SCALABLE HUMANOID WHOLE-BODY CONTROL VIA DIFFERENTIABLE NEURAL NETWORK DYNAMICS

Yu Lei **Zhengyi Luo**[†] **Tairan He** **Jinkun Cao** **Guanya Shi** **Kris Kitani**[†]
Robotics Institute, Carnegie Mellon University
[†] Equal advising

ABSTRACT

Learning scalable humanoid whole-body controllers is crucial for applications in animation and embodied intelligence. While popular model-free reinforcement learning methods are capable of learning controllers to track large-scale motion databases, they require an exorbitant amount of samples and long training times. Conversely, learning a robust world model has emerged as a promising alternative for efficient and generalizable policy learning. In this work, we learn a neural dynamics model and propose a novel framework **SuperDyno** that combines supervised learning and reinforcement learning for scalable humanoid controller learning. Our method achieves significantly higher sample efficiency and lower tracking error compared to prior approaches, scaling seamlessly to datasets with tens of thousands of motion clips. We further show that medium-sized neural dynamics models can serve as a differentiable neural simulator for accurate prediction and effective policy optimization. We also demonstrate the effectiveness of our framework on sparse reward tasks and the transferability of learned neural dynamics model to diverse tasks. [1]

1 INTRODUCTION

How to learn a robust humanoid whole-body controller capable of achieving diverse motion skills has emerged as a crucial problem for applications like character animation (Peng et al., 2018) and enabling agile loco-manipulation for embodied intelligence (Luo et al., 2024). The task of motion tracking, where a policy learns to imitate selected motion capture clips, is a natural way to acquire the motor skills necessary to perform human-like actions. Most popular method use model-free deep reinforcement learning (RL) algorithms like Proximal Policy Optimization (PPO) due to recent advances in GPU-based parallel simulation. However, these model-free RL algorithms are sample inefficient, require long training time, and are sensitive to hyper-parameter settings, reward, and feature designs. For example, the state-of-the-art method motion tracker that has scaled up to the AMASS dataset (Mahmood et al., 2019) - PHC (Luo et al., 2023a) needs to be trained on an A100 GPU for around one week. The data amount required by model-free methods scales rapidly with the complexity of problems and the diversity of motions. This makes imitating motion from a large dataset, such as AMASS (ten thousand clips, 40 hours of motion), with a single policy difficult.

On the other hand, model-based approaches are generally regarded as being more efficient (Deisenroth et al., 2013), which incorporate approximate models, or the so-called *"world models"*, to predict the future states of the system given its history states and actions. Common usages of these learned models include planning to control (Nagabandi et al., 2018; Hansen et al., 2022) or training model-free algorithms with generated data (Ha & Schmidhuber, 2018; Sekar et al., 2020). However, the performance of these methods heavily depends on the accuracy of learned models, and how to learn accurate dynamics models for planning is still an open question. We argue that one key property of the learned neural network dynamics model is differentiability. The differentiable world models bridge the gap between the simulation and control policy, through which the gradients of the loss function can be back-propagated, thus allowing the training objectives to supervise the policy directly. This can also be called as first-order method for policy learning. Some prior works (Fussell et al.,

[1] Our project website: `https://dynamics-humanoid.github.io/dynamics-humanoid/`

2021; Yao et al., 2022) utilize such property of neural dynamics to track complex human motions, but "perform less efficiently in learning large-scale motion dataset".

In this work, our aim is to create an efficient humanoid controller that scales up to ten thousand motion clips. We propose a novel model-based learning framework that interleaves between reinforcement learning and supervised learning (SL). Different from some previous first-order methods (Georgiev et al., 2024), we train the policy and neural dynamics model simultaneously without the need of collecting huge amount of offline data. In order to keep the policy learning harder and harder motions efficiently, we utilize a hard negative mining strategy during the training process, making the policy pay attention to harder motions. We find that a single policy that achieves a high success rate on motion imitation with much higher sample efficiency and lower tracking error compared with previous SOTA can be achieved by our framework. We systematically analyze the impact of different settings within our framework, showing the right recipe to scale up efficiently. Besides, to demonstrate the power of this framework, we show the qualitative results on other challenging sparse reward tasks. At last, we explore the scaling law of the learned world model with different numbers of motions and prove the transferability of the neural dynamics model.

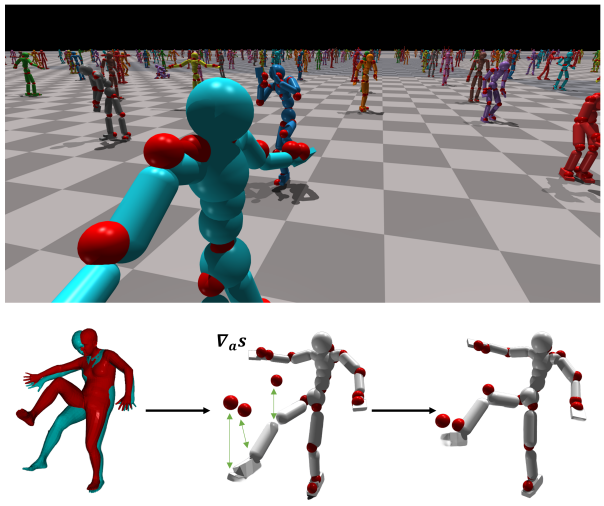


Figure 1: **Our method can effectively scale up to ten thousands of motions.** By learning a neural dynamics model, the training objective can directly supervise the joint actions with $\nabla_a s$.

To summarize, our contributions are as follows: (1) we propose an efficient and well-scalable humanoid controller **SuperDyno** that can imitate more than **99**%of the AMASS dataset, accompanied with a neural dynamics model with fairly good prediction ability within only 80 hours on a single A6000 GPU; (2) we present a novel model-based framework with supervised learning that is capable of solve sparse reward locomotion tasks, and provide the right recipe to utilize; (3) we demonstrate the good transferability of our policy brought by the neural dynamics model, and also investigate its scaling law on policy learning.

