Neural Robot Dynamics

Jie Xu[1], **Eric Heiden**[1], **Iretiayo Akinola**[1], **Dieter Fox**[1,2], **Miles Macklin**[1], **Yashraj Narang**[1]
[1]NVIDIA [2]University of Washington
https://neural-robot-dynamics.github.io

Abstract: Accurate and efficient simulation of modern robots remains challenging due to their high degrees of freedom and intricate mechanisms. Neural simulators have emerged as a promising alternative to traditional analytical simulators, capable of efficiently predicting complex dynamics and adapting to real-world data; however, existing neural simulators typically require application-specific training and fail to generalize to novel tasks and/or environments, primarily due to inadequate representations of the global state. In this work, we address the problem of learning generalizable neural simulators for robots that are structured as articulated rigid bodies. We propose *NeRD* (Neural Robot Dynamics), learned robot-specific dynamics models for predicting future states for articulated rigid bodies under contact constraints. *NeRD* uniquely replaces the low-level dynamics and contact solvers in an analytical simulator and employs a robot-centric and spatially-invariant simulation state representation. We integrate the learned *NeRD* models as an interchangeable backend solver within a state-of-the-art robotics simulator. We conduct extensive experiments to show that the *NeRD* simulators are stable and accurate over a thousand simulation steps; generalize across tasks and environment configurations; enable policy learning exclusively in a neural engine; and, unlike most classical simulators, can be fine-tuned from real-world data to bridge the gap between simulation and reality.

Keywords: Robot Model Learning; Robotics Simulation; Neural Simulation

1 Introduction

Simulation plays a crucial role in various robotics applications, such as policy learning [1, 2, 3, 4, 5, 6, 7], safe and scalable robotic control evaluation [8, 9, 10, 11], and computational optimization of robot designs [12, 13, 14]. Recently, neural robotics simulators have emerged as a promising alternative to traditional analytical simulators, as neural simulators can efficiently predict robot dynamics and learn intricate physics from real-world data. For instance, neural simulators have been leveraged to capture complex interactions challenging for analytical modeling [15, 16, 17, 18], or have served as learned world models to facilitate sample-efficient policy learning [19, 20].

However, existing neural robotics simulators typically require application-specific training, often assuming fixed environments [20, 21] or simultaneous training alongside control policies [22, 23]. These limitations primarily stem from their end-to-end frameworks with inadequate representations of the global simulation state, *i.e.*, neural models often substitute the entire classical simulator and directly map robot state and control actions (*e.g.*, target joint positions, target link orientations) to the robot's next state. Without encoding the environment in the state representation, the learned simulators have to implicitly memorize the task and environment details. Additionally, utilizing controller actions as input causes the simulators to overfit to particular low-level controllers used during training. Consequently, unlike classical simulators, these neural simulators often fail to generalize to novel state distributions (induced by new tasks), unseen environment setups, and customized controllers (*e.g.*, novel control laws or controller gains).

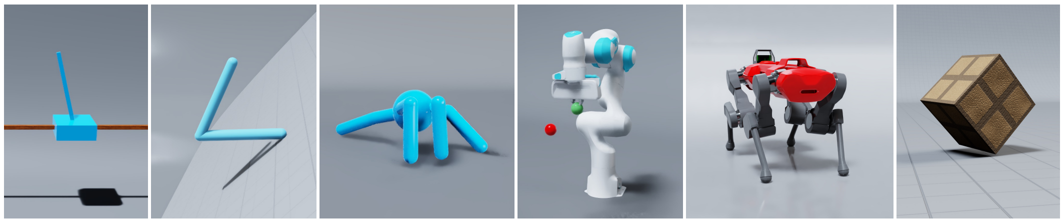


Figure 1: We propose ***NeRD***, learned robot-specific dynamics models for generalizable articulated rigid body simulation. We demonstrate our approach by training *NeRD* models on six diverse robotic systems, from left: *Cartpole, Double Pendulum, Ant, Franka, ANYmal, Cube Toss*.

In this work, we address the problem of learning generalizable neural simulators for articulated rigid-body robots. We envision a future where each robot is equipped with a neural simulator pretrained from analytical simulations. Such a simulator could conduct lifelong fine-tuning as the robot interacts with the real environment to accommodate wear-and-tear and environmental changes, and facilitate versatile skill learning in a digital twin powered by the continuously-updated simulator.

Toward this goal, we propose *Neural Robot Dynamics* (*NeRD*), a learned robot-specific dynamics model for predicting the evolution of articulated rigid-body states under contact constraints. *NeRD* is characterized by two key innovations: (1) a *hybrid prediction framework*, where *NeRD* uniquely replaces only the *application-agnostic* simulation modules – *i.e.*, the low-level forward dynamics and contact solvers – and leverages a general and compact representation describing the world surrounding robots; (2) a *robot-centric state representation*, where *NeRD* further improves the simulation state representation to explicitly enforce dynamics invariance under translation and rotation around the gravity axis, thus further enhancing *NeRD*'s spatial generalizability and training efficiency. Once trained, our *NeRD* models can (1) provide stable and accurate predictions over hundreds to thousands of simulation steps; (2) generalize to different tasks, environments, and low-level controllers; and (3) effectively fine-tune from real-world data to bridge sim-to-real gaps. Additionally, with our hybrid and modular design of *NeRD*, we integrate *NeRD* as an interchangeable backend solver within a state-of-the-art robotics simulator [24], enabling users to effortlessly reuse existing policy-learning environments and activate *NeRD* as a new physics backend through a single-line switch.

We train *NeRD* models on six different robotic systems (Fig. 1) to illustrate the broad applicability of our proposed methodology. We evaluate the trained *NeRD* models on extensive experiments in both simulation and real-world scenarios, including long-horizon dynamics prediction and policy learning on a diverse set of tasks that are unseen during *NeRD* model training. Due to the long-horizon stability and accuracy of *NeRD*, we demonstrate – for the first time, to the best of our knowledge – that robotic policies learned exclusively within a pretrained neural simulator can successfully achieve zero-shot deployment in the analytical simulator and even transfer directly to the real world.

2 Related Work

Neural physics simulations have been studied across diverse simulation domains, including cloth [25, 26, 27], fluid [28, 29], and continuum dynamics [30, 31]. Our work focuses on the subfield of neural simulation for articulated rigid-body dynamics in robotics.

Neural physics engines for single rigid bodies have modeled object-ground and inter-object interactions [15, 32, 33, 34]. ContactNets [15] learns implicit signed distance functions to capture the discontinuous cube-ground dynamics, while a subsequent method [33] employs graph neural networks (GNNs) to improve the prediction accuracy of the same task. Allen et al. [34] further model inter-object collisions with face interaction graph networks. Despite their advancements, these approaches are not readily extendable to articulated rigid bodies, limiting practicality in robotics applications.

Neural models predicting dynamics of articulated rigid bodies have been explored in model-based reinforcement learning and planning, known as world models. Most model-based RL methods [19, 20, 22, 23, 35, 36, 37, 38] predict future robot states directly from the current robot state and control actions, without explicit environment modeling, and jointly train the neural world models with control policies. Consequently, these world models lack generalizability to novel tasks, environments, and controllers. Some works in model-based planning decouple the simulation model training from planning. For instance, GNNs [39] have been utilized to model generalizable physics across articulated rigid bodies. But this approach still directly predicts state transition from robot state and action, primarily targeting 2D systems or contact-free dynamics. A concurrent work [16] pretrains a Bayesian network for modeling the dynamics of a loco-manipulation system but relies on analytical modeling for this particular robot, restricting its applicability to broader robot systems.

A recent work, LARP [21], couples dynamics and contact networks for modeling humanoid-ball interactions, but targets human motion reconstruction in computer vision, with the accuracy and applicability to robotic policy learning unverified. Physics-informed neural networks [40, 41] incorporate physics laws into learning generalizable dynamics of simple articulations, but their reliance on expert-modeled physics laws of each individual system limits their use in complex robot designs.

Hybrid neural simulation frameworks have also been proposed. NeuralSim [42] integrates neural models in localized components of a rigid-body simulator to improve friction and passive force modeling. But models' generalizability is limited by robot-state-only representations. Neural contact clustering [43] and neural collision detectors [44] accelerate contact algorithms. Residual physics is also studied [17, 18] for bridging sim-to-real gaps. These techniques complement ours, as collision detection generates contact information consumed by *NeRD*, and the residual physics augments *outputs* of an analytical simulator while we propose a generalizable *input* for a neural simulator.

