# ART: AUXILIARY REPRESENTATION TRANSFER FOR MULTI-AGENT POLICY OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Actor-critic methods in multi-agent reinforcement learning (MARL) typically rely solely on scalar rewards to shape actor representations. This narrow training signal leaves rich joint system dynamics unexploited, which severely limits performance and adaptability, especially under partial observability. We overcome this by introducing Auxiliary Representation Transfer (ART), a novel framework that aligns decentralized actor features with the predictive latent coordinates of a centralized Koopman autoencoder. Unlike prior methods that feed global models directly into the policy, ART regresses these coordinates through a linear auxiliary head during training to embed global predictive dynamics directly into the action selection process. Crucially, this transfer requires no execution-time overhead and leaves the actors' local observations and network architectures entirely unchanged. Experiments on Multi-Agent MuJoCo demonstrate that ART substantially improves the sample efficiency and final performance of Multi-Agent Proximal Policy Optimization (MAPPO). Notably, ART yields up to a 76% performance gain on Ant under full observation and more than doubles the final return on Ant $2 \times 4$ under partial observation.

## 1 INTRODUCTION

Deep reinforcement learning requires an agent to learn not only which actions to take, but also which features of its environment are most useful for making those decisions. Typically, these internal representations are shaped entirely through return-driven optimization, guided implicitly by policy and value objectives. However, relying solely on action selection and return estimation to drive feature learning often falls short (Schwarzer et al., 2021; Guo et al., 2020). Studies on feature-rank reduction and capacity loss demonstrate that this narrow training signal can severely limit the diversity of representations a network learns, limiting its ability to generalize and adapt to complex dynamics (Kumar et al., 2020; Lyle et al., 2022).

The need for robust representations becomes even more critical when scaling to multi-agent reinforcement learning (MARL), particularly under partial observability. In cooperative continuous control, the consequences of an agent's actions depend heavily on the underlying system configuration and the concurrent actions of its teammates. Standard MARL methods, such as MAPPO (Yu et al., 2022) and HAPPO (Kuba et al., 2022) , attempt to solve this by learning actor features through the same return-driven optimization, leaving rich trajectory data and transition dynamics fundamentally underutilized. While recent predictive representation learning methods, such as MAPO-LSO (Huh & Mohapatra, 2025) and TIMAR (Feng et al., 2025), have successfully used forward prediction or self-supervised objectives to improve multi-agent performance, they often fall short in deployment. These approaches typically feed the learned representations directly into the policy network. However, they learn these representations only from the agents' own observations, and MAPO-LSO also requires communication between agents at execution.

To bridge this gap, we introduce Auxiliary Representation Transfer (ART), a framework that leverages Koopman operator theory (Koopman, 1931) to provide explicit representation supervision without altering the actor's execution pathway (Figures 1 and 2). The core intuition is twofold. First, instead of using arbitrary predictive targets, we seek coordinates whose temporal evolution under joint actions is simple to predict. Deep Koopman autoencoders (Lusch et al., 2018) achieve this by lifting the state into a space where dynamics act linearly, paired with a bilinear control approxima-



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tion to model state-dependent action effects. This provides highly informative, structurally stable representation targets. Second, rather than feeding these learned Koopman coordinates into the policy as inputs, ART uses a linear auxiliary head on the actor's backbone to regress against them during training. By matching the actor's features to the autoencoder's learned coordinates via an auxiliary loss, the system's dynamics shape action selection implicitly. This cleanly separates representation learning from representation transfer: the actor absorbs the dynamical context during training, but relies strictly on its local observations at execution, completely free of the autoencoder.

Compared with prior work, our approach has several advantages. While Phasic Policy Gradient (PPG) (Cobbe et al., 2021) uses auxiliary heads for value targets, and multi-agent methods like TIMAR or MAPO-LSO learn representations that directly enter the policy input, ART is unique in its integration of independently learned dynamical coordinates with training-only actor supervision. Our primary contribution is a novel framework for cooperative control that seamlessly handles both full and partial observation settings. We evaluate ART using MAPPO on Multi-Agent MuJoCo tasks including Ant, HalfCheetah, and Walker2d. Our findings demonstrate that ART substantially improves upon MAPPO's sample efficiency and mean final return across all nine tested combinations of task, agent partition, and observation setting. Notably, ART yields a 31–76% performance gain on Ant under full observation, and more than doubles the final return on the $2 \times 4$ partition under partial observation settings.

## 2 Related Work

**Privileged information in MARL.** Multi-agent actor-critic methods use centralized training with decentralized execution: a centralized critic sees the full state, while each actor acts on its own observation. MADDPG (Lowe et al., 2017) conditions each agent's critic on all agents' observations and actions; MAPPO (Yu et al., 2022) applies PPO with a centralized value function and shared actor parameters; and HAPPO (Kuba et al., 2022) updates agents sequentially to obtain a monotonic improvement guarantee. In these approaches, the actor benefits from the privileged global state only indirectly, through the critic's advantage estimates, and a centralized critic does not by itself improve actor updates (Lyu et al., 2023). ART establishes a direct pathway: the actor's features are explicitly trained to match the Koopman autoencoder's representation of the global state through auxiliary regression.

**Predictive and auxiliary representation learning.** Self-predictive objectives train features to predict their own future latent states, and stop-gradient targets keep these features from collapsing (Ni et al., 2024). A temporal-consistency loss alone can train a latent dynamics model that supports both planning and policy learning (Zhao et al., 2023). Recent work extends these ideas to multi-agent settings. MAPO-LSO (Huh & Mohapatra, 2025) and TIMAR (Feng et al., 2025) model joint transitions, with recurrent networks and Transformers respectively, and feed the resulting features to the policy. MA2CL (Song et al., 2023) and MA$^2$E (Kang et al., 2025) use masked reconstruction and contrastive learning to infer missing information about other agents under partial observation. These methods learn the representation from the agents' own observations, and MAPO-LSO also exchanges it between agents at execution. ART takes a different route. It leaves the actor's inputs and execution-time architecture unchanged and adds only the gradient of an auxiliary regression onto the encoding of a dynamics model trained alongside the policy. Because this needs only a linear head on the actor's features, ART could in principle be attached to the representation modules of methods such as TIMAR or MAPO-LSO, giving them full-state dynamical targets at no cost at execution. We therefore evaluate ART as an addition to a fixed policy optimizer rather than as a replacement for these architectures.

**Koopman models for reinforcement learning.** Koopman models have been used for offline data augmentation (Weissenbacher et al., 2022) and long-horizon prediction for planning (Mondal et al., 2024). Closest method to ours is KIPPO (Cozma et al., 2025) which trains a Koopman autoencoder with its own losses and feeds the encoded state to the actor and critic, and SKooP (D'Elia et al., 2026) provides Koopman predictions to the critic as privileged observations; both are single-agent, and in both the Koopman output is an input to the actor or critic. In ART no network receives it: the actor only regresses onto the autoencoder's encoding through an auxiliary head, so its inputs are unchanged and the autoencoder is not needed at execution. To our knowledge, ART is the first

method to use a learned bilinear Koopman model as a source of auxiliary supervision for actor features, and we study it in cooperative multi-agent policy optimization under both full and partial observation.

