
Technical Appendix:

Revisiting Multi-Agent World Modeling from a Diffusion-Inspired Perspective

Anonymous Author(s)

Affiliation

Address

email

1 A Proof of Theorem 2

2 *Proof.* Given the log-likelihood of the global state transition, we have the following:

$$\begin{aligned}
 \log P(s_{t+1}|s_t, a_t^{1:N}) &= \log \int p(s_{t+1}, s_{t+1}^{(1):(n)}|s_t, a_t^{1:n}) ds_{t+1}^{(1):(n)} \\
 &= \log \int \frac{p(s_{t+1}, s_{t+1}^{(1):(n)}|s_t, a_t^{(1):(n)}) \hat{q}(s_{t+1}^{(1):(n)}|s_t, a_t^{1:n}, s_{t+1})}{\hat{q}(s_{t+1}^{(1):(n)}|s_t, a_t^{1:n}, s_{t+1})} ds_{t+1}^{(1):(n)} \\
 &= \log \mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_t, a_t^{1:n}, s_{t+1})} \left[\frac{p(s_{t+1}, s_{t+1}^{(1):(n)}|s_t, a_t^{1:n})}{\hat{q}(s_{t+1}^{(1):(n)}|s_t, a_t^{1:n}, s_{t+1})} \right] \\
 &\geq \mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_t, a_t^{1:n}, s_{t+1})} \left[\log \frac{p(s_{t+1}, s_{t+1}^{(1):(n)}|s_t, a_t^{1:n})}{\hat{q}(s_{t+1}^{(1):(n)}|s_t, a_t^{1:n}, s_{t+1})} \right], \tag{11}
 \end{aligned}$$

3 where the last inequality results from Jensen's inequality. Under the definition and property of the
 4 conditional Markovian forward diffusion process \hat{q} in Eqs. (2)–(5) and Assumption 1, we can rewrite
 5 Eq. (11) as follows,

$$\log P(s_{t+1}|s_t, a_t^{1:N}) \geq \mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_{t+1})} \left[\log \frac{p(s_{t+1}^{(n)}) \prod_{k=1}^n p(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_t, a_t^k)}{\prod_{k=1}^n \hat{q}(s_{t+1}^{(k)}|s_{t+1}^{(k-1)})} \right], \tag{12}$$

6 where we denote $s_{t+1} := s_{t+1}^{(0)}$. Then, RHS of Eq. (12) can be further simplified,

$$\begin{aligned}
 &\mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_{t+1})} \left[\log \frac{p(s_{t+1}^{(n)}) \prod_{k=1}^n p(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_t, a_t^k)}{\prod_{k=1}^n \hat{q}(s_{t+1}^{(k)}|s_{t+1}^{(k-1)})} \right] \\
 &= \mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_{t+1})} \left[\log \frac{p(s_{t+1}^{(n)}) p(s_{t+1}^{(0)}|s_{t+1}^{(1)}, s_t, a_t^1)}{\hat{q}(s_{t+1}^{(1)}|s_{t+1}^{(0)})} + \log \prod_{k=2}^n \frac{p(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_t, a_t^k)}{\hat{q}(s_{t+1}^{(k)}|s_{t+1}^{(k-1)})} \right] \\
 &= \mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_{t+1})} \left[\log \frac{p(s_{t+1}^{(n)}) p(s_{t+1}^{(0)}|s_{t+1}^{(1)}, s_t, a_t^1)}{\hat{q}(s_{t+1}^{(1)}|s_{t+1}^{(0)})} + \log \prod_{k=2}^n \frac{p(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_t, a_t^k)}{\frac{\hat{q}(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_{t+1}^{(0)}) \hat{q}(s_{t+1}^{(k)}|s_{t+1}^{(0)})}{\hat{q}(s_{t+1}^{(k-1)}|s_{t+1}^{(0)})}} \right] \\
 &= \mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_{t+1})} \left[\log \frac{p(s_{t+1}^{(n)}) p(s_{t+1}^{(0)}|s_{t+1}^{(1)}, s_t, a_t^1)}{\hat{q}(s_{t+1}^{(1)}|s_{t+1}^{(0)})} + \log \frac{\hat{q}(s_{t+1}^{(1)}|s_{t+1}^{(0)})}{\hat{q}(s_{t+1}^{(1)}|s_{t+1}^{(0)})} + \log \prod_{k=2}^n \frac{p(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_t, a_t^k)}{\hat{q}(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_{t+1}^{(0)})} \right] \\
 &= \mathbb{E}_{\hat{q}(s_{t+1}^{(1):(n)}|s_{t+1})} \left[\log \frac{p(s_{t+1}^{(n)}) p(s_{t+1}^{(0)}|s_{t+1}^{(1)}, s_t, a_t^1)}{\hat{q}(s_{t+1}^{(1)}|s_{t+1}^{(0)})} + \sum_{k=2}^n \log \frac{p(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_t, a_t^k)}{\hat{q}(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_{t+1}^{(0)})} \right]
 \end{aligned}$$

