LATENT JOINT-ACTION MODELING FOR CENTRALIZED MULTI-AGENT WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

World models have become an effective tool for improving sample efficiency in multi-agent reinforcement learning by enabling policy learning from imagined trajectories. However, centralized multi-agent world models must directly model complex joint actions, whose dimensionality and interaction structure grow with the number of agents. Moreover, different components of the joint action can be strongly correlated and need not contribute independently to future environment dynamics. Inspired by recent advances in latent action modeling in video world models and vision-language-action models, we propose **LJAM**, a **L**atent **J**oint-**A**ction **M**odeling approach that represents the complex joint-action space through a structured latent effect space. Specifically, LJAM contextualizes each agent action with the current state and factorizes the resulting action tokens into a fixed set of collective, dynamics-relevant latent effects. We further introduce a predictive-error-aware confidence mechanism for weighting imagined policy updates. Experiments on multi-agent continuous-control benchmarks show that LJAM achieves state-of-the-art performance, while ablations demonstrate the importance of latent joint-action modeling and confidence-weighted updates.

1 INTRODUCTION

World models enable agents to improve their policies through imagined trajectories (Hafner et al., 2020; 2021; 2025). By learning predictive environment dynamics, model-based reinforcement learning (MBRL) can reduce costly interactions and improve sample efficiency. This is particularly attractive in multi-agent reinforcement learning (MARL), where learning is further complicated by interactions among multiple agents (Wang et al., 2022; Venugopal et al., 2024).

In a typical centralized multi-agent setting, world models predict environment dynamics from globally available state and action information (Egorov & Shpilman, 2022; Venugopal et al., 2024). Unlike single-agent world models, they must reason about joint actions from multiple agents acting simultaneously. As the number of agents grows, these actions involve increasingly complex cross-agent dependencies, while different action components can be strongly correlated and need not contribute independently to future dynamics. Recent multi-agent world models have explored different ways of organizing global and agent-wise dynamics information (Zeng & Zhang, 2025; Xue et al., 2026; Zhang et al., 2025b), yet representing the coupled effects of simultaneous actions remains challenging. As illustrated in Figure 1, one strategy is to directly condition centralized dynamics prediction on the complete joint action. Another is to introduce sequential structure across agents. For example, DIMA progressively reveals agents' actions to resolve uncertainty in multi-agent dynamics (Zhang et al., 2025c), while SeqWM factorizes joint dynamics into autoregressive agent-wise world models conditioned on preceding predicted trajectories (Zhao et al., 2026). These approaches improve the modeling of cross-agent dependencies, but they still operate directly on explicit agent actions or their sequential factorization.

We therefore hypothesize that different components of the raw joint action may contain correlated or redundant information with respect to future dynamics. The world model need not treat all joint-action components independently, but instead should preserve action information relevant to predicting future state transitions. This motivates learning a structured latent joint-action representation before dynamics prediction. Recent advances in latent-action modeling support this perspective. Latent-action world mod-

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.

els learn compressed action representations that preserve behaviorally or dynamically relevant information (Wang et al., 2026a), while hierarchical world models represent temporally extended behaviors through latent macro-actions for long-horizon prediction and planning (Zhang et al., 2026b). Building on these advances, we seek a structured latent effect space that captures how simultaneous multi-agent actions collectively influence future environment dynamics.

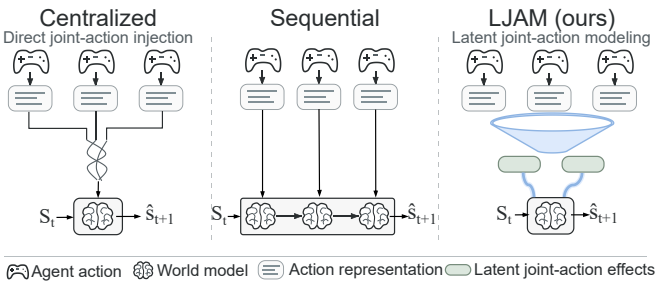


Figure 1: Conceptual comparison of representative action-conditioning strategies. Joint actions can be provided directly or structured sequentially across agents, whereas LJAM models simultaneous actions in a latent joint-action space before centralized dynamics prediction.

Based on this idea, we propose **LJAM**, a **L**atent **J**oint-**A**ction **M**odeling approach. LJAM first constructs state-conditioned action tokens using the current state, agent identity, and action time, and then organizes them into a fixed set of latent action effects for centralized dynamics prediction. This design reflects the observation that different agents may induce redundant, complementary, or similar dynamical consequences, so the number of agents need not correspond to the number of distinct action effects. The latent effects are learned jointly with the world model, allowing the representation to preserve information most relevant to predicting future dynamics.