# 3 PRELIMINARIES

## 3.1 COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING

We formulate the problem as a cooperative multi-agent system where $n$ agents share a common team reward. Time indices appear as subscripts and agent indices as superscripts. At each timestep $t$, the environment is in state $s_t$. Each agent $i$ receives a local observation $o_t^i \in \mathcal{O}_i$ and samples an action $a_t^i \sim \pi_\theta(\cdot \mid o_t^i)$. The environment then transitions to a new state $s_{t+1} \sim P(\cdot \mid s_t, \mathbf{a}_t)$ based on the joint action $\mathbf{a}_t$. Our baseline algorithm, Multi-Agent PPO (MAPPO) (Yu et al., 2022), relies on a centralized critic $V_\omega(s_t)$ to estimate advantages $\hat{A}_t$. Each actor $\pi_\theta$ is updated using the probability ratio $\rho_t^i(\theta) = \pi_\theta(a_t^i \mid o_t^i)/\pi_{\theta,\text{old}}(a_t^i \mid o_t^i)$. The clipped surrogate objective for agent $i$ is:

$$\mathcal{L}_{\text{clip},i}(\theta) = \mathbb{E}_t \left[ \min \left( \rho_t^i(\theta)\hat{A}_t, \text{clip}\big(\rho_t^i(\theta), 1-\epsilon, 1+\epsilon\big)\, \hat{A}_t \right) \right]. \tag{1}$$

## 3.2 AUXILIARY REPRESENTATION LEARNING

Auxiliary learning supplements the reinforcement-learning objective with additional tasks that train the network's shared features. For instance, Self-Predictive Representations (SPR) train an agent to predict its own latent state representations multiple steps into the future (Schwarzer et al., 2021), while Predictions of Bootstrapped Latents (PBL) focus on capturing environment dynamics by predicting latent embeddings of future observations (Guo et al., 2020). There are various ways this auxiliary information can be incorporated into the policy. Many architectures incorporate auxiliary tasks directly as an additive loss alongside the standard RL objective, updating the shared features concurrently. The auxiliary task then shapes the features at every update, but its gradients can conflict with those of the policy objective. To mitigate this, other methods decouple the auxiliary optimization from the policy update (Cobbe et al., 2021; Stooke et al., 2021).

Phasic Policy Gradient (PPG) provides a concrete example of how regression can shape a policy representation. PPG attaches an auxiliary value head to the policy network, sharing its hidden layers with the action head. Using PPG's notation, $s_t$ is the state at timestep $t$, $V_{\theta_\pi}(s_t)$ is the policy network's auxiliary value prediction, and $\hat{V}_t^{\text{targ}}$ is a value target computed using generalized advantage estimation. The following defines the auxiliary regression loss: $L_{\text{aux}} = \frac{1}{2}\,\hat{\mathbb{E}}_t \left[ \left( V_{\theta_\pi}(s_t) - \hat{V}_t^{\text{targ}} \right)^2 \right]$. This regression updates both the auxiliary head and the shared features used for action selection, but the auxiliary prediction itself is not a policy input. PPG optimizes this loss in a separate auxiliary phase, holding the value targets fixed and adding a policy-cloning KL penalty to limit changes in action probabilities. PPG also notes that its auxiliary loss could in principle be any auxiliary objective, although it uses only the value function. ART builds on the same intuition that a regression target can shape the features used for action selection. In ART, the target is the output of an encoder, and the matching loss encourages actor features to retain the information needed to reproduce it. ART adds this loss directly to the actor loss at every policy update, with no separate phase and no KL penalty.

## 3.3 KOOPMAN OBSERVABLES AND CONTROLLED DYNAMICS

Koopman operator theory offers a framework for describing complex nonlinear dynamics through linear operations. An observable $g$ is defined as a function of the state. The controlled stochastic Koopman operator maps $g$ to its expected value at the next timestep under a fixed joint action $\mathbf{a}$:

$$(\mathcal{K}_{\mathbf{a}} g)(s) = \mathbb{E}[g(s_{t+1}) \mid s_t = s, \mathbf{a}_t = \mathbf{a}]. \tag{2}$$

For deterministic environments governed by $s_{t+1} = f(s_t, \mathbf{a}_t)$, the composition rule simplifies to $(\mathcal{K}_{\mathbf{a}} g)(s) = g(f(s, \mathbf{a}))$.

Because the exact Koopman operator typically acts on an infinite-dimensional function space, practical implementations rely on a finite dictionary of $d$ observable functions, resulting in a lifted vector

$\mathbf{z}(s) = [g_1(s), \ldots, g_d(s)]^\top$. Deep Koopman approaches jointly learn this dictionary using an encoder $\psi$ and a decoder $\phi$ to ensure the coordinates are both predictable and retain essential state information (Lusch et al., 2018; **?**).

To account for state-dependent action effects in continuous-time control-affine dynamics, we utilize a discrete-time bilinear approximation (Bruder et al., 2021):

$$\hat{\mathbf{z}}_{t+1} = K\mathbf{z}_t + B\mathbf{a}_t + \sum_{k=1}^{d_a} a_{t,k} N_k \mathbf{z}_t, \qquad d_a = \dim(\mathbf{a}_t). \tag{3}$$

Here, the joint action $\mathbf{a}_t \in \mathbb{R}^{d_a}$ concatenates the actions of all agents, and $a_{t,k}$ is its $k$-th scalar component. The matrix $K$ models the autonomous state evolution, $B$ models additive control, and each $N_k$ models the bilinear interaction, scaling the effect of an action by the current observables.

### 3.4 Extended Dynamic Mode Decomposition (EDMD)

Extended Dynamic Mode Decomposition (EDMD) is a data-driven method for estimating a finite Koopman approximation from recorded transition snapshots (Williams et al., 2015). Given paired observations $(s_t, s_{t+1})$, lifted to $\mathbf{z}(s_t)$ and $\mathbf{z}(s_{t+1})$, the successor map $K^\star$ is fitted via regularized regression with positive ridge coefficient $\alpha$ and Frobenius norm $\|\cdot\|_F$.

$$K^\star = \arg\min_K \left\{ \sum_t \|\mathbf{z}(s_{t+1}) - K\mathbf{z}(s_t)\|_2^2 + \alpha \|K\|_F^2 \right\}. \tag{4}$$

For controlled dynamics, this regression is expanded to include the paired actions and their bilinear products. In our framework, we use this closed-form EDMD regression as a one-time initialization for the autoencoder's transition operators, which are subsequently refined alongside the encoder and decoder via gradient descent.

## 4 Methodology

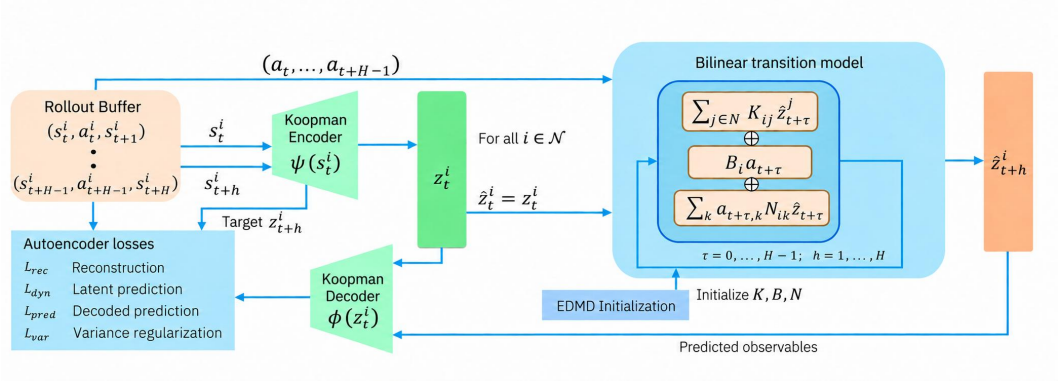



Figure 1: Training the deep Koopman autoencoder on recorded trajectories. The Koopman encoder maps each agent's input into learned coordinates, or observables, whose joint evolution the bilinear model predicts by applying the same operators repeatedly with the recorded actions. The Koopman decoder maps the current and predicted observables back to their states. Encoder and decoder parameters are shared across agents and are trained with reconstruction, prediction, and variance objectives. The learned encoder produces the cached targets in Figure 2, and its training continues as new trajectories are collected.

Auxiliary Representation Transfer (ART) decouples the learning of system dynamics from the transfer of this knowledge to the policy. The framework consists of two separately optimized modules. First, a centralized deep Koopman autoencoder is trained on recorded trajectories to produce lifted

representations that retain global state information and support future transition prediction (Figure 1). Second, a linear auxiliary head is added to the decentralized actor's backbone (Figure 2). By training this head to reproduce the autoencoder's outputs during the policy update, ART shapes the policy's features without altering its partial observations, rewards, or critic dependencies at execution time.