Therefore, the evidence of dynamics transition can be bounded as follows:

$$\begin{aligned} \log P(s_{t+1}|s_t, a_t^{1:N}) &\geq \mathbb{E}_{\hat{q}(s_{t+1}^{(1)}|s_{t+1}^{(0)})} [\log p(s_{t+1}^{(0)}|s_{t+1}^{(1)}, s_t, a_t^1)] - \text{D}_{\text{KL}}[\hat{q}(s_{t+1}^{(n)}|s_{t+1}^{(0)})||p(s_{t+1}^n)] \\ &\quad - \sum_{k=2}^n \mathbb{E}_{\hat{q}(s_{t+1}^{(k)}|s_{t+1}^{(0)})} \left[\text{D}_{\text{KL}}(\hat{q}(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, s_{t+1}^{(0)})||p(s_{t+1}^{(k-1)}|s_{t+1}^{(k)}, a_t^k, s_t)) \right]. \end{aligned} \quad (13)$$

As shown by [25], the conditional forward diffusion process \hat{q} behaves identically to the unconditional one q . Therefore, we can substitute the \hat{q} with the q in Eq. (13), concluding our proof. \square

B EDM Preconditioners and Noise Scheduler

To keep input and output signal magnitudes fixed to the same scale and avoid large variance in gradient magnitudes on a per-sample basis, Karras et al. [21] introduced the following preconditioners for normalization and re-scaling output to stabilize and improve the training dynamics of the network:

$$c_{\text{in}}^\tau = \frac{1}{\sqrt{\sigma(\tau)^2 + \sigma_{\text{data}}^2}} \quad (14)$$

$$c_{\text{out}}^\tau = \frac{\sigma(\tau)\sigma_{\text{data}}}{\sqrt{\sigma(\tau)^2 + \sigma_{\text{data}}^2}} \quad (15)$$

$$c_{\text{noise}}^\tau = \frac{1}{4} \log(\sigma(\tau)) \quad (16)$$

$$c_{\text{skip}}^\tau = \frac{\sigma_{\text{data}}^2}{\sigma_{\text{data}}^2 + \sigma(\tau)^2}, \quad (17)$$

where $\sigma_{\text{data}} = 0.5$ in our experiment hyperparameter setup. The noise scheduler for training the diffusion model follows the same design in [21], described as follows:

$$\sigma(\tau) = \tau, \log(\sigma) \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2), \quad (18)$$

where $P_{\text{mean}} = -0.4$ and $P_{\text{std}} = 1.2$.

C Behavior Learning Details

Inspired by the success of MARIE [14], we adopt MAPPO [34] to train both the actor and critic inside the imaginations of DIMA. A key distinction from MARIE is that our model explicitly predicts the global state, enabling seamless integration with CTDE techniques as well as actor-critic architectures commonly used in model-free MARL. Therefore, we implement both the actor ψ and critic ξ with two 3-layer MLPs together with ReLU activation and Layer Normalization, respectively. Similar to off-the-shelf CTDE model-free MARL algorithms, we adopt actor parameter sharing across agents.