3 NeRD: Neural Robot Dynamics

We now present *NeRD*, robot-specific neural dynamics models for articulated rigid-body simulation. We start by presenting the typical workflow of a classical articulated rigid-body simulator in §3.1. Next, in §3.2, we introduce the *hybrid prediction framework* of *NeRD*, which leverages a general and compact simulation state representation describing the world to enable generalization across applications. In §3.3, we further improve the representation by proposing a *robot-centric simulation state representation* to enforce spatial generalizability and improve the training efficiency of *NeRD*.

3.1 Preliminary: A Typical Robotics Simulation Workflow

We illustrate a typical workflow of a classical robotics simulator in Fig. 2(a). The user first sets up the simulator by importing a robot model with its initial state, and specifying an environment configuration (*e.g.*, ground, objects) and a low-level controller (*e.g.*, joint position control, end-effector control). At each time step t, the simulator takes as input the robot model, current robot state $\boldsymbol{s}_t$, the action command fed to the robot $\boldsymbol{a}_t$, and the scene configuration. It then performs collision detection to identify contact information for interacting physical parts, and executes the low-level controller to convert the action command into joint-space torques. Those, along with the robot state, serve as intermediate quantities that are processed by the dynamics and contact solvers, where physics equations are formulated and a numerical solver is employed to calculate the acceleration. Finally, the simulator performs time integration to obtain the new state of the robot.

Previous neural robotics simulators [19, 20, 22, 37] often adopt an *end-to-end* framework that substitutes the entire simulation engine with a neural model and directly maps the robot state $\boldsymbol{s}_t$ and action command $\boldsymbol{a}_t$ to the next robot state $\boldsymbol{s}_{t+1}$, without leveraging information regarding the scene and the controller, *i.e.*, $E2E(\boldsymbol{s}_t, \boldsymbol{a}_t) \rightarrow \boldsymbol{s}_{t+1}$. Such neural simulators are therefore forced to memorize the scene and the controller used for training and lack generalizability to novel applications.

3.2 Hybrid Prediction Framework of Neural Robot Dynamics

To train a generalizable neural simulator, we need a comprehensive representation to encode the scene and controller that generalizes across diverse applications. Inspired by the observation that the low-level dynamics and contact solvers in a classical simulator are application-agnostic, *NeRD*

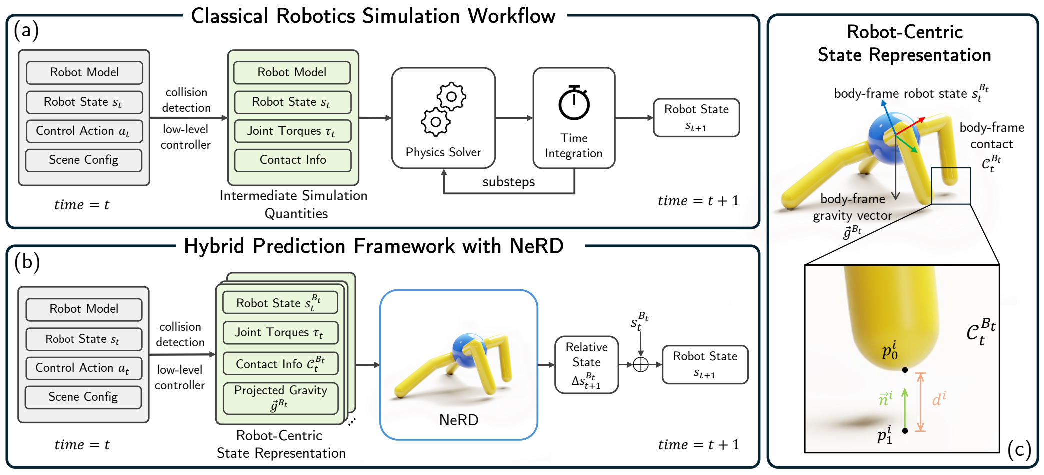


Figure 2: **Framework overview for Neural Robot Dynamics (*NeRD*). (a)** Workflow of a classical robotics simulator. The quantities shaded in green are application-agnostic. **(b)** Hybrid prediction framework of the *NeRD*-integrated simulator. Inputs to *NeRD* are the robot-centric state representations (illustrated in (**c**)) within a history window.

employs a *hybrid prediction framework* that replaces only the core physics components in a conventional simulator (Fig. 2(b)). This hybrid framework allows *NeRD* to leverage intermediate simulation quantities (*i.e.*, robot state, contact information, and joint-space torques) as a general and compact representation describing the full simulation state, providing all necessary information to evolve the robot dynamics regardless of the applications (*e.g.*, tasks, scenes, and controllers).

Formally speaking, let $\boldsymbol{s}_t = (\boldsymbol{x}_t, \boldsymbol{R}_t, \boldsymbol{q}_t, \boldsymbol{\phi}_t, \dot{\boldsymbol{q}}_t)$ denote the robot state at time t, where $\boldsymbol{x}_t$ and $\boldsymbol{R}_t$ are the position and orientation (represented as a quaternion) of the robot base, $\boldsymbol{q}_t$ denotes articulated joint angles, $\boldsymbol{\phi}_t$ is the spatial twist of the base (*i.e.*, 6D velocity), and $\dot{\boldsymbol{q}}_t$ are joint velocities. We define $\boldsymbol{\tau}_t$ as the joint-space torque and $\boldsymbol{\mathcal{C}}_t$ as contact-related quantities. We construct $\boldsymbol{\mathcal{C}}_t = \{\boldsymbol{c}_t^i\}$ by reusing the collision detection module in the classical simulator. For each pre-specified contact point $\boldsymbol{p}_0^i$ on the robot, we obtain its contact event quantities $\boldsymbol{c}_t^i = (\boldsymbol{p}_0^i, \boldsymbol{p}_1^i, \vec{\boldsymbol{n}}^i, d^i)$. Here $\boldsymbol{p}_1^i$ is the contact point on a non-robot shape, $\vec{\boldsymbol{n}}^i$ is the contact normal, and d^i is the contact distance.

Our neural robot dynamics model is a parametric function $NeRD_\theta\left(\{\boldsymbol{s}_k, \boldsymbol{\mathcal{C}}_k, \boldsymbol{\tau}_k\}_{k=t-h+1}^t\right)$ that maps the robot states, contacts, and joint torques within a history window of length h to the state difference $\Delta \boldsymbol{s}_{t+1} \triangleq \boldsymbol{s}_{t+1} \ominus \boldsymbol{s}_t$; here, $\ominus$ is defined to be the rotation difference $\mathbf{R}_{t+1}\mathbf{R}_t^{-1}$ for the base orientation and the subtraction operator for other state dimensions. The model is trained by minimizing the mean squared error between the prediction and the ground-truth state difference $\widehat{\Delta \boldsymbol{s}_{t+1}}$:

$$\mathcal{L}_\theta = \frac{1}{NS}\sum_N \|NeRD_\theta\left(\{\boldsymbol{s}_k, \boldsymbol{\mathcal{C}}_k, \boldsymbol{\tau}_k\}_{k=t-h+1}^t\right) - \widehat{\Delta \boldsymbol{s}_{t+1}}\|^2, \tag{1}$$

where N is the batch size and S is the dimension of the robot state. The next state $\boldsymbol{s}_{t+1}$ is then computed by $\boldsymbol{s}_{t+1} = \boldsymbol{s}_t \oplus NeRD_\theta\left(\{\boldsymbol{s}_k, \boldsymbol{\mathcal{C}}_k, \boldsymbol{\tau}_k\}_{k=t-h+1}^t\right)$. This concise state representation $\{\boldsymbol{s}_t, \boldsymbol{\mathcal{C}}_t, \boldsymbol{\tau}_t\}$ is a carefully designed outcome resulting from deeply integrating the neural models into the classical simulation framework and reusing the application-agnostic intermediate simulation quantities, thereby fundamentally providing the generalizability across diverse applications.

3.3 Robot-Centric State Representation

The dynamics of a robot remain invariant under spatial translation, as well as rotation around the gravity axis, provided that its interaction with the environment, such as contact forces, remains unchanged in the robot's body frame. Inspired by this, we enhance the simulation state representation by introducing a *robot-centric parameterization* to explicitly enforce such spatial invariance.

