# Sim-to-Real Transfer for Muscle-Actuated Robots via Generalized Actuator Networks

Jan Schneider[1], Mridul Mahajan[2], Le Chen[1], Simon Guist[1],
Bernhard Schölkopf[1,3], Ingmar Posner[4], and Dieter Büchler[1,5,6,7]

[1]MPI for Intelligent Systems, Tübingen, [2]Boston University, [3]ELLIS Institute Tübingen, [4]University of Oxford,
[5]CIFAR AI Chair, [6]University of Alberta, [7]Alberta Machine Intelligence Institute (Amii)

***Abstract*—Tendon drives paired with soft muscle actuation enable faster and safer robots while potentially accelerating skill acquisition. Still, these systems are rarely used in practice due to inherent nonlinearities, friction, and hysteresis, which complicate modeling and control. So far, these challenges have hindered policy transfer from simulation to real systems. To bridge this gap, we propose a sim-to-real pipeline that learns a neural network model of this complex actuation and leverages established rigid body simulation for the arm dynamics and interactions with the environment. Our method, called Generalized Actuator Network (GeAN), enables actuation model identification across a wide range of robots by learning directly from joint position trajectories rather than requiring torque sensors. Using GeAN on PAMY2, a tendon-driven robot powered by pneumatic artificial muscles, we successfully deploy precise goal-reaching and dynamic ball-in-a-cup policies trained entirely in simulation. To the best of our knowledge, this result constitutes the first successful sim-to-real transfer for a four-degrees-of-freedom muscle-actuated robot arm.**

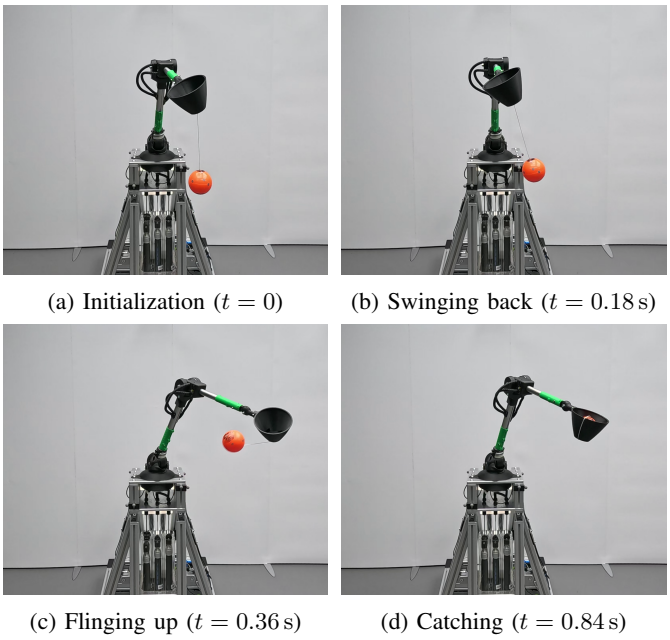
## I. Introduction

Tendon-driven robot arms paired with soft actuation present a promising alternative to classical rigid and motor-driven systems [1], [2], [3], [4]. These designs allow for reducing the moving masses significantly by placing the actuation in the base. Combined with powerful pneumatic actuators, e.g., pneumatic artificial muscles (PAMs), these systems excel at athletic tasks [2], [3]. Furthermore, the lightweight design and mechanical compliance greatly reduce contact forces upon collision [3], making these robots safer to operate around humans even at higher speeds. Muscle actuation can also facilitate more sample-efficient skill learning [5].

Despite these advantages and the prevalence of muscle and tendon actuation in biological systems, modern robots are rarely equipped with such actuation. The primary obstacle lies in inherent modeling challenges that impede precise control. Muscle actuation is highly nonlinear, subject to hysteresis, and can exhibit time-varying properties, e.g., due to temperature fluctuations. Tendon drives introduce additional friction to the system that depends on the joint positions, as tendons are routed through the robot. Consequently, existing works often resort to learning-based approaches, such as iterative learning control [6] or reinforcement learning (RL) [7], when generating behaviors with these robots.

Many recent successes in robot learning, including locomotion [8], [9], table tennis [10], and soccer [11] have been enabled by sim-to-real transfer. Such methods allow for learning complex behaviors without requiring vast amounts of interactions with the physical robot. Through the use of massive parallelization enabled by GPU-based simulators, these techniques also have the potential to vastly reduce the training time, sometimes from hours or days to mere minutes [9], [12]. Prolonged motion execution on a robot is also energy-intensive, accelerates mechanical wear, and typically necessitates extensive safety considerations. Moreover, automatically resetting the physical environment to a fixed initial condition can be difficult, especially if the robot is handling external objects.

A common technique to bridge the *sim-to-real gap*, i.e., the difference between simulated and real dynamics, is *domain randomization* [13], [14]. Domain randomization adds noise to the physics parameters during training to robustify the RL policy to dynamics variations. However, for muscle-



(a) Initialization ($t = 0$) (b) Swinging back ($t = 0.18\,\mathrm{s}$)

(c) Flinging up ($t = 0.36\,\mathrm{s}$) (d) Catching ($t = 0.84\,\mathrm{s}$)

Fig. 1: The ball-in-a-cup policy on a tendon-driven and muscle-actuated robot. The behavior was trained entirely in simulation with a learned actuator model and transferred zero-shot to the physical robot.

actuated robots, the sim-to-real gap is vastly greater due to the aforementioned modeling challenges. Even domain randomization techniques rely on approximately correct dynamics models, which have remained elusive for PAM-actuated robots [15], [16]. Compensating for this wide sim-to-real gap with domain randomization would require increasing the noise on the dynamics parameters significantly. However, excessive noise on the parameters generally degrades policy performance [17]. An alternative to sim-to-real learning for muscle-actuated systems is hybrid-sim-and-real training [7], where only the objects are simulated and the robot remains real. However, even with the sample-efficiency improvements by Guist et al. [18], these techniques still require many hours of real-world interactions, limiting the scalability to more complex tasks.

The core idea of this work is to utilize a known dynamics model for the analytically tractable model components while learning the complicated actuation dynamics from data, making use of expressive neural network models. Concretely, we introduce the Generalized Actuator Network (GeAN), inspired by the actuator network framework [19]. In contrast to the original method, which relies on torque sensors and focuses on more well-behaved series elastic actuators, our method utilizes only joint position measurements, generalizing the technique to a wide range of robots with different actuation types. To the best of our knowledge, we demonstrate *the first* sim-to-real transfer of ball-in-a-cup policies for a tendon-driven robot powered by PAMs (see Figure 1). Our contributions are threefold:

1) We expand the applicability of learned actuator models to robots without torque sensors by introducing the Generalized Actuator Network (GeAN), which learns actuator dynamics from joint position trajectories.
2) Using GeAN, we demonstrate the first successful sim-to-real transfer for a four-degrees-of-freedom robot arm with muscle and tendon actuation.
3) We explore the utility of GeAN ensembles for preventing policy overfitting to model uncertainty, especially in low-data regimes.

## II. Related work

This paper tackles sim-to-real reinforcement learning for muscle-actuated systems and extends upon ideas from actuator model learning. In this section, we discuss the relation to the existing literature in these fields.

### A. Sim-to-real transfer with learned actuator models

The idea of utilizing *actuator networks*, i.e., learned actuator models, for sim-to-real learning was first introduced by Hwangbo et al. [19] in the context of quadruped locomotion. They train a neural network to predict joint torques produced by the series elastic actuation of a quadruped robot. For the target labels, they use torque measurements, which limit their method to robots that are equipped with torque sensors. They demonstrate zero-shot sim-to-real transfer for locomotion and fall recovery behaviors. The approach was then used in a series of further works on quadruped locomotion [12], [20], [21], [22], demonstrating its utility for learning agile locomotion in diverse terrains.

Despite these successes, the application of actuator networks beyond quadruped locomotion is only slowly gaining traction. Spinelli et al. [23] learn end-effector control of an excavator using a neural network model of its hydraulic actuation. Yuryev and Hughes [24] learn a model of forces transferred via a tendon, and use it for trajectory tracking with a one-degree-of-freedom motor-actuated finger. Fey et al. [25] treat the actuator network optimization as an RL task, where the agent attempts to produce torques that minimize the error between simulated and real trajectories. They use the method to model friction and hysteresis effects in a robot arm with harmonic drives and demonstrate successful sim-to-real transfer in dynamic whole-body control tasks. In contrast to their work, our method enables gradient-based optimization, thereby simplifying training and yielding a more accurate model. As a result, our method is capable of modeling muscle dynamics, which are highly nonlinear and subject to complicated, configuration-dependent friction along the tendons.

### B. Sim-to-real transfer for muscle-actuated systems

Existing works on sim-to-real transfer for muscle actuation target systems with relatively simple kinematics and are limited to reaching-type tasks. Tao et al. [26] utilize a combination of system identification with an analytic dynamics model and domain randomization [14] to learn reaching policies for single-joint robotic systems. Biyajima et al. [27] demonstrate a successful sim-to-real transfer with an analytic dynamics model for a one-degree-freedom percussion robot. Wang et al. [28] learn a Deep Lagrangian Network [29] model of PAM dynamics to train a goal-reaching policy for a single muscle with a weight attached.

Schumacher et al. [30] approach muscle actuation from a different direction by emulating muscles in software on a motor-driven quadruped. Thereby, they simplify the sim-to-real transfer as the muscle dynamics are known exactly at training time. However, since the muscles are only emulated, this approach sacrifices some of the advantages of muscle actuation, such as zero-delay compliance. Beyond sim-to-real transfer, Büchler et al. [7] demonstrate that simulations can facilitate learning a dynamic table tennis task with a complex muscle-actuated robot. They circumvent the challenges of simulating the muscle and tendon dynamics by keeping the robot real and simulating only the ball. Even though Guist et al. [18] improve the sample efficiency of the approach by simulating multiple balls during each stroke, the training still requires many hours of interaction with the real robot.

To advance sim-to-real learning for muscle-actuated robot arms toward more realistic robot applications, it is paramount to develop methods suitable for more complex and capable robots. Extending existing methods to multiple joints introduces significant modeling challenges, such as mechanical coupling between the degrees of freedom and friction dependent on the configuration of the robot due to tendon routing.

This work tackles these modeling challenges by leveraging the expressiveness of neural network models, thereby unlocking sim-to-real learning for more complex muscle-actuated robots. To the best of our knowledge, we demonstrate the first successful sim-to-real transfer for a four-degrees-of-freedom muscle-actuated robot.

## III. Sim-to-real pipeline

This work presents a novel approach to simulating robots with complex actuator dynamics that leverages the known arm dynamics to learn an actuator model, enabling sim-to-real learning for these systems. Concretely, our pipeline consists of three phases. First, we collect a dataset of open-loop motions with the real robot and train a *Generalized Actuator Network (GeAN)* that maps robot states and control signals to resulting joint torques. Then, we utilize this model together with a simulator of the arm dynamics to train an RL policy entirely in simulation. Lastly, we deploy this policy zero-shot on the real robot. See Figure 2 for an overview of the pipeline.