Critic loss function. We utilize λ -return in DreamerV1 [1], which employs an exponentially weighted average of different k -steps TD targets to balance bias and variance as the regression target for the critic. Given an imagined trajectory $\{\hat{s}_t, \hat{o}_t^{1:n}, a_t^{1:n}, \hat{r}_t, \hat{\gamma}_t\}_{t=1}^H$ over all agents, λ -return is calculated recursively as,

$$V_\lambda(\hat{s}_t) = \hat{r}_t^i + \hat{\gamma}_t \cdot \begin{cases} (1 - \lambda)V_\xi(\hat{s}_t) + \lambda V_\lambda(\hat{s}_{t+1}) & \text{if } t < H \\ V_\xi(\hat{s}_t) & \text{if } t = H \end{cases} \quad (19)$$

The objective of the critic ξ is to minimize the mean squared difference \mathcal{L}_ξ with λ -returns over imagined trajectories, as

$$\mathcal{L}_\xi = \mathbb{E}_{\pi_\psi} \left[\sum_{t=1}^{H-1} \left(V_\xi(\hat{s}_t) - \text{sg}(V_\lambda(\hat{s}_t)) \right)^2 \right], \quad (20)$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operation. We optimize the critic loss with respect to the critic parameters ξ using the Adam optimizer.

Table 1: Behaviour learning hyperparameters.

Hyperparameter	Value
Common	
Imagination horizon (H)	15
λ	0.95
Clipping parameter ϵ	0.1
MAMuJoCo	
Discount factor γ	0.99
η	0.001
Bi-DexHands	
Discount factor γ	0.95
η	0.01

Actor loss function. The objective for the actor $\pi_{\psi}^i(\cdot|\hat{o}_t^i) := \pi_{\psi}(\cdot|\hat{o}_t^i)$ is to output actions that maximize the prediction of long-term future rewards made by the critic. To incorporate intermediate rewards more directly, we train the actor to maximize the same λ -return that was computed for training the critic. In terms of the non-stationarity issue in multi-agent scenarios, we adopt PPO updates, which introduce importance sampling for actor learning. The actor loss function for agent i is:

$$\mathcal{L}_{\psi}^i = -\mathbb{E}_{\pi_{\psi_{\text{old}}^i}} \left[\sum_{t=0}^{H-1} \min \left(r_t^i(\psi) A_t, \text{clip}(r_t^i(\psi), 1 - \epsilon, 1 + \epsilon) A_t \right) + \eta \mathcal{H}(\pi_{\psi}^i(\cdot|\hat{o}_t^i)) \right] \quad (21)$$

where $r_t^i(\psi) = \pi_{\psi}^i / \pi_{\psi_{\text{old}}^i}$ is the policy ratio and $A_t = \text{sg}(V_{\lambda}(\hat{s}_t) - V_{\xi}(\hat{s}_t))$ is the advantage. Unlike MAPPO, we choose not to design agent-specific global states, as such designs are overly hand-crafted and inject task-specific human priors, which undermines the generality and soundness of the approach. Instead, we retain the environment’s original agent-agnostic global state shared among all agents, and feed it into the value function V_{ξ} . As a result, the estimated advantage function A_t is also shared across all agents during actor updates. We optimize the actor loss with respect to the actor parameters ψ using the Adam optimizer. Overall hyperparameters are shown in Table 1.

D Illustrations of Experimental Environments

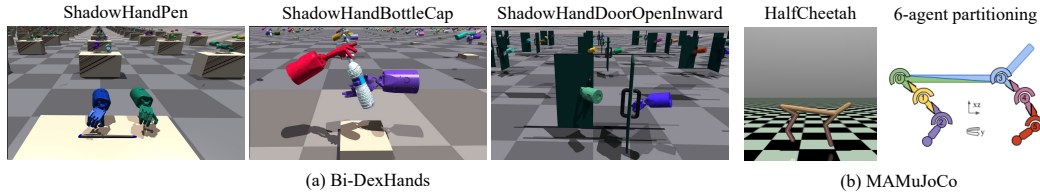


Figure 9: Illustrations of the experimental environments in our work. **Left:** Visualizations of three Bi-DexHands tasks: removing a pen cap, opening a bottle cap, and opening a door inwards. **Right:** Visualization of 6-agent partitioning w.r.t. HalfCheetah in MAMuJoCo.

