\mathcal{L}_{\text{corrupt}}$ (Eq. (22)) and prior $\bar{X}_{t+1|t}$ (Eq. (14))
  end if
  Corrupt y_t with noise
  Compute ν_{t+1} (Eq. (15)) and $g_{\nu,t+1}$ (Eq. (16))
  Estimate $\bar{X}_{t+1|t+1}$ via $\bar{\mu}_\phi$ with $\bar{X}_{t+1|t}$, U_t, Y_{t+1}, ν_{t+1}, $g_{\nu,t+1}$
end for
Update θ using total SNS dynamics loss (Eq. (18)) and α_0

Algorithm 3 Estimator training step

Require: Trajectory batch $\{x_t,u_t,y_t\}_{t=0}^{T+H} \sim \mathcal{D}$, Initial state estimate covariance $\Sigma_{\bar{X}}$, measurement noise, estimator smoothness budget $\bar{c}_{\text{ub}}$, horizon T, history length H, SNS dynamics $\hat{\mu}_\theta$, SNS estimator $\bar{\mu}_\zeta$, learning rate α_1, loss weight λ_4

Perturb initial state history estimate $\bar{X}_{t|t} \sim \mathcal{N}(X_t, \Sigma_{\bar{X}})$
for $t = H+1$ to $T+H$ **do**
  Predict x_{t+1} via $\hat{\mu}_\theta$ with $\texttt{stop\_grad}(\bar{X}_{t|t})$, U_t for prior $\bar{X}_{t+1|t}$ (Eq. (14))
  Corrupt y_t with noise
  Compute ν_{t+1} (Eq. (15)) and $g_{\nu,t}$ (Eq. (16))
  Estimate $\bar{X}_{t+1|t+1}$ via $\bar{\mu}_\phi$ with $\bar{X}_{t+1|t}$, U_t, Y_{t+1}, ν_{t+1}, $g_{\nu,t+1}$
end for
Update ζ using total SNS estimator loss (Eq. (23)) and α_1

where $\mathrm{roll}(a,b)$ discards the oldest element of a and appends b. Measurements are predicted via the prior and a typical measurement model, then used to calculate the innovation:

$$\hat{y}_t = h(\bar{X}_{t|t-1}), \qquad \nu_t = y_t - \hat{y}_t. \tag{15}$$

Then, the gradient of the total squared innovation provides sensitivities for the neural estimator:

$$g_{\nu,t} = \begin{bmatrix}\frac{\partial(\nu_t^\top\nu_t)}{\partial\bar{X}_{t-1|t-1}} & \frac{\partial(\nu_t^\top\nu_t)}{\partial U_{t-1}}\end{bmatrix}^\top. \tag{16}$$

Correction step. The SNS estimator applies a correction to the prior using all available information:

$$\begin{aligned}X_t &= \bar{X}_{t|t-1} + d\bar{\mu}_\zeta\big(\bar{X}_{t|t-1},U_t,Y_t,\nu_t,g_{\nu,t}\big) + \varepsilon,\ \varepsilon\sim\mathcal{C}(0,\hat{\Sigma}) \\ &\approx \bar{\mu}_\zeta\big(\bar{X}_{t|t-1},U_t,Y_t,\nu_t,g_{\nu,t}\big)\,.\end{aligned} \tag{17}$$

Where $\{Y_t : y_{t-H:t} \cap X_t = \emptyset\}$ are the measurement histories that do not overlap with the state histories. This would include, for example, base acceleration. For directly measurable state components (e.g., joint angles and velocities), we fill the corresponding entries of $\bar{X}_{t|t}$ with their noise corrupted sensor readings rather than a learned correction. In other words, the estimator is only trained to estimate unmeasurable or "privileged" states while noisy measurements flow directly to the dynamics without correction.

D. Loss Functions and Concurrent Training

We train the dynamics model and estimator *concurrently*, with each module receiving its own loss while also influencing the

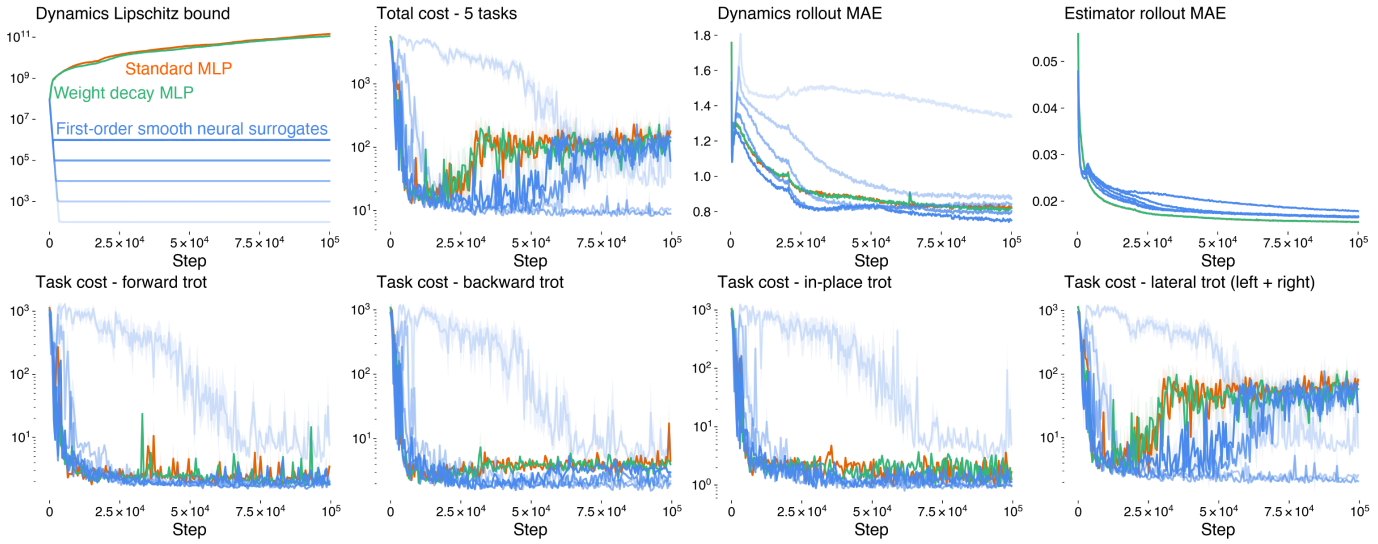

Fig. 7. **Smooth neural surrogates excel at gradient-based MPC for legged robots.** First-order smooth neural surrogate dynamics with varying Lipschitz constraints are compared to dynamics modeled with a standard MLP and an MLP with weight-decay regularization. The nonsmooth networks become progressively less smooth throughout training, whereas the smooth neural surrogates quickly converge and remain within their allotted sensitivity budgets. A Pareto front emerges between smoothness and control: models with moderate sensitivity budgets are best for planning quadruped locomotion. Excessively smooth models learn slowly, while nonsmooth models become too stiff for planning as they converge. Among the smooth neural surrogates, the relationship between dynamics rollout error and smoothness is monotonically decreasing, smoother surrogates exhibit larger prediction errors. The standard and weight-decay MLPs achieve comparable rollout errors to the moderately smooth surrogates. Thus, smooth neural surrogates with loose Lipschitz constraints effectively regularize the model without introducing bias, but these models remain too stiff for reliable planning.

TABLE III
LOSS COMPONENTS FOR DYNAMICS AND ESTIMATION

Loss Term	Purpose
$\mathcal{L}_{\text{step}}$	One-step predictions
$\mathcal{L}_{\text{rollout}}$	Multi-step predictions
$\mathcal{L}_{\text{corrupt}}$	Robustness to poor state estimates
$\mathcal{L}_{\text{estimator}}$	Multi-step estimation accumulation
$\mathcal{L}_{k\text{-SNS}}$	Smoothness constraints/regularization

other's inputs during rollout. This is not unlike concurrent methods in model-free RL [63]. Table III gives a high-level overview of the role of each loss, while Algorithms 2 and 3 detail how they were computed during training.

Dynamics loss. The dynamics model is trained with

$$\mathcal{L}_{\text{dynamics}} = \lambda_0 \mathcal{L}_{\text{step}} + \lambda_1 \mathcal{L}_{\text{rollout}} + \lambda_2 \mathcal{L}_{\text{corrupt}} + \lambda_3 \mathcal{L}_{k\text{-SNS}}. \tag{18}$$

For robust optimization under heavy-tailed residuals we use the mean Cauchy error (MCE)

$$\mathcal{L}_{\text{MCE}} \propto -\log \mathcal{C}(x_{t+1}; \hat{\mu}_\theta, \hat{\Sigma}_\mathcal{D}), \tag{19}$$

with homoscedastic dispersion $\hat{\Sigma}_\mathcal{D} = \hat{\sigma}^2 I$ estimated via the replay buffer and the MAD. The same MAD and median values are used for fixed "normalization" and "denormalization" layers for the input and output. In the homoscedastic case, (19) corresponds to dropping the $\log|\Sigma|$ term in (10) and averaging across the state and batch dimensions.

The single-step loss along a trajectory segment of length T is

$$\mathcal{L}_{\text{step}} = \frac{1}{T} \sum_{t=0}^{T} \mathcal{L}_{\text{MCE}}(x_{t+1}, \hat{\mu}_\theta, \hat{\Sigma}_\mathcal{D}), \tag{20}$$

while the multi-step rollout loss is computed autoregressively with discount γ:

$$\mathcal{L}_{\text{rollout}} = \frac{1}{T-1} \sum_{t=1}^{T} \mathcal{L}_{\text{MCE}}(x_{t+1}, \hat{\mu}_{\theta,t+1}, \hat{\Sigma}_{\mathcal{D},\gamma^t}). \tag{21}$$

This prevents long-horizon drift [5], [60]. The corrupted-input loss uses estimator outputs as dynamics inputs:

$$\mathcal{L}_{\text{corrupt}} = \frac{1}{T-1} \sum_{t=1}^{T} \mathcal{L}_{\text{MCE}}(x_{t+1},\ \hat{\mu}_\theta(\texttt{stop\_grad}(\bar{X}_{t|t}), \cdot), \hat{\Sigma}_\mathcal{D}) \tag{22}$$

to ensure robustness to poor estimates. The estimates are treated as plain data to avoid instability via the `stop_grad` operator. The algorithm for a full training step of the dynamics model is given in Algorithm 2.

Estimator loss. The estimator is trained in parallel using multi-step rollouts, starting from noise-perturbed histories:

$$\mathcal{L}_{\text{estimator}} = \frac{1}{H(T-1)} \sum_{t=1}^{T} \sum_{h=0}^{H} \mathcal{L}_{\text{MCE}}(x_{t-h:t},\ \bar{\mu}_\zeta(\bar{X}_{t|t-1}, \cdot),\ \hat{\Sigma}_\mathcal{D}) + \lambda_4 \mathcal{L}_{k\text{-SNS}}. \tag{23}$$

As before, we detach the estimates to prevent exploding gradients during the rollout. The algorithm for a full training step of the estimator model is given in Algorithm 3.