### A. Data collection

We first collect an exploration dataset of 2500 open-loop trajectories, each two seconds in length, for a total of about $1.4\,\mathrm{h}$ of robot data. For each trajectory, we sample control signals every 0.5 seconds and fit a cubic spline between these commands to obtain smooth but diverse exploration trajectories that span the robot's workspace and contain different velocity profiles. At each step, we record the current joint position $\boldsymbol{q}_t$ and control signal $\boldsymbol{u}_t$. We compute joint velocities $\dot{\boldsymbol{q}}_t$ and accelerations $\ddot{\boldsymbol{q}}_t$ from the positions via backward differences and central differences, respectively. See Appendix A for an explanation of this choice.

### B. Actuator network training

We split the dataset from Section III-A into $80\%$ training data and $20\%$ validation data, which we use for early stopping. Using this data, we train a GeAN $\hat{\boldsymbol{\tau}}_t = f_{\boldsymbol{\theta}}(\boldsymbol{q}_{t-H:t}, \boldsymbol{u}_{t-H:t})$ that maps from control signals to the resulting joint torques $\hat{\boldsymbol{\tau}}_t$. Similar to [19], we pass an $H$-step history of joint positions $\boldsymbol{q}_{t-H:t}$ and control signals $\boldsymbol{u}_{t-H:t}$ as input to the network to model hysteresis effects. Furthermore, we assume that we have access to a simulator of the arm dynamics with a step function $\boldsymbol{q}_{t+1} = \mathrm{step}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{\tau}_t)$ and an inverse dynamics function $\boldsymbol{\tau}_t = \mathrm{invdyn}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \ddot{\boldsymbol{q}}_t)$. Note that this simulator is entirely torque-driven since we do not have access to an accurate analytic model of the muscle and tendon dynamics.

Hwangbo et al. [19] propose to use sparse histories $(\boldsymbol{x}_t, \boldsymbol{x}_{t-s}, \ldots, \boldsymbol{x}_{t-s\cdot H})$ with stride length $s = 4$, mentioning overfitting as problem of dense histories. We hypothesize that overfitting occurs for dense histories since two consecutive measurements tend to be very similar to each other, making it difficult for the network to make proper use of such histories. To mitigate this issue, we replace the histories $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \ldots, \boldsymbol{x}_{t-H})$ in the network input with *delta histories*, i.e., representing the same sequences by the differences to the current value $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1} - \boldsymbol{x}_t, \ldots, \boldsymbol{x}_{t-H} - \boldsymbol{x}_t)$ and normalize all inputs to mean 0 and standard deviation 1, which amplifies the differences between consecutive values. The analysis in Appendix B indeed shows that short strides tend to work best in our setting. For the remainder of the paper, we denote the normalized delta histories simply as $\boldsymbol{q}_{t-H:t}$ and $\boldsymbol{u}_{t-H:t}$ for conciseness.

In the following, we propose two losses to train the GeAN via supervised learning. A *torque loss* that directly measures errors in torque space and a *position loss* that passes the torques through the simulator to compute the difference between predicted and true next position.

*1) Torque loss:* We first sample a sequence of joint positions and control signals $\boldsymbol{q}_{t-H:t+1}, \boldsymbol{u}_{t-H:t}$ from the dataset and compute the joint velocities $\dot{\boldsymbol{q}}_t$ and accelerations $\ddot{\boldsymbol{q}}_t$. Since we have no torque measurements, we compute torque labels via the inverse dynamics $\boldsymbol{\tau}_t = \mathrm{invdyn}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \ddot{\boldsymbol{q}}_t)$. Note that we assume that there are no external forces on the robot, except gravity. To deal with differences in torque magnitudes across joints, we standardize the torque labels $\boldsymbol{\tau}_t^{\mathrm{std}} = (\boldsymbol{\tau}_t - \boldsymbol{\mu})/\boldsymbol{\sigma}$, where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation across the torques of the training set and the multiplication and division are elementwise. To obtain the torque predictions, we invert the standardization when we deploy the GeAN $f_{\boldsymbol{\theta}}(\boldsymbol{q}_{t-H:t}, \boldsymbol{u}_{t-H:t}) = f_{\boldsymbol{\theta}}^{\mathrm{std}}(\boldsymbol{q}_{t-H:t}, \boldsymbol{u}_{t-H:t})\boldsymbol{\sigma} + \boldsymbol{\mu}$. We train the network with the following squared loss

$$\mathcal{L}_{\mathrm{tor}}(\boldsymbol{\theta}) = \left\| f_{\boldsymbol{\theta}}^{\mathrm{std}}(\boldsymbol{q}_{t-H:t}, \boldsymbol{u}_{t-H:t}) - \boldsymbol{\tau}_t^{\mathrm{std}} \right\|^2. \tag{1}$$

*2) Position loss:* When deploying the GeAN, it is important that the joint positions resulting from the predicted torques are accurate. However, the loss in Equation (1) does not directly optimize the position accuracy. In Appendix C, we derive that the position error and the torque error are related as follows

$$\boldsymbol{q}_{t+1} - \hat{\boldsymbol{q}}_{t+1} = \Delta t^2 \boldsymbol{M}(\boldsymbol{q}_t)^{-1}(\boldsymbol{\tau}_t - \hat{\boldsymbol{\tau}}_t), \tag{2}$$

where $\hat{\boldsymbol{q}}_{t+1} = \mathrm{step}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \hat{\boldsymbol{\tau}}_t)$ is the next position resulting from predicted torque $\hat{\boldsymbol{\tau}}_t = f_{\boldsymbol{\theta}}\big(\boldsymbol{q}_{t-H:t}, \boldsymbol{u}_{t-H:t}\big)$, $\Delta t$ is the simulator time step, and $\boldsymbol{M}(\boldsymbol{q}_t)$ is the mass matrix of the robot in position $\boldsymbol{q}_t$. The equation implies that an ideal actuator network with zero torque error would also result in zero position error. However, errors from imperfect torque predictions are scaled by the inverse of the mass matrix, which can have significant off-diagonal elements. Therefore, there are directions in which the torque errors compensate each other, while they add in other directions, which means that not only the torque error magnitude but also the error direction matters. To take this insight into account, we introduce the following position loss to directly optimize the error that is relevant during deployment.

$$\mathcal{L}_{\mathrm{pos}}(\boldsymbol{\theta}) = \left\| \hat{\boldsymbol{q}}_{t+1} - \boldsymbol{q}_{t+1} \right\|^2 \tag{3}$$

To compute gradients for updating the network, we differentiate Equation (2), which is equivalent to differentiating through the simulator for one step. We also experimented with a multi-step variant of this loss, but we found that it does not yield consistent gains; refer to Appendix E for details.

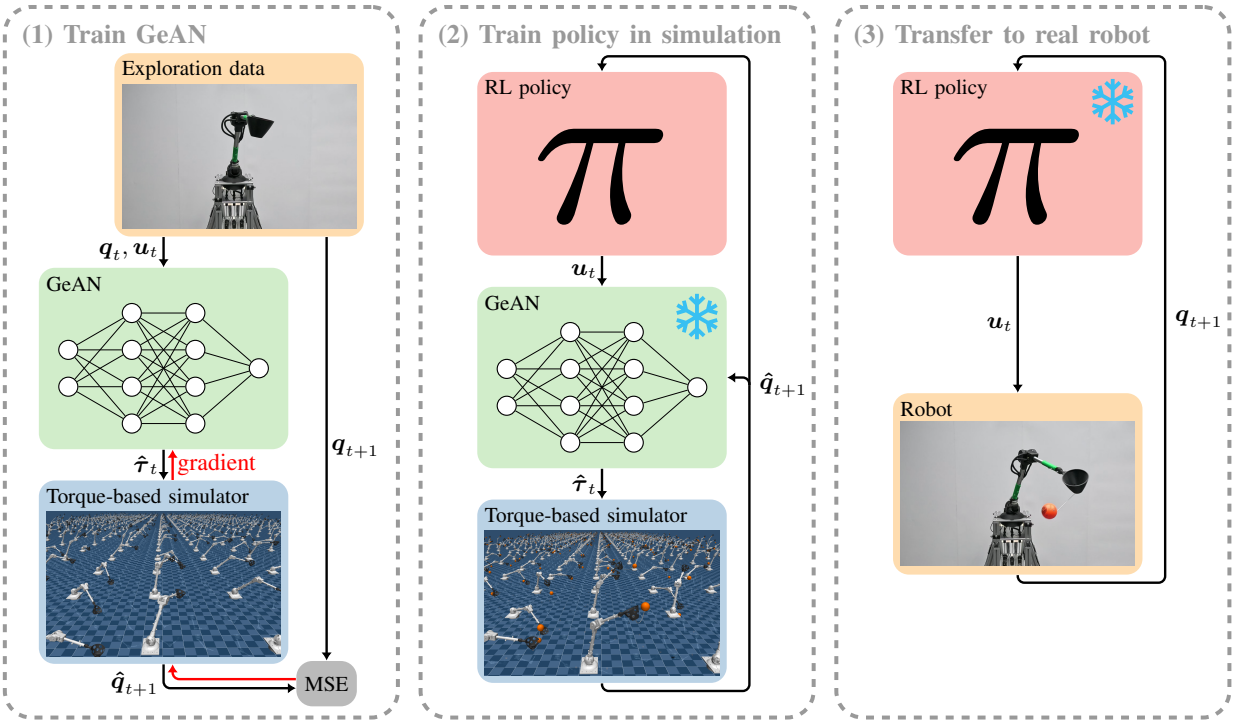



Fig. 2: Overview of the sim-to-real pipeline. (1) Actuator network training with the position loss, where the network is trained to produce torques so that the simulated joint positions match the exploration data. (2) RL training in simulation, where the trained actuator network converts the policy's control signals into torques, which are fed into a torque-based simulator of the arm and external objects. (3) Zero-shot transfer to the real system.

### C. Simulated RL environment

To obtain a realistic simulation environment for training policies, we deploy the GeAN with a torque-based simulator of the arm dynamics and the objects in the scene. The arm and objects follow simple rigid body dynamics, which can be simulated accurately with analytic models. Therefore, we use the learned model only for the complex and hard-to-model tendon and muscle dynamics. In contrast to learning the entire simulator end-to-end, this scheme allows leveraging the extensive prior knowledge about rigid body dynamics and enables changing the scene, e.g., by adding or removing objects, without retraining the network. Instead of a single network, we use an ensemble of 5 GeANs, each trained according to Section III-B. Each model in the ensemble is initialized with a different random seed and trains on a different permutation of the exploration dataset. The ensemble disagreement constitutes a measure of the model's epistemic uncertainty. Similar to Janner et al. [31], we sample a random network from the GeAN ensemble for each simulation step to mitigate policy overfitting to model uncertainty.