2 Related Work

Using a simulated character to track reference motion has a long history in the vision and graphics community (Peng et al., 2018; 2021; Won et al., 2020; Chentanez et al., 2018; Gong et al., 2022; Winkler et al., 2022; Yuan & Kitani, 2020a; Luo et al., 2021; 2022; 2023a) and recently making its way into robotics (He et al., 2024a;b; Cheng et al., 2024). This task involves in controlling a humanoid to track kinematic motions in simulation or the real world. The source of the motion can be either keyframe animation (Peng et al., 2018) or motion capture (Chentanez et al., 2018). Tracking the reference motion can also be done at the style level (Peng et al., 2018) or per-frame level (Won et al., 2020). From tracking a single sequence of motion (Peng et al., 2018), to thousands (Won et al., 2020; Fussell et al., 2021), to tens of thousands of motion sequences (Luo et al., 2023a), the community has come a long way for the motion tracking task despite the diversity and complexity of the humanoid motion datasets. However since physics simulation is often not differentiable, popular methods rely on model-free RL to optimize policies. Due to the sample complexity of RL methods, (Luo et al., 2023a; Won et al., 2020) takes billions of samples and weeks of wall clock time to train. Although there have been attempts to use supervised learning (Fussell et al., 2021) and differentiable simulators (Ren et al., 2023), these methods struggle to scale to large-scale datasets.

3 Methodology

In Sec 3.1, we first set up the our problem formulation; then in Sec 3.2, we introduce our supervised learning based framework with differentiable neural dynamics model.

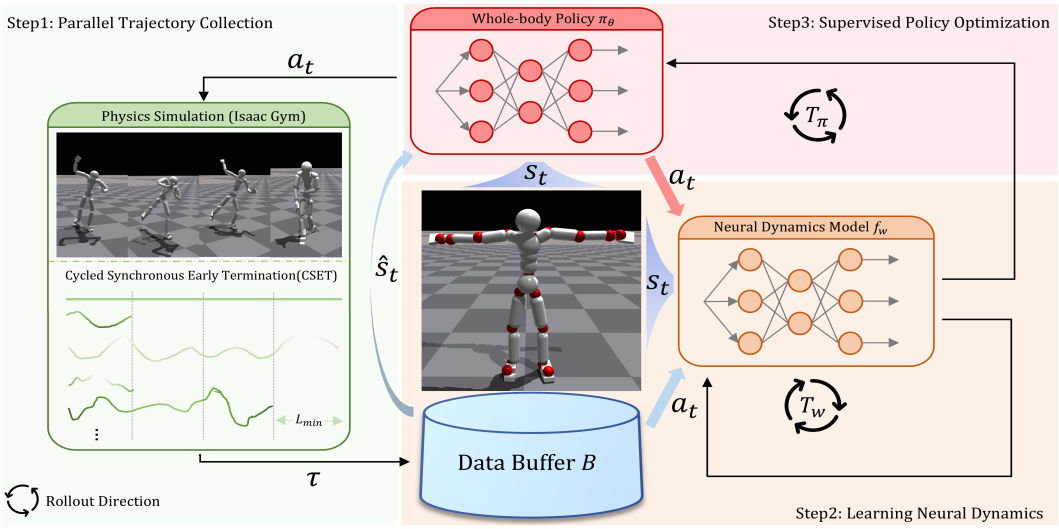


Figure 2: **Overview of Our Framework.**

3.1 Problem Formulation

Similar to prior work (Luo et al., 2021; Peng et al., 2018), we formulate the task as a Markov Decision Process (MDP) defined by the tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{G}, \mathcal{R}, \gamma >$ where $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ are continuous states and actions , $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the transition function determined by physics simulator, $\mathcal{G}$ denotes the space of goals, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \mapsto \mathbb{R}$ is a reward function associated with a particular goal g, and γ is the discount factor. The objective of our goal is to derive a parameterized goal-conditioned policy $\pi_\theta : \mathcal{S} \times \mathcal{G} \mapsto \mathcal{A}$ that maximizes the expectation of the discounted cumulative reward:

$$\max_\theta J(\pi_\theta) = \max_\theta \mathbb{E}_{\pi_\theta, s_0 \sim \rho_0}[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t, g)] \tag{1}$$

where ρ_0 is the initial state distribution. Following (Luo et al., 2023a), we define the reference pose as $\hat{q}_t \triangleq (\hat{\theta}_t, \hat{p}_t)$, consisting of 3D joint rotation $\hat{\theta}_t \in \mathbb{R}^{J\times 6}$ and position $\hat{p}_t \in \mathbb{R}^{J\times 3}$ of all J links on humanoid (we use the 6 DoF rotation representation (Zhou et al., 2019)). From reference poses $\hat{q}_{1:T}$, we can compute the reference velocities $\hat{\dot{q}}_{1:T}$ through finite difference, where $\hat{\dot{q}}_{1:T} \triangleq (\hat{w}_t, \hat{v}_t)$ consist of angular $\hat{w}_t \in \mathbb{R}^{J\times 3}$ and linear velocities $\hat{v}_t \in \mathbb{R}^{J\times 3}$. As a notation convention, we use $\hat{\cdot}$ to denote the ground truth kinematic quantities from Motion Capture (MoCap), $\tilde{\cdot}$ to represent the prediction of dynamics model and normal symbols without accents for values from the physics simulation. For other representation details, please refer to Appdx. A.1.