Specifically, we transform the robot state $\boldsymbol{s}_t$ and contact-related quantities $\boldsymbol{\mathcal{C}}_t$ into the robot's base frame $\boldsymbol{B}_t = (\boldsymbol{x}_t, \boldsymbol{R}_t)$, as shown in Fig. 2(c). For the robot articulation, we use the reduced coordinate state, which is spatially invariant; thus, we only need to transform the state of the robot base (*i.e.*, $\boldsymbol{x}_t$, $\boldsymbol{R}_t$, and $\boldsymbol{\phi}_t$) into the robot's base frame. To properly account for gravity when the robot rotates about axes other than the gravity axis, we treat gravity as an external force and augment the simulation state with gravity expressed in the robot's base frame. Additionally, the predicted state difference $\Delta \boldsymbol{s}_{t+1}$ (*i.e.*, network's output) is also expressed in the robot's base frame $\boldsymbol{B}_t$. Intuitively, this robot-centric representation encodes the world from the robot's local, myopic view – knowing its joint state, how it contacts the environment locally, and how external forces (*i.e.*, gravity) are applied to it. *NeRD* then uses this information to evolve the robot's dynamics within the local frame. By using this robot-centric parameterization, we reformulate our loss function from Eq. 1 as:

$$\mathcal{L}_\theta = \frac{1}{NS} \sum_N \left\| NeRD_\theta \left(\{ \boldsymbol{s}_k^{\boldsymbol{B}_k}, \boldsymbol{\mathcal{C}}_k^{\boldsymbol{B}_k}, \boldsymbol{\tau}_k, \vec{\boldsymbol{g}}^{\boldsymbol{B}_k} \}_{k=t-h+1}^{t} \right) - \widehat{\Delta \boldsymbol{s}_{t+1}^{\boldsymbol{B}_t}} \right\|, \tag{2}$$

where $\vec{\boldsymbol{g}}$ is the unit gravity vector, and the superscript $\boldsymbol{B}_k$ (or $\boldsymbol{B}_t$) means the corresponding quantity is expressed in the robot base frame at timestep k (or t). The robot state is then updated by

$$\boldsymbol{s}_{t+1} = \mathcal{T}_{\boldsymbol{B}_t}^{w} \left(\boldsymbol{s}_t^{\boldsymbol{B}_t} \oplus NeRD_\theta \left(\{ \boldsymbol{s}_k^{\boldsymbol{B}_k}, \boldsymbol{\mathcal{C}}_k^{\boldsymbol{B}_k}, \boldsymbol{\tau}_k, \vec{\boldsymbol{g}}^{\boldsymbol{B}_k} \}_{k=t-h+1}^{t} \right) \right), \tag{3}$$

where $\mathcal{T}_{\boldsymbol{B}_t}^{w}(\cdot)$ is the transformation from robot base frame at time step t to the world frame.

The spatial invariance property of our robot-centric representation explicitly enforces the spatial generalizability of the learned robot dynamics models. In addition, it reduces the state space, substantially enhancing the training and data efficiency of *NeRD* by eliminating the need to exhaustively sample all spatial positions and orientations of the robot during model training.

4 Implementation

NeRD is compatible with most articulated rigid-body simulation frameworks, as it uses intermediate quantities commonly computed in a standard simulator. We validate our approach by integrating *NeRD* into a state-of-the-art robotics simulator, NVIDIA's Warp simulator [24], since Warp's modular design enables implementing *NeRD* as an interchangeable solver module in Python and keeps it transparent to simulation users. We use a GPU-parallelized collision detection algorithm adapted from the one in Warp.

Training Datasets We generate the training datasets for *NeRD* in a task-agnostic manner using Warp with the Featherstone solver [45]. For each robot instance in our experiments, we collect 100K random trajectories, each consisting of 100 timesteps. These trajectories are generated using randomized initial states of the robot, random joint torque sequences within the robot's motor torque limits, and optionally, randomized environment configurations.

Network and Training Details We model *NeRD* using a causal Transformer architecture, specifically a lightweight implementation of the GPT-2 Transformer [46, 47]. We use a history window size $h = 10$ for all tasks in our experiments. During training, we sample batches of sub-trajectories of length h and train the model using a teacher-forcing approach [48]. To prevent the loss from being dominated by high-variance velocity terms, we normalize the output prediction, using the mean and standard deviation statistics computed from the dataset. Ablation experiments (see Appendix C.5) show that output normalization is critical for improving the accuracy and long-horizon stability of *NeRD*. We also apply normalizations to the model's input to regularize the ranges of the inputs, improving the stability of model training. We report training hyperparameters in the Appendix B.

5 Experiments

We train *NeRD* models on six distinct robotic systems (Fig. 1) and conduct extensive experiments to validate the capabilities of the trained *NeRD* models [1]. We investigate the following questions:

[1]View the supplementary video to observe qualitative performance throughout the following examples.

Robot	**Cartpole**			**Ant**
Trajectory Horizon	100	500	1000	500
Base Position Err. (*m*)	0.0002	0.004	0.033	0.057
Base Orientation Err. (*rad*)	-	-	-	0.095
Non-Base Joint Err. (*rad*)	0.0004	0.013	0.075	0.077

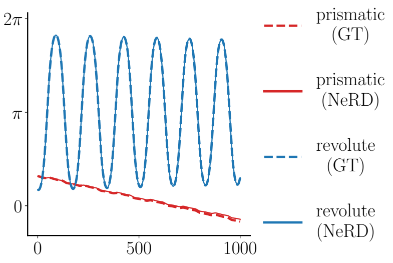


Figure 3: **Evaluation of *NeRD* on long-horizon passive motions.** Left: Full report of the measured errors. Right: 1000-step cartpole state trajectories simulated by *NeRD* and ground-truth simulator.

Can *NeRD* reliably and accurately simulate long-horizon robotic trajectories (§5.1)? Does *NeRD*'s hybrid prediction framework enable it to generalize across different contact configurations (§5.2)? Can a single *NeRD* model generalize to diverse tasks, customized robotic controllers, and spatial regions that are unseen during training? Can we train robotic policies for diverse tasks entirely in a *NeRD* simulator and successfully deploy the learned policies in the ground-truth simulator and even in the real world (§5.3, §5.4)? Finally, can we effectively fine-tune the pretrained *NeRD* models from real-world data (§5.5)? We also provide a comprehensive ablation study in Appendix C.5, highlighting critical design decisions essential for successfully training *NeRD* models.

5.1 Long-Horizon Stability and Accuracy: Cartpole and Ant with Passive Motion

We first evaluate *NeRD*'s long-horizon performance using open-loop passive motions of *Cartpole* and *Ant*. While *Cartpole* is a contact-free system that serves as a tractable problem for analysis, *Ant* assesses *NeRD*'s performance when a floating base, high DoFs (14), and contact are all involved. The robots start from randomized initial states and apply zero joint torques, $\boldsymbol{\tau}_t = 0$. We compute the temporally averaged errors between the trajectories generated by *NeRD* and the ground-truth simulators. We report the errors averaged from 2048 trajectories for each test in Fig. 3 (left).

For *Cartpole*, we evaluate trajectories of 100, 500, and 1000 steps. We measure the errors of the prismatic base joint (reported as *Base Position Err.*) and the non-base revolute joint. As a reference for the prismatic joint error, the pole length is 1 *m*. To visualize the simulation accuracy, Fig. 3 (right) compares the state trajectories generated by *NeRD* and the ground-truth simulator from the same initial state. The results demonstrate high long-horizon accuracy of *NeRD* on *Cartpole*, with accumulated error of 0.075 *rad* (smaller than $5°$) for the revolute joint and 0.033 *m* for the prismatic joint, even after 1000 steps (*i.e.*, equivalent to 16.67 seconds of passive motion). For *Ant*, we evaluate on 500-step trajectories, as the motion typically converges to a static state within this duration. We measure the base's position and orientation errors, and the mean error of non-base revolute joints. *NeRD* achieves an average base angular error of 0.095 *rad* and positional error of 0.057 *m* after 500 steps of simulation ($1.2\,m$ full body width). The minimal prediction errors indicate *NeRD*'s capability to accurately predict the motion of robots with a floating base for extended horizons.

5.2 Contact Generalizability: Double Pendulum with Varying Contact Environments

We validate *NeRD*'s generalizability across varying contact configurations using a *Double Pendulum* example, in which a randomized planar ground (random normal direction and position) is placed beneath the double pendulum. Different combinations of ground configurations and initial pendulum states yield distinct modes of motion: *contact-free chaotic motion* [49] when the ground is distant; *sliding contact motion*, occurring when the pendulum lightly touches and slides along the ground surface; and *collision-induced stopping motion* when the ground is positioned closely enough that the pendulum rapidly comes to rest after contact. Such varied contact configurations pose challenges for prior methods, as typical state representation without encoding the environment provides insufficient clues to determine contact timing and mode. To test *NeRD*, we evaluate seven different ground setups – one contact-free and six involving potential pendulum-ground contact. For each ground

Table 1: Quantitative evaluation of policies trained exclusively in *NeRD* simulators, when deployed in both *NeRD* simulators *and* the ground-truth simulator.