### 4.1 KOOPMAN AUTOENCODER TRAINING

We follow centralized training with decentralized execution (CTDE). For each agent $i$, the input to the Koopman autoencoder is the full environment state $s_t$ concatenated with the agent's one-hot identifier $e_i$, written $s_t^i = \mathrm{concat}(s_t, e_i)$. Conversely, each actor receives only its observation $o_t^i$; under full observability $o_t^i = s_t^i$, and under partial observability $o_t^i$ is restricted to a local view. The Koopman autoencoder thus uses the same full state information as the centralized critic. Because one shared encoder lifts every agent's input and $s_t$ is identical across agents, the identifier is what lets it produce a distinct block of coordinates for each agent. We normalize the autoencoder inputs before encoding and leave this operation implicit in the equations. Figure 1 shows how the Koopman autoencoder creates the lifted coordinates used for the auxiliary loss.

**Lifting and prediction.** We lift each agent's state $s_t^i$ individually with a shared Koopman encoder $\psi$ into $d_z$ learned observable coordinates, $z_t^i = \psi(s_t^i)$. After lifting, we stack the blocks of all $n$ agents to create a global lifted state $\mathbf{z}_t \in \mathbb{R}^{nd_z}$. The bilinear model then propagates these lifted coordinates forward for $H$ steps under the recorded actions; we write $\hat{z}_{t+h}^i$ for agent $i$'s block of the predicted coordinates after $h$ steps, $h = 1, \ldots, H$. The shared Koopman decoder $\phi$ decodes the current coordinates and the predicted coordinates at every step up to the $H^{\text{th}}$ step of the horizon, mapping each back to the state. Decoding the current coordinates gives the reconstruction loss:

$$\mathcal{L}_{\text{rec}} = \mathrm{MSE}\big(\phi(z_t^i), s_t^i\big). \tag{5}$$

Decoding the predicted coordinates gives the decoded prediction loss, which compares them with the recorded future states:

$$\mathcal{L}_{\text{pred}} = \frac{1}{H}\sum_{h=1}^{H} \mathrm{MSE}\big(\phi(\hat{z}_{t+h}^i), s_{t+h}^i\big). \tag{6}$$

Because every step up to the horizon is decoded, errors that build up over the rollout are penalized, not only the error of a single step. This multi-step supervision is expected to encourage the encoder to retain information useful for predicting longer-term evolution, yielding more temporally consistent representation targets for auxiliary transfer.

We denote the bilinear map defined in Section 3 (Equation 3) by $F$ for simplicity. Given the stacked lifted state and the recorded joint action, $\hat{\mathbf{z}}_{t+1} = F(\mathbf{z}_t, \mathbf{a}_t)$ contains one predicted block $\hat{z}_{t+1}^i$ for each agent. EDMD initializes the operators of $F$ once, at the first model update, by ridge regression (Section 3.4) on the encodings produced by the current encoder. On windows of $H$ transitions sampled within individual episodes, $F$ is applied recursively with the recorded actions, as in Equation 7.

$$\hat{\mathbf{z}}_t = \mathbf{z}_t, \qquad \hat{\mathbf{z}}_{t+h} = F(\hat{\mathbf{z}}_{t+h-1}, \mathbf{a}_{t+h-1}), \quad h = 1, \ldots, H. \tag{7}$$

The latent prediction loss in Equation 8 compares each predicted block with the encoding of the recorded future observation, shown as the target $z_{t+h}^i$ in Figure 1; $\mathrm{sg}(\cdot)$ denotes the stop-gradient operator, which treats its argument as a constant during backpropagation.

$$\mathcal{L}_{\text{dyn}} = \frac{1}{H}\sum_{h=1}^{H} \mathrm{MSE}\big(\hat{z}_{t+h}^i, \mathrm{sg}(z_{t+h}^i)\big). \tag{8}$$

After initialization, the encoder, decoder, and operators are refined jointly through the objective below, without repeating the EDMD fit.

**Autoencoder objective.** The latent prediction loss alone has a trivial solution: if the encoder maps every state to nearly the same point, the future of that point is easy to predict and $\mathcal{L}_{\text{dyn}}$ becomes small, but the coordinates no longer tell one state from another. The variance term in Equation 9 prevents this. For each agent $i$ and each latent coordinate $r = 1, \ldots, d_z$, it measures the standard

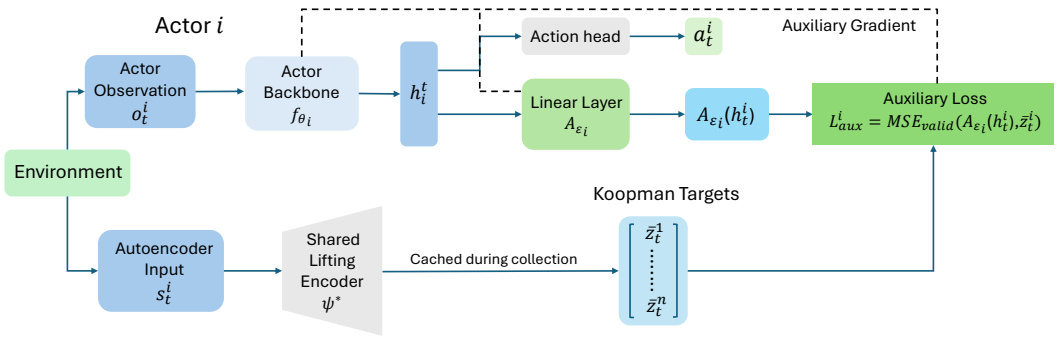



Figure 2: Auxiliary representation transfer. The actor's observation is mapped to features shared by the action head and a linear auxiliary head. The notation $\psi^* = \mathrm{sg}(\psi)$ denotes stop-gradient evaluation of the learned encoder for target construction, not permanently frozen encoder parameters. During rollout collection, the encoder trained in Figure 1 maps the training-time state to Koopman coordinates (Equation 11), which are cached in the rollout buffer and held fixed during the policy update. The auxiliary head regresses each sampled observation onto its agent's stored target, which encodes the current state. Auxiliary gradients update the backbone and auxiliary head, never the autoencoder or the action head.

deviation $\sigma_{\mathcal{B}}(z_{t,r}^i)$ of that coordinate over the starting states of the windows in a minibatch $\mathcal{B}$, and penalizes it when it falls below one:

$$\mathcal{L}_{\mathrm{var}} = \frac{1}{nd_z} \sum_{i=1}^{n} \sum_{r=1}^{d_z} \max\bigl(0, 1 - \sigma_{\mathcal{B}}(z_{t,r}^i)\bigr)^2 . \tag{9}$$

Every coordinate therefore keeps varying from one state to the next, so the lifted state keeps carrying information about the state of the robot. This term is adapted from VICReg (Bardes et al., 2022); unlike VICReg, we add no covariance penalty, so coordinates are allowed to be correlated. The autoencoder objective is the weighted sum of the four losses:

$$\mathcal{L}_{\mathrm{model}} = \lambda_{\mathrm{rec}}\mathcal{L}_{\mathrm{rec}} + \lambda_{\mathrm{dyn}}\mathcal{L}_{\mathrm{dyn}} + \lambda_{\mathrm{pred}}\mathcal{L}_{\mathrm{pred}} + \lambda_{\mathrm{var}}\mathcal{L}_{\mathrm{var}}. \tag{10}$$

The reconstruction and prediction terms are the standard deep Koopman autoencoder losses (Lusch et al., 2018). All four losses train the encoder. Reconstruction and decoded prediction train the decoder, while both prediction losses, $\mathcal{L}_{\mathrm{dyn}}$ and $\mathcal{L}_{\mathrm{pred}}$, train the operators through the recursive rollout. Gradients are stopped through future latent targets. Because the encoder produces the auxiliary targets, reconstruction and prediction shape these targets; the auxiliary regression then encourages their information to be recoverable from the features used for action selection.

### 4.2 Auxiliary Representation Transfer

The second module transfers the autoencoder's representation into the actors (Figure 2). Following Section 3.2, a linear auxiliary head on the actor backbone predicts the Koopman encoder's current coordinates, in place of the value target used by PPG. Neither the encoder nor its predicted trajectories enter the policy's input path. Because the actor only regresses onto these targets and never receives them as input, they can come from the full state even when the actor sees only a partial observation. The full state reaches the actor only through the gradient of the auxiliary loss, so execution stays decentralized.