3.2 SL-based Motion Imitation with Differentiable Neural Dynamics Model

Usually we have our policy gradient as follows:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\pi_\theta, s_0 \in \rho_0}[\nabla_\theta \left(\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t, g) \right)] \\ &= \mathbb{E}_{\pi_\theta, s_0 \in \rho_0}[\sum_{t=0}^{T} \gamma^t \nabla_\theta \mathcal{R}(s_t, a_t, g)] \end{aligned} \tag{2}$$

Different from the situation where we have no access to the transition functions of the physics simulator, by learning a neural dynamics model $f_w(s_t, a_t)$, we can expand the policy gradients further as follows:

$$\nabla_\theta \mathcal{R}(s_t, a_t, g) = \nabla_{s_t} \mathcal{R} \nabla_\theta s_t + \nabla_{a_t} \mathcal{R} \nabla_\theta \pi_\theta(a_t|s_t) \tag{3}$$

$$\nabla_\theta s_t = \nabla_{s_{t-1}} f_w \nabla_\theta s_{t-1} + \nabla_{a_{t-1}} f_w \nabla_\theta a_{t-1} \tag{4}$$

where $\nabla_{s_{t-1}} f_w, \nabla_{a_{t-1}} f_w$ are exactly the Jacobian matrix of the learned neural dynamics model. With these two components, we can back-propagate through time and hopefully have much more efficient policy optimization process.

3.2.1 NEURAL NETWORK DYNAMICS MODEL

The neural dynamics model $f_w(s_t, a_t)$ plays a key role in our policy optimization as it "controls" the direction of optimization (by multiplying the policy gradient with its own Jacobian matrix). Here we describe two important factors of the dynamics model based on our experience.

Injecting Physics Prior. A straightforward parameterization of f_w would be taking the current state s_t and action a_t as input, and output the next state prediction $\tilde{s}_{t+1}$. But this function can be difficult to learn when the states s_t and s_{t+1} are too similar and the action has seemingly little effect on the output. Prior work (Nagabandi et al., 2018) proposes to predict the change of state over the time step instead, which equals to $\tilde{s}_{t+1} = s_t + f_w(s_t, a_t)$. However, one problem introduced by such modeling is that there exist little constraint during the optimization for different physics variables like velocity and position, possibly resulting in predictions disobey the physical laws. Although this phenomenon will decrease as the data amount increases, the cost brought by the massive data collection is not comparable to the corresponding improvement in its performance. To this end, we propose to introduce human physics knowledge prior into the dynamics modeling. More specifically, our neural dynamics model only predicts the partial state $\tilde{v}_t$ and $\tilde{w}_t$. Then we integrate them into the current state to get the next state. Denote the integration operation as Φ:

$$\tilde{w}_{t+1}, \tilde{v}_{t+1} = f_w(s_t, a_t) \tag{5}$$

$$\Phi := \begin{cases} \tilde{p}_{t+1} = \tilde{v}_{t+1}\Delta t + \tilde{p}_t \\ \tilde{\theta}_{t+1} = \exp(\tilde{w}_{t+1}\frac{\Delta t}{2}) \otimes \tilde{\theta}_t \end{cases} \tag{6}$$

$$\implies \quad \tilde{s}_{t+1} = \Phi(f_w, s_t) \tag{7}$$

In the integration we transform the rotation from 6 DoF representation into quaternions, and $\otimes$ represents the quaternion-vector product.

Autoregressive Training. Another key factor influences the performance of neural dynamics model is the training manner. In our framework, we construct the training data by sliding a window of size T_w over collected trajectories. We propagate the neural dynamics model forward T_w times to make multi-step open-loop predictions. Then we calculate the T_w-step errors in the world frames as the loss objective:

$$\mathcal{L}_w = \sum_{t=0}^{T_w-1} \|\tilde{s}_{t+1} - \Phi(f_w, s_t)\|^2 \tag{8}$$

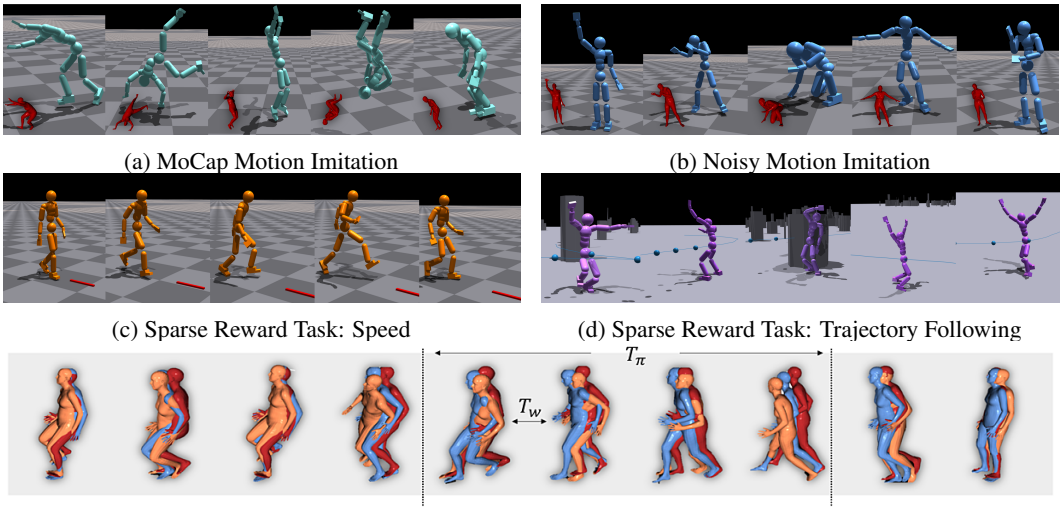
With the strong temporal characteristics shown by motion data, another popular choice of learning dynamics is to train with teacher forcing (Williams & Zipser, 1989). Actually teacher forcing can be viewed as a special case of autoregressive training with $T_w = 1$, which boosts training with higher parallelization. However through our experiments in Sec. 5d, we find that teacher forcing is unsuitable for our usage. Please refer to the discussion in Sec. 4.2.