Robot	**Cartpole**	**Franka**	**Ant**			**ANYmal**	
Task	Swing Up	Reach	Running	Spinning	Spin Tracking	Forward Walk	Sideways Walk
GT Reward	1212.5 ± 210.4	89.3 ± 10.5	2541.5 ± 309.1	2624.7 ± 641.0	1630.2 ± 203.1	1323.4 ± 60.5	1360.2 ± 81.2
NeRD Reward	1212.6 ± 210.2	91.1 ± 9.9	2649.5 ± 227.4	3076.2 ± 433.5	1670.5 ± 192.6	1323.1 ± 62.4	1359.2 ± 70.4
Reward Err. (%)	+0.01%	+2.02%	+4.25%	+17.21%	+2.47%	-0.02%	-0.07%

configuration, 2048 passive-motion trajectories of 100 steps are simulated with random initial states of the pendulum and zero joint torques. Due to space constraints, visualizations of the seven ground configurations and detailed error metrics for *NeRD* are provided in the Appendix C.2. Among all seven ground configurations, the maximum mean joint error after 100-step simulation is 0.056 *rad* ($3.2°$), with joint errors typically below $1°$ for most cases. The results demonstrate that a single *NeRD* model effectively generalizes across diverse contact scenarios.

5.3 Task, Controller, and Spatial Generalizability: Robotic Policy Learning via RL

To evaluate the task, controller, and spatial generalizability of *NeRD*, we conduct extensive RL policy-learning experiments across diverse tasks for four robotic systems: a swing-up task for *Cartpole*, an end-effector reach task for *Franka*, three different tasks (running, spinning, and spin tracking) for *Ant*, and forward and sideways velocity-tracking for *ANYmal* [50]. While detailed task descriptions are provided in the Appendix C.1, we highlight key aspects here. First, each task explores specialized robot state distributions that is never covered in the *NeRD* training dataset for that robot, which *only* includes randomly-generated motions. Conducting policy learning in those tasks requires the trained *NeRD* models to make accurate predictions under unseen state distributions. Second, to verify trained *NeRD* models' generalizability to low-level controllers, we use *joint-torque control* for *Cartpole* and *Ant*, and *joint-position control* for *Franka* and *ANYmal*. Third, in the *Ant* running task and *ANYmal* velocity-tracking tasks, the robots reach a spatial region that is exceptionally far from the range covered by the training datasets, thus examining *NeRD*'s spatial generalizability. Fourth, the horizons of the tasks vary from several hundred steps to 1000 (*ANYmal* tasks), assessing the stability and accuracy of the *NeRD* models over extremely long horizons.

For each task, we use PPO [51] to train three policies with different random seeds *entirely* within the *NeRD* simulators. We then evaluate each learned policy over 2048 trajectories in *both* the *NeRD* and the ground-truth simulators, and report the average rewards and the standard deviations in Table 1. Despite training purely from random trajectories, the results show that the trained *NeRD* models can support high-performing policy learning for diverse tasks (see supplementary video for policy behaviors). Furthermore, the *NeRD*-trained policies have remarkably similar rewards when deployed in the *NeRD* simulator and in the ground-truth simulator (without any fine-tuning or adaptation phase), further confirming the long-horizon predictive accuracy of *NeRD* models.

5.4 Sim-to-Real Transfer of Franka Reach Policy

We further evaluate the accuracy of *NeRD* models via zero-shot sim-to-real transfer of a *Franka* reach policy trained exclusively in the *NeRD* simulator in §5.3 (Fig. 4). The goal of this task is to move the robot's end-effector to a randomly-specified target position. The robot is controlled via a *joint-position controller*. See the Appendix for detailed task and reward settings. We command the robot to move to 50 different target positions sampled within the robot's workspace, and we evaluate the policy's performance by measuring the distance to the targets at steady-state. As a baseline, we repeat the same experiment using a policy trained in the ground-truth simulator. Deployment results show that policies trained with both *NeRD* and the ground-truth (**GT**) simulator achieve low steady-state error, with mean and standard deviation: **NeRD:** 1.927 ± 0.699 *mm*, **GT:** 4.647 ± 2.667 *mm*. We show the distance-to-goal plots of ten executions of *NeRD*-trained policies in Fig. 4. These results validate that the *NeRD* model can effectively learn policies that transfer to the real world.

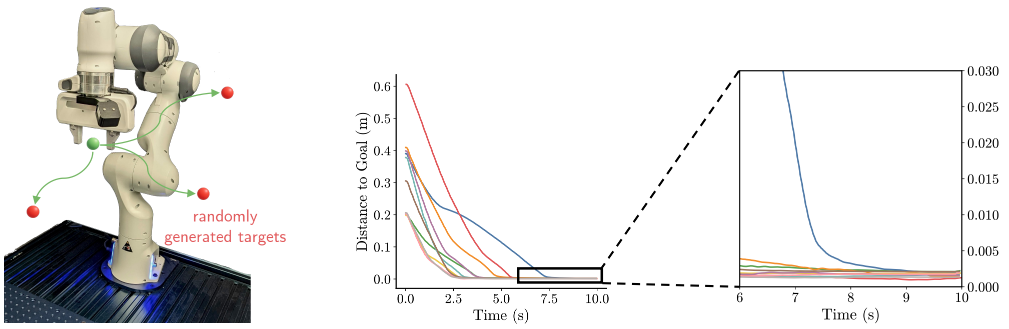
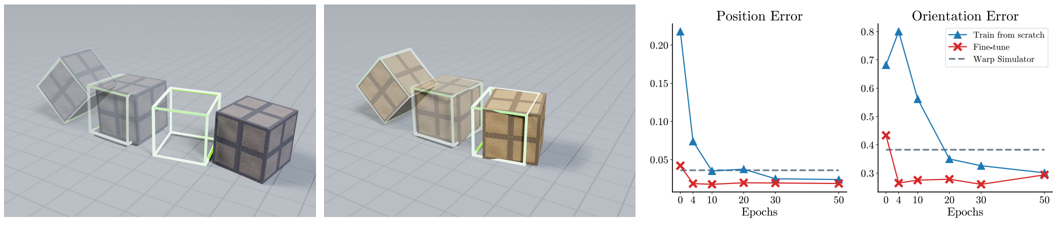


Figure 4: **Zero-shot sim-to-real transfer of a Franka reach policy.** The real-world setup is shown in the **left** figure. The plot on the **middle** visualizes the evolution of distance-to-goal measurements when 10 *NeRD*-trained policies are executed, with a zoomed-in plot in the **right**.



(a) Warp Simulator (b) Fine-tuned NeRD Simulator (c) Fine-tune v.s. Train from scratch

Figure 5: **Fine-tuning of a pretrained *NeRD* model on real-world cube-tossing data. (a-b)** Cube-tossing trajectories simulated by the Warp simulator and by the fine-tuned *NeRD* simulator. The light-green frames are ground-truth poses. **(c)** Comparison of fine-tuning a pretrained *NeRD* model (red) against training a *NeRD* model from scratch (blue) on the real dataset.

5.5 Fine-tunability on Real-World Data: Cube Tossing

We evaluate *NeRD*'s fine-tunability using a real-world cube-tossing dataset [15], where a cube is tossed with a random initial state and collides with the ground. We first replicate this cube-tossing environment in the Warp simulator and generate a synthetic dataset of cube-tossing trajectories for pretraining a *NeRD* model. After pretraining, we fine-tune the *NeRD* model on the real-world dataset. As a comparison, we also train a *NeRD* model from scratch using only the real-world cube-tossing dataset (*i.e.*, no simulation data). We provide more details in the Appendix C.4.

We evaluate trained models on 85 held-out real-world trajectories, and measure the average cube COM position error and orientation error along the trajectory. Since the **Warp simulator** does not fully capture real-world dynamics, it has a position and orientation error of 0.036 *m* and 0.383 *rad*, respectively. Both the fine-tuned *NeRD* model and the *NeRD* model trained from scratch outperform Warp. Specifically, the **fine-tuned** *NeRD* model has errors of 0.018 *m* and 0.266 *rad*, and the **model trained from scratch** has errors of 0.023 *m* and 0.276 *rad*. Fig. 5(a) and (b) qualitatively compare the trajectories generated by Warp and the fine-tuned *NeRD* model. In addition, Fig. 5(c) shows that fine-tuning the pretrained *NeRD* converges in fewer than five training epochs, which is **$10\times$ faster** than training from scratch; thus, pretraining the *NeRD* model on a large-scale simulation dataset enables efficient adaptation to real-world dynamics with a small amount of real-world data.

We further evaluate two baselines designed specifically for this dataset: (1) *GNN-Rigid* [33] and (2) *ContactNets* [15]. For *GNN-Rigid*, we used the released model and inference code (training code unavailable). The measured position error is 0.032 *m*; rotation error is not measured due to missing code. For *ContactNets*, we used the released code to train the model. The evaluated position error is 0.017 *m* and rotation error is 0.242 *rad*. These comparisons show that *NeRD* achieves comparable real-world fine-tuning results to specialized models while offering key advantages: (1) *NeRD* is widely applicable to diverse systems, including articulated and single rigid bodies; (2) Fine-tuning *NeRD* took <10 min for the *Cube Tossing* dataset, compared to 12 h for *ContactNets*.