E Additional Results

E.1 Final Returns of All Methods on MAMuJoCo and Bi-DexHands

Table 2: **Comparison of final episode returns across MAMuJoCo and Bi-DexHands benchmarks.** We report the mean final episode return and standard deviation over 4 random seeds. DIMA consistently outperforms all baselines across all chosen tasks on both MAMuJoCo and Bi-DexHands. The best result per task is highlighted in bold and shaded in blue color, while the second-best is underlined.

Tasks	Steps	Methods					
		DIMA (Ours)	MARIE	MAMBA	HASAC	HAPPO	MAPPO
<i>MAMuJoCo</i>							
Ant-2x4	1M	4881 \pm 756	4471 \pm 553	1314 \pm 756	1344 \pm 282	1716 \pm 449	859 \pm 47
Ant-4x2		4766 \pm 450	1173 \pm 136	1618 \pm 931	850 \pm 126	1917 \pm 253	854 \pm 41
HalfCheetah-2x3		6370 \pm 121	4045 \pm 275	2813 \pm 1580	2499 \pm 1081	2628 \pm 893	3196 \pm 75
HalfCheetah-3x2		6175 \pm 212	2380 \pm 1145	3029 \pm 798	2872 \pm 890	3402 \pm 317	2936 \pm 766
HalfCheetah-6x1		5643 \pm 163	1738 \pm 1213	1848 \pm 220	2044 \pm 110	2939 \pm 1113	925 \pm 121
Walker2d-2x3		3329 \pm 1056	2822 \pm 997	124 \pm 19	1135 \pm 210	1007 \pm 282	752 \pm 216
Walker2d-3x2		4084 \pm 357	604 \pm 349	466 \pm 103	958 \pm 715	932 \pm 513	1004 \pm 480
<i>Bi-DexHands</i>							
BottleCap	300K	259.9 \pm 4.1	-	203.8 \pm 5.2	210.9 \pm 6.1	100.7 \pm 3.8	104.0 \pm 2.3
DoorOpenInward		290.4 \pm 29.0	-	225.0 \pm 79.4	246.3 \pm 7.0	30.7 \pm 2.5	65.8 \pm 6.9
DoorOpenOutward		367.1 \pm 19.4	-	177.4 \pm 43.1	221.9 \pm 7.3	58.8 \pm 4.6	96.4 \pm 8.5
BottleCap		24.4 \pm 11.4	-	4.3 \pm 0.4	0.0 \pm 0.0	0.0 \pm 0.0	0.0 \pm 0.0

E.2 Experiment Details of Imagination Evaluation across Different Conditioning Orders

To evaluate DIMA’s imagination under different conditioning orders on the 2-agent Ant [2x4] task, we collect 10 episodes by using the final policy induced by our algorithm, and randomly sample 100 trajectory segments to form our trajectory segment dataset. For each segment, we generate imagined rollouts using DIMA with different action conditioning orders.

As the EDM framework decouples inference-time sampling from training, the number of denoising steps need not match the number of agents. Thus, we set the number of denoising steps equal to 4, i.e., twice the number of agents. Letting the agent set be $\{1, 2\}$, we consider three conditioning orders: (i) random order: (2, 1, 1, 2), (ii) ascending order w.r.t. agent id: (1, 1, 2, 2), and (iii) descending order w.r.t. agent id: (2, 2, 1, 1).

To provide a quantitative evaluation in Figure 7 (right), we compute the L1 error per observation dimension at each timestep between the 100 sampled trajectory segments and their corresponding imagined rollouts, and accumulate the errors over the prediction horizon. All observation L1 errors are averaged across 2 agents.