3.2.2 POLICY OPTIMIZATION

As the supervision signal can directly back-propagate through the neural dynamics model, we opt to use supervised learning to optimize the policy. As the same as training the dynamics model, we also train the policy model autoregressively. More specifically, we take out the initial states and reference states from the data buffer by sliding a window of size T_π. Then we rollout the current policy in the neural dynamics model for T_π times to make multi-step close-cloop predictions. Lastly we calculate the T_π-step errors in the local frames as the loss objective:

$$\mathcal{L}_\theta = \sum_{t=0}^{T_\pi-1} \|\tilde{s}_{t+1} - \Phi(f_w(s_t, \pi_\theta), s_t)\|^2 \tag{9}$$

It is worth noting that compared with previous RL-based methods in motion imitation, although we don't incorporate any discriminator loss like Adversarial Motion Prior (AMP) (Peng et al., 2021) into the policy optimization, it is totally compatible with our framework.



(a) MoCap Motion Imitation

(b) Noisy Motion Imitation

(c) Sparse Reward Task: Speed

(d) Sparse Reward Task: Trajectory Following

(e) Neural Dynamics Model Open-Loop Prediction

Figure 3: **Qualitative results.** (a) Imitating hard MoCap - forward flip. (b) Imitating noisy motion dataset H36M, which is estimated from video. (c, d) Solving sparse reward tasks with high-quality. (e) Prediction visualization of neural dynamics model. The red one is reference motion, the blue one is our policy's tracking and the orange one is the open-loop prediction of dynamics model.

3.2.3 SCALABLE TRAINING FRAMEWORK

Based on the literature above, we establish our supervised-learning-based motion tracking framework with a neural dynamics model. During training, we first collect a number of simulated trajectories then update the neural dynamics model for N_w times and the policy model for N_π times in tandem as illustrated in Algo. 1. However, naively utilizing such framework will lead to less efficient performance on large scale dataset. We describe two additional key designs that enable the policy to scale on AMASS efficiently in this section. The overview of our system is shown in Fig. 2.

Parallel Efficient Trajectory Collection. To allow for enough data to be collected we set up a training environment that closely resembles an RL gym similar in style to some previous works (Luo et al., 2022). We use reference state initialization (RSI) (Peng et al., 2018) during training and randomly select a starting point of a motion clip for imitation. To avoid the data buffer $\mathcal{D}$ being filled with bad quality trajectories at start, we follow PHC (Luo et al., 2023a) and terminate the episode when the joints are more than 0.5 meters globally on average from the reference motion. Unlike PHC, we only terminate the episode after the minimum episode length has been obtained since we need to train the model autoregressively. Prior works like SuperTrack (Fussell et al., 2021) and ControlVAE (Yao et al., 2022) propose to dump the samples into the data buffer once any episode terminates. However, collecting in this way will be very inefficient under the situation that we have many environments and need to learn on a large dataset, as all other environments are paused during the process of data dumping and post-processing, which is attributed to the asynchronous termination between different environments. So to collect more high quality trajectories with less time, we propose Cycled Synchronous Early Termination (CSET), which is designed especially for situations where we need to collect as many complete trajectories with minimum length as possible. For the trajectory reaching the termination distance, we will keep propagating it in the next epoch until the length reaches a multiple of the roll-out steps per epoch. This strategy greatly improves the sampled data quality and massively shortens the data collection time. In our default setting, it only takes $\sim$**1.5** seconds to collect enough data while it takes more than **7** seconds for previous sampling methods. The training speed of our framework gets **doubled** with this design.

Hard Negative Mining. When learning from a large motion dataset, it is essential to train on harder sequences in the later stages of training to gather more informative experiences. We use a similar hard negative mining procedure as in PHC (Luo et al., 2023a). At the start, we assign equal chance to each motion in the dataset. During the training process, we adjust the sampled probability of different motions according to the failed times of current policy on imitating it by evaluating over the entire dataset $p_i = \frac{N_{failed,i}}{\sum_i N_{failed,i}}$.

Algorithm 1 : SuperDyno

Input: Reference motion dataset $\hat{\mathcal{Q}}$, initialized empty data buffer $\mathcal{B}$, and randomly initialized f_w, π_θ
while *not converged* **do**
  $\tau \leftarrow$ roll out current policy π_θ in the environment with $\hat{\mathcal{Q}}$, with Cycled Synchronous Early Termination
  Store τ into $\mathcal{B}$
  for $i = 1$ **to** N_w **do**
    Take out *states* and *actions* from $\mathcal{B}$ with sliding window T_w
    Propagate f_w forward and train f_w with $\mathcal{L}_w$ (Eq. 8)
  end for
  for $i = 1$ **to** N_π **do**
    Take out *states* and *targets* from $\mathcal{B}$ with sliding window T_π
    Roll out π_θ in f_w and train with $\mathcal{L}_\pi$ (Eq. 9)
  end for
end while

4 EXPERIMENTS

We evaluate our efficient and scalable humanoid controller's ability to imitate large scale high-quality MoCap sequences in Sec. 4.1. In Sec. 4.2, we exhaustively ablate on different settings of our framework and demonstrate the capability of our key designs. In Sec. 4.3, we investigate into the transferability and scaling law of the neural dynamics model. Lastly, we try to evaluate the prediction ability of the learned neural dynamics model.

Implementation Details. In the parallel trajectory collection stage, we simulate 1024 environments with minimum episode length as 32, which produces ∼15000 samples per second, and store them into a large cyclic data buffer of around 150000 samples. Both our policy model and neural dynamics model are grounded as vanilla MLPs. In our default setting, we update the neural dynamics model 8 times and then update the policy model 4 times each epoch. For full details of the hyper-parameters used please refer to Table. 2. Each of our experiments are conducted on a single NVIDIA A6000 GPU. But by decreasing the batchsize for a little bit, our framework can be trained on a single NVIDIA RTX 3090, which is more friendly to the demand for computation. Once trained, the policy runs at >30 FPS. Physics simulation is carried out in NVIDIA's Isaac Gym (Makoviychuk et al., 2021). The control policy is run at 30 Hz, while simulation runs at 60 Hz. For evaluation, we do not consider body shape variation and use the mean SMPL body shape.