6 Limitations and Future Work

In this work, we present Neural Robot Dynamics (*NeRD*), learned robot-specific dynamics models capable of stable and accurate simulation over thousands of time steps. Our neural dynamics models can be fine-tuned from real-world data and generalize across different tasks, environments, and low-level controller configurations.

Although our experiments clearly demonstrate the effectiveness of *NeRD*, several promising directions remain for future research. First, while we evaluate *NeRD* on a 14-DoF *Ant* robot and an 18-DoF *ANYmal* robot, we have yet to test it on some of the most complex robotic systems, such as humanoid robots. These advanced robot designs typically have 20-50 degrees of freedom and complex mechanical structures that are difficult to model analytically and simulate efficiently. Applying *NeRD* on these robots can further highlight the efficiency and accuracy benefits of a neural simulation approach.

Another interesting direction to explore is the trajectory-sampling strategy for generating the synthetic training dataset. Currently, we adopt a random sampling strategy to generate task-agnostic datasets, ensuring the trained *NeRD* model is not limited to specific state distributions. However, random sampling may become ineffective when the state dimensionality grows, particularly for complex robots such as humanoids. Exploring more effective dataset construction strategies that still preserve task-agnostic characteristics of the datasets and the generalizability of the resulting *NeRD* models is a compelling direction for future research.

Furthermore, our current fine-tuning process assumes we have access to the same state space in the real world as we do in simulation (*i.e.*, full robot state, and environment setups for collision detection). However, real-world robot data is often only partially observable due to sensor limitations. Investigating methods to fine-tune a pretrained *NeRD* model from partially observable real-world data is another exciting direction for future research.

References

[1] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 2020.

[2] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 2023.

[3] T. Haarnoja, B. Moran, G. Lever, S. H. Huang, D. Tirumala, J. Humplik, M. Wulfmeier, S. Tunyasuvunakool, N. Y. Siegel, R. Hafner, M. Bloesch, K. Hartikainen, A. Byravan, L. Hasenclever, Y. Tassa, F. Sadeghi, N. Batchelor, F. Casarini, S. Saliceti, C. Game, N. Sreendra, K. Patel, M. Gwira, A. Huber, N. Hurley, F. Nori, R. Hadsell, and N. Heess. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 2024.

[4] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[5] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. 2021.

[6] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 2020.

[7] B. Tang, I. Akinola, J. Xu, B. Wen, A. Handa, K. Van Wyk, D. Fox, G. S. Sukhatme, F. Ramos, and Y. Narang. Automate: Specialist and generalist assembly policies over diverse geometries. In *Robotics: Science and Systems*, 2024.

[8] N. Funk, C. Schaff, R. Madan, T. Yoneda, J. U. De Jesus, J. Watson, E. K. Gordon, F. Widmaier, S. Bauer, S. S. Srinivasa, et al. Benchmarking structured policies and policy optimization for real-world dexterous object manipulation. *IEEE Robotics and Automation Letters*, 2021.

[9] J. Gu, F. Xiang, X. Li, Z. Ling, X. Liu, T. Mu, Y. Tang, S. Tao, X. Wei, Y. Yao, X. Yuan, P. Xie, Z. Huang, R. Chen, and H. Su. Maniskill2: A unified benchmark for generalizable manipulation skills. In *The Eleventh International Conference on Learning Representations*, 2023.

[10] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kirmani, S. Levine, J. Wu, C. Finn, H. Su, Q. Vuong, and T. Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

[11] Z. Liu, W. Liu, Y. Qin, F. Xiang, M. Gou, S. Xin, M. A. Roa, B. Calli, H. Su, Y. Sun, et al. Ocrtoc: A cloud-based competition and benchmark for robotic grasping and manipulation. *IEEE Robotics and Automation Letters*, 2021.

[12] T. Du, A. Schulz, B. Zhu, B. Bickel, and W. Matusik. Computational multicopter design. *ACM Trans. Graph.*, 2016.

[13] M. Li, R. Antonova, D. Sadigh, and J. Bohg. Learning tool morphology for contact-rich manipulation tasks with differentiable simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[14] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Proceedings of Robotics: Science and Systems*, 2021.

[15] S. Pfrommer, M. Halm, and M. Posa. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. In *Proceedings of the 2020 Conference on Robot Learning*, 2021.

[16] B. Hoffman, J. Cheng, C. Li, and S. Coros. Learning more with less: Sample efficient dynamics learning and model-based rl for loco-manipulation. *arXiv preprint arXiv:2501.10499*, 2025.

[17] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[18] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 2020.

[19] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

[20] C. Li, A. Krause, and M. Hutter. Robotic world model: A neural network simulator for robust policy optimization in robotics, 2025.

[21] M. Andriluka, B. Tabanpour, C. D. Freeman, and C. Sminchisescu. Learned neural physics simulation for articulated 3d human pose reconstruction. In *Computer Vision – ECCV 2024: 18th European Conference, Proceedings, Part LXXXIV*, 2024.

[22] L. Fussell, K. Bergamin, and D. Holden. Supertrack: motion tracking for physically simulated characters using supervised learning. *ACM Trans. Graph.*, 40(6), 2021.

[23] N. Hansen, X. Wang, and H. Su. Temporal difference learning for model predictive control. In *International Conference on Machine Learning, PMLR*, 2022.

[24] M. Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, 2022.

[25] H. Bertiche, M. Madadi, and S. Escalera. Neural cloth simulation. *ACM Trans. Graph.*, 2022.

[26] Y. Jin, D. Omens, Z. Geng, J. Teran, A. Kumar, K. Tashiro, and R. Fedkiw. A neural-network-based approach for loose-fitting clothing. *arXiv preprint arXiv:2404.16896*, 2024.

[27] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[28] L. Ladickỳ, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 2015.

[29] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, 2020.

[30] J. Li, Y. Gao, J. Dai, S. Li, A. Hao, and H. Qin. Mpmnet: A data-driven mpm framework for dynamic fluid-solid interaction. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[31] X. Li, Y.-L. Qiao, P. Y. Chen, K. M. Jatavallabhula, M. Lin, C. Jiang, and C. Gan. Pac-nerf: Physics augmented continuum neural radiance fields for geometry-agnostic system identification. *International Conference on Learning Representations (ICLR)*, 2023.

[32] Y. Jiang, J. Sun, and C. K. Liu. Data-augmented contact model for rigid body simulation. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, 2022.

[33] K. R. Allen, T. L. Guevara, Y. Rubanova, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. In *Proceedings of The 6th Conference on Robot Learning*, 2023.

[34] K. R. Allen, Y. Rubanova, T. Lopez-Guevara, W. F. Whitney, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff. Learning rigid dynamics with face interaction graph networks. In *The Eleventh International Conference on Learning Representations*, 2023.

[35] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

[36] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[37] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024.

[38] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 2023.

[39] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International conference on machine learning*, 2018.

[40] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 2019.

[41] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. 2022.

[42] 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)*, 2021.

[43] M. Kim, J. Yoon, D. Son, and D. Lee. Data-driven contact clustering for robot simulation. In *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[44] D. Son and B. Kim. Local object crop collision network for efficient simulation of non-convex objects in gpu-based simulators. *arXiv preprint arXiv:2304.09439*, 2023.

[45] R. Featherstone. *Rigid Body Dynamics Algorithms*. Springer-Verlag, 2007.

[46] A. Karpathy. nanoGPT, 2023. URL `https://github.com/karpathy/nanoGPT`.

[47] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[48] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, 2014.

[49] T. Shinbrot, C. Grebogi, J. Wisdom, and J. A. Yorke. Chaos in a double pendulum. *American Journal of Physics*, 1992.

[50] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger. Anymal - a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[51] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[52] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

Neural Robot Dynamics: Appendix

Appendix Contents

A Additional *NeRD* Details

We provide additional details about *NeRD* that are not covered in the main paper due to space constraints.