## IV. Evaluation on a Muscle-Actuated Robot

We evaluate our method on PAMY2 [3], a PAM-actuated, tendon-driven robot arm with four degrees of freedom (DoFs). Each DoF is actuated by an antagonistic muscle pair. Similar to [32], we actuate both muscles in a correlated antagonistic fashion with a single control signal to reduce the number of actions to one per DoF. Increasing the control signal decreases the desired agonist pressure and simultaneously increases the antagonist pressure.

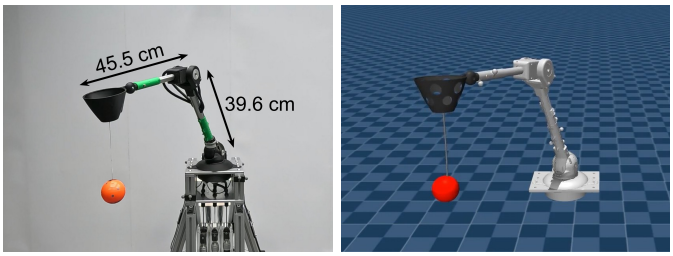



Fig. 3: The 4-DoF muscle-actuated robot PAMY2 [3] (left) with its simulated counterpart (right).

Figure 3 shows the robot and its virtual counterpart, which we simulate in MuJoCo XLA (MJX) [33], an efficient GPU-based simulator. Note that the MJX simulation itself is torque-driven, and the actuator model maps from the pressure commands $\boldsymbol{u}$ to torques $\boldsymbol{\tau}$ to be applied in the simulator.

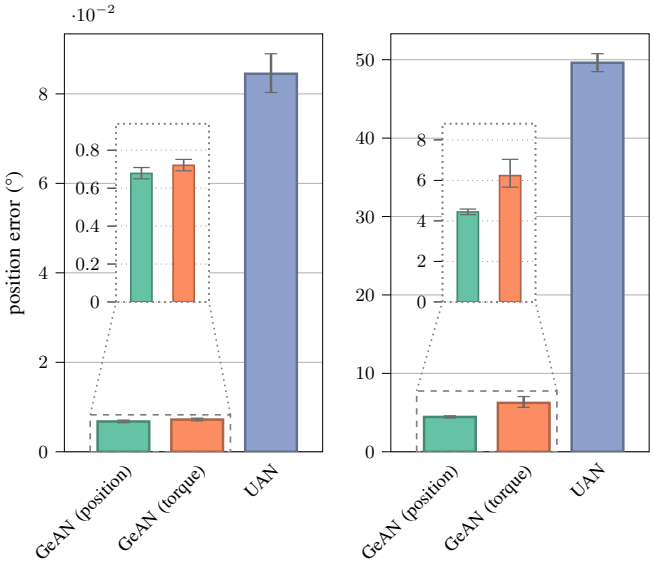
### A. Actuator net accuracy

We compare the networks resulting from the two training losses in Equations (1) and (3) to the Unsupervised Actuator Net (UAN) [25], which frames the actuator net optimization as an RL problem. In this framework, the UAN is the policy whose observations are histories of the system state and

control signals, and the actions are torques. Each episode corresponds to one trajectory from the training dataset, and the task of the policy is to output torques that produce the same trajectory in the simulator. The reward is the negative distance between the simulated and dataset trajectories, plus a smoothness term. Note that the authors assume a motor-driven system controlled by a PD-controller with known gains. Hence, the control signals are desired positions, and the torques commanded by the controller are known. The agent then learns residuals between the commanded and true torques, which are caused by friction losses in the gears. For the muscle-actuated robot, we do not know the torques produced by the actuators and, therefore, adapt the method to directly predict the full joint torques. To ensure a fair comparison, we ran a hyperparameter search for UAN, sweeping over 500 sets of RL hyperparameters, history lengths, network architectures, and reward weights. For our method, we found that relatively little hyperparameter tuning is required. Refer to Appendix D-A for the values that we use.

To test whether the actuator networks faithfully model the actuation dynamics of PAMY2, we collected a test set of 800 trajectories on the real robot with the exploration policy from Section III-A. We set the simulator state to the first configuration of this trajectory and apply the same commands. Then we measure the deviation between the simulated trajectory and the dataset trajectory. With this data, we can assess the quality of the actuator-net-augmented simulation not only for a single step but also the robustness to error accumulation during multi-step rollouts, where the network gets joint states resulting from its previous predictions. Figure 4 compares the mean absolute position error for the two training losses from Equations (1) and (3) to the UAN. In both the single-step prediction case and the multi-step rollouts, the GeAN trained with the position loss produces the most accurate trajectories, yielding errors that are 6% and 29% lower, respectively, than those of the model trained with the torque loss. The UAN fails to capture the robot dynamics and has a vastly higher simulation error both in the single-step and multi-step cases. We believe that the UAN is unable to model the dynamics accurately since the RL agent gets only incomplete information, causing the optimization to be brittle. As the agent observes only controls and the positions resulting from its previous predictions, but not the true positions, it cannot reliably infer rewards or values based on the observations alone, making the task partially observable. Also note that we cannot provide the true positions as input, as this information is not available during deployment of the network.

Since the position loss results in the highest accuracy of the two losses defined in Section III-B, we use this training method for the remainder of this paper.




(a) Error after 1 step / 2 ms (b) Error after 500 steps / 1 s

Fig. 4: Position error for the GeAN trained with the position and torque losses, compared to UAN [25]. For each of the 800 real test trajectories, we reset the simulator to the same initial configuration and apply the same command sequence. The plots show the mean absolute error between simulated and real trajectories after (a) a single step (2 milliseconds) and (b) 500 steps (1 second). The error bars visualize the 95% confidence intervals, obtained via bootstrapping. While the position error results in more accurate simulations, both training methods vastly outperform the UAN baseline.

### B. Tasks

To validate the suitability of the GeAN-augmented simulator for training policies for PAMY2, we learn two tasks: *reacher* and *ball-in-a-cup* [34]. The reacher task involves only the robot and, therefore, directly evaluates the precision of the learned motions independent of external error sources, such as object tracking noise. Conversely, the ball-in-a-cup task tests the method's robustness to the extra weight and external forces caused by the ball, which are not seen during the GeAN training. For both tasks, we run the simulator at 500 Hz for numerical stability but query the RL agent at 100 Hz by repeating each action for 5 steps. We found that the lower agent frequency makes policy learning more efficient and robust. The episode is truncated after 2 seconds, which corresponds to 200 agent steps. For policy learning, we use the skrl [35] implementation of Proximal Policy Optimization (PPO) [36] in a GPU-accelerated, massively parallelized simulation.

*1) Reacher:* In the *reacher* task, the robot is initialized with a random control signal and has to reach randomly sampled target joint positions $\boldsymbol{g}$, see Figure 5a. The observations $\boldsymbol{o}_t = (\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_{t-1}, \boldsymbol{g})$ include the current position $\boldsymbol{q}_t$ and velocity $\dot{\boldsymbol{q}}_t$, the last command $\boldsymbol{u}_{t-1}$, and the goal pose $\boldsymbol{g}$ for all joints. The policy actions are desired changes to the control signal $\boldsymbol{a}_t = \Delta \boldsymbol{u}_t$. The reward consists of four terms.

$$r_t = r_t^{\text{dist}} + c^{\text{act}} r_t^{\text{act}} + c^{\text{disag}} r_t^{\text{disag}} + c^{\text{lim}} r_t^{\text{lim}} \tag{4}$$

$r_t^{\text{dist}} = -\|\boldsymbol{q}_t - \boldsymbol{g}\|$ is the main reward that incentivizes moving to the goal position, $r_t^{\text{act}} = -\|\boldsymbol{a}_t\|^2$ is an action penalty that discourages rapidly changing controls, $r_t^{\text{disag}} = -\sum_{j=1}^{4} \boldsymbol{\sigma}_j^{\tau}$

with $\boldsymbol{\sigma}^\tau = \mathrm{SD}_i \left[ f_{\boldsymbol{\theta}_i}(\boldsymbol{q}_{t-H:t}, \boldsymbol{u}_{t-H:t}) \right]$ penalizes disagreement in the GeAN ensemble, measured by the standard deviation across the outputs of the ensemble members. $r_t^{\mathrm{lim}}$ applies a penalty if the agent gets too close to the joint limits. For the weighting constants, we use $c^{\mathrm{act}} = 1250$, $c^{\mathrm{disag}} = 0.025$, and $c^{\mathrm{lim}} = 1$. Note that $r^{\mathrm{dist}}$ is awarded every step and therefore encourages the agent to move quickly to the goal. Refer to Appendix D-B for more details on the reacher task.

*2) Ball-in-a-cup:* In the ball-in-a-cup task, the robot has to swing a ball on a string into a cup at its end effector (see Figure 1). This is a challenging task that requires speed to fling up the ball and precision to catch it.

Similar to reacher, the observations include the robot state $(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$ and the last command $\boldsymbol{u}_{t-1}$. The task-specific observations are the ball position and velocity $\boldsymbol{x}_t^b, \dot{\boldsymbol{x}}_t^b$, resulting in the observations $\boldsymbol{o}_t = (\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_{t-1}, \boldsymbol{x}_t^b, \dot{\boldsymbol{x}}_t^b)$. The reward function for the task is defined as

$$r_t = r_t^{\mathrm{cup}} + c^{\mathrm{act}} r_t^{\mathrm{act}} + c^{\mathrm{vel}} r_t^{\mathrm{vel}} + c^{\mathrm{disag}} r_t^{\mathrm{disag}} + c^{\mathrm{lim}} r_t^{\mathrm{lim}}, \quad (5)$$

where

$$r_t^{\mathrm{cup}} = \begin{cases} 10, & \text{if the ball is in the cup} \\ 0, & \text{otherwise} \end{cases}$$

is a sparse success reward, and $r_t^{\mathrm{vel}} = -\|\dot{\boldsymbol{q}}_t\|^2$ is a joint velocity penalty that discourages overly aggressive motions with $c^{\mathrm{vel}} = 0.0025$. All other reward terms match the reacher task and use identical weights.