Dataset&Metrics. We train our method on the training split of the AMASS (Mahmood et al., 2019) dataset. We follow PHC and remove sequences that are noisy or involve interactions of human objects, resulting in 11313 high-quality training sequences and 140 test sequences. To evaluate our policy fairly, we use the same metrics as in PHC. We report the success rate (Succ), deeming imitation unsuccessful when, at any point during imitation, the body joints are on average >$0.25m$ from the reference motion. Succ measures whether the humanoid can track the reference motion without losing balance or significantly lags behind. We also report the root-relative mean per-joint position error (MPJPE) E_{mpjpe} and global MPJPE $E_{g-mpjpe}$ (in mm), measuring the ability to imitate the reference motion both locally and globally. To show physical realism, we compare acceleration E_{acc} (mm/frames^2) and velocity E_{vel} (mm/frame) difference between simulated and MoCap motion.

4.1 EXPERIMENTS ON HUMANOID WHOLE-BODY CONTROL

4.1.1 MOTION TRACKING ON AMASS

Through the experiments in this subsection, we try to demonstrate the effectiveness of our framework on learning large scale motions. We compare with the SOTA motion imitator PHC (Luo et al., 2023a) and use the official implementation. Table. **??** reports our motion imitation results on the AMASS train and test dataset. Our method outperforms PHC on almost all metrics across training and test datasets, even within far less time, which is around 3 days while PHC takes a whole week. We achieve higher success rate, constantly much lower error on MPJPE, velocity and acceleration, which

Table 1: **Quantitative results on imitating MoCap motion sequences** (* indicates removing sequences containing human-object interaction). AMASS-Train* and AMASS-Test* contains 11313 and 140 high-quality MoCap sequences respectively. FT represents “future tracks".

	AMASS-Train*						AMASS-Test*					
Method	Succ↑	$E_{g-mpjpe}\downarrow$	$E_{mpjpe}\downarrow$	$E_{acc}\downarrow$	$E_{vel}\downarrow$	Time↓	Succ↑	$E_{g-mpjpe}\downarrow$	$E_{mpjpe}\downarrow$	$E_{acc}\downarrow$	$E_{vel}\downarrow$
PHC	98.9	37.5	26.9	3.3	4.9	7 days	96.4	47.4	30.9	6.8	9.1
PHC+	100	26.6	21.1	2.7	3.9	7+ days	99.2	36.1	24.1	6.2	8.1
PULSE	99.8	39.2	35.0	3.1	5.2	None(distillation)	97.1	54.1	43.5	7.0	10.3
Ours*	**99.2**	**27.7**	**15.5**	**2.1**	**2.7**	**3 days**	**98.5**	**32.1**	**23.9**	**5.5**	**7.1**
Ours(FT)*	99.0	20.3	16.1	1.9	2.6	3 days	98.1	27.0	20.2	5.0	6.4
PHC+*(env 3072)	98.4	30.9	23.6	2.4	3.6	2 days	97.8	30.5	23.5	5.6	8.0
Ours+*(env 1024)	98.8	16.9	13.3	2.3	2.7	2 days	97.9	26.5	19.5	5.6	6.5

reflects the advantage of supervised learning based framework. Also, we only use one single model in contrast to the multiplicative policy model in PHC. Besides, we implement “future tracks" on top of our method, which is shown in the last row of the main table. In “future tracks", the input state of policy model not only includes the next frame target, but also several future frame targets more, which means $s_t = (s_t^p, s_t^g, s_{t+\Delta t}^g, \cdots, s_{t+n\Delta t}^g)$. Specifically in our experiments, we set $\Delta t = 10$ frames and $n = 2$. As shown by the results, the performance gets improved by incorporating more future goals into the conditions. One of our explanation is that more informative gradients can back-propagate through time after we explicitly compute the losses within the future frames. And the policy model is optimized to take actions considering future goals. It is also worth noting that we run PHC with the default setting, in which the environment number is 1536, while ours is just 1024. Some qualitative results are shown in Fig. 3.

Sample Efficiency. We argue that another advantage of our framework is the superior sample efficiency, which is related to the required number of transition pairs from the physics simulator to achieve similar performance. We show the success rate curve comparison between PHC and our method below in Fig. 4. We expect our sample efficiency to be actually much higher because our method can exhibit similar performance with fewer environments as proved by our experiments in Fig. 6b. Although physics simulation is becoming faster with the advancement of modern GPU hardware, our model-based learning process mainly consumes the synthetic data by learning a world model to approximate the environment. We expect our method to be extended to field with scarce data like real-world robotics.

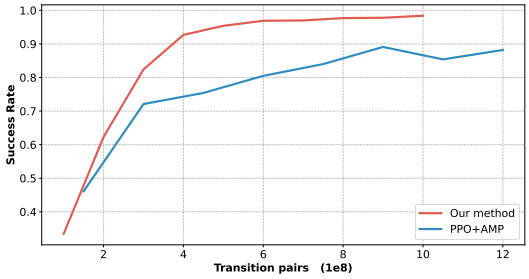


Figure 4: **Sample efficiency comparison.** It takes at least three times more samples for PHC to achieve 90% success rate than our method.