E. Experiments

We evaluate our approach (Algorithm 1) on a quadruped robot by validating its distributional assumptions and replacing individual components with standard MBRL with MPC methods. Our experiments examine: (1) the effect of neural smoothness

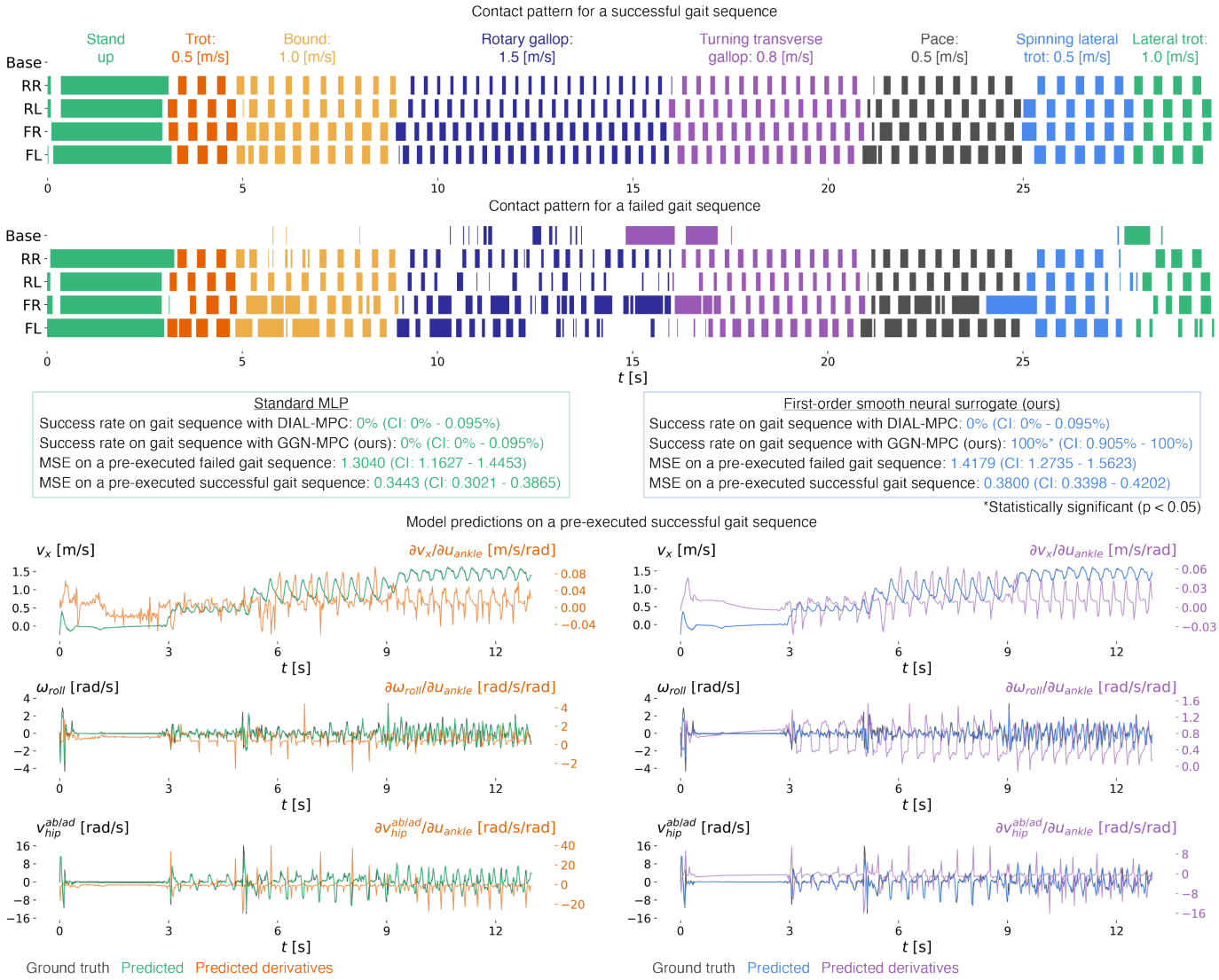

Fig. 8. **First-order model characteristics correspond to planning performance, while zeroth-order error can be misleading. Top:** Contact patterns for a failed and successful gait sequence generated at test time. The successful sequence is synthesized using an SNS dynamics-estimation stack, whereas the failed sequence uses a standard MLP dynamics-estimation. **Middle:** The standard MLP failed to produce a gait without base or hip contact in 30 trials using either DIAL-MPC [26], or our generalized Gauss–Newton solver (Section VII). In contrast, the SNS achieved a 100% success rate with GGN-MPC. Both models exhibit comparable errors along the successful and failed trajectories, with overlapping confidence intervals. However, the SNS shows a slightly higher mean error on both sequences, reflecting a small bias introduced by the imposed Lipschitz bound. **Bottom:** Zeroth-order predictions from both models appear nearly identical, aside from a few rapid transients that the standard MLP captures more sharply. Nevertheless, the standard MLP derivatives are substantially noisier and stiffer (sometimes $\sim 3\times$ greater range), resulting in poor planning performance.

on prediction and control, (2) residual distribution models, (3) predictor–corrector vs. direct estimation, and (4) their combined effect. All models are tested on 5 basic trotting variants using an evaluation cost different from the training cost, averaged over 10 seeds. Additional comparisons include 7 different gaits from galloping to pacing. When applicable, error bars are calculated from the log-transformed data. More detailed discussion on the hyperparameters and costs used for the following experiments are provided in Section VIII and Appendix C.

Smooth and nonsmooth neural networks. Fig. 7 shows that the Lipschitz bounds of standard and weight-decay MLPs grow rapidly during training, and these models perform poorly in gradient-based trajectory optimization. Smooth neural surrogates remain within their sensitivity budgets and exhibit a Pareto front between smoothness and control: overly smooth models learn slowly, loose bounds behave like standard MLPs, and moderate bounds work best for planning through contact.

For the least smooth models, control performance degrades with training not due to overfitting—the replay-buffer error continues to drop, and test-time error on unseen trajectories stays low even after 10^5 updates (Fig. 8)—but because their dynamics become increasingly stiff as they better represent rapid changes in the ground truth dynamics.

Consistent with prior work [5], models with the lowest prediction error are not necessarily the best for planning. Loose Lipschitz SNS models achieve the lowest replay-buffer error but remain too stiff for reliable MPC, while the best

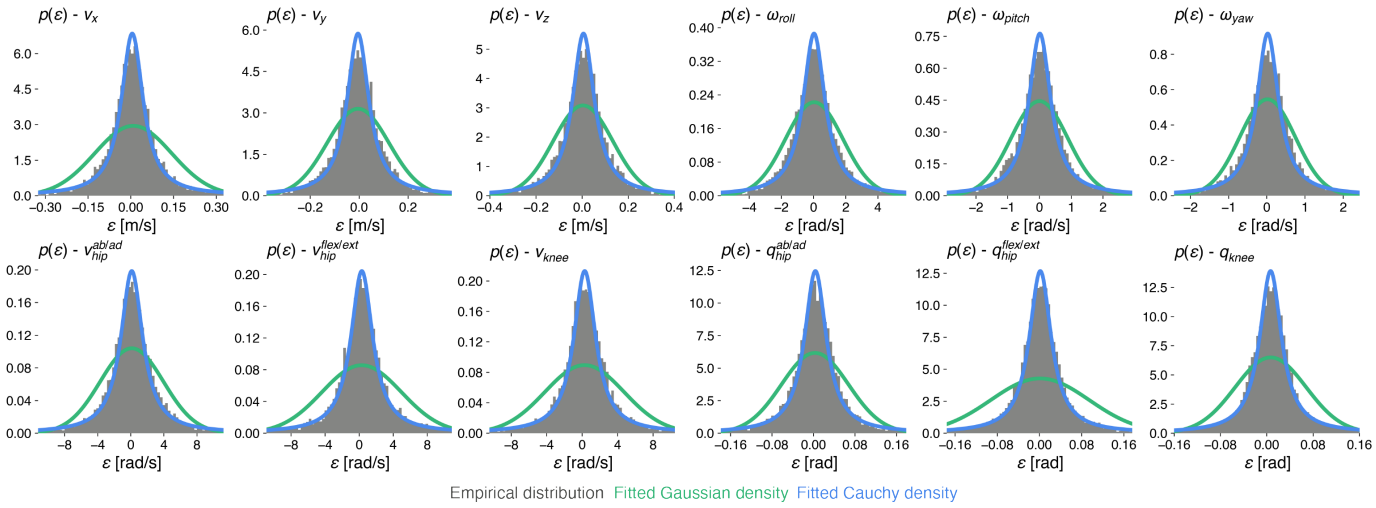

Fig. 9. **Errors for learned models of quadruped dynamics follow a heavy-tailed Cauchy distribution.** Empirical residuals of one-step dynamics predictions, ε, across representative quadruped state variables. The residuals, computed from the replay buffer, align more closely with a Cauchy density than with a Gaussian. This indicates that Gaussian loss functions such as the MSE may be poorly suited for learning legged robot dynamics.

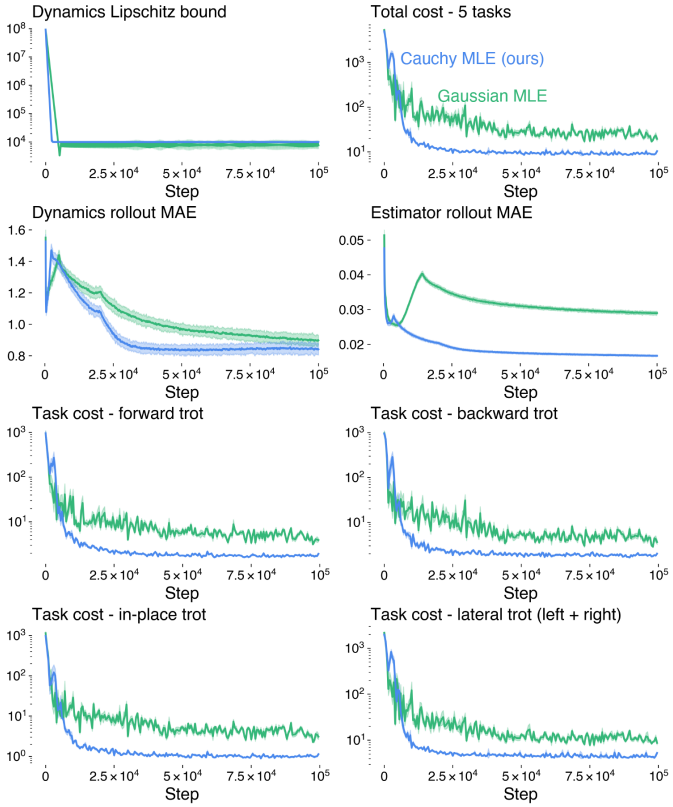

Fig. 10. **Learning with heavy-tailed maximum likelihood estimation improves training stability, motion planning, and model accuracy.** Gaussian MLE, common in model-based RL, fails to capture the number of large errors we observe while training stiff legged-robot dynamics. In contrast, heavy-tailed Cauchy MLE yields more stable convergence, higher prediction accuracy, and lower overall control cost.

control models show prediction errors similar to standard and weight-decay MLPs.