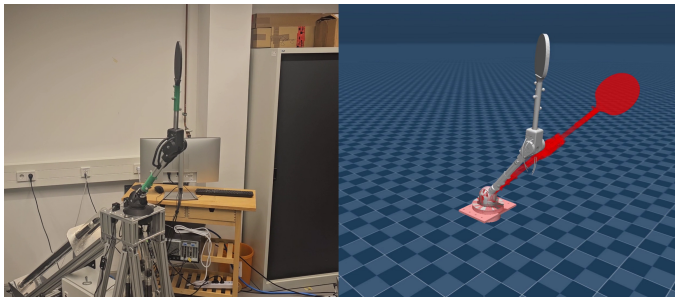
To obtain the real ball position and velocity during the transfer, we use a Vicon object tracking system. Placing 15 flat reflective markers in an irregular pattern onto the ball yielded reasonably reliable tracking results. We observe two issues with this solution: the reflectiveness of the ball's material and occlusions when the ball is near or inside the cup. The tracking software occasionally misidentifies reflections on the ball as markers, resulting in small errors in the ball position measurements, while the occlusions cause missing detections. To make the agent more robust to these errors, we inject zero-mean Gaussian noise with $\sigma^b = 0.5\,\mathrm{cm}$ into the ball positions during the training in simulation, and with a probability of 5 percent, we omit the ball position entirely. Both in simulation and on the real system, we maintain a buffer of the last 5 ball positions and provide the last successful position measurement to the agent, as well as the finite difference ball velocity averaged over the buffer. Refer to Appendix D-C for more details on the ball-in-a-cup task.

### C. Transfer to the real robot

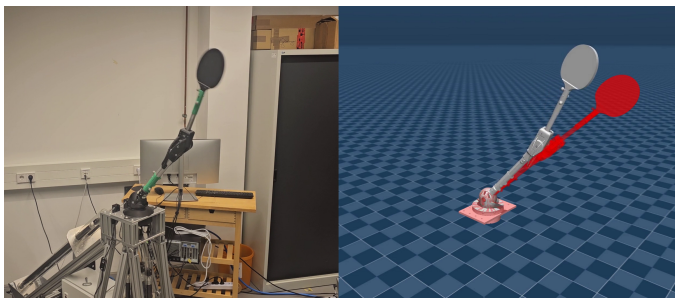
After training purely in simulation, we transfer the policies to the real robot in a zero-shot manner. To judge the reacher agent's performance, we define the following success criterion

$$\frac{1}{4} \sum_{j=1}^{4} \left| \boldsymbol{q}_T^{(j)} - \boldsymbol{g}^{(j)} \right| < 2^\circ. \quad (6)$$

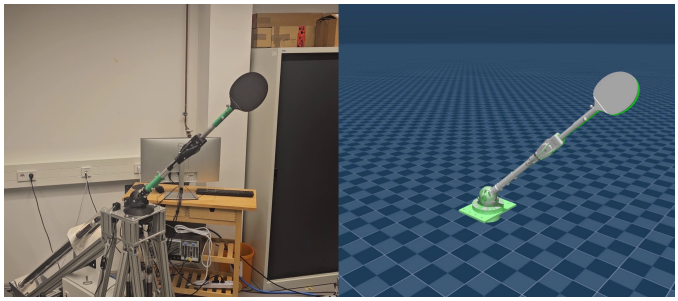
This criterion checks whether the average error between the joint positions at the last step $T$ of the episode and the goal is less than $2^\circ$. Figure 5 visualizes an example of a successful episode on the real system. In the ball-in-a-cup task, we consider an episode a success if the robot gets the ball into the cup. See Figure 1 for an example of a successful episode.



(a) Initial configuration



(b) The robot moves toward the goal configuration



(c) The robot has reached the goal configuration

Fig. 5: Visualization of the reacher task. The motions of the real robot (left) are copied to the MuJoCo environment (right) to visualize the distance to the goal. The goal marker turns from red to green when the deviation is below the success threshold, defined in Equation (6).

Figure 6 shows the agent's success rate for the reacher and ball-in-a-cup tasks on the real robot. Each policy was rolled out 100 times, and the plots show the mean and 95% confidence intervals for the trials. In addition to the main configuration, we show two ablations. The "no ensemble" ablation uses only a single GeAN instead of the ensemble of five networks. The "low action penalty" configuration reduces the action penalty weight $c^{\mathrm{act}}$ in the reward (Equation (4)) from 1250 to 250. In both tasks, the main and "no ensemble" configurations perform comparably: both achieve high success rates (90 % and 93 % for reacher and 75 % and 74 % for ball-in-a-cup). To the best of our knowledge, this is the first successful sim-to-real transfer

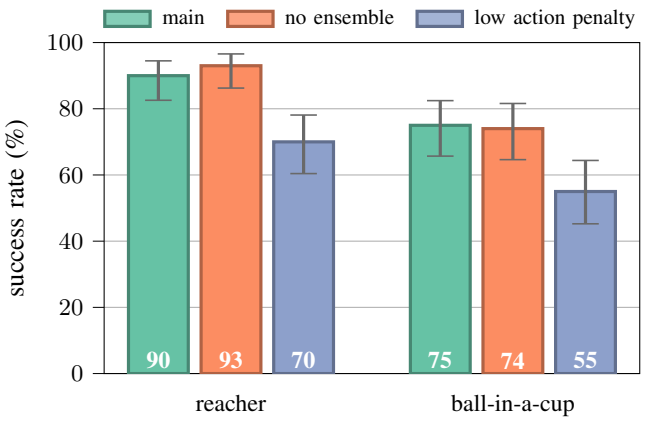



Fig. 6: Success rates for the reacher and ball-in-a-cup policies on the real robot (higher is better). Results are computed for 100 trials. The error bars visualize the 95% confidence intervals across trials computed with the Wilson score interval. For both tasks, the main and "no ensemble" configurations result in similar performance. Reducing the action penalty degrades the transfer performance to some extent.

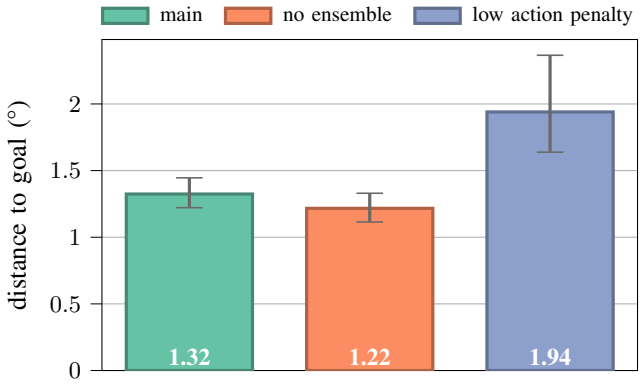
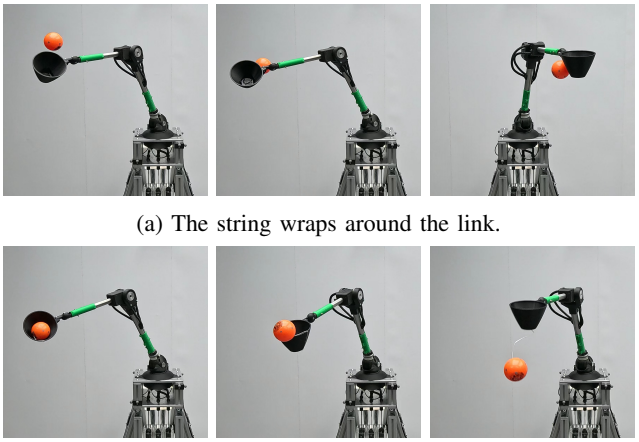



Fig. 7: Mean absolute distance between the final joint positions and the goal for the reacher task (lower is better). The error bars visualize the 95% confidence intervals across 100 trials computed via bootstrapping. The main and "no ensemble" configurations result in similar goal distances, but reducing the action penalty increases the distances.

with a muscle-actuated robot for tasks of this complexity, underscoring the utility of learned actuator models for sim-to-real learning with muscle-actuated systems. Since the GeAN is trained without the influence of the ball, the successful ball-in-a-cup transfer also showcases that the model is robust to certain changes to the end effector weight and unseen external forces. Reducing the action penalty degrades the performance to a success rate of 70 % for reacher and 55 % for ball-in-a-cup. The training trajectories for the GeAN (see Section III-A) are relatively smooth, and overly jittery trajectories, therefore, are out-of-distribution for the GeAN, making the simulation less realistic. Smooth policies are desirable for real robots in any case, as they consume less energy and reduce wear.



(a) The string wraps around the link.

(b) The ball bounces out of the cup.

Fig. 8: Common failure modes of the ball-in-a-cup policy caused by differences in the ball and string dynamics.

Figure 7 displays the average joint position error for the reacher task, measured as the joint-space distance to the goal at the final step of the episode. The main and "no ensemble" policies achieve low final position errors of 1.32° and 1.22°, respectively, despite the challenges in precise control inherent to the tendon-driven and muscle-actuated design of the robot. Decreasing the action penalty increases the error to 1.94°.

Successful ball-in-a-cup episodes typically involve the robot first moving slightly in one direction before rapidly reversing to swing up the ball and then catching it midair (see Figure 1). Common failure modes of the policy include the string wrapping around the link (see Figure 8a) and the ball bouncing out of the cup after an attempted catch (see Figure 8b). These failure modes originate from differences in the ball and string dynamics between the simulated and real environment and are, therefore, independent of the actuator modeling. For simplicity, we use a MuJoCo tendon to simulate the string, which does not model collisions with the robot geometry. Therefore, the agent does not learn how to recover in this scenario by unwinding the string. Using a more sophisticated string model could enable the agent to learn effective recovery maneuvers for these situations. Preventing the ball from bouncing out of the cup is challenging, as the ball is partly occluded while in the cup, which makes tracking less reliable. Nevertheless, carefully tweaking the contact properties of the ball and cup in simulation to match the real dynamics could mitigate the risk of the ball bouncing out of the cup.

### *D. Influence of the GeAN training dataset size*

In this section, we investigate the impact of the GeAN training set size on the simulation accuracy and the policy transfer. Figure 9 shows the position error of the GeAN-augmented simulation on the test set from Section IV-A for different training set sizes. The errors for both the single-step and multi-step predictions decrease for larger datasets, as expected, but the decrease stops at around 1500 trajectories, signifying diminishing returns from additional data.

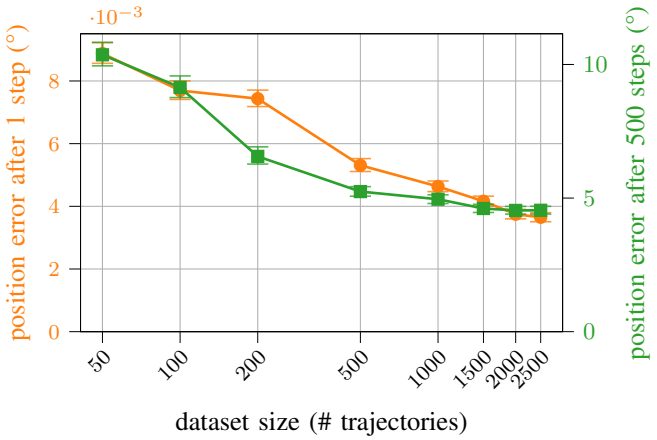



Fig. 9: Mean absolute position error of the GeAN-augmented simulation for networks trained on datasets of different sizes (lower is better). The error markers visualize the 95% confidence intervals across trajectories, obtained via bootstrapping. Smaller training datasets degrade the position accuracy both after a single step (2 milliseconds simulation time; orange plot) and after 500 steps (1 second simulation time; green plot).