During rollout collection, the encoder is evaluated without gradient tracking and maps each agent's state $s_t^i$ to its block, which is stored in the rollout buffer as the auxiliary target in Equation 11 (Figure 2).

$$\bar{z}_t^i = \mathrm{sg}\bigl(\psi(s_t^i)\bigr) . \tag{11}$$

The stored targets do not change during the policy update that uses them. The actor backbone $f_\theta$ maps the observation to features $h_t^i = f_\theta(o_t^i)$. These features feed both the action head and a linear

auxiliary head $A_\xi$ with parameters $\xi$ (Figure 2). The head maps $h_t^i$ to a $d_z$-dimensional output $A_\xi(h_t^i)$, which is trained to match the stored target $\bar{z}_t^i$. The actor backbone and auxiliary head share parameters across agents; agent indices identify samples, not separate networks.

Let $\mathcal{M}_i$ denote the set of timesteps of agent $i$ in the current PPO minibatch. For each $t \in \mathcal{M}_i$, the binary mask $m_t^i \in \{0, 1\}$ equals 1 when a valid target is available and 0 otherwise. A target is invalid before the first autoencoder update, when no encoder output exists yet, if it contains non-finite values, or where MAPPO's active mask excludes the sample. The auxiliary objective in Equation 12 averages the matching error over valid samples and target coordinates:

$$\mathcal{L}_{\text{aux},i} = \frac{1}{d_z N_i} \sum_{t \in \mathcal{M}_i} m_t^i \left\| A_\xi(h_t^i) - \bar{z}_t^i \right\|_2^2, \qquad N_i = \max\Big(1, \sum_{t \in \mathcal{M}_i} m_t^i\Big). \tag{12}$$

Here $N_i$ counts the valid samples; taking the maximum with one keeps the loss defined, and equal to zero, when a minibatch contains no valid target. Exact recovery from local observations is not assumed under partial observability. Each actor minimizes its MAPPO actor loss plus the weighted auxiliary loss, Equation 13,

$$\mathcal{L}_{\text{actor},i}^{\text{ART}} = \mathcal{L}_{\text{actor},i} + \lambda_{\text{aux}} \mathcal{L}_{\text{aux},i}, \tag{13}$$

where $\mathcal{L}_{\text{actor},i} = -\mathcal{L}_{\text{clip},i} - \beta\, \mathcal{H}[\pi_\theta(\cdot \mid o_t^i)]$ is the MAPPO actor loss with entropy weight $\beta$. Auxiliary gradients update the backbone and auxiliary head; the autoencoder, stored targets, and critic receive no auxiliary gradients (Figure 2). The action head is trained by the policy objective using the shared features. Execution uses only the decentralized actors.

Together, the two modules give each actor information about the system's dynamics that the return alone does not provide. A single shared Koopman encoder gives each agent its own block of latent coordinates, and the bilinear model (Equation 3) couples these blocks through the joint action. The auxiliary targets therefore describe how the agents move together. The auxiliary loss trains the actor's own features to predict these targets, so the features used for action selection carry this information. The stop-gradient on the stored targets also keeps the policy objective from changing the dynamics model.

## 5 Experiments

Our evaluation addresses two questions: whether ART improves policy learning relative to MAPPO, and which components of the framework account for the observed differences. All experiments use MAPPO as the underlying policy optimizer, so differences between runs reflect the auxiliary supervision rather than the choice of MARL algorithm.

### 5.1 Experimental Setup

We use Multi-Agent MuJoCo (Peng et al., 2021), in which a single robot is partitioned into agents that each control a subset of its joints. We write $n \times m$ for $n$ agents with $m$ joints each. The tasks are Ant ($2 \times 4$, $4 \times 2$, $8 \times 1$), HalfCheetah ($3 \times 2$, $6 \times 1$), and Walker2d ($6 \times 1$), shown in Appendix Figure 3. All agents receive the environment's shared team reward. Appendix A.1 lists the joints each agent controls and the observation sizes, and Appendix A.2 gives implementation details of the Koopman autoencoder and the auxiliary head.

Under *full observation*, we use the configuration released with HAPPO (Kuba et al., 2022): every actor receives the full environment observation together with a one-hot agent identifier. Under *partial observation*, each actor receives only the local observation of its own joints and their immediate neighbours in the morphology graph ($k = 1$). For Ant, this consists of the agent's own joint positions and velocities, the positions of the neighbouring agents' hip joints, and the torso position and velocity. Partial observation is evaluated on Ant and HalfCheetah $6 \times 1$.

To isolate the contribution of each component of ART, we change one element at a time and keep the auxiliary head, the loss weights, and the PPO hyperparameters fixed. We first replace the auxiliary target. The fixed linear target replaces the learned Koopman encoder $\psi$ with a static linear projection $P$, initialized with Gaussian entries and unit-norm rows. It receives the same full-state input and has the same dimension as the Koopman target, but it is not shaped by the dynamics. The value target replaces the Koopman coordinates with the critic's generalized advantage estimation (GAE) return,

standardized across each rollout, which is the target of PPG's auxiliary value head. We then keep the Koopman target and remove individual terms from the autoencoder objective $\mathcal{L}_{\text{model}}$ (Equation 10), keeping all other terms active. The results tables list the terms removed in each variant.

We report the final evaluation return with deterministic actions as the mean $\pm$ population standard deviation over seeds. Main results use six seeds per method. Ablations compare all variants on matched seeds: six for the target ablation on Ant $4 \times 2$ and three otherwise. Appendix A.3 lists all hyperparameters.

### 5.2 Results

Table 1: Final evaluation return and sample efficiency. Returns are mean $\pm$ population standard deviation over six seeds; bold marks the higher mean, not statistical significance. Steps is the mean number of environment steps (millions) at which ART seeds first reach MAPPO's mean final return, with the number of seeds that reach it; AUC is the ratio of areas under the ART and MAPPO learning curves.

| Environment | Partition | MAPPO | ART | Steps (M) | AUC |
|---|---|---|---|---|---|
| *Full observation* | | | | | |
| Ant | $2 \times 4$ | $3{,}385 \pm 716$ | $\mathbf{4{,}434 \pm 758}$ | 5.90 (6/6) | 1.48 |
| Ant | $4 \times 2$ | $3{,}226 \pm 648$ | $\mathbf{5{,}128 \pm 163}$ | 4.17 (6/6) | 1.76 |
| Ant | $8 \times 1$ | $2{,}862 \pm 479$ | $\mathbf{5{,}040 \pm 319}$ | 4.03 (6/6) | 1.90 |
| HalfCheetah | $3 \times 2$ | $5{,}270 \pm 426$ | $\mathbf{5{,}447 \pm 821}$ | 6.70 (3/6) | 1.09 |
| HalfCheetah | $6 \times 1$ | $4{,}889 \pm 473$ | $\mathbf{5{,}180 \pm 695}$ | 6.23 (4/6) | 1.08 |
| Walker2d | $6 \times 1$ | $4{,}449 \pm 641$ | $\mathbf{4{,}990 \pm 153}$ | 4.97 (6/6) | 1.26 |
| *Partial observation* | | | | | |
| Ant | $2 \times 4$ | $1{,}454 \pm 101$ | $\mathbf{3{,}267 \pm 660}$ | 2.57 (6/6) | 1.84 |
| Ant | $4 \times 2$ | $1{,}342 \pm 119$ | $\mathbf{1{,}997 \pm 187}$ | 2.03 (6/6) | 1.51 |
| HalfCheetah | $6 \times 1$ | $4{,}434 \pm 442$ | $\mathbf{4{,}665 \pm 458}$ | 6.59 (5/6) | 1.02 |

ART improves on MAPPO in every setting (Table 1), with the largest gains on Ant. Under partial observation on Ant, every ART seed ends above every MAPPO seed. ART also learns faster. On every Ant task and on Walker2d, all ART seeds reach MAPPO's final return well before the end of training (Appendix Figure 4). On HalfCheetah the margin is smaller, under both full and partial observation, and lies within the variation across seeds. Across all settings, the autoencoder trains stably alongside the policy (Appendix Figure 5).

Table 2: Ablations: final evaluation return, mean $\pm$ population standard deviation over matched seeds (six on Ant $4 \times 2$, three on Ant $2 \times 4$). (a) ART with its Koopman target replaced by a fixed linear target. (b) ART on Ant $2 \times 4$ under partial observation, trained with the value target or with the listed autoencoder losses removed. All Ant $2 \times 4$ ablations share the same three seeds, so their MAPPO and ART values differ from the six-seed values in Table 1. Full-observation loss ablations are in Appendix A.5.