Fig. 8 compares variable gait sequences generated by a standard MLP and an SNS-MLP. Using our generalized Gauss-Newton solver (Section VII), the smooth surrogate executes the sequence online with a 100% success rate over 30 trials. The standard MLP fails all 30 trials under both GGN-MPC and DIAL-MPC. Failure is defined as unintended contact of the robot's base or hips with the ground.

Both models achieve similar MSE on successful and failed sequences, with slightly higher mean error for the smooth surrogate due to the Lipschitz constraint, but the difference is not statistically significant. Their zeroth-order predictions are nearly identical, visually. The key difference lies in first-order behavior: the standard MLP exhibits noisy, jagged derivatives, sometimes spanning nearly triple the range of the smooth neural surrogate (Fig. 8). While the failure of sampling-based MPC is likely a combination of both the imperfect learned dynamics and the planner, smooth neural surrogates provide a steady gradient signal for planning.

Gaussian and Cauchy error models. We next examine empirical residual distributions and how their corresponding likelihoods influence learning, prediction, and control. Fig. 9 shows that one-step dynamics residuals (computed from the replay buffer) across representative quadruped states are heavy-tailed and match a Cauchy density more closely than a Gaussian, indicating that MSE-style losses are poorly aligned with the data.

Fig. 10 compares two otherwise identical training setups, differing only in their residual model and using learning rates that were retuned to the best of our ability. In the Gaussian case, the Cauchy terms in (18) and (23) are replaced with the mean Mahalanobis error

$$\mathcal{L}_{\text{MME}} = \frac{1}{nN_{\mathcal{D}}} \sum_{x \in \mathcal{D}} \frac{1}{2}(x-\mu)^{\top}\Sigma^{-1}(x-\mu), \qquad (24)$$

and where the median and MAD are replaced by the mean and

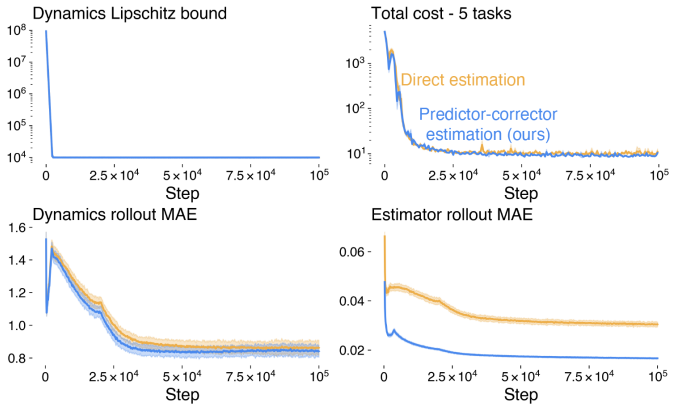

Fig. 11. **Leveraging model predictions for neural estimation.** Direct state-estimation networks, cannot accumulate information over time or leverage predictions from the dynamics model to refine their estimates. In contrast, predictor–corrector strategies accumulate information and integrate model predictions into the estimation loop. yielding more accurate state estimates and lower overall control cost for a quadruped robot.

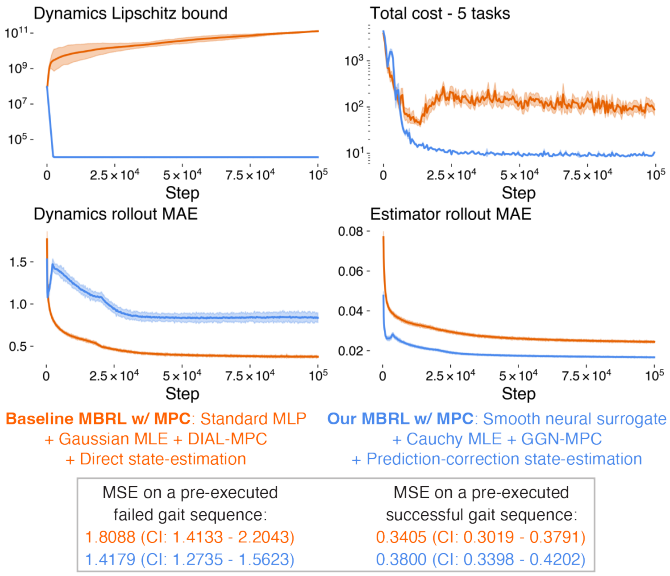

Fig. 12. **Baseline model-based RL with MPC.** The baseline uses standard MLP dynamics and estimator networks, Gaussian MLE, and a sampling-based controller. Our smooth neural surrogates with Cauchy MLE and derivative-based MPC achieve substantially lower control cost. Sampling-based MPC produces a replay buffer with reduced variation that is easier to predict, which slightly improves accuracy on successful gait sequences (see Fig. 8) but increases error on stiff failure trajectories. Overall, reduced exploration plays a minor role compared to the gains from Lipschitz-constrained networks, robust MLE, and derivative-based control and estimation

standard deviation for the normalization layers and loss calculation. We preserve the scaling for fairness instead of using the more common MSE (i.e., $\mathcal{L}_{\text{MME}}(x, \mu, I)$). Models trained with the heavy-tailed Cauchy likelihood achieve lower prediction error, faster and more stable convergence, and substantially lower control cost than those trained with Gaussian MLE. This suggests that, in the setting we examined, heavy-tailed residual models are better suited to train supervised models for legged robots.

State estimation. We next compare the model-based estimation framework against a direct, model-free estimation strategy [63]. This ablation is not intended to identify the optimal estimator, but rather to demonstrate that derivatives from the smooth neural surrogate can be used to improve estimation quality and downstream control performance. The model-free direct estimation approach uses a history of measurements and actions to reconstruct the full state-history:

$$X_t = \bar{\mu}_\zeta(y_{t-H:t}, U_{t-1}) + \varepsilon. \tag{25}$$

For the direct approach, we also assume heavy-tailed errors for ε fair comparison. Fig. 11 shows that our prediction–correction strategy yields lower and less variable state-estimation error compared to direct estimation. The improved estimation enables faster dynamics learning, as both models are trained jointly. As a result, the total control cost for locomotion also decreases marginally.

Baseline comparisons. Fig. 12 summarizes our baseline, which uses standard MLP dynamics and direct estimator networks with Gaussian MLE and a sampling-based MPC controller. We use this setup to isolate the total effect of ignoring derivatives and heavy-tails during training. Sampling-based MPC during training and deployment did not resolve the issues with planning using standard MLPs. Although sampling-based MPC produces a replay buffer with lower variation (Fig. 12), this only slightly simplifies prediction during training and does not meaningfully improve planning accuracy, and may even hurt generalization. Gradient-based MPC, produces very stiff actions during the initial training steps, but even with smoother transitions, Gaussian losses still underestimate the error kurtosis.

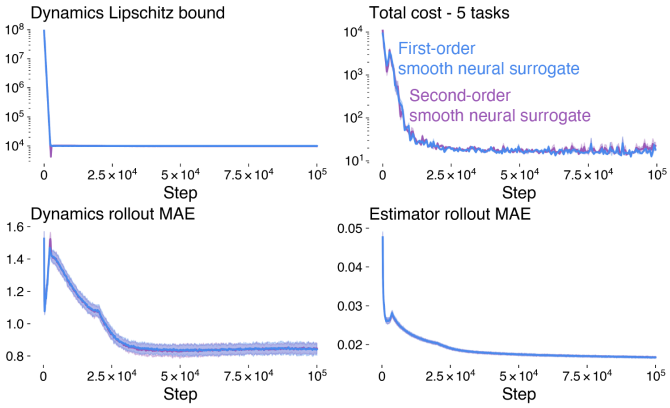

Fig. 13. **First- and second-order smooth neural surrogates.** Both networks achieve comparable prediction and planning performance in our Gauss–Newton MPC setup, as the solver primarily relies on stable first-order derivatives.

First- and second-order smoothing. Lastly, we compare first- and second-order smooth neural surrogates and find that, in our quadruped setting, the smoothing order does not substantially influence prediction accuracy or planning performance (Fig. 13). This result could be expected as our generalized Gauss–Newton solver depends on stable first-order derivatives through the forward simulation. We anticipate that second-order smoothing will be more consequential in settings that rely explicitly on Hessian information such as full Newton methods, or differentiable optimization.

VII. Gradient-Based Neural MPC Enabled by Smooth Neural Surrogates

A. Generalized Gauss-Newton MPC

Our basic single-shooting setup (Eq. 8) allows the use of any general-purpose nonlinear optimizer. We choose a generalized Gauss–Newton method [68] with sequential quadratic programming, which has demonstrated strong performance numerical optimization and MPC in prior work [69], [70].

We refer to our approach as *gray-box*: aside from requiring differentiability and convex costs, the internal structure of the learned dynamics model is irrelevant to the optimizer. This stands in contrast to popular Gauss–Newton trajectory optimizers such as the iterative Quadratic Gauss-Newton (iLQG) [71], which impose specific assumptions on the temporal nature of the dynamics, actions, and cost. While our method is inspired by iLQG, it does not rely on those structural constraints, making it better suited for neural dynamics. In addition to the generalized Gauss–Newton procedure, we incorporate GPU-parallelization to accelerate solve times.

We parameterize the action (control) sequence $u_{0:T-1}$ using a set of k knots u_κ:

$$u_t = \texttt{spline}(u_\kappa)(t), \quad u_\kappa \in \mathbb{R}^{n\cdot k}, \quad k \leq T. \tag{26}$$

This parameterization is common in gradient- and sampling-based methods as it enforces continuity and physical plausibility while reducing the number of decision variables.

Constraints are handled using an augmented objective following [69]:

$$\min_{u_\kappa} \quad \ell(u_\kappa) + \sum_{i=0}^{G} \beta_i \log_\delta(g_i(u_\kappa)). \tag{27}$$

Here, $\log_\delta(\cdot)$ denotes a relaxed logarithmic barrier:

$$\log_\delta(g_i) = \begin{cases} -\log(-g_i), & g_i < -\delta_i, \\ -\log\delta_i + \frac{1}{2}\left(\frac{g_i+2\delta_i}{\delta_i}\right)^2 - \frac{1}{2}, & g_i \geq -\delta_i, \end{cases} \tag{28}$$

The relaxation parameters δ_i and β_i regulate the strength of constraint enforcement. As $\delta_i \to 0$ and $\beta_i \to 0$, feasible solutions incur vanishing barrier cost, whereas infeasible solutions diverge. In practice, we keep these parameters fixed for each constraint. GGN-MPC performs a backward pass and forward pass with each iteration.