62 F Overview of DIMA with Learning in Imaginations

63 We summarize the overall training procedure of DIMA paired with learning in imaginations in
 64 Algorithm 1 below. We denote as \mathcal{D} the replay databuffer which stores data collected from the real
 65 environment.

66 G Training Details and Hyperparameters

67 G.1 Model Architecture Details

68 **State decoder.** To enable decentralized execution of policies trained within DIMA’s imagination
 69 rollouts, each agent must make decisions based solely on its local observation rather than the shared
 70 global state. To support such policy learning, we introduce a necessary state decoder that maps the
 71 global state s_t into the corresponding joint local observations $o_t^{1:n}$.

72 Due to our online model-based MARL setup, the state decoder must be continually updated under
 73 a non-stationary data distribution which also shifts continually. Using a vanilla MLP as the state
 74 decoder in this setting may lead to issues such as overfitting or mode collapse. To mitigate these risks,
 75 we incorporate additional regularization into the decoder design by adopting a Vector Quantized
 76 Variational Autoencoder (VQ-VAE) [30], which enforces a compact latent codebook representation
 77 via vector quantization. Among various VQ-VAE variants, we choose Finite Scalar Quantization
 78 (FSQ) [31] as our final implementation as it removes any auxiliary losses and achieves remarkably
 79 high codebook utilization, which indicates its strong and effective regularization.

80 Our implementation is based on the open-source repository: [https://github.com/lucidrains/](https://github.com/lucidrains/vector-quantize-pytorch)
 81 `vector-quantize-pytorch`. We simply build the encoder E_φ and decoder D_φ as MLPs to deal
 82 with continuous non-vision global states and joint local observations. The decoder is designed with
 83 the same hyperparameters as the encoder. The loss function for learning the autoencoder is as follows:

$$\mathcal{L}_{\text{FSQ}}(E_\varphi, D_\varphi) = \mathbb{E}_{(s_t, o_t^{1:n}) \sim \mathcal{D}} [\|o_t^{1:n} - D_\varphi(E_\varphi(s_t) + \text{sg}(\text{round}(f(E_\varphi(s_t)))) - E_\varphi(s_t))\|^2], \quad (22)$$

84 where f is a bounding function such that i -th channel/entry in $\hat{z}_t = \text{round}(f(E_\varphi(s_t)))$ takes one of
 85 L_i unique values (here $f : z \rightarrow \lfloor L_i/2 \rfloor \tanh(z)$ for i -th channel/entry) and round is the operation
 86 to map real-valued inputs to the nearest integers. Therefore, we have an implicit codebook \mathcal{C} with
 87 $|\mathcal{C}| = \prod_{i=1}^d L_i$. After training the VAE, our state decoder can be expressed by $g_\varphi(o_t^{1:n} | s_t) =$
 88 $D_\varphi(\text{round}(f(E_\varphi(s_t))))$. The hyperparameters are listed in Table 3.

89 **Diffusion model for dynamics modeling.** We use the 1-D variant adapted from the U-Net 2D
 90 in DIAMOND [18] as the backbone of the diffusion model D_θ . To predict the next state s_{t+1} , the
 91 diffusion model D_θ is initially conditioned on the current global s_t , joint action $a_t^{1:n}$ and the diffusion
 92 time τ . To improve next-global-state prediction accuracy, we empirically augment the temporal
 93 context by additionally incorporating the last 2 global states and joint actions, extending it from s_t
 94 and $a_t^{1:n}$ to $s_{t-2:t}$ and $a_{t-2:t}^{1:n}$. Note that the effect of sequential denoising is confined to the joint
 95 action $a_t^{1:n}$ conditioning at the current timestep t , and does not extend to the past joint actions.

96 Inspired by the success of DIAMOND, we directly adopt the same conditioning mechanism in
 97 DIAMOND, and use temporal stacking for global state conditioning and adaptive group normalization
 98 for joint action and diffusion time conditioning. The hyperparameters are listed in Table 3.