*(a) Fixed linear target*

| | MAPPO | Fixed linear | **Koopman (ART)** |
|---|---|---|---|
| Ant $4 \times 2$, full obs. | $3{,}226 \pm 648$ | $4{,}515 \pm 650$ | $\mathbf{5{,}128 \pm 163}$ |
| Ant $2 \times 4$, partial obs. | $1{,}420 \pm 123$ | $3{,}189 \pm 233$ | $\mathbf{3{,}723 \pm 336}$ |

*(b) Value target and removed autoencoder losses, Ant $2 \times 4$, partial obs.*

| | Target | Removed losses | | | | |
|---|---|---|---|---|---|---|
| MAPPO | Value | $\mathcal{L}_{\text{dyn}}, \mathcal{L}_{\text{pred}}$ | $\mathcal{L}_{\text{dyn}}$ | $\mathcal{L}_{\text{pred}}$ | $\mathcal{L}_{\text{rec}}$ | **ART** |
| $1{,}420 \pm 123$ | $2{,}013 \pm 292$ | $2{,}957 \pm 53$ | $2{,}645 \pm 93$ | $2{,}759 \pm 499$ | $2{,}904 \pm 944$ | $\mathbf{3{,}723 \pm 336}$ |

Table 2 and Appendix Figure 6 show why the Koopman target works better than simpler alternatives. Replacing it with a fixed linear projection of the state keeps the full-state information and removes

only the dynamics training. This variant already improves on MAPPO, so the auxiliary regression alone is an effective way to bring full-state information into the actor. The value target, a PPG-style regression onto the return, helps much less. A single return estimate carries far less information about the state than a vector of coordinates, so what the head regresses matters, not only that there is a head. The Koopman target is the best of the three in both observation settings, and under full observation it beats the fixed linear target on every seed. This additional gain comes from training the coordinates to be predictable under the joint action. When both prediction losses are removed, the autoencoder only reconstructs the current state and its target performs at the level of the fixed linear one.

Every term of the autoencoder objective contributes, but how much each one matters varies with the partition. Under partial observation, where the autoencoder sees information the actors do not, removing any loss lowers the return (Table 2(b)). Under full observation, the full objective is clearly better than latent prediction alone on Ant $8 \times 1$, while on the coarser partitions the reduced objectives stay close to it (Appendix Figure 7).

## 6 Conclusion

We introduced ART, a new way to improve policy learning in cooperative control. Rather than using a model of the dynamics to plan or to feed the policy, ART uses it only as a source of training targets for the actor's features. A Koopman autoencoder trained on the team's joint transitions learns coordinates that describe the whole system and how it evolves under the joint action, and a single linear head with one loss term carries this description into each actor. This separation is what makes the approach easy to adopt. The autoencoder can use any information available during training, including the full state that a partially observing actor never sees, while execution stays fully decentralized and each actor keeps its own inputs and architecture. ART therefore adds a training signal on top of existing policy optimizers and actor architectures instead of replacing them. Our experiments show that this signal is useful across robots, partitions, and observation settings, and that its value comes from the dynamics the target encodes rather than from the extra loss alone. Furthermore, ART consistently preserves or enhances performance. In every setting, ablation, and hyperparameter choice we tested, its mean return remains strictly higher than MAPPO's. This includes HalfCheetah, where the gain is smallest, showing that ART can be added to an existing policy optimizer without compromising performance.

**Limitations.** ART also has limitations that point to future work. We apply it only to MAPPO, and how the auxiliary signal interacts with other on-policy, off-policy, and value-decomposition methods remains open. Our experiments cover locomotion in Multi-Agent MuJoCo, where every agent controls part of a single robot. Teams of separate bodies, sparse rewards, or discrete actions may call for different targets. The Koopman autoencoder uses a finite learned dictionary and a bilinear model, so how well its coordinates describe systems with stronger nonlinearities is not yet clear. Finally, the contribution of each autoencoder loss varies with the partition, and which properties make a target most useful to the actor is an open question.

**Future directions.** ART separates target construction from policy optimization, which makes it possible to study representation transfer beyond MAPPO. Integrating the auxiliary head with HAPPO or off-policy actor–critic methods would test how its benefits depend on the policy update rule, and combining it with recurrent observation encoders could extend it under partial observability. Cooperative manipulation by separate robots, where interactions arise through contact and shared objects, would show when joint dynamics modeling helps individual actors. Finally, the transfer interface permits targets beyond current-state encodings, such as action-conditioned future representations weighted by model uncertainty, without adding a model to the execution pathway.

### AI use statement

We used a large language model (LLM) coding assistant throughout this project. It wrote and debugged parts of the training code, the job scripts used to launch experiments, the analysis and plotting scripts, and the code that renders the figures. It also helped search and summarize related work, proposed diagnostics and ablations, and drafted and edited parts of the paper text and its LaTeX

formatting. The research question, the method, and the choice of experiments were decided by the authors, who accepted or rejected every suggestion from the assistant. All reported numbers come from the raw training logs of the runs, extracted by scripts that the authors checked, and none were written by the model. The authors reviewed all AI-assisted code, text, and citations, and take full responsibility for the content of this work.

### Reproducibility statement

We have taken several steps to make our results reproducible. Section 5.1 describes the tasks, the two observation settings, the number of seeds, and the evaluation protocol, and every reported return is the final evaluation return averaged over seeds with the population standard deviation. We build on the TRPO-in-MARL codebase of Kuba et al. (2022), available at `https://github.com/cyanrain7/TRPO-in-MARL`, which provides Multi-Agent MuJoCo and the MAPPO baseline, and MAPPO and ART share every hyperparameter they have in common. Appendix A.1 lists the agent partitions and the exact input of every actor, critic, and autoencoder, including simulator details that affect the observation. Appendix A.2 gives the architecture of the Koopman autoencoder and the auxiliary head, Appendix A.3 lists all hyperparameters, and Algorithm 1 in Appendix A.4 summarizes the training loop. Appendix A.5 reports the learning curves and the training losses behind every table, so that a reimplementation can be checked against them during training and not only at the end.

## References


Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning, 2022. URL `https://arxiv.org/abs/2105.04906`.

Daniel Bruder, Xun Fu, and Ram Vasudevan. Advantages of bilinear koopman realizations for the modeling and control of systems with unknown dynamics. *IEEE Robotics and Automation Letters*, 6(3):4369–4376, 2021.

Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pp. 2020–2027. PMLR, 2021.

Andrei Cozma, Landon Harris, and Hairong Qi. Kippo: Koopman-inspired proximal policy optimization. In James Kwok (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 4994–5002. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/556. URL `https://doi.org/10.24963/ijcai.2025/556`. Main Track.

Evelyn D'Elia, Weishu Zhan, Giulio Turrisi, Giulio Romualdi, Giuseppe L'Erario, Raffaello Camoriano, Wei Pan, and Daniele Pucci. Skoop: Symmetric koopman predictions for faster and more generalizable legged robot locomotion with reinforcement learning. *arXiv preprint arXiv:2607.11624*, 2026.

Mingxiao Feng, Yaodong Yang, Wengang Zhou, and Houqiang Li. Timar: Transition-informed representation for sample-efficient multi-agent reinforcement learning. *Neural Networks*, 184: 107081, 2025.

Zhaohan Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-Bastien Grill, Florent Altché, Remi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3875–3886. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/guo20g.html`.

Dom Huh and Prasan Mohapatra. Representation learning for efficient deep multi-agent reinforcement learning. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pp. 1–8, 2025.

Sehyeok Kang, Yongsik Lee, Gahee Kim, Song Chong, and Se-Young Yun. Ma ̃2 e: Addressing partial observability in multi-agent reinforcement learning with masked auto-encoder. In *International Conference on Learning Representations*, volume 2025, pp. 20145–20165, 2025.

B. O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931. doi: 10.1073/pnas.17.5.315. URL https://www.pnas.org/doi/abs/10.1073/pnas.17.5.315.

Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning, 2022. URL https://arxiv.org/abs/2109.11251.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *arXiv preprint arXiv:2010.14498*, 2020.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018. doi: 10.1038/s41467-018-07210-0.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. *arXiv preprint arXiv:2204.09560*, 2022.

Xueguang Lyu, Andrea Baisero, Yuchen Xiao, Brett Daley, and Christopher Amato. On centralized critics in multi-agent reinforcement learning. *J. Artif. Int. Res.*, 77, June 2023. ISSN 1076-9757. doi: 10.1613/jair.1.14386. URL https://doi.org/10.1613/jair.1.14386.

Arnab Kumar Mondal, Siba Smarak Panigrahi, Sai Rajeswar, Kaleem Siddiqi, and Siamak Ravanbakhsh. Efficient dynamics modeling in interactive environments with koopman theory, 2024. URL https://arxiv.org/abs/2306.11941.

Tianwei Ni, Benjamin Eysenbach, Erfan Seyedsalehi, Michel Ma, Clement Gehring, Aditya Mahajan, and Pierre-Luc Bacon. Bridging state and history representations: Understanding self-predictive rl. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 23555–23569, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/666c1861d709bd84e20b6e0e02a2c223-Paper-Conference.pdf.

Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in neural information processing systems*, 34:12208–12221, 2021.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations, 2021. URL https://arxiv.org/abs/2007.05929.

Haolin Song, Mingxiao Feng, Wengang Zhou, and Houqiang Li. Ma2cl: Masked attentive contrastive learning for multi-agent reinforcement learning. *arXiv preprint arXiv:2306.02006*, 2023.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9870–9879. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/stooke21a.html.

Matthias Weissenbacher, Samarth Sinha, Animesh Garg, and Kawahara Yoshinobu. Koopman q-learning: Offline reinforcement learning via symmetries of dynamics. In *International conference on machine learning*, pp. 23645–23667. PMLR, 2022.

Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and YI WU. The surprising effectiveness of ppo in cooperative multi-agent games. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24611–24624. Curran Associates, Inc., 2022. doi: 10.52202/068431-1787. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9c1535a02f0ce079433344e14d910597-Paper-Datasets_and_Benchmarks.pdf.

Yi Zhao, Wenshuai Zhao, Rinu Boney, Juho Kannala, and Joni Pajarinen. Simplified temporal consistency reinforcement learning. In *International Conference on Machine Learning*, pp. 42227–42246. PMLR, 2023.

# A Appendix

## A.1 Environments, Partitions, and Observations

All tasks come from Multi-Agent MuJoCo (Peng et al., 2021) with the Gym v2 robots. Each task is a single robot, and the partition only decides which actuated joints each agent controls. Every agent outputs torques for its own joints, and the joint action is the concatenation of these torques. Episodes last 1000 steps, and every agent receives the robot's standard Gym reward as the shared team reward. Table 3 lists the partitions and the size of each agent's observation, and Figure 3 shows the three robots with the joints of each agent drawn in the same colour. The partition decides only which joints an agent drives. Every agent still acts on the same body, so the torque of one agent changes the motion of the joints that the other agents control.

Table 3: Partitions and actor observation sizes. Joints are listed per agent. Partial observation is evaluated only on the three tasks with an entry in the last column.

| Task | Joints controlled by each agent | Full obs. | Partial obs. |
|---|---|---|---|
| Ant $2 \times 4$ | {hip1, ankle1, hip2, ankle2}, {hip3, ankle3, hip4, ankle4} | 113 | 25 |
| Ant $4 \times 2$ | $\{\text{hip}_i, \text{ankle}_i\}$ for leg $i = 1, \ldots, 4$ | 115 | 24 |
| Ant $8 \times 1$ | one joint each: hip1, ankle1, …, hip4, ankle4 | 119 | – |
| HalfCheetah $3 \times 2$ | {bfoot, bshin}, {bthigh, ffoot}, {fshin, fthigh} | 20 | – |
| HalfCheetah $6 \times 1$ | one joint each: bfoot, bshin, bthigh, ffoot, fshin, fthigh | 23 | 15 |
| Walker2d $6 \times 1$ | one joint each: right foot, leg, thigh; left foot, leg, thigh | 23 | – |

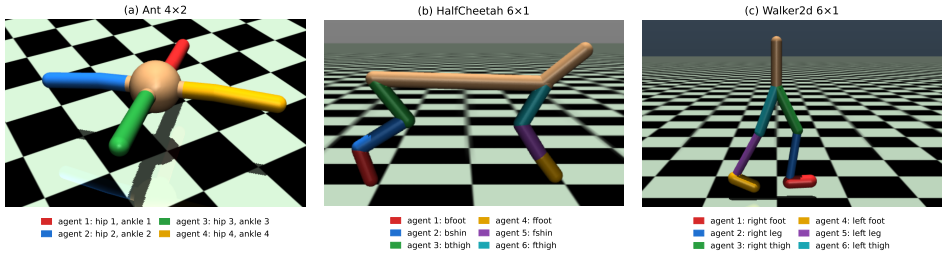



Figure 3: The three Multi-Agent MuJoCo robots. Joints with the same colour belong to the same agent. (a) Ant $4 \times 2$, one agent per leg, each driving the hip and the ankle. (b) HalfCheetah $6 \times 1$ and (c) Walker2d $6 \times 1$, one agent per joint.

Under full observation, we use the observation of the released HAPPO configuration (Kuba et al., 2022): every actor receives the robot's full Gym observation with its one-hot agent identifier appended. The Gym observation has 111 entries for Ant (torso height and orientation, joint angles, all velocities, and 84 external contact forces) and 17 entries for HalfCheetah and Walker2d (joint angles and velocities, without the torso's horizontal position). With the identifier this gives $111 + n$ or $17 + n$ entries for $n$ agents. In the simulator version we use, the 84 contact-force entries of the Ant observation are always zero; we keep them so that the observation matches the released benchmark.

Under partial observation, each actor receives the positions and velocities of its own joints, the positions of the joints one step away in the morphology graph ($k = 1$), the position and velocity of the torso, and its one-hot identifier. On Ant the neighbours of a leg are the hips of the other legs, which connect to it through the torso. The actor never receives the ankle angles or the joint velocities of legs controlled by other agents.

In both settings the centralized critic and the Koopman autoencoder receive the full-observation vector of agent $i$, that is, the full Gym observation with the identifier $e_i$; this is the input $s_t^i$ of Section 4.1. Under partial observation the actor never receives this vector, during training or at execution; the full state reaches the actor only through the gradient of the auxiliary loss.

### A.2 Implementation of the Koopman Autoencoder and the Auxiliary Head

The Koopman encoder $\psi$ is an MLP with two hidden layers of 256 ReLU units and a 32-dimensional output, followed by a layer normalization without learned scale or offset. Its input is normalized per dimension with statistics estimated at the first model update and then kept fixed. The Koopman decoder has the same hidden layers and reconstructs the robot's physical state (27 entries for Ant, 17 for HalfCheetah and Walker2d) together with the agent identifier. The prediction loss $\mathcal{L}_{\text{pred}}$ scores the physical state only, since the identifier does not change over time.

Besides the 32 learned coordinates, the lifted state contains a small set of fixed contact features built from the normal contact force $f$ on each foot, read from the simulator. Each force is first scaled to $\bar{f} = \tanh(f/c_0)$, where $c_0$ is the 90th percentile of that foot's force. Three Gaussian radial basis functions (RBFs) of $\bar{f}$, one of them centred at zero for a foot in the air, describe each agent's own foot, and eight more, centred by $k$-means on the vector of all foot values $\bar{f}$, describe the contact pattern of the whole robot. The centres and widths are set once from the first batch of data and then kept fixed. The autoencoder lifts these features together with the other state variables, and a contact decoder reconstructs the foot forces from them. These features do not improve the return (Appendix A.5).

The operators $K$, $B$, and $N_k$ in Equation 3 are dense: every agent's block of the next lifted state depends on all agents' blocks, on the full joint action, and on all products $a_{t,k}\mathbf{z}_t$. EDMD initializes them once, at the first model update, and they are then trained by gradient descent together with the encoder and decoders, using Adam.