Backward pass / quadratic approximation. In the backward pass, we compute a local descent direction for the spline knots. Let the objective in (27) be written as a sum over residuals ε_i:

$$\mathcal{J}(\varepsilon(u_\kappa)) = \sum_{i=0}^{R} \ell_i\big(\varepsilon_i(u_\kappa)\big). \tag{29}$$

We seek an update direction δu_κ at point u_κ^- by solving the convex quadratic subproblem:

$$\min_{\delta u_\kappa} \ \tfrac{1}{2}\,\delta u_\kappa^\top H\,\delta u_\kappa \ + \ q^\top \delta u_\kappa. \tag{30}$$

where the gradient is

$$q = \nabla_{u_\kappa}\mathcal{J}(\varepsilon(u_\kappa^-)) = \left(\frac{\partial \varepsilon}{\partial u_\kappa}\right)^\top \nabla_\varepsilon \mathcal{J}(\varepsilon(u_\kappa^-)). \tag{31}$$

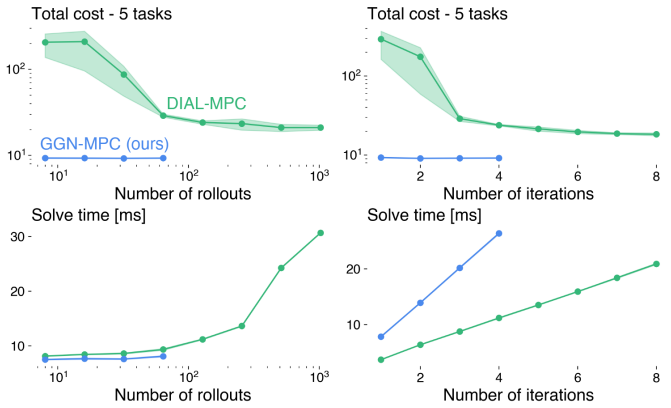

Fig. 14. **Convergence and computational efficiency: gradient-based vs. sampling-based MPC with SNS. Left:** With a fixed iteration count (4 for DIAL-MPC, 1 for GGN-MPC), DIAL-MPC performance depends strongly on rollout count, while GGN-MPC remains consistently low-cost. **Right:** With a fixed rollout count (128 vs. 16), GGN-MPC shows diminishing returns from additional iterations, whereas DIAL-MPC requires more iterations and still fails to reach comparable cost.

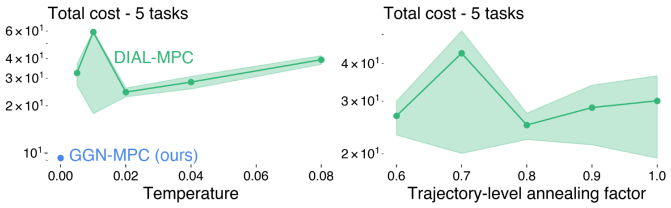

Fig. 15. **Sensitivity to hyperparameters: gradient-based vs. sampling-based MPC with SNS. Left:** GGN-MPC is greedy (temperature = 0) and stable. DIAL-MPC performs poorly in the greedy limit and is highly sensitive to temperature. **Right:** Additional diffusion-related hyperparameters further affect DIAL-MPC, whereas our method remains robust without tuning any parameters.

and the GGN Hessian is approximated by

$$\begin{aligned} H &\approx \nabla^2_{u_\kappa}\mathcal{J}(\varepsilon(u_\kappa^-)) \\ &= \left(\frac{\partial \varepsilon}{\partial u_\kappa}\right)^\top \nabla^2_\varepsilon \mathcal{J}(\varepsilon(u_\kappa^-)) \left(\frac{\partial \varepsilon}{\partial u_\kappa}\right). \end{aligned} \tag{32}$$

If each $\ell_i(\varepsilon)$ is convex in the residual, then its residual-space Hessian is PSD $\nabla^2_\varepsilon \mathcal{J}(\varepsilon) \succeq 0$. By congruence, H is also PSD. This holds for common objectives such as squared loss or the relaxed logarithmic barrier $\log_\delta(\cdot)$. When the objective is not twice differentiable (e.g., absolute value or Huber loss), the curvature term may be omitted, and only the first-order (subgradient) contribution is used [70]. The convexity of (30) ensures a unique global minimizer, numerical stability in the backward pass, and a well-defined update direction.

The linear system

$$H\,\delta u_\kappa = -q \tag{33}$$

is solved efficiently using Cholesky factorization. We also find that no additional regularization (e.g., Levenberg-Marquardt) is required with smooth neural surrogate dynamics.

Forward pass / greedy parallel line search. Line search stabilizes updates when the full Gauss-Newton step is unreliable due to nonlinearities in the rollout or cost landscape. In the

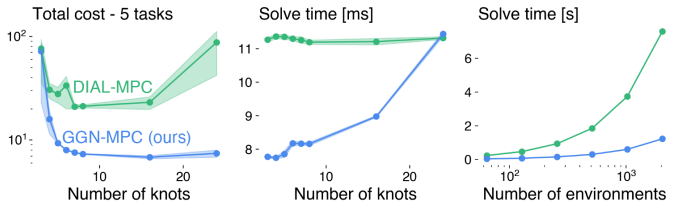

Fig. 16. **Scalability: gradient-based vs. sampling-based MPC with SNS. Left:** GGN-MPC maintains low cost across action-trajectory resolutions; DIAL-MPC fails as dimensionality increases. **Middle:** GGN-MPC solve time grows quadratically with knots yet remains efficient and achieves much lower mean cost. **Right:** In parallel environments, sampling-based MPC scales poorly due to large rollout requirements, while GGN-MPC remains more efficient.

forward pass, the control knots are updated along the descent direction with step size $\alpha \in [0, 1]$:

$$u_\kappa^+ = u_\kappa^- + \alpha\, \delta u_\kappa. \tag{34}$$

The step size is selected by solving

$$\alpha^* = \arg\min_{0 \le \alpha \le 1} \mathcal{J}\big(\varepsilon(u_\kappa^- + \alpha\, \delta u_\kappa)\big)\,, \tag{35}$$

with the full nonlinear rollouts and cost.

We approximate this 1D subproblem using a *greedy parallel line search*: R candidate step sizes $\boldsymbol{\alpha}$ are sampled uniformly in $[0, 1]$, and the corresponding rollout trajectories evaluated in parallel on GPU:

$$\hat{\mathbf{x}}_{0:T} = \texttt{vmap}\big(\texttt{rollout}_\theta(u_\kappa^- + \alpha\, \delta u_\kappa)\big)(\boldsymbol{\alpha}). \tag{36}$$

where $\hat{\mathbf{x}}_{0:T} \in \mathbb{R}^{R \times T \times n}$ is the batch of candidate trajectories.

B. Experiments

We compare GGN-MPC to DIAL-MPC [26] as a state-of-the-art representative of sampling-based MPC. Both controllers use single-shooting and spline-parameterized controls with the same number of knots unless noted otherwise. Experiments compare solve time and cumulative closed-loop performance over full episodes, using identical costs, SNS dynamics, and estimators for a fair comparison. We note the benefits of GGN-MPC only emerge with smooth neural surrogates; without them, gradient-based neural MPC in our setting is brittle, as shown earlier.

Overall performance and sensitivity to hyperparameters. GGN-MPC and DIAL-MPC share the number of rollouts as a key forward-pass hyperparameter. DIAL-MPC samples candidates from an annealed isotropic Gaussian, while GGN-MPC evaluates deterministic proposals along the descent direction from the backward pass. Although sampling-based MPC improves with more rollouts, Fig. 14 shows that GGN-MPC selects far more informative candidates, achieving low cost with an order of magnitude fewer evaluations. As a result, its cumulative control cost remains significantly lower across all tested numbers of rollouts.

GGN-MPC also gains little from additional iterations, whereas DIAL-MPC depends heavily on repeated solves. A single DIAL-MPC iteration is faster, but multiple iterations are required for convergence, reducing overall efficiency. Fig. 15 highlights the effect of additional hyperparameters on sampling-based methods: DIAL-MPC's performance varies widely with the temperature parameter, that governs its soft-min reduction, and with diffusion-related hyperparameters. In contrast, GGN-MPC applies the lowest-cost candidate directly (equivalent to temperature $= 0$), has no extra hyperparameters, and remains stable under all tested settings.

Scalability. We compare scalability along two axes: the dimensionality of the decision vector and parallelization across environments during RL training. In our setup, DIAL-MPC is run for four iterations with 128 rollouts per iteration, whereas GGN-MPC uses a single iteration with only 16 rollouts. As shown in Fig. 16, both methods initially benefit from increasing the action-trajectory resolution, reaching peak performance at roughly ten spline knots (~ 120 decision variables). Beyond this point, DIAL-MPC degrades rapidly—an instance of the curse of dimensionality that is characteristic of sampling-based methods. GGN-MPC, exhibits quadratic growth in solve time due to Cholesky factorization but remains faster at most resolutions and sustains low costs even with high-dimensional decision vectors, where DIAL-MPC fails, despite using only a single iteration.

Parallelization becomes essential in large-scale RL. Our JAX implementation vectorizes both optimizers across environments using $\texttt{vmap}(\texttt{MPC}_\theta)$. Because GGN-MPC requires significantly fewer rollouts and thus less computation per agent, it scales more effectively in GPU-based parallel training, as illustrated in Fig. 16.

VIII. Online Quadruped Behavior Synthesis

In this section, we detail how the flexibility of learning and MPC are combined with smooth neural surrogates on the Go2 quadruped for versatile control and sim-to-real transfer.

A. States, Actions, and Measurements.

We select states to support downstream planning while remaining compatible with the Lipschitz constraints of our smooth neural surrogate models. The augmented state (Table IV) includes standard proprioceptive terms together with the 6D orientation representation $r_{\mathcal{W}}^{6D}$ and the signed distances ϕ from each body geometry to the terrain (visualized in Fig. 17). These distances allow the robot to infer proximity to contact across varying terrain profiles for better awareness and can be used for collision avoidance or gait tracking costs and constraints. The resulting learned dynamics has 60 states. The full list of measured quantities, predicted measurements, and actions is listed in Table V.

We represent base orientation using the continuous 6D parameterization [72] to ensure all discontinuities in the data are physical, not mathematical artifacts that occur with Euler angles or quaternions. Let

$$R_{\mathcal{W}} = [\, r_1 \;\; r_2 \;\; r_3 \,] \in SO(3) \tag{37}$$

denote the base rotation matrix. The forward map retains the first two columns of the rotation matrix:

$$r_{\mathcal{W}}^{6D} = \Pi_{6\mathrm{D}}(R_{\mathcal{W}}) = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}. \tag{38}$$

Fig. 17. **A simple strategy for whole-body and terrain-aware planning with learned MPC.** By augmenting the states in our neural dynamics model with signed distances from the robot to the terrain, the controller can plan using both classical generalized coordinates and self-terrain geometry. Training with randomized terrain inclines makes the state estimates and plans robust to uneven ground, varying slopes, and stepped terrain in the real world. Even on sloped, slippery ground with deep debris, the model demonstrates precise awareness of collision geometry and contact dynamics, allowing it to regain footing after a fall.