4.1.2 Generalizable on Sparse Reward Tasks

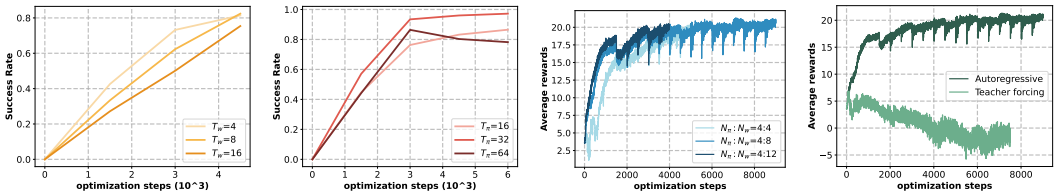
While our focus is on the task of whole-body motion tracking, we argue that our framework is also feasible for other sparse reward tasks, which can be proved theoretically as in Sec. 3.2. Following PULSE (Luo et al., 2023b), we study two popular downstream tasks – reaching a certain x-directional speed (between 0 m/s and 5m/s), and following a trajectory with obstacles around. For the task of speed, denote $\vec{\mathbf{v}}^*$ as the target velocity vector and h_0^* as the falling threshold of root height, the loss function is defined simply as:

$$\mathcal{L}_{speed} = w_v|\vec{\mathbf{v}}^* - \vec{\mathbf{v}}|^2 + w_h \max(h_0^* - h_0, 0) \tag{10}$$

For the task of trajectory following, given $p^* \in \mathbb{R}^2$ as the next target position on the $xy-$plane and p_0 as the xy coordinate of root, the loss function is defined as:

$$\mathcal{L}_{traj} = w_p|p^* - p_0|^2 + w_h \max(h_0^* - h_0, 0) \tag{11}$$

To make behaviors more human-like, we learn the policy based on PULSE. Another choice is incorporating AMP into training. We provide the qualitative results in the Fig. 3. What makes us excited is that prior model-based methods like SuperTrack (Fussell et al., 2021) rely on explicit trajectory following datasets while we do not. We also find that we can speed up training by utilizing the learned neural dynamics model from whole-body motion tracking.



(a) Different window size for dynamics model.

(b) Different window size for policy model.

(c) Different update ratio per epoch.

(d) Ablation on training manner.

Figure 5: **Ablation study on different settings of our framework.**

4.2 Ablation Study

In this section, we study the effects of different settings of our framework using the whole-body tracking task.

Window size. Our window-based training scheme allows the gradient back-propagate through time to learn long range predictions more stably than other single-step based method. For effective policy training, a larger window is necessary for learning motions that require long-term awareness. However, the policy's action quality over extended time horizons also depends on the dynamics model's ability to maintain accurate long-term predictions, which is hard to achieve if the window is too short for the dynamics model. On the other hand, if the dynamics model's training window is too large, the predicted states may diverge from target states, potentially leading the model to disregard the actions taken. Fig. 5a and 5b provide the effect of changing the relative training window size of the neural dynamics model and policy. We find that a policy window of 32 and dynamics model window of 8 are the most stable and efficient across all the experiments we tried.

Update ratio of policy and dynamics. Different from some previous model-based methods (Nagabandi et al., 2018; Georgiev et al., 2024) which collect massive offline data and train the dynamics model separately, our neural dynamics model is learned accompanied with the policy model. In each epoch, we update the neural dynamics for N_w times and policy for N_π times. As our performance depends on whether the neural dynamics model could offer correct gradients, we set N_w a little bit larger than N_π. Although averaged rewards increase faster with larger N_w, it might hurt the performance for two reasons. firstly, setting N_w too large will cause the training process inefficient. Actually training the dynamics model offline is equal to setting N_w as total training steps to some extent. Secondly, the dynamics model will be overfit on the current dynamics distribution in the data buffer, rather than bounding the model to learn only the subspace of possible dynamics – within the neighborhood of policy model. Ablation experiments are shown in the Fig. 5c, and our default setting is $\frac{N_w}{N_\pi} = 2$.

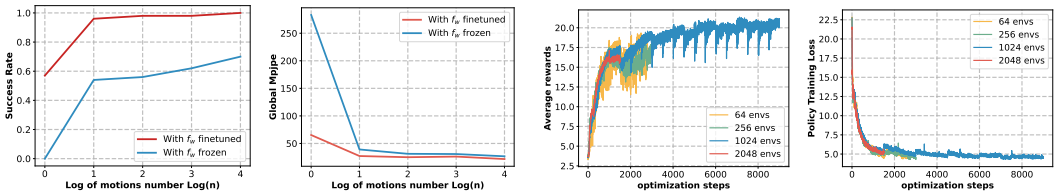
Autoregressive training. We want to emphasize that autoregressive training is so crucial for our framework that the performance could degrade severely if we use the popular teacher forcing instead, which is as shown in the Fig. 5d. We believe this is related to our continuous and random state/action spaces, which increases the challenge for stable open-loop dynamics prediction. while autoregressive training is an effective strategy to stabilize it.

4.3 A Glimpse into Properties of Neural Dynamics Model

Except for accelerating and improving the policy optimization process, we are interested in more wonderful properties of neural dynamics model. In this section, through the lens of motion tracking, we investigate into the scaling law and efficient transferability of the dynamics model.

4.3.1 Scaling Law on Policy Learning.

As scaling has been a key driver behind the rapid advancements across many fields in deep learning, we are pleased to have a glimpse on the scaling law of neural dynamics model. In this subsection, we simply explore the *"data"* dimension of scaling. In experiments shown by Fig. 6, we train the neural dynamics model with our framework on different number of motion clips in AMASS-Train*, then utilize this pre-trained dynamics model to learn a new policy on 100 randomly selected motions in AMASS-Test*. We can find that more diverse motion data will make the dynamics model more



(a) Scaling on motion diveristy. (b) Scaling on environment numbers.

Figure 6: **Scaling law on policy learning.**

efficient. In experiments shown by Fig. 6b, we train our policy and dynamics model from scratch with different number of simulated environments. We find that our method can perform similarly with much fewer environments, which proves the opinion about sample efficiency in Sec. 4.1.1.