Beyond joint-action representation, another challenge arises when the learned world model is used for policy optimization. Actor and critic updates depend on imagined trajectories, whose reliability deteriorates as model errors accumulate over rollout. Once imagined transitions are used for learning, prediction errors become optimization errors: inaccurate transitions can bias both value targets and policy gradients. We therefore introduce a prediction-error-aware confidence mechanism that estimates transition reliability and reweights actor and critic updates accordingly.

Our contributions can be summarized as follows: **(1)** We introduce a new view of multi-agent world modeling in which the joint-action space is represented through a compact **latent effect space**. LJAM contextualizes each agent action with the current state and factorizes the resulting action tokens into a fixed set of collective, dynamics-relevant latent effects. **(2)** We develop a **prediction-error-aware confidence mechanism** for imagination-based policy optimization, which modulates actor and critic updates according to the estimated reliability of model-generated transitions and mitigates the propagation of world-model errors into policy learning. **(3)** We demonstrate across multi-agent continuous-control benchmarks that LJAM improves dynamics prediction and policy performance. Detailed ablations and representation analyses attribute these gains to latent effect factorization and confidence-weighted policy optimization.

2 Related Work

2.1 Multi-Agent World Models

World models have recently been extended to multi-agent reinforcement learning to improve sample efficiency through model-generated trajectories. MAMBA applies model-based learning to cooperative multi-agent settings and leverages inter-agent communication to support world-model learning and imagined policy optimization (Egorov & Shpilman, 2022). MABL further introduces a bi-level latent-variable world model, where global information available during training is incorporated into a global latent state that guides the learning of agent-specific latent states, while preserving decentralized policy execution (Venugopal et al., 2024).

Subsequent work has explored alternative ways of organizing multi-agent dynamics modeling. MARIE learns decentralized local dynamics while aggregating representations from multiple agents through a centralized module, aiming to combine scalable local modeling with global interaction information (Zhang et al., 2025b). Other approaches move further toward decentralized world mod-

eling: Zeng and Zhang study efficient information sharing among decentralized agent-wise world models (Zeng & Zhang, 2025), while DMAWM learns disentangled multi-agent latent representations for decentralized control (Xue et al., 2026).

These methods mainly differ in how global information and agent-wise dynamics are organized within the world model. LJAM instead focuses on a complementary problem: how simultaneous multi-agent actions should be represented before centralized dynamics prediction. Specifically, LJAM organizes the joint action into a structured set of latent action effects that capture information relevant to future dynamics.

2.2 Sequential Multi-Agent World Modeling

Recent work has explored sequential formulations to reduce the difficulty of modeling complex multi-agent dynamics. DIMA introduces a diffusion-inspired world model that progressively reveals agents' actions to resolve uncertainty in state transitions (Zhang et al., 2025c), while SeqWM decomposes joint dynamics into autoregressive agent-wise world models, where each agent conditions on the predicted trajectories of preceding agents (Zhao et al., 2026). Although their mechanisms differ, both approaches introduce sequential structure over agents to simplify multi-agent world modeling. LJAM takes a complementary direction: instead of sequentially decomposing the prediction process, it retains centralized dynamics prediction and represents simultaneous agent actions through a structured latent effect space.

2.3 Latent Actions and Action Abstraction

Latent action representations have emerged as a way to capture action-relevant structure from observations without requiring explicit action labels. LAPO recovers latent action information directly from observed dynamics (Schmidt & Jiang, 2024), while Genie learns a discrete latent action space from unlabeled videos to enable action-controllable environment generation (Bruce et al., 2024). LAPA further uses discrete latent actions as supervision for vision-language-action pretraining (Ye et al., 2025). Recent analyses have examined what such representations actually capture, highlighting the challenge of separating controllable changes from exogenous visual dynamics (Zhang et al., 2025a), while LAOF improves robustness to action-irrelevant distractors using optical-flow constraints (Bu et al., 2026).

Latent actions have become prominent in vision-language-action models. Moto represents visual dynamics using latent motion tokens (Chen et al., 2025), and UniVLA learns task-centric latent actions for cross-embodiment policy transfer (Bu et al., 2025). Recent approaches strengthen the connection between latent representations and executable control: villa-X grounds latent actions in proprioceptive dynamics and jointly models latent and robot actions (Chen et al., 2026), while ViPRA predicts both future observations and motion-centric latent actions before decoding them into continuous robot controls (Routray et al., 2026). In generative world modeling, AdaWorld conditions an autoregressive world model on self-supervised latent actions (Gao et al., 2025), while CoLA-World jointly learns the latent action model and world model (Wang et al., 2026a). FLAM extends latent-action modeling to multi-entity scenes through factor-specific latent actions (Wang et al., 2026b). DiLA disentangles action-relevant structure from visual content to learn transferable latent actions (Zhang et al., 2026a), whereas Motus unifies latent-action learning, video generation, and robot control within a single world model (Bi et al., 2026). Another related direction is temporal action abstraction: HWM compresses sequences of primitive actions into latent macro-actions for long-horizon planning (Zhang et al., 2026b), while FAST uses a compression-based tokenization scheme for robot action sequences (Pertsch et al., 2025). LJAM differs from these approaches in that it operates on observed simultaneous multi-agent actions and organizes their collective dynamical effects into a structured latent effect space before centralized dynamics prediction. Although FLAM also employs slot-based factorization, the two methods address different modeling problems. FLAM infers factor-specific latent actions from action-free multi-entity videos, whereas LJAM starts from observed agent actions and factorizes their collective dynamical consequences into latent action effects for centralized multi-agent dynamics prediction.