TABLE IV
AUGMENTED STATE SPACE

Quantity	Description
z [m]	Base height above terrain
$r_{\mathcal{W}}^{6D}$ [–]	6D base orientation representation
q_J [rad]	Joint positions (12 DoF)
v_J [rad/s]	Joint velocities
$v_{\mathcal{B}}$ [m/s]	Base linear velocity (body frame)
$\omega_{\mathcal{B}}$ [rad/s]	Base angular velocity (body frame)
ϕ [m]	Signed distances (23 collision geoms)

TABLE V
MEASUREMENTS AND CONTROL INPUTS

Quantity	Description
y	Measurements $[q_J,\, v_J,\, r_{\mathcal{W}}^{6\mathrm{D}},\, \omega_{\mathcal{B}},\, \dot{v}_{\mathcal{B}}]$
$\hat{y}$	Predicted measurements $[\hat{q}_J,\, \hat{v}_J,\, \hat{r}_{\mathcal{W}}^{6\mathrm{D}},\, \hat{\omega}_{\mathcal{B}},\, \Delta\hat{v}_{\mathcal{B}}]$
$\dot{v}_{\mathcal{B}}$ [m/s^2]	Base linear acceleration (body frame)
$\Delta\hat{v}_{\mathcal{B}}$ [m/s^2]	Predicted acceleration $(\hat{v}_{\mathcal{B},t} - \hat{v}_{\mathcal{B},t-1})/dt$
u [rad]	Desired joint positions q_J^d

The inverse map orthonormalizes these vectors:

$$r_1' = r_1/\|r_1\|, \quad \tilde{r}_2 = r_2 - (r_1'^{\top} r_2) r_1', \quad r_2' = \tilde{r}_2/\|\tilde{r}_2\|, \tag{39}$$

to reconstruct the valid rotation

$$R_{\mathcal{W}} = \Phi_{6\mathrm{D}}(r_{\mathcal{W}}^{6D}) = [\, r_1' \;\; r_2' \;\; r_1' \times r_2' \,]. \tag{40}$$

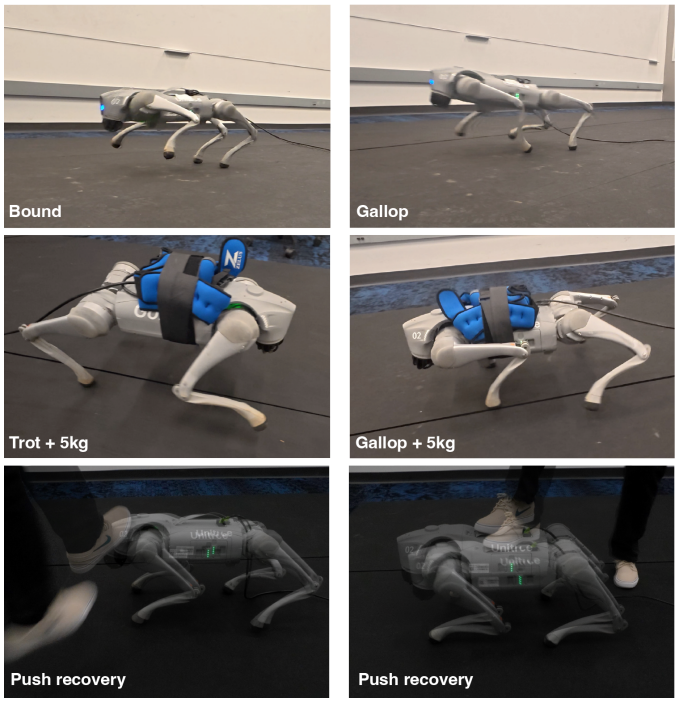

Fig. 18. **Sample indoor experiments. Top:** Bounding and galloping gaits. **Middle:** Locomotion with external payload. **Bottom:** Push recovery from different angles.

B. Terrain Randomization

Signed-distance states provide more useful information when the terrain varies meaningfully during training. We randomize the ground-plane slope by applying a uniformly sampled 3D rotation, creating broad terrain variation without explicit curricula. Combined with the signed-distance representation, this serves as our primary mechanism for encouraging whole-body and terrain awareness (Fig. 17). Although richer terrain models might further improve sim-to-real transfer, this simple scheme proved sufficient for our purposes. Additional domain-randomization details are provided in Appendix C.

C. Control

Costs and constraints. We use quadratic tracking objectives during training, validation, and deployment, with task-specific weights in Table VI. The cost penalizes deviations in base and

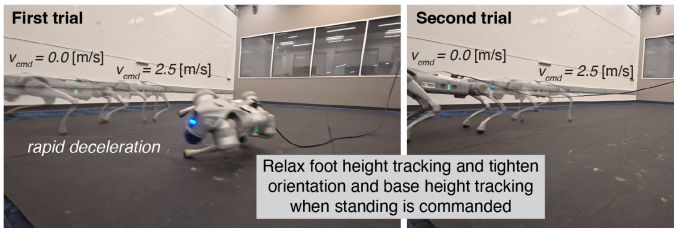

Fig. 19. **Cost tuning on hardware.** The standing controller, which is engaged when no velocity is commanded, is not tested in simulation; instead, we begin with the trotting cost using a zero commanded foot height. This formulation fails to decelerate the robot at high speeds because it over-penalizes foot height and under-penalizes orientation and height tracking, causing the robot to fall by not taking an extra stabilizing step. After adjusting these weights, the controller achieves stable deceleration from sprinting.

joint motion, torque effort, gait phase/foot height, and includes a trajectory-drift term that reduces long-horizon bias.

Training intentionally uses a simple objective—no gait tracking, no tuned coefficients, and no behavior-specific shaping—to avoid implicit biases toward particular gaits. At deployment, locomotion behaviors (trot, bound, pace, gallop, rear, tripod, and transitions) are obtained solely by adding cost terms, adding constraints, or adjusting weight magnitudes. Constraints are required for bounding, galloping, and tripod gaits. Bounding and galloping use collision-avoidance constraints on the thigh and upper shank, $\phi_{\text{thigh}}, \phi_{\text{shank}} > 0$, with relaxation $\delta = 5 \cdot 10^{-4}$ and coefficient $\beta = 10^{-3}$. Tripod additionally enforces base-height bounds $-0.02 < \varepsilon_z < 0.08$ m, with relaxation $\delta = 0.01$ and coefficient $\beta = 1.0$. Checkpoints for evaluation are selected using cumulative trotting performance on flat ground across randomized initial heights and directions.

Behavior stitching and synthesis. Costs, constraints, and commands are regenerated at each time step and appended to the end of the planning horizon, making the trajectory optimization time-varying. Directly applying new gait parameters (e.g., switching from a 2-Hz trot to a 3-Hz gallop in one step) produces jerky motions or failures because gait-specific foot-height commands do not guarantee a safe transition. To avoid this, a transition-specific cost (Table VI) is activated whenever the cost definition changes. Its role is to temporarily suppress gait-specific tracking, allowing the optimizer to discover a valid transition. This is necessary even when gaits differ only slightly (e.g., bound vs. gallop). Commands and weights are additionally smoothed with an exponential moving average before being appended to the horizon. Most cost weights, constraints, and reference commands were first tuned in simulation after training and then adjusted on hardware to improve behavior quality and sim-to-real transfer.

D. Results

Using a single dynamics model and estimator trained entirely offline and kept frozen at test time, the controller generates a wide range of whole-body behaviors by modifying only costs, constraints, and commands online (Fig. 1 and 8). Domain randomization during training, together with signed-distance state augmentation, enables robust terrain- and body-aware planning. As shown in Fig. 17, the MPC maintains accurate collision-awareness and contact reasoning across uneven slopes, slippery surfaces, and deep debris, recovering from disturbances such as falls.

Indoors, the same model produces diverse locomotion patterns—including bounding, galloping, load-carrying, and multi-directional push recovery (Fig. 18). Cost tuning on hardware further refines behavior quality, allowing the controller to correct failure modes such as high-speed deceleration instability (Fig. 19). Because we include explicit behavior stitching, the system can transition smoothly between gaits (Fig. 20).

Finally, the framework also synthesizes high-coordination behaviors in simulation, such as stable rearing, rotary galloping over rough terrain, and three-legged locomotion, again with no additional training (Fig. 21). Together, these results illustrate how offline-learned smooth neural dynamics, combined with online MPC, yield flexible and versatile behavior generation across a broad set of scenarios.

IX. Conclusion

We set out to understand whether learned full–order dynamics models can support gradient-based model predictive control for legged robots, despite the discontinuities and stiffness introduced by contact and imperfect neural models. Our study shows that standard neural networks reproduce the same optimization pathologies that hinder classical trajectory optimization under contact dynamics, and that datasets for legged-robot dynamics contain heavy-tailed residuals that are poorly modeled by Gaussian objectives. These observations motivated our smooth neural surrogate, a lightweight architecture with tunable first- and second–order smoothness, and a heavy-tailed training objective that better reflects the empirical distribution of model errors.

Across a range of controlled experiments and hardware demonstrations, we showed that smooth neural surrogates provide substantially more informative derivatives for planning with learned dynamics and enable gradient-based methods to outperform strong sampling-based baselines in terms of scalability, convergence, and computational efficiency. Combined with a smooth learned estimator and domain randomization, the approach allows a single frozen model to synthesize a wide variety of legged behaviors at test time, including locomotion on uneven terrain and multi-gait transitions, without behavior-specific training.

Our results suggest that relatively simple architectural and training choices can substantially improve the compatibility between learned dynamics and gradient-based planners, enabling a practical and scalable alternative to both hand-designed models and sampling-heavy MPC pipelines for real world deployment. We believe this opens a path toward more flexible, general-purpose model-based controllers that retain the adaptability of MPC while benefiting from the representational power of deep learning.