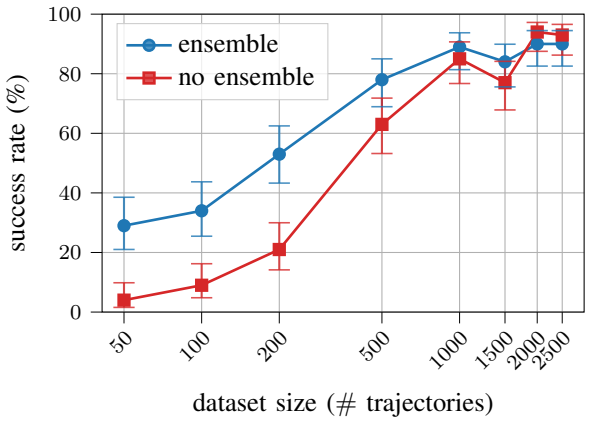



Fig. 10: Reacher policy success rates when using GeANs trained with different dataset sizes (higher is better). The error markers visualize the 95% confidence intervals across trials computed with the Wilson score interval. The dataset size can be reduced by 60% without losing performance. With a single GeAN, instead of the ensemble, the policy performance degrades more severely for smaller datasets.

Next, we investigate the role of the GeAN ensemble in these different data regimes. For the transfer with the full training set, Figure 6 shows no significant performance differences between the configurations that use an ensemble and those that use a single network. We believe that this is the case because the epistemic uncertainty of the GeAN is already low enough due to the large dataset size. To test this hypothesis, we use the networks from Figure 9 to train reacher policies. Figure 10 visualizes the success rates of the policy transfer in relation to the GeAN training set size. The results show that we can reduce the dataset size to 1000 trajectories, i.e., just 33 minutes of robot data, without sacrificing performance. This insight aligns with our earlier finding that the effect of additional data diminishes for large datasets. Transfer performance degrades only with further dataset reduction. Furthermore, the graphs highlight that the performance degradation is less severe for the ensemble configurations, suggesting that the ensemble mitigates the effects of increasing epistemic model uncertainty when data is scarce.

## V. Discussion and Future Work

In this paper, we showed the utility of learned actuator models for tendon-driven robot arms with muscle actuation, demonstrating that these models overcome longstanding modeling challenges that have impeded sim-to-real learning for such complex robots. Our training pipeline requires only joint position measurements, thereby eliminating the need for torque sensors, making it applicable to a wide range of robots with different actuation types. By integrating learned actuator models with standard robotics simulators, we enable training RL policies for zero-shot transfer to physical robots. We validated our approach by learning precise goal-reaching and dynamic ball-in-a-cup policies for a complex 4-DoF tendon-driven and PAM-actuated robot. Notably, these results constitute the first sim-to-real transfer for a multi-joint muscle-actuated robot. We further demonstrated that ensembles of actuator models provide an effective means to handle epistemic uncertainty when training on limited data.

Our approach, Generalized Actuator Network (GeAN), unlocks multiple opportunities for future research. First, it would be interesting to learn a single GeAN for multiple, possibly athletic tasks, such as table tennis, badminton, and ball throwing/catching. Furthermore, the precision achieved in the reacher task suggests that trajectory tracking is now in reach, which would enable teleoperation for muscle-actuated systems. Second, we plan to explore ways of learning actuator models and policies that can adapt to dynamics changes. During our experiments, we observed that the robot dynamics change slowly over time due to tendon elongation, wear, and slight deformations of the 3D-printed components, which likely also occur in other complex systems. Currently, these changes necessitate collecting new data at regular intervals to fine-tune the network when using the robot over long time horizons. An adaptive model could eliminate this limitation and even enable the transfer between robot instances.

## Acknowledgements

We thank Felix Grüninger, Heiko Ott, and Thomas Steinbrenner for support with the robot hardware and 3D printing; Gökce Ergün and Senya Polikovsky for help with the ball tracking; and Leyla Gurbanova for implementing an early version of the ball-in-a-cup task. This work was supported by the Max Planck Institute for Intelligent Systems. Jan Schneider was supported by the Konrad Zuse School of Excellence in Learning and Intelligent Systems (ELIZA), sponsored by the German Federal Ministry of Education and Research.

## References


[1] D. Büchler, H. Ott, and J. Peters, "A lightweight robotic arm with pneumatic muscles for robot learning," in *IEEE International Conference on Robotics and Automation*, IEEE, 2016.

[2] S. Mori, K. Tanaka, S. Nishikawa, R. Niiyama, and Y. Kuniyoshi, "High-speed and lightweight humanoid robot arm for a skillful badminton robot," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, 2018.

[3] S. Guist, J. Schneider, H. Ma, L. Chen, V. Berenz, J. Martus, H. Ott, F. Grüninger, M. Muehlebach, J. Fiene, B. Schölkopf, and D. Büchler, "Safe & Accurate at Speed with Tendons: A Robot Arm for Exploring Dynamic Motion," in *Robotics: Science and Systems*, 2024.

[4] K. Kawaharazuka, S. Makino, K. Tsuzuki, M. Onitsuka, Y. Nagamatsu, K. Shinjo, T. Makabe, Y. Asano, K. Okada, K. Kawasaki, et al., "Component modularized design of musculoskeletal humanoid platform Musashi to investigate learning control systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2019.

[5] I. Wochner, P. Schumacher, G. Martius, D. Büchler, S. Schmitt, and D. Haeufle, "Learning with muscles: Benefits for data-efficiency and robustness in anthropomorphic tasks," in *Conference on Robot Learning*, PMLR, 2023.

[6] H. Ma, D. Büchler, B. Schölkopf, and M. Muehlebach, "A Learning-based Iterative Control Framework for Controlling a Robot Arm with Pneumatic Artificial Muscles," in *Robotics: Science and Systems*, 2022.

[7] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," *IEEE Transactions on Robotics*, vol. 38, no. 6, 2022.

[8] I. Radosavovic, T. Xiao, B. Zhang, T. Darrell, J. Malik, and K. Sreenath, "Real-world humanoid locomotion with reinforcement learning," *Science Robotics*, vol. 9, no. 89, 2024.

[9] Y. Seo, C. Sferrazza, J. Chen, G. Shi, R. Duan, and P. Abbeel, "Learning Sim-to-Real Humanoid Locomotion in 15 Minutes," *arXiv preprint arXiv:2512.01996*, 2025.

[10] Z. Su, B. Zhang, N. Rahmanian, Y. Gao, Q. Liao, C. Regan, K. Sreenath, and S. S. Sastry, "HITTER: A humanoid table tennis robot via hierarchical planning and learning," *arXiv preprint arXiv:2508.21043*, 2025.

[11] Z. Xu, M. Seo, D. Lee, H. Fu, J. Hu, J. Cui, Y. Jiang, Z. Wang, A. Brund, J. Biswas, et al., "Learning Agile Striker Skills for Humanoid Soccer Robots from Noisy Sensory Input," *arXiv preprint arXiv:2512.06571*, 2025.

[12] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*, PMLR, 2022.

[13] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International Conference on Robotics and Automation*, IEEE, 2018.

[14] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers in Robotics and AI*, vol. 9, 2022.

[15] B. Tondu, "Modelling of the McKibben artificial muscle: A review," *Journal of Intelligent Material Systems and Structures*, vol. 23, no. 3, 2012.

[16] D. Büchler, R. Calandra, B. Schölkopf, and J. Peters, "Control of musculoskeletal systems using learned dynamics models," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, 2018.

[17] G. Tiboni, P. Klink, J. Peters, T. Tommasi, C. D'Eramo, and G. Chalvatzaki, "Domain Randomization via Entropy Maximization," in *International Conference on Learning Representations*, 2024.

[18] S. Guist, J. Schneider, A. Dittrich, V. Berenz, B. Schölkopf, and D. Büchler, "Hindsight States: Blending sim and real task elements for efficient reinforcement learning," in *Robotics: Science and Systems*, 2023.

[19] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

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

[21] Y. Ji, G. B. Margolis, and P. Agrawal, "DribbleBot: Dynamic Legged Manipulation in the Wild," in *IEEE International Conference on Robotics and Automation*, IEEE, 2023.

[22] C. Eichmann, S. Bellmann, N. Hügel, L.-E. Enslin, C. Plasberg, G. Heppner, A. Roennau, and R. Dillmann, "LAURON VI: A Six-Legged Robot for Dynamic Walking," in *International Conference on Advanced Robotics and Mechatronics*, 2025.

[23] F. A. Spinelli, P. Egli, J. Nubert, F. Nan, T. Bleumer, P. Goegler, S. Brockes, F. Hofmann, and M. Hutter, "Reinforcement learning control for autonomous hydraulic material handling machines with underactuated tools," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2024.

[24] V. Yuryev and J. Hughes, "Tendon Force Modeling for Sim2Real Transfer of Reinforcement Learning Policies for Tendon-Driven Robots," *arXiv preprint arXiv:2603.04351*, 2026.

[25] N. Fey, G. B. Margolis, M. Peticco, and P. Agrawal, "Bridging the Sim-to-Real Gap for Athletic Loco-Manipulation," in *Robotics: Science and Systems*, 2025.

[26] J. Tao, Y. Zhang, S. K. Rajendran, and F. Zhang, "An Efficient Learning Control Framework With Sim-to-Real for String-Type Artificial Muscle-Driven Robotic Systems," *IEEE/ASME Transactions on Mechatronics*, 2025.

[27] T. Biyajima, R. Yamazaki, and M. Okui, "Development of a Variable-Stiffness Musculoskeletal Percussion Robot and Realization of Single-Stroke Motion through Sim-to-Real Transfer," in *51st Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2025.

[28] S. Wang, R. Wang, Y. Liu, Y. Zhang, and L. Hao, "Dynamic modeling and control of pneumatic artificial muscles via Deep Lagrangian Networks and Reinforcement Learning," *Engineering Applications of Artificial Intelligence*, vol. 148, 2025.

[29] M. Lutter, C. Ritter, and J. Peters, "Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning," in *International Conference on Learning Representations*, 2019.

[30] P. Schumacher, L. Krause, J. Schneider, D. Büchler, G. Martius, and D. Haeufle, "Learning to Control Emulated Muscles in Real Robots: A Software Test Bed for Bio-Inspired Actuators in Hardware," in *10th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics*, IEEE, 2024.

[31] 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.

[32] D. Büchler, R. Calandra, and J. Peters, "Learning to control highly accelerated ballistic movements on muscular robots," *Robotics and Autonomous Systems*, vol. 159, 2022.

[33] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012.