99 **Transformer as reward and termination model.** The Transformer for predicting the reward and
 100 termination is built upon the implementation of minGPT [29]. Given a fixed imagination horizon H ,
 101 it first takes a sequence of length $2H$ composed of global states and joint actions $(\dots, s_t, a_t^{1:n}, \dots)$,
 102 and encodes every single global state and joint action into d_e -dimensional embedding tensor via
 103 2 separate encoding functions. Then the sequence tensor of shape $2H \times d_e$ is forwarded through
 104 fixed Transformer blocks. Finally, the Transformer predicts reward and termination via two separate
 105 3-layer MLP heads $f_\phi(r_t | s_{\leq t}, a_{\leq t}^{1:n})$ and $f_\phi(\gamma_t | s_{\leq t}, a_{\leq t}^{1:n})$, respectively. In general, the loss function
 106 is described by

$$\mathcal{L}_\phi = \mathbb{E} \left[\sum_{t=1}^H -\log f_\phi(r_t | s_{\leq t}, a_{\leq t}^{1:n}) - \log f_\phi(\gamma_t | s_{\leq t}, a_{\leq t}^{1:n}) \right]. \quad (23)$$

107 But in practice, we optimize the reward prediction with a smooth L1 loss function and the termination
108 prediction with a cross-entropy loss function. The hyperparameters are listed in Table 3.

109 **G.2 Computational Resources Used for Training**

110 All our experiments including the evaluation of chosen baselines are run on a machine with a single
111 NVIDIA RTX 4090 GPU, a 24-core CPU, and 256GB RAM.

112 **G.3 Baseline Implementation Details**

113 In our experiments, we reran and evaluated all baseline methods. To ensure fairness for comparisons,
114 we followed the optimal hyperparameters provided in their official implementations, listed below:

- 115 • MARIE: <https://github.com/breez3young/MARIE>;
- 116 • MAMBA: <https://github.com/jbr-ai-labs/mamba>;
- 117 • HASAC, HAPPO and MAPPO: <https://github.com/PKU-MARL/HARL>.

118 **G.4 DIMA hyperparameters**

119 We list the hyperparameters of DIMA paired with learning in imaginations in Table 4.

120 **H Broader Impact**

121 Our work introduces DIMA, a diffusion-inspired multi-agent world model that significantly improves
122 sample efficiency in cooperative multi-agent control environments. By enabling more faithful
123 imagined rollouts, DIMA can accelerate the development of complex autonomous systems—such
124 as multiple real robots coordination, traffic management, and energy-efficient buildings—thereby
125 reducing real-world trial costs. However, these capabilities also carry potential risks: misuse of high-
126 fidelity neural simulators for adversarial planning could exacerbate privacy and security concerns.

Algorithm 1: DIMA paired with learning in imaginations

Procedure training_loop():

```
for epochs do
    collect_experience(steps_collect)
    for steps_state_decoder do
        update_state_decoder()
    for steps_diffusion_model do
        update_diffusion_model()
    for steps_reward_end_model do
        update_reward_end_model()
    for steps_actor_critic do
        update_actor_critic()
```

Procedure collect_experience(n):

```
s0, o01:n ← env.reset()
for t = 0 to n - 1 do
    Sample ati ∼ πψi(ati|oti), ∀ agent i
    st+1, ot+11:n, rt, γt ← env.step(at1:n)
    D ← D ∪ {st+1, ot+11:n, at1:n, rt, γt}
    if γt = 1 then
        st+1, ot+11:n ← env.reset()
```

Procedure update_state_decoder():

```
Sample state-observation pair (st, ot1:n) ∼ D
Compute LFSQ in Eq. (22)
Update State Decoder gφ
```

Procedure update_diffusion_model():

```
Sample sequence (st-L+1, at-L+11:n, ..., st, at1:n, st+1) ∼ D
Sample log(σ) ∼ N(Pmean, Pstd2) and get τ = σ since σ(τ) := τ
Sample st+1τ ∼ N(xt+10, σ2I)
Sample a chosen agent id k ∼ Uniform{1, 2, ..., n}
Compute st+1(0) = Dθ(st+1τ; τ, atk, st-L+1:t, at-L+1:t-11:n)
                                     extra temporal context
Compute loss L(θ) = ||st+1(0) - st+1||2 in Eq. (9)
Update Diffusion Model Dθ
```