A. Limitations and Future Work

While effective in our setting, our approach presents several opportunities for future work that we leave unaddressed:

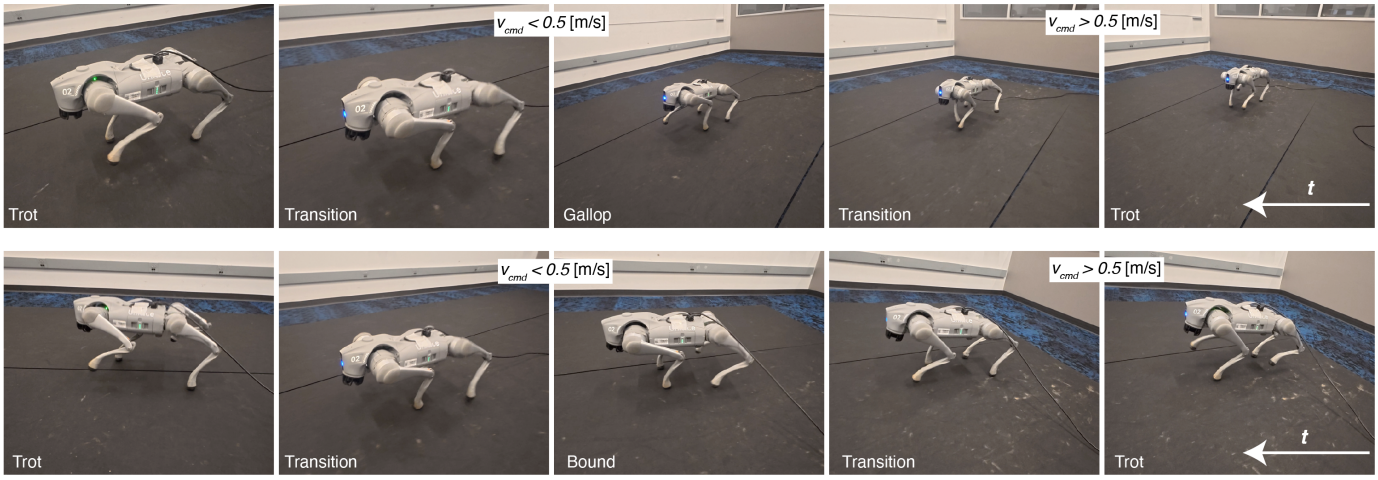

Fig. 20. **Swapping cost and constraints in real time.** At each control step, the MPC solves a distinct objective. Low velocity commands use the default trotting cost, while crossing the 0.5 m/s threshold triggers the transition cost, which provides no explicit gait reference and allows the controller to plan its own steps between gaits. **Top:** For velocity commands above 0.5 m/s, the controller blends the transition cost with the galloping objective using an exponential moving average. **Bottom:** Similarly, velocity commands above 0.5 m/s blend the transition cost with the bounding objective.

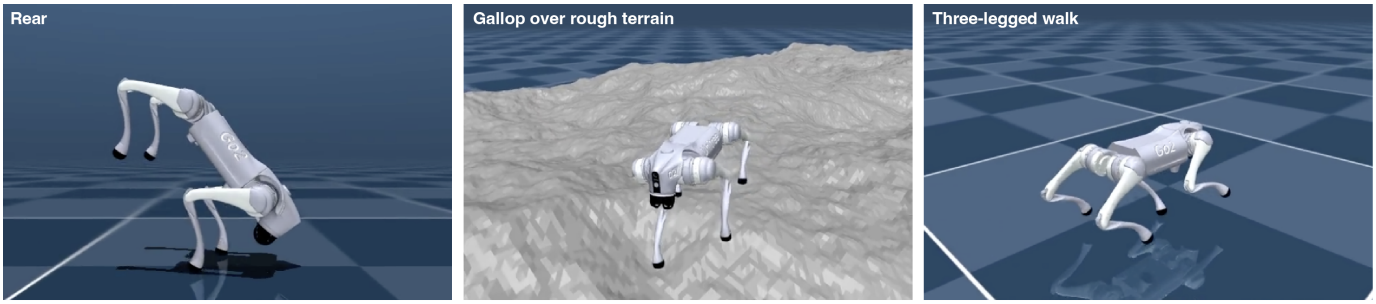

Fig. 21. **High-coordination behaviors synthesized without additional training using smooth neural surrogates and GGN-MPC. Left:** Stable balance in the rearing position. **Middle:** Rotary gallop over rough terrain at 1.5 meters per second. **Right:** Three-legged locomotion.

- **Contact-rich manipulation.** Extending smooth neural surrogates to dexterous manipulation tasks may reveal how neural smoothing and heavy-tailed training interact with richer contact geometries, stick–slip effects, and large numbers of contact points.
- **Policy learning.** Smooth neural surrogates may help stabilize model-based policy learning by providing better-behaved, or even adaptive model gradients, potentially complementing or improving model-based RL strategies that rely on differentiable simulation.
- **Perceptual observations.** Integrating raw perceptual inputs (vision, depth, tactile sensing) into both the dynamics and estimator models would further improve terrain awareness and whole-body planning.
- **Neural architectures.** Although Lipschitz constraints have been applied to a variety of architectures (e.g., CNNs, attention-based models), the benefits of Lipschitz-based weight normalization in higher-capacity networks remain largely unexplored–particularly in robotics and control.

These directions suggest that smooth neural surrogates may have value not only as standalone dynamics models for MPC, but also as building blocks that complement and enhance a broad class of scalable, contact-rich learning and control methods.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation Graduate Research Fellowship Program, ...

APPENDIX A
PROOF FOR CURVATURE BOUND ON SMOOTH NEURAL SURROGATE.

Proposition. Let $f = f_L \circ \cdots \circ f_1$ be a composition of L twice continuously differentiable mappings. For each layer define

$$c_\ell := \sup_x \|Df_\ell(x)\|_p, \qquad L_2(f_\ell) := \sup_x \|D^2 f_\ell(x)\|_p. \quad (41)$$

Assume *layerwise* 2-*smoothness*: there exists a constant $\alpha_2 > 0$ such that

$$L_2(f_\ell) \le \alpha_2 c_\ell^2 \qquad \forall\, \ell. \quad (42)$$

Then the composition satisfies

$$L_2(f) \;\le\; \alpha_2 \Big(\prod_{j=1}^{L} c_j\Big) \sum_{\ell=1}^{L} c_\ell \prod_{j<\ell} c_j. \quad (43)$$

Proof. Let $h_\ell = f_\ell \circ \cdots \circ f_1$. By the chain rule,

$$Df(x) = J_{f_L}(h_{L-1}(x)) \cdots J_{f_1}(x). \quad (44)$$

TABLE VI
COST TERMS AND WEIGHTS USED FOR NEURAL MPC ON THE GO2 QUADRUPED ACROSS BEHAVIORS

Cost Term		Training	Trot	Bound/Gallop	Pace	Tripod	Transition	Stand
Name	Equation	Weight						
Base orientation tracking	$\|\log(\hat{R}_{\mathcal{W}}^{\top} R_{\mathcal{W}}^{\text{cmd}})^{\vee}\|^2$	1	1	0.65	1	0.7	Prev.	2
Base height tracking	$\|\hat{z} - z^{\text{cmd}}\|^2$	5	5	1.25	5	0.5	Prev.	6
Base linear velocity tracking	$\|\hat{v}_{\mathcal{B}} - v_{\mathcal{B}}^{\text{cmd}}\|^2$	$3{\cdot}10^{-2}$	$5{\cdot}10^{-2}$	$5{\cdot}10^{-2}$	$5{\cdot}10^{-2}$	$5{\cdot}10^{-2\dagger\dagger}$	Prev.	$5{\cdot}10^{-2}$
Base angular velocity tracking	$\|\hat{\omega}_{\mathcal{B}} - \omega_{\mathcal{B}}^{\text{cmd}}\|^2$	10^{-3}	10^{-3}	$5{\cdot}10^{-4}$	10^{-3}	$5{\cdot}10^{-4\ddagger}$	Prev.	10^{-3}
Gait / foot height tracking	$\|\hat{\phi}_{\text{foot}} - \phi_{\text{foot}}^{\text{cmd}}\|^2$	0	2	7	4	$7^{\dagger}$	Prev.	0.5
Nominal joint position penalty	$\|\hat{q}_J - q_J^{\text{nom}}\|^2$	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}	$5{\cdot}10^{-4}$	10^{-2}
Joint velocity penalty	$\|\hat{v}_J\|^2$	10^{-8}	10^{-2}	$5{\cdot}10^{-8}$	10^{-2}	$5{\cdot}10^{-8}$	10^{-6}	10^{-2}
Torque penalty	$\|\hat{\tau}\|^2$	$2{\cdot}10^{-6}$	$4{\cdot}10^{-6}$	$4{\cdot}10^{-6}$	$4{\cdot}10^{-6}$	$4{\cdot}10^{-6}$	$4{\cdot}10^{-6}$	$4{\cdot}10^{-6}$
Pos. mech. work penalty	$\|\max(0, \hat{\tau} \cdot \hat{v}_J)\|^2$	0	10^{-6}	10^{-6}	10^{-6}	10^{-6}	Prev.	10^{-6}
Drift penalty*	$\|\sum_{t=0}^{T}(\hat{v}_{xy}(t) - v_{xy}^{\text{cmd}}(t))dt\|^2$	0	10^{-5}	10^{-5}	10^{-5}	10^{-5}	Prev.	10^{-5}

Notes: All costs are stagewise unless marked with *, which denotes a trajectory-level (global) cost. †: FR foot has a weight of 35. ††: Velocity v_y as a weight of 25. ‡: Angular velocity ω_z as a weight of 25.

Differentiating again yields

$$D^2 f(x) = \sum_{\ell=1}^{L} A_\ell \, D^2 f_\ell(h_{\ell-1}(x)) \circ (T_\ell, T_\ell), \tag{45}$$

where

$$A_\ell = J_{f_L} \cdots J_{f_{\ell+1}}, \qquad T_\ell = J_{f_{\ell-1}} \cdots J_{f_1}, \tag{46}$$

where $(T_\ell, T_\ell) : (u, v) \mapsto (T_\ell u, T_\ell v)$ and $D^2 f_\ell(x)$ is a bilinear form. Thus

$$(D^2 f_\ell(x) \circ (T_\ell, T_\ell))(u, v) = D^2 f_\ell(x)\big(T_\ell u, \, T_\ell v\big), \tag{47}$$

which is the pullback of the Hessian by the linear map T_ℓ. Using submultiplicativity $\|AB\| \le \|A\| \, \|B\|$ of the operator norm and the bound $\|B \circ (T_\ell, T_\ell)\| \le \|B\| \, \|T_\ell\|^2$, we obtain

$$\|A_\ell D^2 f_\ell(T_\ell, T_\ell)\| \le \|A_\ell\| \, \|D^2 f_\ell\| \, \|T_\ell\|^2. \tag{48}$$

Since $\|J_{f_j}\| \le c_j$,

$$\|A_\ell\| \le \prod_{j>\ell} c_j, \qquad \|T_\ell\| \le \prod_{j<\ell} c_j. \tag{49}$$

Layerwise 2-smoothness provides $\|D^2 f_\ell\| \le \alpha_2 c_\ell^2$. Hence each summand of (45) is bounded by

$$\alpha_2 \Big(\prod_{j>\ell} c_j \Big) c_\ell^2 \Big(\prod_{j<\ell} c_j \Big)^2. \tag{50}$$

Factor out the full product $\prod_{j=1}^{L} c_j$:

$$\Big(\prod_{j>\ell} c_j \Big) c_\ell^2 \Big(\prod_{j<\ell} c_j \Big)^2 = \Big(\prod_{j=1}^{L} c_j \Big) \Big(c_\ell \prod_{j<\ell} c_j \Big). \tag{51}$$

Summing over ℓ and taking a supremum over x gives

$$L_2(f) \le \alpha_2 \Big(\prod_{j=1}^{L} c_j \Big) \sum_{\ell=1}^{L} c_\ell \prod_{j<\ell} c_j, \tag{52}$$

which completes the proof.