Contact-Related Quantities We use contact-related quantities $\boldsymbol{\mathcal{C}}_t$ as an application-agnostic representation to capture how the surroundings impact the robot's dynamics, without the need to parameterize the whole environment. This is inspired by the dynamics and contact solvers of analytical simulators, as these solvers also use these contact-related quantities to formulate physics equations to evolve the robot dynamics. We construct $\boldsymbol{\mathcal{C}}_t = \{\boldsymbol{c}_t^i\}$ by reusing the collision detection module in the classical simulator. Specifically, in our implementation, we adopt a GPU-parallelized collision detection algorithm adapted from the one in Warp. For each pre-specified contact point $\boldsymbol{p}_0^i$ on the robot, we use the collision detection algorithm to compute its contact event quantities $\boldsymbol{c}_t^i = (\boldsymbol{p}_0^i, \boldsymbol{p}_1^i, \vec{\boldsymbol{n}}^i, d^i)$. Here $\boldsymbol{p}_1^i$ is the contact point on a non-robot shape, $\vec{\boldsymbol{n}}^i$ is the contact normal, and d^i is the contact distance (zero or negative for collisions). We mask a $\boldsymbol{c}_t^i$ to be zero if the associated contact distance is larger than a positive threshold $d^i > \xi$, allowing free-space motion while providing robustness to cases where a collision occurs within the timestep. Quantity ξ is a positive value greater than or equal to the *contact thickness* of a geometry (*i.e.*, a standard collision-detection parameter in classical simulators). Ideally, ξ should also exceed the allowed maximum displacement of the contact point within a timestep, ensuring that all near-contact events are retained. However, in practice, the choice of ξ is very flexible, and we use a fixed setting of $\xi = \max(4 \cdot \textit{contact\_thickness}, 0.1)$ across all our experiments without task-specific tuning.

Robot-Centric State Representation Here we provide the detailed calculation for transforming the robot state into robot's base frame. For the robot state $\boldsymbol{s}_k$ at time step $k \in [t-h+1, t]$, we transform it into robot's base frame, $\boldsymbol{B}_k$, at time step k, where $\boldsymbol{B}_k = (\boldsymbol{x}_k, \boldsymbol{R}_k)$. For the robot articulation, we use the reduced coordinate state, which is spatially invariant; thus, we only need to transform the state of the robot base (*i.e.*, $\boldsymbol{x}_k$, $\boldsymbol{R}_k$, and $\boldsymbol{\phi}_k$) into the robot's base frame. Therefore, we have $\boldsymbol{s}_k^{\boldsymbol{B}_k} = (\boldsymbol{x}_k^{\boldsymbol{B}_k}, \boldsymbol{R}_k^{\boldsymbol{B}_k}, \boldsymbol{q}_k, \boldsymbol{\phi}_k^{\boldsymbol{B}_k}, \dot{\boldsymbol{q}}_k)$, with

$$\boldsymbol{x}_k^{\boldsymbol{B}_k} = \boldsymbol{0}, \tag{4}$$

$$\boldsymbol{R}_k^{\boldsymbol{B}_k} = \textit{Identity}, \tag{5}$$

$$\boldsymbol{\nu}_k^{\boldsymbol{B}_k} = \boldsymbol{R}_k^{-1}(\boldsymbol{\nu}_k - \boldsymbol{x}_k \times \boldsymbol{\omega}_k), \tag{6}$$

$$\boldsymbol{\omega}_k^{\boldsymbol{B}_k} = \boldsymbol{R}_k^{-1}\boldsymbol{\omega}_k, \tag{7}$$

where $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$ are the linear and angular components of a spatial twist $\boldsymbol{\phi}$, respectively.

Additionally, the predicted state difference $\Delta\boldsymbol{s}_{t+1}$ (*i.e.*, the network's output) is expressed in the robot's base frame at time step t instead of $t+1$, *i.e.*, $\Delta\boldsymbol{s}_{t+1}^{\boldsymbol{B}_t} \triangleq \boldsymbol{s}_{t+1}^{\boldsymbol{B}_t} \ominus \boldsymbol{s}_t^{\boldsymbol{B}_t}$. This is because, when expressed in its own base frame, the state of the robot base $(\boldsymbol{x}, \boldsymbol{R})$ is always the identity transformation (*i.e.*, $\boldsymbol{x}_t^{\boldsymbol{B}_t} = \boldsymbol{x}_{t+1}^{\boldsymbol{B}_{t+1}} = \boldsymbol{0}$ and $\boldsymbol{R}_t^{\boldsymbol{B}_t} = \boldsymbol{R}_{t+1}^{\boldsymbol{B}_{t+1}} = \mathbf{I}$), which results in zero state changes for these dimensions, so $\Delta\boldsymbol{x}_{t+1}^{\boldsymbol{B}_{t+1}} = \boldsymbol{0}$ and $\Delta\boldsymbol{R}_{t+1}^{\boldsymbol{B}_{t+1}} = \boldsymbol{0}$. Therefore the learned model cannot predict the motion of the robot base if using $\Delta\boldsymbol{s}_{t+1}^{\boldsymbol{B}_{t+1}}$ as the prediction target.

To compute $\boldsymbol{s}_{t+1}^{\boldsymbol{B}_t} = (\boldsymbol{x}_{t+1}^{\boldsymbol{B}_t}, \boldsymbol{R}_{t+1}^{\boldsymbol{B}_t}, \boldsymbol{q}_{t+1}, \boldsymbol{\phi}_{t+1}^{\boldsymbol{B}_t}, \dot{\boldsymbol{q}}_{t+1})$, we use the following calculations:

$$\boldsymbol{x}_{t+1}^{\boldsymbol{B}_t} = \boldsymbol{R}_t^{-1}(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t), \tag{8}$$

$$\boldsymbol{R}_{t+1}^{\boldsymbol{B}_t} = \boldsymbol{R}_t^{-1}\boldsymbol{R}_{t+1}, \tag{9}$$

$$\boldsymbol{\nu}_{t+1}^{\boldsymbol{B}_t} = \boldsymbol{R}_t^{-1}(\boldsymbol{\nu}_{t+1} - \boldsymbol{x}_t \times \boldsymbol{\omega}_{t+1}), \tag{10}$$

$$\boldsymbol{\omega}_{t+1}^{\boldsymbol{B}_t} = \boldsymbol{R}_t^{-1}\boldsymbol{\omega}_{t+1}, \tag{11}$$

Multi-Substep Prediction To improve the stability of the simulation, a classical simulator often utilizes a smaller timestep (*i.e.*, substep) and runs multiple substeps of solver-integration iterations to obtain the actual state of the robot at the next time step, $\boldsymbol{s}_{t+1}$ (as shown in Fig. 2(a)). Unlike previous neural simulation works [39, 21, 33] that sequentially predict the robot state acceleration at each substep and obtain the robot state at next time step by time integration over substeps, *NeRD* directly predicts the state difference from the current robot state to the state at the next (macro) time step, $\boldsymbol{s}_{t+1}$, which might span multiple substeps in the analytical simulator. This design enables us to learn *NeRD* from a finer-grained simulator with smaller substep sizes without sacrificing the efficiency of the learned model at test time.

B Additional Training Details and Hyperparameters

We adopt a lightweight implementation of causal Transformer [46, 47], and repurpose it as a sequential model for robot dynamics. Past robot-centric simulation states are encoded into embeddings via a learnable linear layer, processed through Transformer blocks with self-attention, and then mapped to latent features from which the robot state difference is predicted as the output.

We use a fixed set of hyperparameters for training *NeRD* across all six robotic systems – including training hyperparameters and Transformer hyperparameters – except for the embedding size of the Transformer model. For robots with fewer degrees of freedom, we use a smaller embedding size to enhance training and inference efficiency. The complete hyperparameter settings used in our experiments are provided in Table 2.

Table 2: Training hyperparameters in the experiments.

Robot		**Cartpole**	**Pendulum**	**Cube Tossing**	**Franka**	**Ant**	**ANYmal**
Training	history window size h	10					
	batch size	512					
	learning rate	linear decay from $1e^{-3}$ to $1e^{-4}$					
Transformer	block size	32					
	num layers	6					
	num heads	12					
	input embedding size	192			384		
	dropout	0					
Output MLP	num layers	1					
	layer size	64					

C Additional Experiment Details

C.1 Details of Policy Learning Tasks

We train a *NeRD* model for each robotic system and use the trained *NeRD* model in all the downstream tasks for the corresponding robotic system. Here we provide details of the policy learning tasks in §5.3.

C.1.1 Cartpole

Swing-Up Task In this task, a *Cartpole* (2-DoF) is controlled to swing up its pole from a randomized initial angle to be upright and maintain the upright pose as long as possible. The *Cartpole* is directly controlled by the commanded 1D joint-space torque of the base prismatic joint. The observation of the policy is 4-dimensional, including:

- 2-dim joint positions: x, θ
- 2-dim joint velocities: $\dot{x}, \dot{\theta}$

A trajectory is terminated if it exceeds the maximum number of steps 300, or the cart position of the *Cartpole* moves outside the $[-4\,m, 4\,m]$ range, or the joint velocity is above 10 *rad/s*.