Procedure update_reward_end_model():

```
Sample sequence (st, at1:n, rt, γt, ..., st+H-1, at+H-11:n, rt+H-1, γt+H-1) ∼ D
for i = t to t + H - 1 do
    Compute r̂i ∼ fφ(r̂i|s≤i, a≤i1:n) and γ̂i ∼ fφ(γ̂i|s≤i, a≤i1:n)
Compute Lφ = ∑i=tt+H-1 CrossEntropy(γ̂i, γi) + SmoothL1(r̂i, ri) corresponding to Eq. (23)
Update Reward and Termination Model fφ
```

Procedure update_actor_critic():

```
Set the joint action condition order ρ = (i1, i2, ..., in)
Sample starting point (st-L+1, ot-L+11:n, at-L+11:n, ..., st, ot1:n) ∼ D of the imagination
Let ôt1:n = ot1:n
for i = t to t + H - 1 do
    Sample aij ∼ πψj(aij|ôij) ∀ agent j
    Sample the reward r̂i and the termination γ̂i with fφ
    Sample the next global state ŝt+1 by iteratively denoising with Dθ and ρ
    Sample the next joint observation state ôt+11:n with gφ
Update actor πψi and critic Vξ via Lξ and Lψi over imaginations {ŝi, ôi1:n, ai1:n, r̂i, γ̂i}i=tt+H-1
```

Table 3: Architecture details.

Hyperparameter	Value
State Decoder (g_φ)	
MLP layers	3
Hidden size	512
Activation	GELU [69]
FSQ Levels L_i	[8, 6, 5]
Diffusion Model (D_θ)	
Global state conditioning mechanism	Temporal stacking
Joint action conditioning mechanism	Adaptive Group Normalization
Diffusion time conditioning mechanism	Adaptive Group Normalization
Residual blocks layers	[2, 2, 2]
Residual blocks channels	[64, 64, 64]
Residual blocks conditioning dimension	256
Reward and Termination Model (f_ϕ)	
Embedding dimension d_e	256
Transformer block layers	6
Attention heads	4
Embedding dropout	0.1
Attention dropout	0.1
Residual dropout	0.1

Table 4: Hyperparameters for DIMA.

Hyperparameter	Value
Batch size for State Decoder training	256
Batch size for Diffusion Model training	64
Batch size for Reward and Termination Model training	128
Optimizer for State Decoder	AdamW
Optimizer for Diffusion Model	AdamW
Optimizer for Reward and Termination Model	AdamW
Optimizer for Actor & critic	Adam
Learning rate for State Decoder	0.0003
Learning rate for Diffusion Model	0.0001
Learning rate for Reward and Termination Model	0.0001
Learning rate for Actor & critic	0.0005
Gradient clipping for State Decoder	10
Gradient clipping for Diffusion Model	1
Gradient clipping for Reward and Termination Model	10
Gradient clipping for Actor & critic	10
Weight decay for State Decoder	0.01
Weight decay for Diffusion Model	0.01
Weight decay for Reward and Termination Model	0.01
λ for λ -return computation	0.95
Discount factor γ	see Table 1
Entropy coefficient	see Table 1
Buffer size (transitions)	2.5×10^5
Training steps per epoch	200
Training steps per epoch for policy learning	4
Sampling Environment steps per epoch	200 in MAMuJoCo 500 in Bi-DexHands
PPO epochs	5
PPO Clipping parameter ϵ	0.1
Number of imagined rollouts	600
Imagination horizon H	15
Diffusion sampling solver	Euler
Number of denoising steps	$\begin{cases} 2 \cdot \mathcal{N} & \text{if } \mathcal{N} \leq 2 \\ \mathcal{N} & \text{if } \mathcal{N} > 2 \end{cases}$