APPENDIX B
SMOOTH NEURAL SURROGATE EXPERIMENTS AND IMPLEMENTATION DETAILS

Curvature budget selection. For second-order smoothness, we construct a curvature budget d_{ub} from the same first-order budget. Although d_{ub} could be chosen independently of c_{ub}, we scale it consistently to compare first- and second-order constraints under similar regularization strength. We define

$$d_{\text{ub}} = c_{\text{ub}} \sum_{\ell=1}^{L} c_{\text{ub}}^{\ell/L}, \tag{53}$$

which corresponds to the curvature bound that would arise if all layers shared equal Lipschitz constants $c_{\text{ub}}^{1/L}$. This construction does not enforce equality across layers; it simply provides a consistent scaling.

2-D shape interpolation. We train neural signed-distance functions in (x, y) conditioned on a latent variable z, yielding $d \approx f_\theta(x, y, z)$. The standard MLP uses an MSE loss, and the SNS and Lipschitz MLPs include a first-order smoothness constraint with $c_{\text{ub}} = 8.0$. Our dataset contains 500K uniformly sampled points per shape. All networks use softplus activations, five hidden layers with 192 units, and we train them for 200 epochs with a learning rate of 0.003.

1-D nonsmooth functions. For Figs. 4 and 5, we train on 15K uniformly sampled (x, y) pairs with $y \approx f_\theta(x)$ using an MSE loss. We train the ReLU example for 3000 epochs with $c_{\text{ub}} = 1.0$, and the nonsmooth-objective example for 1000 epochs with $c_{\text{ub}} = 1.35$ and $\lambda = 0.1$. Both models use softplus activations, four hidden layers with 64 units, and a learning rate of 0.01. The nonsmooth function (Fig. 5) is defined as

$$f(x) = \begin{cases} -0.6x - 2.0, & x < -3, \\ -0.2, & -3 \le x < 1, \\ -0.6, & 1 \le x < 1.3, \\ -0.6 + 0.8(x - 1.3)^2, & 1.3 \le x < 2.2, \\ mx + b, & x \ge 2.2, \end{cases} \tag{54}$$

TABLE VII
DOMAIN RANDOMIZATION DISTRIBUTIONS FOR GO2 QUADRUPED

Quantity	Distribution
Link masses m [kg]	$\mathcal{U}(0.975\,m_0,\ 1.025\,m_0)$
Base COM offset [mm]	$\mathcal{U}(\pm 3)$ per axis
Link COM offsets [mm]	$\mathcal{U}(\pm 1)$ per axis
Joint damping [N·m·s/rad]	$\mathcal{U}(0.0,\ 0.05)$
Joint friction [N·m·s/rad]	$\mathcal{U}(0.0,\ 0.25)$
Joint armature [kg·m^2]	$\mathcal{U}(0,\ 5{\times}10^{-5})$
Foot radius r [m]	$\mathcal{U}(0.95\,r_0,\ 1.05\,r_0)$
Sliding friction μ_{slide}	$\mathcal{U}(0.2,\ 1.0)$
Rolling/torsional friction	Scaled μ_{slide}, clipped to $[0.1,\ 1.0]$
Terrain tilt axis	Uniform on unit sphere
Terrain tilt angle [rad]	$\mathcal{U}(0,\ 0.5)$
Latency [ms]	$\mathcal{U}\{10, 15\}$
Measurement noise:	
$r_{\mathcal{W}}^{6D}$	$\mathcal{U}(\pm 0.001)$
q_J [rad]	$\mathcal{U}(\pm 0.01)$
v_J [rad/s]	$\mathcal{U}(\pm 0.1)$
$\dot{v}_{\mathcal{B}}$ [m/s]	$\mathcal{U}(\pm 0.08)$
$\omega_{\mathcal{B}}$ [rad/s]	$\mathcal{U}(\pm 0.025)$

where the coefficients m and b are chosen so that the final segment connects to the quadratic segment at $x = 2.2$ and satisfies $f(5) = 1$.

Smoothing particle–mass contact dynamics. We generate 500 trajectories (6 s at 50 Hz) from random initial states $x_0 = [q_0, v_0] \sim [\mathcal{U}(0.1, 4.0), \mathcal{U}(-5.0, 5.0)]$ under sinusoidal forcing $u_t = 2g\sin(2\pi\omega t)$, $\omega \sim \mathcal{U}(0.1, 3.0)$. The dynamics follow $\ddot{q} = -g + u$ with inelastic collisions at $q = 0$. Models are trained with a single-step MSE loss $x_{t+1} \approx x_t + f_\theta(x_t, u_t), dt$ using Softplus networks with five hidden layers of 192 units, for 500 epochs at a learning rate of 10^{-3}, with $c_{\text{ub}} = 50$ and $\lambda = 0.2$.

APPENDIX C
QUADRUPED EXPERIMENTS AND IMPLEMENTATION DETAILS

Simulation environments. We run MuJoCo simulation at 200 Hz and perform control and data collection at 50 Hz. We keep MuJoCo's default contact settings, including `impratio` $= 1$, which is known to produce stiff contacts and large derivatives in legged locomotion [36]. These settings ensure that our training data contain the nonsmooth behaviors characteristic of contact dynamics.

At every reset, we randomize physical and terrain parameters as summarized in Table VII, including link masses, centers of mass, joint properties, friction coefficients, foot geometry, and measurement noise. We randomize the ground-plane slope by sampling a uniform tilt axis and angle. For actuator system identification, we train our initial models using MuJoCo's position-actuator interface with gains $K_p \sim \mathcal{U}(23,\ 27)$ N·m/rad and $K_d \sim \mathcal{U}(2.5,\ 3.5)$ N·m·s/rad. We convert all collision geometries in the XML to capsules or spheres to maintain stable signed-distance computations in our MuJoCo version.

Data collection pipeline. Each episode lasts 5.12 s (256 control steps), and we simulate 512 environments in parallel. Our replay buffer stores 20,480 episodes (∼5M transitions) and cycles in a FIFO manner. We vary reference commands across episodes and randomize robot states at every reset. We collect an initial bootstrap buffer using random uniform spline actions, then alternate between collecting on-policy trajectories and updating our models using the replay buffer.

Hybrid simulation and actuator models. Our control pipeline uses a hybrid dynamics model that combines MuJoCo rigid-body simulation with a smooth neural surrogate actuator model. For all 12 joints, we log torque, commanded position, actual position, and velocity at 200 Hz. Following [1], [73], we train a single shared actuator network and introduce randomized actuation latency.

For each joint j, the actuator predicts torque from histories of joint-position error $q_J^e = q_J - q_J^d$ and joint velocities:

$$\hat{\tau}_j = A_\psi\left(q^{e,j}_{J,\,t-P:t},\ v^j_{J,\,t-P:t}\right), \tag{55}$$

using $P = 8$ steps of 200 Hz data. We train a second actuator network on 50 Hz data with $P = 3$ to capture mechanical work and torque penalties accurately during MPC rollouts.

Model training details. All learned models use Mish activations and we update them 500 times between data-collection episodes using the Lion optimizer [74]. Our global loss uses weights $\lambda_0 = \lambda_1 = 0.5$, $\lambda_2 = 0.05$, and smoothness penalties $\lambda_3 = 10.0$, $\lambda_4 = 10^{-5}$. First-order SNS models use sensitivity budgets of 10^4 (dynamics) and 1 (estimator). We initialize the estimator's noise covariance as $\frac{1}{2}\Sigma_{\mathcal{D}}$, computed from the initial replay buffer. All models use history length $H = 8$, rollout horizon $T = 19$, and a batch size of 512 trajectories (Algorithms 2–3).

Architectures and learning rates for experiments (Sec. V–VI). For Section V comparisons:

- Cauchy-loss SNS dynamics / estimator: learning rates 0.0008 / 0.0004,
- Gaussian-loss SNS dynamics / estimator: learning rates 0.0004 / 0.0002,
- Standard and weight-decay MLPs: 0.0001 (dynamics) / 0.00005 (estimator).

All ablation models use four hidden layers with widths $2x_0$ (dynamics) and $1.5y_0$ (estimator), where x_0 and y_0 denote the input dimensions. We use a consistent history length $H = 8$.

We use linear splines with $k = 5$ knots in Section V and evaluate 16 rollouts per greedy GGN-MPC line-search step. For gait-sequence comparisons in Fig. 8, we use $k = 6$ knots; GGN-MPC runs one iteration with 16 rollouts, and DIAL-MPC runs four iterations with 128 rollouts. All comparisons use our best-performing SNS model ($c_{ub} = 10^4$) and the best standard MLP.

Model architectures for deployment. For hardware deployment, we use four-layer MLPs with widths:

- Dynamics: $[1.5x_0,\ 1.5x_0,\ 1.25x_0,\ 1.0x_0]$,
- Estimator: $[1.5y_0,\ 1.25y_0,\ 1.0y_0,\ 0.75y_0]$.

These architectures incorporate the smooth-surrogate constraints described above and are used in all real-world experiments.

Reference commands. A periodic gait generator produces foot-height references $\phi^{\text{cmd}}_{\text{foot}}$ parameterized by cadence, phase, duty ratio, and swing height [4]. In tripod mode, we assign the disabled leg a fixed 0.1 m foot-height command; during rearing, we disable both rear legs.

Base height varies throughout training and deployment with nominal height $z^{\text{cmd}} = 0.27$. Nominal joint positions are $q_J^{\text{nom}} = [0,\, 0.9,\, -1.8] \times 4$, with adjustments for tripod and outdoor trotting. We smooth commanded v_{xy}^{cmd} and ω_z^{cmd} using an exponential moving average. Orientation commands follow a heuristic: pitch upward when moving forward and downward when moving backward.

Hardware deployment. We run all algorithms on a Linux workstation with an Nvidia 4090 GPU and deploy them to the Unitree Go2 via a wired connection. Model quantization would enable onboard-only operation, but we leave this for future work.

To convert MPC outputs into joint torques, we combine impedance and feedforward terms:

$$\begin{aligned}\tau = K_p(q_J^d - q_J) - K_d v_J \\ + K_p^{\text{ff}}(q_{J,t+1}^* - q_J) - K_d^{\text{ff}}(v_{J,t+1}^* - v_J),\end{aligned} \tag{56}$$

where starred quantities denote MPC-predicted next-step joint positions and velocities. These feedforward terms improve tracking during missed footholds or when the sim-to-real gap is large. Deployment uses linear splines with $k = 6$ knots and GGN-MPC with 8 rollouts per greedy line-search step.

REFERENCES

[1] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[2] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *arXiv preprint arXiv:2107.04034*, 2021.

[3] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.

[4] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive sampling: Real-time behaviour synthesis with mujoco," *arXiv preprint arXiv:2212.00541*, 2022.

[5] M. Lutter, L. Hasenclever, A. Byravan, G. Dulac-Arnold, P. Trochim, N. Heess, J. Merel, and Y. Tassa, "Learning dynamics models for model predictive agents," *arXiv preprint arXiv:2109.14311*, 2021.