The stepwise reward function is below:

$$\mathcal{R}_t = 5 - \theta_t^2 - 0.05x_t^2 - 0.1\dot{\theta}_t^2 - 0.1\dot{x}_t^2.$$

C.1.2 Ant

Ant Running In this task, an *Ant* robot (14-DoF) is controlled to move forward as fast as possible. The action space is 8D joint-space torque of Ant's non-base revolute joints, and the *Ant* is directly controlled by the commanded joint-space torque. The observation space has 29 dimensions, including:

- 1-dim height of the base h
- 4-dim orientation of the base represented by a quaternion
- 3-dim linear velocity of the base $\boldsymbol{v}$
- 3-dim angular velocity of the base $\boldsymbol{\omega}$
- 8-dim joint positions
- 8-dim joint velocities
- 2-dim up and heading vector projections: $p_{\text{up}}, p_{\text{heading}}$.

The episode is terminated if it exceeds the maximum number of steps 500, or the height of the base h falls below 0.3 *m* (*i.e.*, $h < 0.3$ *m*).

The stepwise reward function for the running task is defined below:

$$\mathcal{R}_t = \boldsymbol{v}_x + 0.1p_{\text{up}} + p_{\text{heading}}.$$

Ant Spinning An *Ant* is controlled to maximize its spinning speed around the gravity axis (Y-axis in this environment) in this task. It uses the same observation space and the termination condition as the running task. The stepwise reward function is defined as below:

$$\mathcal{R}_t = \boldsymbol{\omega}_y + p_{\text{up}}.$$

Ant Spin Tracking This task requires an *Ant* to track a spinning speed of 5 *rad/s*. It uses the same observation space and termination condition as the running task. The stepwise reward function is defined below:

$$\mathcal{R}_t = 5 \cdot \exp(-(\boldsymbol{\omega}_y - 5)^2 - 0.1\boldsymbol{\omega}_x^2 - 0.1\boldsymbol{\omega}_z^2) + 0.1p_{\text{up}}.$$

C.1.3 Franka

End-Effector Reach Task The goal of this task is to move the Franka robot's end-effector to a randomly-specified target position. The action space is defined as delta joint positions, which are executed via a *joint-position PD controller* (Note: the *NeRD* model is still predicting using the joint-space torques, which are converted from the target joint positions via the joint-position PD controller). We also conducted another experiment with *joint-torque control* in §C.3. The 13-dim observation space consists of the following:

- 7-dim joint positions
- 3-dim end-effector position
- 3-dim target goal position

The episode length of this task is 128. We adopt an exponential reward function from our existing setups, with minimal tuning:

$$\mathcal{R}_t = -d + \frac{1}{\exp(50d) + \exp(-50d) + \epsilon} + \frac{1}{\exp(300d) + \exp(-300d) + \epsilon},$$

where $d = \|\vec{e}\|$ is the end-effector's distance to the goal position and $\epsilon = 0.0001$.

C.1.4 ANYmal

Forward Walk Velocity-Tracking In this task, an *ANYmal* robot [50] (18-DoF) is controlled to track a forward walking speed of 1 *m/s*. The action space is the target joint positions, and the *ANYmal* is controlled by a *joint-position PD controller*. The observation space is similar to *Ant* tasks and has 37 dimensions, including the following:

- 1-dim height of the base h
- 4-dim orientation of the base represented by a quaternion
- 3-dim linear velocity of the base $\boldsymbol{v}$
- 3-dim angular velocity of the base $\boldsymbol{\omega}$
- 12-dim joint positions
- 12-dim joint velocities
- 2-dim up and heading vector projections: $p_{\text{up}}, p_{\text{heading}}$.

The episode is terminated if it exceeds the maximum number of steps 1000, or the height of the base $h < 0.4$ *m*, or the base or the knees hit the ground. We adopt a commonly used reward function [52] with minimal tuning:

$$\mathcal{R}_t = \exp\Big(-\big((\boldsymbol{v}_x - 1)^2 + \boldsymbol{v}_z^2\big)\Big) + 0.5\exp(-\boldsymbol{\omega}_y^2) - (0.002\sum\boldsymbol{\tau})^2,$$

where $\boldsymbol{\tau}$ is the joint-space torque. Note that, in our *ANYmal* environments, $\vec{x}$ is the forward direction, $\vec{z}$ is the sideways direction, and $\vec{y}$ is the upward direction.

Sideways Walk Velocity-Tracking This task requires an *ANYmal* robot to track a sideways walking speed of 1 *m/s*. It uses the same action space, observation space, and termination condition as the *Forward Walk Velocity-Tracking task*. The reward function is below:

$$\mathcal{R}_t = \exp\Big(-\big(\boldsymbol{v}_x^2 + (\boldsymbol{v}_z - 1)^2\big)\Big) + 0.5\exp(-\boldsymbol{\omega}_y^2) - (0.002\sum\boldsymbol{\tau})^2.$$

C.2 Visualization and Full Results for Double Pendulum with Varying Contact Environments

We provide visualizations of the seven ground configurations and the detailed measured errors in this section. The seven contact setups include one contact-free scenario where we put the ground far below the pendulum, and six different planar ground settings where the double pendulum is able to make contact with the ground (as shown in Fig. 6). We use a single trained *NeRD* model for all contact setups and generate 2048 passive-motion trajectories of the *Double Pendulum* over a duration of 100 steps with random initial states and zero joint torques. We report the mean joint-angle errors (in *radians*) over the trajectory in Table 3.

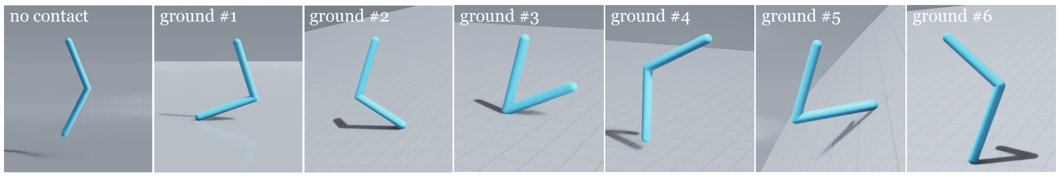


Figure 6: **Seven *Double Pendulum* contact configurations used for testing *NeRD*'s generalizability across different contact environments.**

C.3 Franka Reach Policy Learning with Joint-Torque Control

In §5.3, we conduct RL policy-learning experiments to show the *NeRD* model's generalizability to low-level controllers, where we apply different low-level controllers on different robots: *joint-torque*

Table 3: **Full passive-motion evaluation results on *Double Pendulum*.** For each contact configuration, we report the mean joint-angle error of each joint in radians.

Robot	**Double Pendulum**						
Contact Configuration	no contact	ground #1	ground #2	ground #3	ground #4	ground #5	ground #6
Joint #1 Error (*rad*)	0.004	0.012	0.008	0.005	0.011	0.029	0.008
Joint #2 Error (*rad*)	0.007	0.015	0.011	0.011	0.018	0.056	0.013

Table 4: Quantitative evaluation of *Franka Reach* policies trained exclusively in *NeRD* simulators, when deployed in the *NeRD* simulator *or* the ground-truth simulator.

Robot	**Franka**	
Task	Reach (joint-position control)	Reach (joint-torque)
GT Reward	89.3 ± 10.5	94.9 ± 7.8
NeRD Reward	91.1 ± 9.9	95.0 ± 7.8
Reward Err. (%)	+2.02%	+0.11%

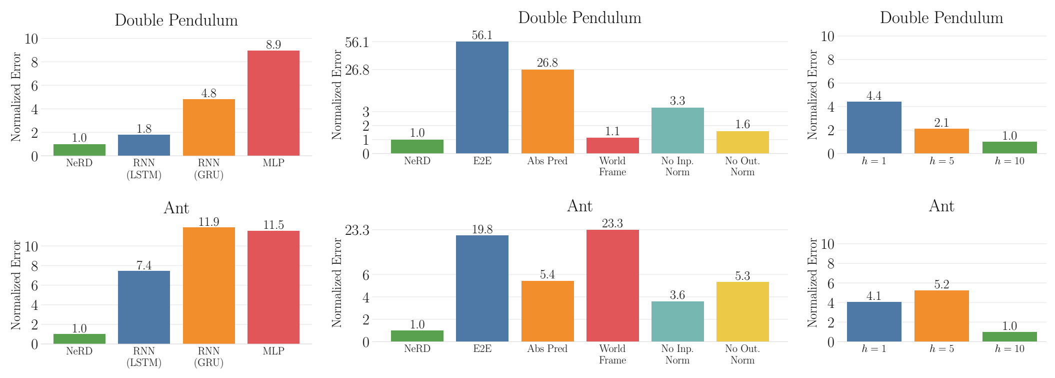
control for *Cartpole* and *Ant*, and *joint-position control* for *Franka* and *ANYmal*. To further verify such generalizability, we conduct another experiment here for the *Franka Reach* task, but with *joint-torque control* instead. We use the same task settings (*e.g.*, reward, observation) as *Franka Reach* with *joint-position* control, with the only change being the action space of the policy, and use the same *NeRD* model trained for *Franka*. Similarly, we train three policies with different random seeds *entirely* within the *NeRD* simulator for *Franka*, and then evaluate each learned policy over 2048 trajectories in the *NeRD* and in the ground-truth simulator and compare the obtained rewards and the standard deviations. Similar to the previous experiments, the results show that the trained *NeRD* model can support high-performing policy learning for different low-level controllers, and the *NeRD* simulator and the ground-truth simulator obtain remarkably similar rewards when evaluating the trained policies (*i.e.*, 0.11% error), as reported in Table 4.