4.3.2 Generalization & Transfer Learning

Unlike RL methods which can often struggle to generalize and transfer to new tasks and new environments (Bengio et al., 2020), the policy and dynamics model in our framework can easily generalize because the dynamics model explicitly models the environment dynamics.

Different Motion Dynamics. We test the transferablity of our policy and dynamics model by pretraining them on AMASS-Train* and then finetuning them on the popular H36M dataset (Ionescu et al., 2013). We use the subset *H36M-Motion** as splited in PHC (Luo et al., 2023a), which contains 140 high-quality MoCap sequences. We compare averaged rewards in the Fig. 7. Dynamics model transfer means we use the learned dynamics model to train a new policy from scratch while policy transfer means we finetune the policy and the dynamics model at the same time. We can find that a learned neural dynamics model could speed up the policy learning.

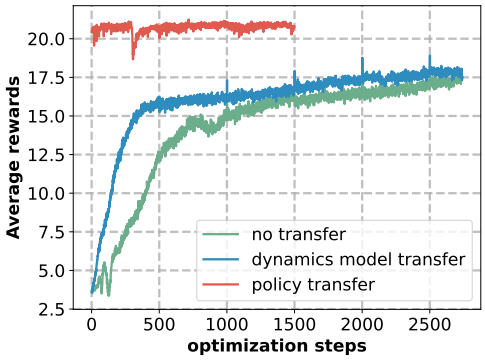


Figure 7: Transfer learning from AMASS to H36M.

4.3.3 Evaluating the Prediction of Dynamics Model.

At last, one important question to be explored is how is the prediction ability of the learned neural dynamics model. We argue that the neural dynamics model doesn't have the necessity to predict a whole long trajectory accurately, but being very useful as long as the prediction error is below the threshold within the policy optimization horizon. Therefore, we provide the open-loop prediction visualization of dynamics model in the Fig. 3. We can find that the neural dynamics model is stable and accurate within the horizon of policy training, though it still fails on longer prediction.

5 Discussions

Limitations. While our proposed framework can efficiently imitate human motion and expose well-scalability, it still does not achieve 100% success rate on AMASS. We find that some highly dynamic motions with large velocity and angle velocity are still challenging, because the dynamics distributions are very different from the rest. Besides, it is relatively harder for our method to discover the behavior such as stepping to prevent from falls than model-free RL, which might be mitigated by combing with more exploratory methods. Furthermore, the modeling of our neural dynamics model is not perfect as we did not consider the changes within the environment. So it is still hard for our model to learn in the highly discontinuous environments such as stairs and rough slopes.

Conclusion and Future Works. We introduce **SuperDyno**, a scalable and efficient framework, which is mainly designed for physics-based motion imitation, but also feasible for other goal-conditioned tasks. Our controller is much more efficient than previous methods, showing better performance, much lower tracking error with fewer samples, even within less time. Also the learned neural

dynamics can effectively serve for future prediction. Future directions include 1) incorporating terrain and scene awareness and updating the dynamics modeling to enable human-object interaction; 2) adapting the current framework to real robots; 3) combining with model-free methods to discover novel behaviors, *etc*.

REFERENCES

Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *International Conference on Machine Learning*, pp. 767–777. PMLR, 2020.

Xuxin Cheng, Yandong Ji, Junming Chen, Ruihan Yang, Ge Yang, and Xiaolong Wang. Expressive whole-body control for humanoid robots. *arXiv preprint arXiv:2402.16796*, 2024.

Nuttapong Chentanez, Matthias Müller, Miles Macklin, Viktor Makoviychuk, and Stefan Jeschke. Physics-based motion capture imitation with deep reinforcement learning. *Proceedings - MIG 2018: ACM SIGGRAPH Conference on Motion, Interaction, and Games*, 2018.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Levi Fussell, Kevin Bergamin, and Daniel Holden. Supertrack: Motion tracking for physically simulated characters using supervised learning. *ACM Transactions on Graphics (TOG)*, 40(6): 1–13, 2021.

Ignat Georgiev, Varun Giridhar, Nicklas Hansen, and Animesh Garg. Pwm: Policy learning with large world models. *arXiv preprint arXiv:2407.02466*, 2024.

Kehong Gong, Bingbing Li, Jianfeng Zhang, Tao Wang, Jing Huang, Michael Bi Mi, Jiashi Feng, and Xinchao Wang. PoseTriplet: Co-evolving 3D human pose estimation, imitation, and hallucination under self-supervision. *CVPR*, March 2022.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

Tairan He, Zhengyi Luo, Xialin He, Wenli Xiao, Chong Zhang, Weinan Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Omnih2o: Universal and dexterous human-to-humanoid whole-body teleoperation and learning. *arXiv preprint arXiv:2406.08858*, 2024a.

Tairan He, Zhengyi Luo, Wenli Xiao, Chong Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Learning human-to-humanoid real-time whole-body teleoperation. *arXiv preprint arXiv:2403.04436*, 2024b.

Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013.

Zhengyi Luo, Ryo Hachiuma, Ye Yuan, and Kris Kitani. Dynamics-regulated kinematic policy for egocentric pose estimation. *Advances in Neural Information Processing Systems*, 34:25019–25032, 2021.

Zhengyi Luo, Shun Iwase, Ye Yuan, and Kris Kitani. Embodied scene-aware human pose estimation. *Advances in Neural Information Processing Systems*, 35:6815–6828, 2022.

Zhengyi Luo, Jinkun Cao, Kris Kitani, Weipeng Xu, et al. Perpetual humanoid control for real-time simulated avatars. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10895–10904, 2023a.

Zhengyi Luo, Jinkun Cao, Josh Merel, Alexander Winkler, Jing Huang, Kris Kitani, and Weipeng Xu. Universal humanoid motion representations for physics-based control. *arXiv preprint arXiv:2310.04582*, 2023b.

