

APPENDIX

A. Domain Randomization in Pre-Training

To improve the robustness and generalization of the pre-trained policy in Figure 2 (a), we utilized the domain randomization technics listed in Table VI.

TABLE VI
DOMAIN RANDOMIZATIONS

Term	Value
Dynamics Randomization	
Friction	$\mathcal{U}(0.2, 1.1)$
P Gain	$\mathcal{U}(0.925, 1.05) \times \text{default}$
Control delay	$\mathcal{U}(20, 40)\text{ms}$
External Perturbation	
Push robot	interval = 10s, $v_{xy} = 0.5$ m/s

B. SysID Parameters

We identify the following representative robot parameters in our simulated model that best align the ones in the real world: base center of mass (CoM) shift (c_x, c_y, c_z) , base link mass offset ratio k_m and low-level PD gain ratios (k_p^i, k_d^i) where $i = 1, 2, \dots, 23$, as shown in Table VII.

TABLE VII
SYSID PARAMETERS

Parameter	Range	Parameter	Range
c_x	$[-0.02, 0.0, 0.02]$	c_y	$[-0.02, 0.0, 0.02]$
c_z	$[-0.02, 0.0, 0.02]$	k_m	$[0.95, 1.0, 1.05]$
k_p^i	$[0.95, 1.0, 1.05]$	k_d^i	$[0.95, 1.0, 1.05]$

C. Implementation of Delta Dynamics Learning

Using the collected real-world trajectory, we replay the action sequence $\{a_0^{\text{real}}, \dots, a_T^{\text{real}}\}$ in simulation and record the resulting trajectory $\{s_0^{\text{sim}}, \dots, s_T^{\text{sim}}\}$. The neural dynamics model f_θ^Δ is trained to predict the difference:

$$s_{t+1}^{\text{real}} - s_{t+1}^{\text{sim}} = f_\theta^\Delta(s_t^{\text{real}}, a_t^{\text{real}}), \quad \forall t.$$

In practice, we compute the mean squared error (MSE) loss in an autoregressive setting, where the model predicts forward for K steps and uses gradient descent to minimize the loss. To balance learning efficiency and stability over long horizons, we implement a schedule that gradually increases K during training. Formally, the optimization objective is:

$$\mathcal{L} = \left\| s_{t+K}^{\text{real}} - \underbrace{f^{\text{sim}}(\dots f^{\text{sim}}(s_t, a_t) + f_\theta^\Delta(s_t, a_t), \dots, a_{t+K})}_{K} \right\|.$$

After training, we freeze the residual dynamics model f_θ^Δ and integrate it into the simulator. During each simulation step, the robot's state is updated by incorporating the delta predicted by the dynamics model. In this augmented simulation environment, we finetune the previously pretrained policy to adapt to the corrected dynamics, ensuring improved alignment with real-world behavior.

D. Derivation of Training-free Methods of Using Delta Action

To formalize the problem, we start by assuming one-step consistency between real and simulated dynamics:

$$f^{\text{real}}(s, \pi(s)) = f^{\text{sim}}(s, \pi(s) + \pi^\Delta(s, \pi(s))).$$

Under this assumption, one-step matching leads to the condition:

$$\pi(s) + \pi^\Delta(s, \pi(s)) = \hat{\pi}(s), \quad (1)$$

$$\Rightarrow \pi(s) = \hat{\pi}(s) - \pi^\Delta(s, \pi(s)). \quad (2)$$

To solve Equation (2), we consider:

- 1) **Fixed-Point Iteration:** We initialize $y_0 = \hat{\pi}(s)$ and iteratively update:

$$y_{k+1} = \hat{\pi}(s) - \pi^\Delta(s, y_k), \quad (3)$$

where y_k converges to a solution after K iterations.

- 2) **Gradient-Based Optimization:** Define the loss function:

$$l(y) = \|y + \pi^\Delta(s, y) - \hat{\pi}(s)\|^2. \quad (4)$$

A gradient descent method minimizes this loss to solve for y .

These methods approximate $\pi(s)$, but suffer from OOD issues when trained on limited trajectories. RL fine-tuning, in contrast, directly optimizes $\pi(s)$ for real-world deployment, resulting in superior performance.

Problem of One-Step Matching. Note that Equation (2) is derived from the one-step matching assumption (i.e., $\pi(s) + \pi^\Delta(s, \pi(s)) = \hat{\pi}(s)$). For multi-step matching, one has to differentiate through f^{sim} , which is, in general, intractable. Therefore, both fixed-point iteration and gradient-based optimization assume one-step matching. This also explains the advantages of RL-based fine-tuning: it effectively performs a gradient-free multi-step matching procedure.