[34] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Advances in Neural Information Processing Systems*, vol. 21, 2008.

[35] A. Serrano-Muñoz, D. Chrysostomou, S. Bøgh, and N. Arana-Arexolaleiba, "skrl: Modular and flexible library for reinforcement learning," *Journal of Machine Learning Research*, vol. 24, no. 254, 2023.

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

## APPENDIX A
## COMPUTING VELOCITIES AND ACCELERATIONS

Let $\boldsymbol{q}_t$ be the real robot trajectory and $\tilde{\boldsymbol{q}}_t$ the simulated trajectory for $t \in \{0, \dots, T\}$. To compute the labels for the torque loss of Equation (1), we need to compute the torques that retrace the real trajectory in the simulator. We use the first two positions of the trajectory to initialize the simulator by setting $\tilde{\boldsymbol{q}}_0 = \boldsymbol{q}_0$ and $\tilde{\boldsymbol{q}}_1 = \boldsymbol{q}_1$ and compute the torques $\boldsymbol{\tau}_t$ that result in $\tilde{\boldsymbol{q}}_{t+1} = \boldsymbol{q}_{t+1}$ for $t \in \{1, \dots, T-1\}$.

MuJoCo and many other dynamics simulators use a symplectic Euler integrator, which combines an explicit integration step for the velocities

$$\dot{\tilde{\boldsymbol{q}}}_{t+1} = \dot{\tilde{\boldsymbol{q}}}_t + \Delta t \, \ddot{\tilde{\boldsymbol{q}}}_t \tag{7}$$

with an implicit step for the positions

$$\tilde{\boldsymbol{q}}_{t+1} = \tilde{\boldsymbol{q}}_t + \Delta t \, \dot{\tilde{\boldsymbol{q}}}_{t+1}. \tag{8}$$

Furthermore, the inverse dynamics equation of a robot manipulator is given by

$$\boldsymbol{\tau}_t = \boldsymbol{M}(\tilde{\boldsymbol{q}}_t)\ddot{\tilde{\boldsymbol{q}}}_t + \boldsymbol{c}\Big(\tilde{\boldsymbol{q}}_t, \dot{\tilde{\boldsymbol{q}}}_t\Big) + \boldsymbol{g}(\tilde{\boldsymbol{q}}_t) \tag{9}$$

$$= \mathrm{invdyn}\Big(\tilde{\boldsymbol{q}}_t, \dot{\tilde{\boldsymbol{q}}}_t, \ddot{\tilde{\boldsymbol{q}}}_t\Big), \tag{10}$$

where $\boldsymbol{M}(\boldsymbol{q}_t)$ is the mass matrix, $c(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$ are the centrifugal and Coriolis forces, and $g(\boldsymbol{q}_t)$ is the gravity vector.
By rearranging, we obtain the forward dynamics equation

$$\ddot{\tilde{\boldsymbol{q}}}_t = \boldsymbol{M}(\tilde{\boldsymbol{q}}_t)^{-1}\left(\boldsymbol{\tau}_t - \boldsymbol{c}\Big(\tilde{\boldsymbol{q}}_t, \dot{\tilde{\boldsymbol{q}}}_t\Big) - \boldsymbol{g}(\tilde{\boldsymbol{q}}_t)\right). \tag{11}$$

Assume an arbitrary $t \in \{1, \dots, T-1\}$, $\tilde{\boldsymbol{q}}_{t-1} = \boldsymbol{q}_{t-1}$, and $\tilde{\boldsymbol{q}}_t = \boldsymbol{q}_t$. We show how to compute the torque $\boldsymbol{\tau}_t$, so that $\tilde{\boldsymbol{q}}_{t+1} = \boldsymbol{q}_{t+1}$. Using Equations (8) and (11), we obtain

$$\boldsymbol{q}_{t+1} = \tilde{\boldsymbol{q}}_{t+1} \tag{12}$$

$$= \tilde{\boldsymbol{q}}_t + \Delta t \, \left(\dot{\tilde{\boldsymbol{q}}}_t + \Delta t \, \ddot{\tilde{\boldsymbol{q}}}_t\right) \tag{13}$$

$$= \tilde{\boldsymbol{q}}_t + \Delta t \left(\dot{\tilde{\boldsymbol{q}}}_t + \Delta t \boldsymbol{M}(\tilde{\boldsymbol{q}}_t)^{-1}\Big(\boldsymbol{\tau}_t - \boldsymbol{c}\Big(\tilde{\boldsymbol{q}}_t, \dot{\tilde{\boldsymbol{q}}}_t\Big)\dot{\tilde{\boldsymbol{q}}}_t - \boldsymbol{g}(\tilde{\boldsymbol{q}}_t)\Big)\right) \tag{14}$$

$$= \tilde{\boldsymbol{q}}_t + \Delta t \, \dot{\tilde{\boldsymbol{q}}}_t + \Delta t^2 \boldsymbol{M}(\tilde{\boldsymbol{q}}_t)^{-1}\Big(\boldsymbol{\tau}_t - \boldsymbol{c}\Big(\tilde{\boldsymbol{q}}_t, \dot{\tilde{\boldsymbol{q}}}_t\Big)\dot{\tilde{\boldsymbol{q}}}_t - \boldsymbol{g}(\tilde{\boldsymbol{q}}_t)\Big). \tag{15}$$

Rearrange the equation to

$$\boldsymbol{\tau}_t = \frac{1}{\Delta t^2}\boldsymbol{M}(\tilde{\boldsymbol{q}}_t)\Big(\boldsymbol{q}_{t+1} - \tilde{\boldsymbol{q}}_t - \Delta t \dot{\tilde{\boldsymbol{q}}}_t\Big) + \boldsymbol{c}\Big(\tilde{\boldsymbol{q}}_t, \dot{\tilde{\boldsymbol{q}}}_t\Big)\dot{\tilde{\boldsymbol{q}}}_t + \boldsymbol{g}(\tilde{\boldsymbol{q}}_t). \tag{16}$$

By rearranging Equation (8), we obtain

$$\dot{\tilde{\boldsymbol{q}}}_t = \frac{\tilde{\boldsymbol{q}}_t - \tilde{\boldsymbol{q}}_{t-1}}{\Delta t}, \tag{17}$$

which we use to compute

$$\boldsymbol{\tau}_t = \frac{1}{\Delta t^2}\boldsymbol{M}(\tilde{\boldsymbol{q}}_t)\left(\boldsymbol{q}_{t+1} - \tilde{\boldsymbol{q}}_t - \Delta t \frac{\tilde{\boldsymbol{q}}_t - \tilde{\boldsymbol{q}}_{t-1}}{\Delta t}\right) + \boldsymbol{c}\left(\tilde{\boldsymbol{q}}_t, \frac{\tilde{\boldsymbol{q}}_t - \tilde{\boldsymbol{q}}_{t-1}}{\Delta t}\right)\frac{\tilde{\boldsymbol{q}}_t - \tilde{\boldsymbol{q}}_{t-1}}{\Delta t} + \boldsymbol{g}(\tilde{\boldsymbol{q}}_t) \tag{18}$$

$$= \boldsymbol{M}(\tilde{\boldsymbol{q}}_t)\left(\frac{\boldsymbol{q}_{t+1} - 2\tilde{\boldsymbol{q}}_t + \tilde{\boldsymbol{q}}_{t-1}}{\Delta t^2}\right) + \boldsymbol{c}\left(\tilde{\boldsymbol{q}}_t, \frac{\tilde{\boldsymbol{q}}_t - \tilde{\boldsymbol{q}}_{t-1}}{\Delta t}\right)\frac{\tilde{\boldsymbol{q}}_t - \tilde{\boldsymbol{q}}_{t-1}}{\Delta t} + \boldsymbol{g}(\tilde{\boldsymbol{q}}_t). \tag{19}$$

With $\tilde{\boldsymbol{q}}_t = \boldsymbol{q}_t$, we obtain

$$\boldsymbol{\tau}_t = \boldsymbol{M}(\boldsymbol{q}_t)\left(\frac{\boldsymbol{q}_{t+1} - 2\boldsymbol{q}_t + \boldsymbol{q}_{t-1}}{\Delta t^2}\right) + \boldsymbol{c}\left(\boldsymbol{q}_t, \frac{\boldsymbol{q}_t - \boldsymbol{q}_{t-1}}{\Delta t}\right)\frac{\boldsymbol{q}_t - \boldsymbol{q}_{t-1}}{\Delta t} + \boldsymbol{g}(\boldsymbol{q}_t) \tag{20}$$

$$= \text{invdyn}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \ddot{\boldsymbol{q}}_t) \tag{21}$$

for

$$\dot{\boldsymbol{q}}_t = \frac{\boldsymbol{q}_t - \boldsymbol{q}_{t-1}}{\Delta t} \tag{22}$$

and

$$\ddot{\boldsymbol{q}}_t = \frac{\boldsymbol{q}_{t+1} - 2\boldsymbol{q}_t + \boldsymbol{q}_{t-1}}{\Delta t^2}. \tag{23}$$

Equations (22) and (23) are the first-order backward and second-order central differences equations.

## Appendix B
## Effect of the history design choices on the simulation accuracy

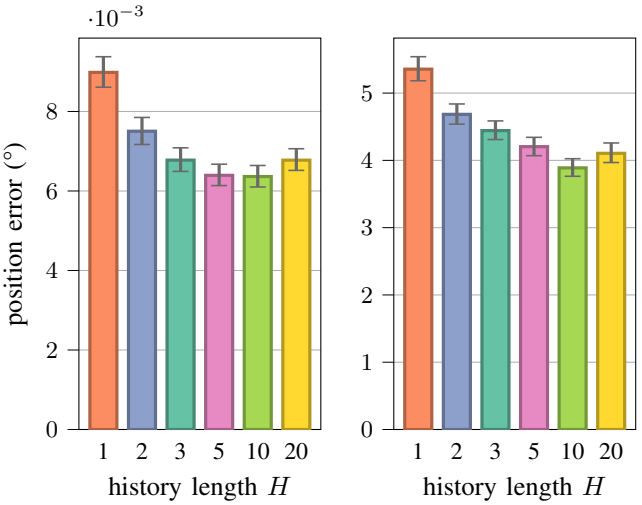
The history input is an important component of the actuator modeling since actuators like PAMs are prone to hysteresis effects. Therefore, in this section, we evaluate the impact of the history length $H$ and stride $s$ for network inputs of the form $(\boldsymbol{q}_t, \boldsymbol{q}_{t-s}, \ldots, \boldsymbol{q}_{t-s\cdot H}, \boldsymbol{u}_t, \boldsymbol{u}_{t-s}, \ldots, \boldsymbol{u}_{t-s\cdot H})$.