[6] M. Posa and R. Tedrake, "Direct trajectory optimization of rigid body dynamical systems through contact," in *Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics*. Springer, 2013, pp. 527–542.

[7] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM review*, vol. 59, no. 4, pp. 849–904, 2017.

[8] G. Kim, D. Kang, J.-H. Kim, S. Hong, and H.-W. Park, "Contact-implicit model predictive control: Controlling diverse quadruped motions without pre-planned contact modes or trajectories," *The International Journal of Robotics Research*, vol. 44, no. 3, pp. 486–510, 2025.

[9] H.-T. D. Liu, F. Williams, A. Jacobson, S. Fidler, and O. Litany, "Learning smooth neural functions via lipschitz regularization," in *ACM SIGGRAPH 2022 Conference Proceedings*, 2022, pp. 1–13.

[10] M. Rosca, T. Weber, A. Gretton, and S. Mohamed, "A case for new neural network smoothness constraints," in *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, ser. Proceedings of Machine Learning Research, J. Zosa Forde, F. Ruiz, M. F. Pradier, and A. Schein, Eds., vol. 137. PMLR, 12 Dec 2020, pp. 21–32. [Online]. Available: https://proceedings.mlr.press/v137/rosca20a.html

[11] C. Anil, J. Lucas, and R. Grosse, "Sorting out lipschitz function approximation," in *International conference on machine learning*. PMLR, 2019, pp. 291–301.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[13] Y. Yoshida and T. Miyato, "Spectral norm regularization for improving the generalizability of deep learning," *arXiv preprint arXiv:1705.10941*, 2017.

[14] X. Song, J. Duan, W. Wang, S. E. Li, C. Chen, B. Cheng, B. Zhang, J. Wei, and X. S. Wang, "Lipsnet: a smooth and robust neural network with adaptive lipschitz constant for high accuracy optimal control," in *International Conference on Machine Learning*. PMLR, 2023, pp. 32 253–32 272.

[15] W. G. Y. Tan and Z. Wu, "Robust machine learning modeling for predictive control using lipschitz-constrained neural networks," *Computers & Chemical Engineering*, vol. 180, p. 108466, 2024.

[16] J. Xu, E. Heiden, I. Akinola, D. Fox, M. Macklin, and Y. Narang, "Neural robot dynamics," *arXiv preprint arXiv:2508.15755*, 2025.

[17] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in neural information processing systems*, vol. 31, 2018.

[18] J. Amigo, R. Khorrambakht, E. Chane-Sane, N. Mansard, and L. Righetti, "First order model-based rl through decoupled backpropagation," *arXiv preprint arXiv:2509.00215*, 2025.

[19] A. Byravan, L. Hasenclever, P. Trochim, M. Mirza, A. D. Ialongo, Y. Tassa, J. T. Springenberg, A. Abdolmaleki, N. Heess, J. Merel *et al.*, "Evaluating model-based planning and planner amortization for continuous control," *arXiv preprint arXiv:2110.03363*, 2021.

[20] M. Parmar, M. Halm, and M. Posa, "Fundamental challenges in deep learning for stiff contact dynamics," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 5181–5188.

[21] P. Roth, J. Frey, C. Cadena, and M. Hutter, "Learned perceptive forward dynamics model for safe and platform-aware robotic navigation," *arXiv preprint arXiv:2504.19322*, 2025.

[22] W. Xiao, H. Xue, T. Tao, D. Kalaria, J. M. Dolan, and G. Shi, "Anycar to anywhere: Learning universal dynamics model for agile and adaptive mobility," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 8819–8825.

[23] P. Tsakalides and C. Nikias, "Maximum likelihood localization of sources in noise modeled as a cauchy process," in *Proceedings of MILCOM'94*. IEEE, 1994, pp. 613–617.

[24] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 2006.

[25] M. P. Kelly, "Transcription methods for trajectory optimization: a beginners tutorial," *arXiv preprint arXiv:1707.00284*, 2017.

[26] H. Xue, C. Pan, Z. Yi, G. Qu, and G. Shi, "Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 4974–4981.

[27] E. Lee, S. A. Moore, and B. Chen, "Sym2real: Symbolic dynamics with residual learning for data-efficient adaptive control," *arXiv preprint arXiv:2509.15412*, 2025.

[28] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on robot learning*. PMLR, 2020, pp. 1101–1112.

[29] J. Alvarez-Padilla, J. Z. Zhang, S. Kwok, J. M. Dolan, and Z. Manchester, "Real-time whole-body control of legged robots with model-predictive path integral control," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 14 721–14 727.

[30] Y. Tian, X. Zheng, X. Zhang, and Y. Jin, "Efficient large-scale multi-objective optimization based on a competitive swarm optimizer," *IEEE Transactions on Cybernetics*, vol. 50, no. 8, pp. 3696–3708, 2019.

[31] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 89–103, 2018.

[32] M. A. Patterson and A. V. Rao, "Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, pp. 1–37, 2014.

[33] W.-C. Huang, A. Aydinoglu, W. Jin, and M. Posa, "Adaptive contact-implicit model predictive control with online residual learning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 5822–5828.

[34] S. Le Cleac'h, T. A. Howell, S. Yang, C.-Y. Lee, J. Zhang, A. Bishop, M. Schwager, and Z. Manchester, "Fast contact-implicit model predictive control," *IEEE Transactions on Robotics*, vol. 40, pp. 1617–1629, 2024.

[35] T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable simulator for robotics," *arXiv preprint arXiv:2203.00806*, vol. 9, no. 2, p. 4, 2022.

[36] J. Z. Zhang, T. A. Howell, Z. Yi, C. Pan, G. Shi, G. Qu, T. Erez, Y. Tassa, and Z. Manchester, "Whole-body model-predictive control of legged robots with mujoco," *arXiv preprint arXiv:2503.04613*, 2025.

[37] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (ToG)*, vol. 31, no. 4, pp. 1–8, 2012.

[38] T. Erez and E. Todorov, "Trajectory optimization for domains with contacts using inverse dynamics," in *2012 IEEE/RSJ International conference on intelligent robots and systems*. IEEE, 2012, pp. 4914–4919.

[39] H. J. T. Suh, T. Pang, and R. Tedrake, "Bundled gradients through contact via randomized smoothing," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4000–4007, 2022.

[40] T. Pang, H. T. Suh, L. Yang, and R. Tedrake, "Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models," *IEEE Transactions on robotics*, vol. 39, no. 6, pp. 4691–4711, 2023.

[41] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin, "Accelerated policy learning with parallel differentiable simulation," *arXiv preprint arXiv:2204.07137*, 2022.

[42] I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg, "Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation," *arXiv preprint arXiv:2405.17784*, 2024.

[43] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[44] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *Advances in neural information processing systems*, vol. 32, 2019.

[45] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[46] R. Huang, H. Balim, H. Yang, and Y. Du, "Flexible locomotion learning with diffusion model predictive control," *arXiv preprint arXiv:2510.04234*, 2025.

[47] S. Pfrommer, M. Halm, and M. Posa, "Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations," in *Conference on Robot Learning*. PMLR, 2021, pp. 2279–2291.

[48] Y. D. Zhong, B. Dey, and A. Chakraborty, "Extending lagrangian and hamiltonian neural networks with differentiable contact models," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 910–21 922, 2021.

[49] A. Hochlehnert, A. Terenin, S. Sæmundsson, and M. Deisenroth, "Learning contact dynamics using physically structured neural networks," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 2152–2160.

[50] B. Sukhija, N. Köhler, M. Zamora, S. Zimmermann, S. Curi, A. Krause, and S. Coros, "Gradient-based trajectory optimization with learned dynamics," *arXiv preprint arXiv:2204.04558*, 2022.

[51] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "Neuralsim: Augmenting differentiable simulators with neural networks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9474–9481.

[52] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[53] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree, "Regularisation of neural networks by enforcing lipschitz continuity," *Machine Learning*, vol. 110, no. 2, pp. 393–416, 2021.

[54] X. Qi, J. Wang, Y. Chen, Y. Shi, and L. Zhang, "Lipsformer: Introducing lipschitz continuity to vision transformers," *arXiv preprint arXiv:2304.09856*, 2023.

[55] T. Mlotshwa, H. van Deventer, and A. S. Bosman, "Cauchy loss function: Robustness under gaussian and cauchy noise," in *Southern African Conference for Artificial Intelligence Research*. Springer, 2022, pp. 123–138.

[56] M. T. El-Melegy, M. H. Essai, and A. A. Ali, "Robust training of artificial feedforward neural networks," in *Foundations of Computational, Intelligence Volume 1: Learning and Approximation*. Springer, 2009, pp. 217–242.

[57] J. T. Barron, "A general and adaptive robust loss function," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4331–4339.

[58] A. Beck, *First-order methods in optimization*. SIAM, 2017.

[59] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM review*, vol. 60, no. 2, pp. 223–311, 2018.

[60] S. A. Moore, B. P. Mann, and B. Chen, "Automated global analysis of experimental dynamics through low-dimensional linear embeddings," *arXiv preprint arXiv:2411.00989*, 2024.

[61] J.-H. Kim, S. Hong, G. Ji, S. Jeon, J. Hwangbo, J.-H. Oh, and H.-W. Park, "Legged robot state estimation with dynamic contact event information," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6733–6740, 2021.

[62] R. Hartley, M. Ghaffari, R. M. Eustice, and J. W. Grizzle, "Contact-aided invariant extended kalman filtering for robot state estimation," *The International Journal of Robotics Research*, vol. 39, no. 4, pp. 402–430, 2020.

[63] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4630–4637, 2022.

[64] G. Kim, Y.-H. Lee, and H.-W. Park, "A learning framework for diverse legged robot locomotion using barrier-based style rewards," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 10 004–10 010.

[65] G. Revach, N. Shlezinger, X. Ni, A. L. Escoriza, R. J. Van Sloun, and Y. C. Eldar, "Kalmannet: Neural network aided kalman filtering for partially known dynamics," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1532–1547, 2022.

[66] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick 1980: proceedings of a symposium held at the University of Warwick 1979/80*. Springer, 2006, pp. 366–381.

[67] H. Abarbanel, *Analysis of observed chaotic data*. Springer Science & Business Media, 2012.

[68] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural computation*, vol. 14, no. 7, pp. 1723–1738, 2002.

[69] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model-predictive control," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3402–3421, 2023.

[70] F. Messerer, K. Baumgärtner, and M. Diehl, "Survey of sequential convex programming and generalized gauss-newton methods," *ESAIM: Proceedings and Surveys*, vol. 71, pp. 64–88, 2021.

[71] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.* IEEE, 2005, pp. 300–306.

[72] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5745–5753.

[73] G. B. Margolis and P. Agrawal, "Walk these ways: Tuning robot control for generalization with multiplicity of behavior," in *Conference on Robot Learning*. PMLR, 2023, pp. 22–31.

[74] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu *et al.*, "Symbolic discovery of optimization algorithms," *Advances in neural information processing systems*, vol. 36, pp. 49 205–49 233, 2023.