The Koopman autoencoder is updated every $C = 125$ environment steps on the transitions collected in that interval, that is, 125 steps from each of the 32 parallel environments. Each update takes $G = 8$ gradient steps on minibatches of 2048 windows of $H = 10$ consecutive transitions; windows never cross an episode boundary.

The actor network is shared across agents, as in MAPPO, and the one-hot identifier tells it which agent it controls. The linear auxiliary head $A_\xi$ maps the 128-dimensional output of the shared actor backbone to the 32-dimensional target. The target and its validity mask are stored with the actor's observation in the rollout buffer, but they are removed before the observation enters the network, so they never enter the action-selection input.

### A.3 Hyperparameters

Table 4 lists the settings used in all experiments. MAPPO and ART use identical values for every shared setting.

### A.4 Training Loop

Algorithm 1 summarizes one run of ART with MAPPO. Each training iteration has two phases: rollout collection, during which the Koopman autoencoder is trained, and the policy update, during which the actors are trained with the auxiliary loss.

Each training iteration starts with rollout collection (lines 2–8). The actors act in the environment for $T$ steps, and every transition is added to a buffer $\mathcal{D}$ of recent transitions. Whenever $\mathcal{D}$ holds $C$ steps, the autoencoder is updated (lines 4–6). With $T = 1000$ and $C = 125$, this happens eight times during every rollout, while the actors are updated only once, after the rollout ends. At the first update only, EDMD fits the operators $K$, $B$, and $N$ by ridge regression on the encodings of the untrained encoder (line 5). Each update then takes $G$ gradient steps on $\mathcal{L}_{\text{model}}$ (line 6). Each step samples $H$-step windows from $\mathcal{D}$, encodes the first state of each window, rolls the bilinear model forward with the recorded actions, and trains the encoder, decoder, and operators together. The buffer is then emptied, so the autoencoder always trains on the most recent $C$ steps. After every environment step, the current encoder maps each agent's next state to its target $\bar{z}^i_{t+1}$, and the mask $m^i_{t+1}$ records whether this target is valid. Both are stored with the policy transition (line 8). Because the autoencoder is updated during collection, targets stored at different steps of one rollout can come from slightly different encoders, but each target stays fixed once it is stored.

Table 4: Hyperparameters.

| Setting | Value |
|---|---|
| *Policy optimization (MAPPO and ART)* | |
| Environment steps | $10^7$ |
| Parallel environments | 32 |
| Episode length | 1000 |
| Actor and critic networks | two-layer MLPs, 128 units |
| Actor parameter sharing | across agents |
| PPO epochs | 5 |
| Clip ratio $\epsilon$ | 0.2 |
| Entropy weight $\beta$ | 0.01 |
| Discount | 0.99 |
| GAE parameter | 0.95 |
| Learning rate (actor / critic) | $5 \times 10^{-4}$ / $5 \times 10^{-3}$ |
| Value loss | Huber, with value normalization |
| *Koopman autoencoder* | |
| Encoder and decoder | MLPs, 256 hidden units |
| Latent dimension per agent $d_z$ | 32 |
| Optimizer | Adam, learning rate $10^{-3}$ |
| Gradient-norm clip | 10 |
| Update interval $C$ | 125 environment steps |
| Gradient steps per update $G$ | 8 |
| Minibatch | 2048 windows |
| Prediction horizon $H$ | 10 |
| Loss weights $\lambda_{\mathrm{rec}}, \lambda_{\mathrm{dyn}}, \lambda_{\mathrm{pred}}, \lambda_{\mathrm{var}}$ | 1 |
| EDMD ridge coefficient $\alpha$ | $10^{-2}\times$ mean diagonal of the Gram matrix |
| Contact force scale $c_0$ | 90th percentile of each foot's force |
| Contact RBFs | 3 per foot, 8 pattern RBFs ($k$-means) |
| Contact decoder loss weight | 1 |
| *Auxiliary transfer* | |
| Auxiliary head | linear |
| Auxiliary weight $\lambda_{\mathrm{aux}}$ | 0.1 |
| *Evaluation* | |
| Evaluation interval | $8 \times 10^5$ environment steps |
| Episodes per evaluation | 32, deterministic actions |
| Reported point | final evaluation, $9.63 \times 10^6$ steps |

After collection, the critic computes advantages and value targets as in MAPPO (line 10). For each PPO epoch and minibatch, the actor backbone computes the features $h_t^i$, and the actor parameters $\theta$ and the auxiliary head $\xi$ are updated with the ART actor loss of Equation 13 (line 11). The stored targets and the autoencoder are not changed by this update. The critic is updated with the standard MAPPO value loss (line 12). Apart from the auxiliary term in the actor loss and the stored targets, this phase is identical to a MAPPO update.

After training, only the actors are kept (line 14). The autoencoder, the bilinear model, and the auxiliary heads are discarded, so each agent acts on its own observation exactly as a MAPPO actor does.

### A.5 Additional Results and Ablations

Figure 4 shows the learning curves behind Table 1, Figure 6 those behind Table 2, and Figure 5 shows the losses of the same ART runs during training. Every autoencoder loss drops sharply in the first million steps and then changes only slowly, although the policy, and with it the distribution of visited states, keeps changing. The autoencoder therefore tracks the data from the current policy without diverging. The latent and decoded prediction losses stay the largest, because predicting

**Algorithm 1** Auxiliary Representation Transfer (ART) with MAPPO

**Require:** Actors $\pi_\theta$ with linear auxiliary head $A_\xi$, critic $V_\omega$, Koopman autoencoder $(\psi, \phi)$ with bilinear model $F$

**for** each training iteration **do**
  **for** $t = 0, \dots, T-1$ **do** ▷ collect a rollout
    Sample $a_t^i \sim \pi_\theta(\cdot \mid o_t^i)$ for every agent and step the environment
    **if** $C$ new steps have been collected **then** ▷ train the Koopman autoencoder
      Initialize $F$ by EDMD (first update only)
      Take $G$ gradient steps on $\mathcal{L}_{\text{model}}$ (Equation 10) over $H$-step windows
    **end if**
    Store the target $\bar{z}_{t+1}^i = \text{sg}\big(\psi(s_{t+1}^i)\big)$ and its mask $m_{t+1}^i$
  **end for**
  Compute advantages with $V_\omega$ ▷ update the policy
  Update the actors with $\mathcal{L}_{\text{actor},i}^{\text{ART}}$ (Equation 13), targets and autoencoder fixed
  Update the critic with the MAPPO value loss
**end for**
**return** the actors $\pi_\theta$ only

ten steps ahead is harder than reconstructing the current state. The variance term stays close to zero, which shows that the coordinates keep their spread and do not collapse. The auxiliary loss is small under full observation, where the actor sees the state that the target encodes. Under partial observation it stays several times larger, since the actor has to infer the full-state target from its local view. On Ant this is also the setting where ART gains the most.

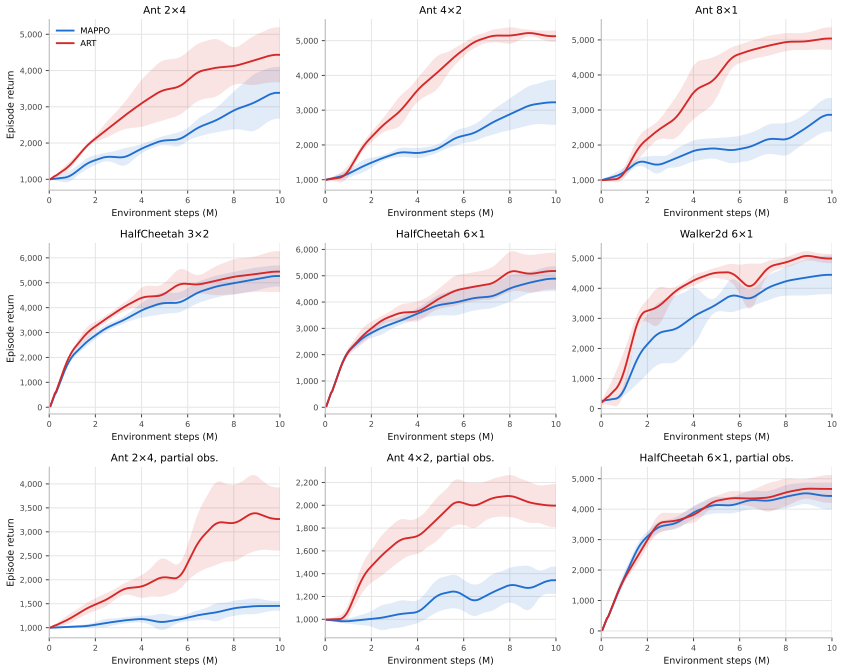



Figure 4: Learning curves of MAPPO and ART. Curves show the mean evaluation return over seeds and shaded regions one population standard deviation. The last three panels use partial observation.