Figure 11 evaluates the simulator position error for GeAN trained with different history lengths. Generally, longer histories result in lower errors in both the single-step and 500-step position error metrics. There is a strong improvement for $H = 2$ over $H = 1$, indicating that one-step histories are insufficient to model the complex dynamics of PAMs. Beyond $H = 2$ the gains from longer histories seem to get gradually smaller. Histories of length $H = 20$ seem to be too long, resulting in a slight performance decrease. For all other experiments, we use $H = 3$ as a tradeoff between accuracy and computational efficiency.

In the original actuator networks paper [19], the authors propose to use a sparse, i.e., strided, history, mentioning overfitting as a problem of dense histories. Figure 12 shows the simulator accuracy for different stride lengths $s$. Generally, shorter strides seem to perform best, with only $s = 2$ leading to a slightly lower error than $s = 1$ in the multi-step error metric. In the single-step error metric, $s = 1$ outperforms all longer strides. As the experiment was conducted on an unseen test dataset, this result indicates that overfitting is not an issue for GeANs with short strides.

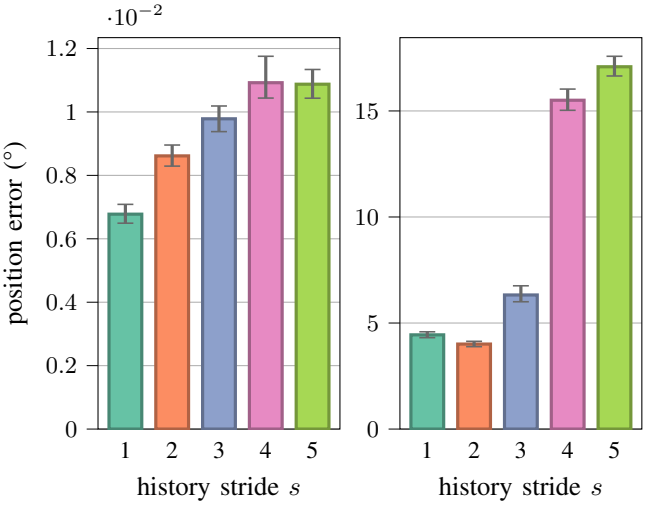
## Appendix C
## Relation between the torque and position loss

Let $\boldsymbol{q}_t$ for $t \in \{0, \ldots, T\}$ be a trajectory from the dataset and $\dot{\boldsymbol{q}}_t$ and $\ddot{\boldsymbol{q}}_t$ the corresponding finite difference velocities and accelerations, computed with Equations (22) and (23) for $t \in \{1, \ldots, T-1\}$ . Furthermore, let $\hat{\boldsymbol{q}}_{t+1}$ be the next simulated position resulting from the dataset position $\hat{\boldsymbol{q}}_t = \boldsymbol{q}_t$




(a) Error after 1 step / 2 ms (b) Error after 500 steps / 1 s

Fig. 11: Position error for GeANs trained with different history lengths $H$, measured as the mean absolute error between 800 simulated trajectories and real robot trajectories (lower is better). Stride length is set to $s = 1$ for all configurations. The error bars visualize the 95% confidence intervals across samples computed via bootstrapping. Up to $H = 10$, GeANs with longer histories tend to perform better. There is a large difference between the accuracy for $H = 1$ and $H = 2$ with progressively smaller gains beyond $H = 2$.




(a) Error after 1 step / 2 ms (b) Error after 500 steps / 1 s

Fig. 12: Position error for GeANs trained with different stride lengths $s$, measured as the mean absolute error between 800 simulated trajectories and real robot trajectories (lower is better). History length is set to $H = 3$ for all configurations. The error bars visualize the 95% confidence intervals across samples computed via bootstrapping. Generally, shorter strides seem to work best. Only in the multi-step loss, yields $s = 2$ a slightly lower error than $s = 1$.

and velocity $\hat{\dot{\boldsymbol{q}}}_t = \dot{\boldsymbol{q}}_t$ after applying the torque $\hat{\boldsymbol{\tau}}_{t+1}$ predicted by the GeAN for all $t \in \{1, \dots, T-1\}$. Starting from the position error $\boldsymbol{q}_{t+1} - \hat{\boldsymbol{q}}_{t+1}$, we derive Equation (2) by utilizing the integration step for the position and velocity from Equations (7) and (8).

Analogous to Equations (7) and (8), we obtain the integration step

$$\dot{\boldsymbol{q}}_{t+1} = \dot{\boldsymbol{q}}_t + \Delta t\, \ddot{\boldsymbol{q}}_t \tag{24}$$

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \Delta t\, \dot{\boldsymbol{q}}_{t+1} \tag{25}$$

and with $\hat{\boldsymbol{q}}_t = \boldsymbol{q}_t$ and $\hat{\dot{\boldsymbol{q}}}_t = \dot{\boldsymbol{q}}_t$, we get

$$\hat{\dot{\boldsymbol{q}}}_{t+1} = \hat{\dot{\boldsymbol{q}}}_t + \Delta t\, \hat{\ddot{\boldsymbol{q}}}_t \tag{26}$$

$$= \dot{\boldsymbol{q}}_t + \Delta t\, \hat{\ddot{\boldsymbol{q}}}_t \tag{27}$$

$$\hat{\boldsymbol{q}}_{t+1} = \hat{\boldsymbol{q}}_t + \Delta t\, \hat{\dot{\boldsymbol{q}}}_{t+1} \tag{28}$$

$$= \boldsymbol{q}_t + \Delta t\, \hat{\dot{\boldsymbol{q}}}_{t+1} \tag{29}$$

Analogous to Equation (11), the forward dynamics for these two cases are given by

$$\ddot{\boldsymbol{q}}_t = \boldsymbol{M}(\boldsymbol{q}_t)^{-1} \left(\boldsymbol{\tau}_t - \boldsymbol{c}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) - \boldsymbol{g}(\boldsymbol{q}_t)\right) \tag{30}$$

$$\hat{\ddot{\boldsymbol{q}}}_t = \boldsymbol{M}(\hat{\boldsymbol{q}}_t)^{-1} \left(\hat{\boldsymbol{\tau}}_t - \boldsymbol{c}\Big(\hat{\boldsymbol{q}}_t, \hat{\dot{\boldsymbol{q}}}_t\Big) - \boldsymbol{g}(\hat{\boldsymbol{q}}_t)\right) \tag{31}$$

$$= \boldsymbol{M}(\boldsymbol{q}_t)^{-1} \left(\hat{\boldsymbol{\tau}}_t - \boldsymbol{c}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) - \boldsymbol{g}(\boldsymbol{q}_t)\right). \tag{32}$$

By first inserting Equations (25) and (29), then Equations (24) and (27), and finally Equations (30) and (32), we obtain Equation (2).

$$\begin{aligned}
&\boldsymbol{q}_{t+1} - \hat{\boldsymbol{q}}_{t+1} \\
&\quad = \boldsymbol{q}_t + \Delta t\, \dot{\boldsymbol{q}}_{t+1} - \boldsymbol{q}_t - \Delta t \hat{\dot{\boldsymbol{q}}}_{t+1} \\
&\quad = \Delta t(\dot{\boldsymbol{q}}_{t+1} - \hat{\dot{\boldsymbol{q}}}_{t+1}) \\
&\quad = \Delta t(\dot{\boldsymbol{q}}_t + \Delta t\, \ddot{\boldsymbol{q}}_t - \dot{\boldsymbol{q}}_t - \Delta t\, \hat{\ddot{\boldsymbol{q}}}_t) \\
&\quad = \Delta t^2(\ddot{\boldsymbol{q}}_t - \hat{\ddot{\boldsymbol{q}}}_t) \\
&\quad = \Delta t^2 \big(\boldsymbol{M}(\boldsymbol{q}_t)^{-1} \left(\boldsymbol{\tau}_t - c(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) - g(\boldsymbol{q}_t)\right) \\
&\qquad\qquad - \boldsymbol{M}(\boldsymbol{q}_t)^{-1} \left(\hat{\boldsymbol{\tau}}_t - c(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) - g(\boldsymbol{q}_t)\right)\big) \\
&\quad = \Delta t^2 \boldsymbol{M}(\boldsymbol{q}_t)^{-1} (\boldsymbol{\tau}_t - \hat{\boldsymbol{\tau}}_t)
\end{aligned}$$

## APPENDIX D
## IMPLEMENTATION DETAILS AND HYPERPARAMETERS

This section describes implementation details and hyperparameters for the RL tasks. First, we describe general choices and then details specific to reacher and ball-in-a-cup.

To obtain diverse but stable and realistic conditions in the simulator at the start of each episode, we sample initial control signals $\boldsymbol{u}_{\text{init}} \sim \mathcal{U}(\boldsymbol{u}_{\text{init}}^{\min}, \boldsymbol{u}_{\text{init}}^{\max})$. We then set the robot joint position to the intermediate angles $(0, 45°, 45°, 0)$ and apply $\boldsymbol{u}_{\text{init}}$ for 500 steps, i.e., 1 second in simulation time. In contrast to just sampling random initial positions, this scheme ensures that the combination of controls and positions corresponds to a stable configuration on the real system. The RL episode starts after these initial 500 steps. During the transfer experiments, we follow a similar procedure. We sample initial controls from the same distribution to test the policy in diverse initial conditions. The only difference is that we ramp the controls linearly to $\boldsymbol{u}_{\text{init}}$ over 2 seconds to avoid unnecessarily aggressive motions during the reset.

The actions $\boldsymbol{a}$ are squashed to the range $[-1, 1]$ by passing the output of the policy network through the $\tanh$ function. Afterward, the result is scaled by $\Delta u^{\max} = 0.01$ to prevent the policy from executing overly aggressive motions, resulting in the following mapping from policy output $\hat{\boldsymbol{a}}_t$ to control signal change $\Delta \boldsymbol{u}_t$.

$$\Delta \boldsymbol{u}_t = \Delta \boldsymbol{u}^{\max} \tanh(\hat{\boldsymbol{a}}_t) \tag{33}$$

### A. GeAN training and evaluation

We split the dataset collected according to Section III-A into $80\,\%$ training and $20\,\%$ validation data by assigning trajectories randomly to the two splits. Splitting at the step level instead would mean that the network is trained on data points that are potentially very similar to the validation data points due to the temporal correlation within the trajectory. We train the GeAN for 150 epochs, which typically takes around 25 minutes on an Nvidia A100 GPU. After the training, we select the model with the lowest validation loss.

The test set of 800 trajectories was collected separately after the transfer experiments of Section IV-C. Due to the gradual dynamics changes mentioned in Section V, the performance on the test set is potentially a conservative estimate of the GeAN accuracy directly after training. The hundreds of interactions with the real system during the transfer experiments could already have led to slight dynamics changes in the robot, which would increase the position errors measured in Section IV-A and Appendix B.

The complete configuration of hyperparameters that we use for the GeANs in our experiments is listed in Table I.