Zhengyi Luo, Jinkun Cao, Sammy Christen, Alexander Winkler, Kris Kitani, and Weipeng Xu. Grasping diverse objects with simulated humanoids. *arXiv preprint arXiv:2407.11385*, 2024.

Naureen Mahmood, Nima Ghorbani, Nikolaus F Troje, Gerard Pons-Moll, and Michael J Black. Amass: Archive of motion capture as surface shapes. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5442–5451, 2019.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. Learning predict-and-simulate policies from unorganized human motion data. *ACM Transactions on Graphics (TOG)*, 38(6):1–11, 2019.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

Jiawei Ren, Cunjun Yu, Siwei Chen, Xiao Ma, Liang Pan, and Ziwei Liu. Diffmimic: Efficient motion mimicking with differentiable physics. 2023.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International conference on machine learning*, pp. 8583–8592. PMLR, 2020.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Alexander Winkler, Jungdam Won, and Yuting Ye. QuestSim: Human motion tracking from sparse sensors with simulated avatars. September 2022.

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. A scalable approach to control diverse behaviors for physically simulated characters. *ACM Trans. Graph.*, 39(4), 2020.

Jungdam Won, Deepak Gopinath, and Jessica Hodgins. Physics-based character controllers using conditional vaes. *ACM Transactions on Graphics (TOG)*, 41(4):1–12, 2022.

Heyuan Yao, Zhenhua Song, Baoquan Chen, and Libin Liu. Controlvae: Model-based learning of generative controllers for physics-based characters. *ACM Transactions on Graphics (TOG)*, 41(6): 1–16, 2022.

Ye Yuan and Kris Kitani. Residual force control for agile human behavior imitation and extended motion synthesis. (NeurIPS), June 2020a.

Ye Yuan and Kris Kitani. Residual force control for agile human behavior imitation and extended motion synthesis. *Advances in Neural Information Processing Systems*, 33:21763–21774, 2020b.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5745–5753, 2019.

A APPENDIX

A.1 OTHER REPRESENTATIONS DETAILS

State. We follow the same setting in PHC. The simulation state $s_t \triangleq (s_t^p, s_t^g)$ consists of humanoid proprioception s_t^p and goal state s_t^g. Proprioception $s_t^p \triangleq (q_t, \dot{q}_t, \beta)$ contains the 3D body pose q_t, velocity $\dot{q}_t$, and (optionally) body shape β. When trained with different body shape, β contains information about the length of the limb of each body link. Our goal state s_t^g is defined as the difference between the next step reference quantitives and their simulated counterpart:

$$s_t^g \triangleq (\hat{\theta}_{t+1} \ominus \theta_t, \hat{p}_{t+1} - p_t, \hat{v}_{t+1} - v_t, \hat{w}_t - w_t, \hat{\theta}_{t+1}, \hat{p}_{t+1})$$

Before being input to the policy network, s_t^g and s_t^p are normalized with respect to the humanoid's current facing direction and root position (Won et al., 2022; Luo et al., 2021).

Loss Function. For the neural dynamics model, we share a common loss function to train the dynamics, which is set as:

$$\mathcal{L}_w = \sum_{t=0}^{T_w} (w_{pos}\|\tilde{p}_t - p_t\| + w_{vel}\|\tilde{q}_t \ominus q_t\| + w_{vel}\|\tilde{v}_t - v_t\| + w_{ang}\|\tilde{w}_t - w_t\|)$$

The weights $w_{pos}, w_{vel}, w_{rot}, w_{ang}$for different physics variables are tweaked to give roughly equal contribution from all losses at the beginning of training and $\ominus$ represents quaternion difference.

For the policy optimization, our loss function is defined similar to the neural dynamics model. But the difference is that our loss is computed in the local frame, which means the states are transformed into local observation:

$$\mathcal{L}_\pi = \sum_{t=0}^{T_\pi} (w_{lpos}\|\hat{p}'_t - p'_t\| + w_{lvel}\|\hat{q}'_t \ominus q'_t\| + w_{lvel}\|\hat{v}'_t - v'_t\| + w_{lang}\|\hat{w}'_t - w'_t\| + w_{act}\|a_t\| + w_{height}\|\hat{h_0} - h_0\|)$$

Also we set the weights $w_{pos}, w_{vel}, w_{rot}, w_{ang}, w_{height}$ to balance the loss of different parts, and set w_{act} smaller by two orders of magitude.

Action. Following PHC, we use a proportional derivative (PD) controller at each DoF of the humanoid and the action a_t specifies the PD target. With the target joint set as $q_t^d = a_t$, the torque applied at each joint is $\tau^i = k^p \circ (a_t - q_t) - k^d \circ \dot{q}_t$. This is different from the residual action representation (Yuan & Kitani, 2020b; Park et al., 2019; Luo et al., 2021) used in prior motion imitation methods, where the action is added to the reference pose:$q_t^d = \hat{q}_t + a_t$ to speed up training. We do not use any external forces or meta-PD control.

Humanoid. Our humanoid controller can support any human kinematic structure, and we use the SMPL kinematic structure. The SMPL body contains 24 rigid bodies, of which 23 are actuated, resulting in an action space of $a_t \in \mathbb{R}^{23\times 3}$. The body proportion can very based on a body shape parameter $\beta \in \mathbb{R}^{10}$.

A.2 HYPER-PARAMETERS IN THE EXPERIMENTS

Table 2: Hyper-parameter settings used in our framework.

Trajectory Collection	Minimum Episode Length	36
	Termination Distance (m)	0.5
	Maximum Episode Length	512
	Action Noise Scale	0.05
Neural Dynamics Model	Hidden Layers	2
	Hidden Units	512
	Activation	SiLU
	Batchsize	4096
	Learning Rate	2e-3
	Window T_w	8
Policy Model	Hidden Layers	2
	Hidden Units	[1024,512]
	Activation	SiLU
	Batchsize	4096
	Learning Rate	1e-5
	Window T_π	32