Table 5 varies the auxiliary weight and Table 6 the prediction horizon, both on Ant $4 \times 2$ under full observation. The return rises from $\lambda_{\text{aux}} = 0.01$ to $0.1$ and stays flat up to $1.0$. Every horizon improves on MAPPO by at least half; $H = 5$ has the highest mean and $H = 1$ the lowest.

Figure 7 removes autoencoder losses under full observation on the three Ant partitions. Every reduced objective still improves on MAPPO by a large margin, and the full objective is within one standard deviation of the best variant on every partition. On Ant $4 \times 2$, removing any single loss

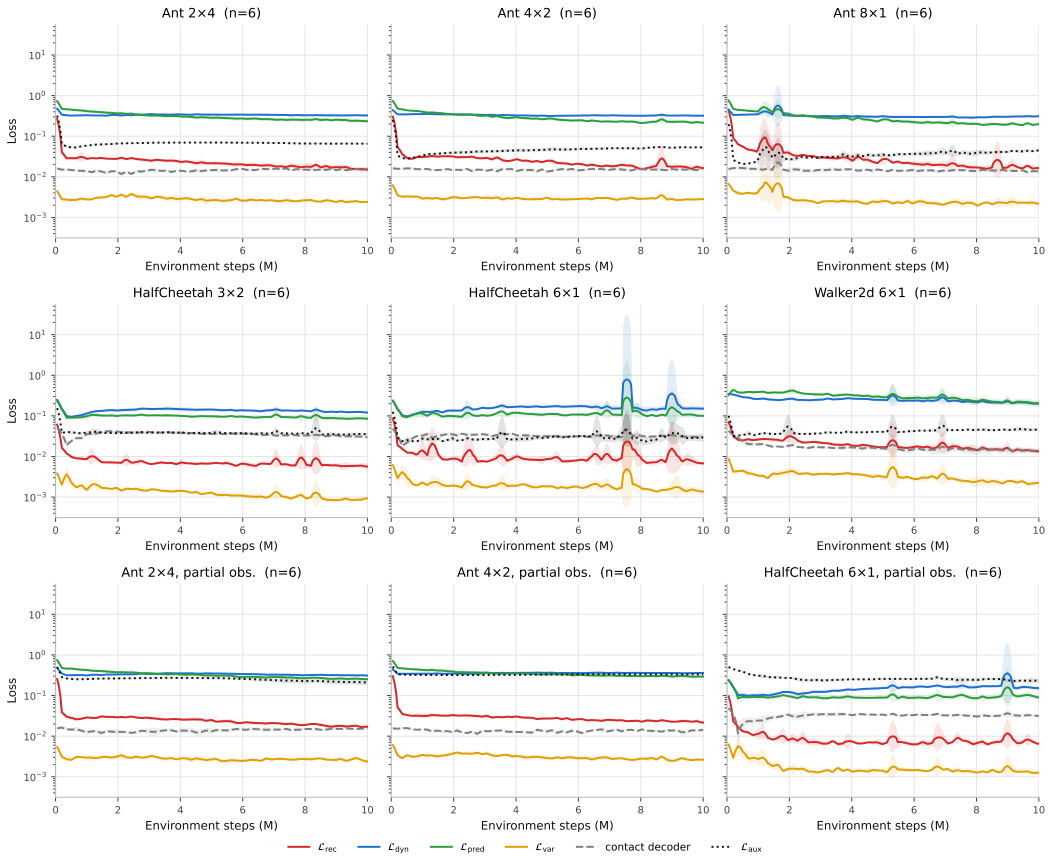



Figure 5: Training losses of the ART runs in Table 1: the four autoencoder losses of Equation 10, the contact decoder loss (Appendix A.2), and the unweighted auxiliary loss of Equation 12. Curves show the mean over seeds on a logarithmic scale and shaded regions one standard deviation of the logarithm.

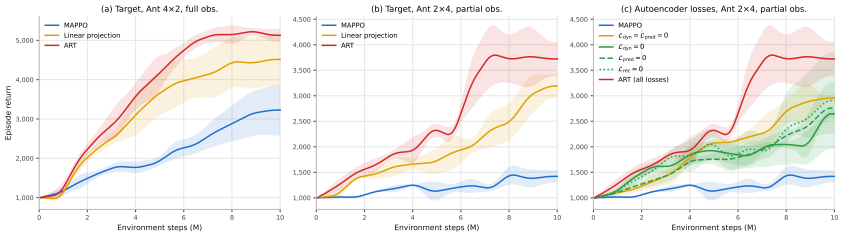



Figure 6: Learning curves for the ablations in Table 2. (a, b) ART with its Koopman target replaced by a fixed linear target. (c) Removing autoencoder losses under partial observation. Curves show the mean over matched seeds (six in panel a, three in panels b and c) and shaded regions one population standard deviation.

Table 5: Sensitivity to the auxiliary weight on Ant $4 \times 2$ under full observation. Values are mean $\pm$ population standard deviation over three seeds; the default is marked in bold.

| $\lambda_{\text{aux}}$ | 0.01 | 0.03 | **0.1** | 0.3 | 1.0 |
|---|---|---|---|---|---|
| Return | $3{,}506 \pm 647$ | $4{,}355 \pm 198$ | $5{,}128 \pm 223$ | $5{,}164 \pm 223$ | $5{,}150 \pm 166$ |

Table 6: Sensitivity to the prediction horizon $H$ on Ant $4 \times 2$ under full observation. Final evaluation return, mean $\pm$ population standard deviation over three seeds; the default is marked in bold.

| $H$ | 1 | 5 | **10** | 20 |
|---|---|---|---|---|
| Return | $4{,}867 \pm 146$ | $5{,}297 \pm 248$ | $5{,}128 \pm 223$ | $4{,}890 \pm 317$ |

leaves the return unchanged: $5{,}262 \pm 199$ without $\mathcal{L}_{\text{dyn}}$, $5{,}271 \pm 256$ without $\mathcal{L}_{\text{pred}}$, $5{,}105 \pm 261$ without $\mathcal{L}_{\text{rec}}$, and $5{,}335 \pm 54$ without the contact features, against $5{,}128 \pm 223$ for the full objective and $3{,}242 \pm 471$ for MAPPO on the same seeds. On Ant $2 \times 4$, an autoencoder trained with reconstruction alone reaches $4{,}847 \pm 128$, against $4{,}613 \pm 745$ for the full objective and $3{,}082 \pm 535$ for MAPPO. When every actor already observes the full state, any of these targets gives the actor a useful training signal. On Ant $8 \times 1$, an autoencoder trained with latent prediction alone reaches $3{,}588 \pm 548$, still well above MAPPO ($2{,}462 \pm 98$) but $31\%$ below the full objective ($5{,}239 \pm 221$). On the finest partition, reconstruction and decoded prediction are needed for the full gain.

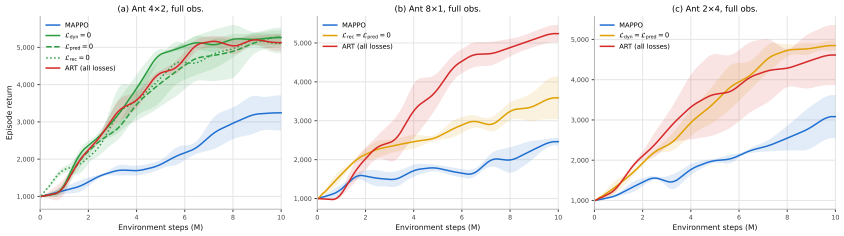



Figure 7: Autoencoder-loss ablations under full observation. Variants are named by the removed terms. Curves show the mean over three matched seeds and shaded regions one population standard deviation.