TABLE I: Hyperparameters for the GeAN training

| | Hyperparameter | Value |
|---|---|---|
| Input | History length | 3 |
| | History stride | 1 |
| Architecture | Hidden layers | 2 |
| | Neurons per layer | 512 |
| | Activation function | tanh |
| | Ensemble size | 5 |
| Training | Optimizer | Adam |
| | Learning rate | $1 \times 10^{-4}$ |

### B. Reacher task

For the reacher task, we sample the goal positions according to $\boldsymbol{g} \sim \mathcal{U}(\boldsymbol{g}^{\min}, \boldsymbol{g}^{\max})$. The complete configuration of environment and agent hyperparameters that we use for the reacher task is given in Table II.

TABLE II: Hyperparameters for the reacher task

| | Hyperparameter | Value |
|---|---|---|
| Environment | Parallel instances | 1024 |
| | Episode length | 2 s |
| | Action repeat | 5 |
| | $\Delta u^{\max}$ | 0.01 |
| | $\boldsymbol{u}_{\text{init}}^{\min}$ | $(-0.5, -0.6, -0.6, -0.5)$ |
| | $\boldsymbol{u}_{\text{init}}^{\max}$ | $(0.5, 0.0, 0.4, 0.5)$ |
| | $\boldsymbol{q}^{\min}$ | $(-90°, -75°, -85°, -85°)$ |
| | $\boldsymbol{q}^{\max}$ | $(90°, 85°, 85°, 85°)$ |
| | $\boldsymbol{g}^{\min}$ | $(-50°, 20°, -50°, -50°)$ |
| | $\boldsymbol{g}^{\max}$ | $(50°, 60°, 50°, 50°)$ |
| PPO (skrl) | discount_factor | 0.9801 |
| | lambda | 0.95 |
| | learning_rate | $3.949 \times 10^{-5}$ |
| | entropy_loss_scale | 0.025 |
| | ratio_clip | 0.1521 |
| | rollouts | 64 |
| | mini_batches | 32 |
| | learning_epochs | 10 |
| | observation_preprocessor | RunningStandardScaler |
| | value_preprocessor | RunningStandardScaler |
| | grad_norm_clip | 1.0 |
| | value_clip | 0.2 |
| | value_loss_scale | 1.0 |
| | kl_threshold | 0.008 |
| Policy | Hidden layers | 4 |
| | Neurons per layer | 64 |
| | Activation function | LeakyReLU |

TABLE III: Hyperparameters for the ball-in-a-cup task

| | Hyperparameter | Value |
|---|---|---|
| Environment | Parallel instances | 1024 |
| | Episode length | 2 s |
| | Action repeat | 5 |
| | $\Delta u^{\max}$ | 0.01 |
| | $\boldsymbol{u}_{\text{init}}^{\min}$ | $(-0.5, -0.6, -0.6, -0.5)$ |
| | $\boldsymbol{u}_{\text{init}}^{\max}$ | $(0.5, 0.0, 0.4, 0.5)$ |
| | $\boldsymbol{q}^{\min}$ | $(-90°, -75°, -85°, -85°)$ |
| | $\boldsymbol{q}^{\max}$ | $(90°, 85°, 85°, 85°)$ |
| | String length | 20 cm |
| | Sim. ball noise std $\sigma^b$ | 0.5 cm |
| | Sim. ball dropout rate | 5 % |
| | Ball pos. buffer size | 5 |
| PPO (skrl) | discount_factor | 0.9835 |
| | lambda | 0.95 |
| | learning_rate | $1.201 \times 10^{-4}$ |
| | entropy_loss_scale | 0.005 |
| | ratio_clip | 0.05882 |
| | rollouts | 1024 |
| | mini_batches | 128 |
| | learning_epochs | 10 |
| | observation_preprocessor | RunningStandardScaler |
| | value_preprocessor | RunningStandardScaler |
| | grad_norm_clip | 1.0 |
| | value_clip | 0.2 |
| | value_loss_scale | 1.0 |
| | kl_threshold | 0.008 |
| Policy | Hidden layers | 3 |
| | Neurons per layer | 128 |
| | Activation function | ELU |

### C. Ball-in-a-cup task

In the simulated ball-in-a-cup environment, we initialize the ball uniformly on the sphere surrounding the string attachment point. The radius of the sphere is the string radius. We chose this initialization over initializing the ball only below the cup to increase the diversity of the initial conditions. Note, however, that the ball naturally always starts below the cup in the real environment. There is no domain randomization on the parameters concerning the ball and string dynamics. Applying domain randomization here could further increase the robustness of the policy to the differences between the simulated and real environments and could, therefore, be explored in future work. The complete configuration of environment and agent hyperparameters that we use for the ball-in-a-cup task is given in Table III.

## APPENDIX E
## MULTI-STEP POSITION LOSS

In Section III-B2, we define a single-step position loss for training the GeAN. A possible extension of this idea rolls out the simulator for $R$ steps, starting from some step $t \in \{H, \dots, T-R\}$. We define $\hat{\boldsymbol{q}}_{t\to r}$ as notation for simulating $r$ steps from $\boldsymbol{q}_t$ into the future. For ease of notation, we define $\hat{\boldsymbol{q}}_{t\to r} = \boldsymbol{q}_{t+r}$ and $\dot{\hat{\boldsymbol{q}}}_{t\to r} = \dot{\boldsymbol{q}}_{t+r}$ for all $r \in \{-H, \dots, 0\}$. We then simulate

$$\hat{\boldsymbol{q}}_{r\to r+1} = \text{step}(\hat{\boldsymbol{q}}_{t\to r}, \dot{\hat{\boldsymbol{q}}}_{t\to r}, \hat{\boldsymbol{\tau}}_{t\to r}) \tag{34}$$

$$\dot{\hat{\boldsymbol{q}}}_{t\to r+1} = \frac{\hat{\boldsymbol{q}}_{t\to r+1} - \hat{\boldsymbol{q}}_{t\to r}}{\Delta t} \tag{35}$$

$$\hat{\boldsymbol{\tau}}_{t\to r} = f_{\boldsymbol{\theta}}\big(\hat{\boldsymbol{q}}_{t\to r-H:r}, \boldsymbol{u}_{t+r-H:t+r}\big) \tag{36}$$

for all $r \in \{0, \dots, R-1\}$, where we use the notation $\hat{\boldsymbol{q}}_{t\to r-H:r} = (\hat{\boldsymbol{q}}_{t\to r-H}, \hat{\boldsymbol{q}}_{t\to r-H+1}, \dots, \hat{\boldsymbol{q}}_{t\to r})$. By adding the losses for the individual simulation steps, we obtain the following *multi-step position loss*

$$\mathcal{L}_{\text{pos,mul}}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r=1}^{R}\left\| \frac{\hat{\boldsymbol{q}}_{t\to r} - \boldsymbol{q}_{t+r}}{\mathbf{c}_r} \right\|^2, \tag{37}$$

where the division is elementwise and $\mathbf{c}_r$ is a normalization constant, described below.

The normalization is required since the position errors for different timestamps are typically on vastly different scales, which results in the network ignoring errors early in the

rollouts in favor of reducing the later errors. For the normalization, we first compute positions $\bar{\boldsymbol{q}}_{t\to r}$ resulting from applying constant zero torques for each timestep of the rollout by setting $\bar{\boldsymbol{q}}_{t\to 0} = \boldsymbol{q}_t$ and $\dot{\bar{\boldsymbol{q}}}_{t\to 0} = \dot{\boldsymbol{q}}_t$ and simulating

$$\bar{\boldsymbol{q}}_{t\to r+1} = \mathrm{step}(\bar{\boldsymbol{q}}_{t\to r}, \dot{\bar{\boldsymbol{q}}}_{t\to r}, \boldsymbol{0}) \tag{38}$$

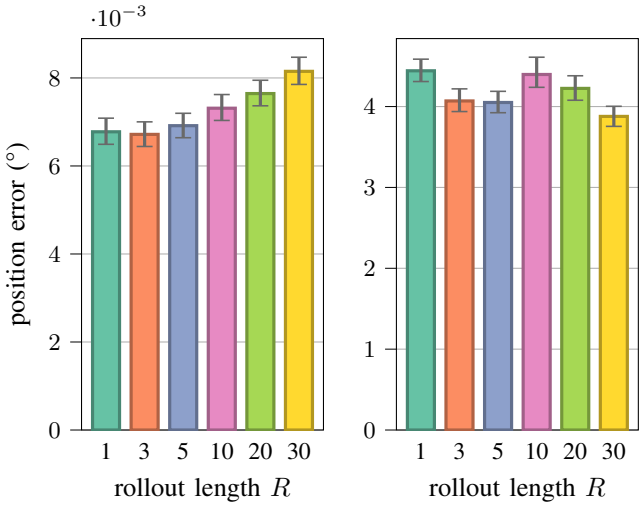
$$\dot{\bar{\boldsymbol{q}}}_{t\to r+1} = \frac{\bar{\boldsymbol{q}}_{t\to r+1} - \bar{\boldsymbol{q}}_{t\to r}}{\Delta t} \tag{39}$$

for all $r \in \{1, \dots, R-1\}$. The normalization constant is then computed by calculating the absolute position error per joint, averaged across all possible $r$-step rollouts in the training dataset.

$$\boldsymbol{c}_r^{(j)} = \left| \bar{\boldsymbol{q}}_{t\to r}^{(j)} - \boldsymbol{q}_{t\to r}^{(j)} \right| \tag{40}$$

The notation $\boldsymbol{x}^{(j)}$ denotes selecting the value for the $j$th joint from the vector.

This normalization essentially compares the error of the network with that of the simplest possible, i.e., constant, model. Naturally, the constant predictions result in increasing errors over the rollouts. Dividing by these errors, therefore, puts more weight on predictions early in the rollouts and mitigates the effects of the error magnitude imbalance described above.




(a) Error after 1 step / 2 ms (b) Error after 500 steps / 1 s

Fig. 13: Position error for GeANs trained with different the position loss rolled out for different rollout lengths $R$, measured as the mean absolute error between 800 simulated trajectories and real robot trajectories (lower is better). In the single-step error, shorter rollouts perform best, while in the multi-step error, longer rollout lengths tend to perform best.

Figure 13 compares the position accuracy of GeANs trained with the multi-step loss of Equation (37) for different rollout lengths $R$. While for the multi-step error, the plots show a slight downward trend for models trained with longer rollouts; in the single-step case, this trend reverses, and shorter rollout lengths result in higher accuracy. Furthermore, the multi-step training is computationally significantly more demanding as it requires differentiating through the simulator for multiple steps, resulting in vastly longer training times. While the single-step training completes in about 25 minutes, the multi-step training with $R = 30$ takes about 12 hours to converge on an Nvidia A100 GPU. Overall, the multi-step loss does not yield clear improvements, and we deem it not worth the additional computational cost. Therefore, we use the single-step loss throughout the main text.