C.4 Details of Cube Tossing Experiment Setups

We evaluate the fine-tunability of the *NeRD* model using a real-world dataset of cube tossing [15], where a cube is tossed with a random initial state and collides with the ground. We first replicate this cube-tossing environment in the Warp simulator by manually tuning the contact and inertia parameters to best replicate the observed dynamics in the dataset. We then generate a dataset comprising 10K randomly simulated cube-tossing trajectories of length 100, and pretrain a *NeRD* model from this synthetic dataset. After pretraining, we fine-tune the *NeRD* model on the real-world cube-tossing dataset. The real-world cube-tossing dataset contains 570 trajectories of varying lengths, corresponding to a total of 60K dynamics transitions. We split the dataset into 400 trajectories for training, 85 trajectories for validation, and 85 held-out trajectories for testing. To evaluate the fine-tuned model's prediction accuracy, we extract all sub-trajectories of length 80 (the minimum length of the trajectories in the dataset) from the testing dataset and use the simulator integrated with the fine-tuned *NeRD* model to generate predicted trajectories from the same initial states. We measure the average cube COM position error and average orientation error (in radians) along the trajectory. As a comparison, we also train a *NeRD* model from scratch using only the real-world cube-tossing dataset (*i.e.*, no simulation data).

C.5 Ablation Study

The success of *NeRD* relies on several critical design decisions made during development. In this section, we analyze these design decisions through a series of ablation experiments. Specifically,



(a) Ablations on Neural Network Models

(b) Ablations on Other Design Decisions

(c) Ablations on History Window Size *h*

Figure 7: **Ablation Study.** We evaluate ablation variants on two test cases: contact-free passive motion of the *Double Pendulum* and policy evaluation on the *Ant* running task. We normalize the errors by the error of *NeRD* ($h = 10$). **(a)** Ablations of different neural network architectures; **(b)** Ablations of other critical design decisions in *NeRD*. **(c)** Ablations on the history window size h.

we conduct our study using two evaluation test cases: (1) contact-free passive motion of *Double Pendulum*; and (2) policy evaluation on the *Ant* running task. All ablation models are trained on the same datasets as the corresponding *NeRD* models.

For test case #1, we compute the temporally-averaged mean joint-angle error (average error of two joints) of *Double Pendulum* for each ablation model, from 2048 passive motion trajectories of the *Double Pendulum* over a duration of 100 steps with random initial states and zero joint torques. Then we normalize the errors by the error of the *NeRD* model, and report the values in Fig. 7 (first row).

For test case #2, we execute the three *Ant* running policies trained in our policy-learning experiments (§5.3) and evaluate the average reward obtained using the simulator integrated with each ablation neural dynamics model (2048 trajectories for each policy in each ablation neural dynamics model). For each ablation model, we then compute the reward differences compared to the reward obtained in the ground-truth simulator. The reward differences are then normalized by the reward difference of the *NeRD* model and reported in Fig. 7 (second row).

C.5.1 Network Architecture

During development, we found the Transformer architecture to be the most effective for modeling neural robot dynamics. We demonstrate this by comparing it against three other architectures in Fig. 7(a): **MLP**: a baseline model that predicts state changes from the robot-centric simulation state of the current step; **GRU** and **LSTM**: two RNN architectures that leverage historical state information in their predictions. Although the ground-truth simulator computes the dynamics in a stateless way (*i.e.*, the next state only depends on the current state and torques), we found that sequence modeling is important to achieve high accuracy of the neural robot dynamics model. We hypothesize that the high variance of the velocity inputs is a challenge for the model; by including the historical states as input, the neural model is able to infer a smoothed version of velocity and combine it with the actual velocity input for a better prediction. Furthermore, based on the comparisons for policy evaluation on the *Ant* running task, the causal Transformer model helps achieve a reward that is much closer to the ground-truth simulator, compared to RNN architectures.

C.5.2 Hybrid Prediction Framework

To demonstrate the effectiveness of our *Hybrid Prediction Framework*, we compare *NeRD* against an *End-to-End* prediction baseline (**E2E**), which directly maps robot state and action to the next robot state. This *end-to-end* framework is commonly adopted by prior neural simulators for rigid bodies [20, 22, 42]. We reimplemented it in the Warp simulator. Specifically, in our implementation, the **E2E** baseline maps the robot state and the joint torques to the relative next robot state, and the robot state is expressed in the world frame (as an *End-to-End* approach is not aware of contact information and has to rely on world-frame state to track the possible collisions in a fixed environment). We replace the action input commonly used in *End-to-End* approaches with joint-torque input so that we can use the same training dataset as *NeRD* for a fair comparison. Fig. 7(b) shows that the **E2E** baseline has large prediction errors in both test cases. This is because, in the *Double Pendulum* case, the training dataset consists of varying scenarios of contact configurations. However, the **E2E** state representation without encoding the environment provides insufficient clues to differentiate distinct contact configurations during training, thus resulting in poor performance. In the *Ant* test case, the **E2E** baseline fails because the world-frame robot state cannot make a reliable prediction when the *Ant* moves far away from the origin and reaches regions outside the range of the training dataset.

C.5.3 Relative Robot State Prediction

The third key design decision is the use of relative robot state prediction, where the model predicts the state difference between the current robot state and the robot state in the next time step, rather than directly predicting the absolute next robot state. Predicting the relative state changes effectively reduces the range of values of the model output, thus stabilizing model training. We compare *NeRD* against its **Abs Pred** variant which predicts the absolute next state of the robot, in Fig. 7(b). The results in the figure show that even for the low-dimensional system like *Double Pendulum*, predicting the absolute state significantly increases training difficulty and results in a prediction error 26$\times$ larger than the error of predicting with relative state changes.

C.5.4 Robot-Centric State Representation

Next, we design experiments to demonstrate the importance of the robot-centric and spatially-invariant simulation state representation. For this ablation study, we train a model with the robot state and contact quantities represented in world space (**World Frame** variant in Fig. 7(b)), *i.e.*, the loss formulation in Eq. 1. As shown in the results, using the world-frame representation does not degrade the model's performance in the *Double Pendulum* case of contact-free motion. This is because the pendulum has a revolute base joint that remains fixed in position, limiting the visited states to the domain covered by the training dataset. In contrast, the world-frame representation fails entirely in the *Ant* running task, as the *Ant* moves far away from the origin during running and quickly reaches regions outside the training dataset's distribution, causing the model to make unreliable predictions.

C.5.5 Model Input and Output Normalization

We then show the critical role of normalizing both the inputs and outputs of the neural robot dynamics models. As shown by the **No Inp. Norm** and **No Out. Norm** variants in Fig. 7(b), removing either input or output normalization degrades the performance of the *NeRD* model. This is because the input normalization effectively regularizes the ranges of the inputs to the model, making model training stable and efficient. Meanwhile, output normalization mitigates the dominance of the high-magnitude and high-variance velocity terms in the loss function, balancing prediction accuracy across the different state dimensions.

C.5.6 History Window Size h

NeRD generally gains slight performance improvements when the history window size h increases. We provide a comparison on history window sizes $h = 1$, $h = 5$, and $h = 10$ in Fig. 7(c). We choose $h = 10$, as we find $h = 10$ consistently provides stable training and generally achieves the best performance across all tasks in our experiments. Though quite rare, we also notice that further increasing h (*e.g.*, $h = 20$) will occasionally result in an exploded training loss.

C.6 Computation Speed of *NeRD*

With 512 parallel Ant envs, Warp (with 16 substeps, which is the number of substeps we use in Warp for generating training data for Ant) achieves 28K FPS, and *NeRD* achieves 46K FPS. We do not view this comparison as definitive, as both Warp and *NeRD* can be further accelerated; however, as a neural model, *NeRD* benefits from continuous advances in AI hardware and community efforts in ML software acceleration (*e.g.,* TensorRT). Additionally, the policy-learning experiments demonstrate that *NeRD* is fast enough for large-scale on-policy RL.