3 PRELIMINARIES

Problem Formulation. We consider a fully cooperative multi-agent decision process with N agents. At timestep t, the environment is characterized by a global state $\mathbf{s}_t \in \mathcal{S}$, and each agent i receives a local observation $\mathbf{o}_t^i \in \mathcal{O}^i$ and selects an action $\mathbf{a}_t^i \in \mathcal{A}^i$ according to its policy $\pi^i(\mathbf{a}_t^i \mid \mathbf{o}_t^i)$. The simultaneous actions form the joint action $\mathbf{a}_t = (\mathbf{a}_t^1, \ldots, \mathbf{a}_t^N) \in \mathcal{A} = \prod_{i=1}^N \mathcal{A}^i$. The environment transitions according to $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ and produces a shared reward r_t. The objective is to learn a joint policy $\pi = \prod_{i=1}^N \pi^i$ that maximizes the expected discounted return $\mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t r_t\right]$.

Centralized World Modeling. A centralized multi-agent world model learns predictive environment dynamics using global information and the joint action (Egorov & Shpilman, 2022; Venugopal et al., 2024). Let $\mathbf{c}_t$ denote the state context constructed from the current state or a recent state history, and let $\mathbf{z}_t^a = g_\phi(\mathbf{a}_t)$ denote an encoded joint-action representation. A conventional centralized transition model can then be written as

$$\hat{\mathbf{s}}_{t+1} = f_\theta\left(\mathbf{c}_t, \mathbf{z}_t^a\right), \qquad \mathbf{z}_t^a = g_\phi(\mathbf{a}_t), \tag{1}$$

where f_θ denotes the learned dynamics model. Repeated application of f_θ produces imagined trajectories for downstream policy optimization (Hafner et al., 2020; 2021; 2025). However, directly encoding the raw joint action still requires the model to capture increasingly complex cross-agent interactions as the number of agents grows. Section 4 introduces LJAM to learn a latent joint-action representation that reduces redundant action information before centralized dynamics prediction.

4 METHOD

Figure 2 illustrates the overall architecture of LJAM. Given the state context and simultaneous actions of multiple agents, LJAM maps the raw joint action into a structured latent joint-action representation, aiming to reduce redundant action information while preserving information relevant to future environment dynamics. The resulting latent action effects are integrated with the state context for centralized dynamics prediction and multi-step imagined rollouts. In addition, a predictive-error-aware confidence mechanism estimates the reliability of imagined transitions and reweights actor and critic updates during policy learning.

4.1 LATENT JOINT-ACTION MODELING

Directly conditioning a centralized world model on the raw joint action requires the transition model to jointly represent agent actions and their cross-agent dependencies. Moreover, different components of the joint action may contain correlated or redundant information with respect to future dynamics. LJAM therefore maps simultaneous agent actions into a latent joint-action representation before centralized dynamics prediction. Specifically, we first construct state-conditioned and identity-aware action tokens and then aggregate them into a set of *latent action effects*.

State-conditioned action representations. At each timestep, all agents share the same global state, while their actions differ across agents. Directly concatenating the same state with every agent action would therefore repeatedly encode identical state information. We instead encode the shared state once and use it to generate Feature-wise Linear Modulation (FiLM) parameters, which modulate each agent-specific action representation. In this way, the action features remain agent-specific while being explicitly conditioned on the current environment state.

Specifically, each agent action $\mathbf{a}_t^i$ is encoded by a two-layer MLP shared across agents:

$$\tilde{\mathbf{x}}_t^i = \mathrm{MLP}_a\left(\mathbf{a}_t^i\right). \tag{2}$$

To account for the state-dependent effect of each action, we apply FiLM (Perez et al., 2018) using the state context $\mathbf{c}_t$:

$$(\boldsymbol{\gamma}_t, \boldsymbol{\beta}_t) = \mathrm{MLP}_{\mathrm{FiLM}}(\mathbf{c}_t), \tag{3}$$

where $\boldsymbol{\gamma}_t$ and $\boldsymbol{\beta}_t$ denote the feature-wise scaling and shifting parameters, respectively. The modulated action representation is then given by

$$\mathbf{h}_t^i = \mathrm{LN}\big((1 + \boldsymbol{\gamma}_t) \odot \tilde{\mathbf{x}}_t^i + \boldsymbol{\beta}_t + \mathbf{u}^i + \boldsymbol{\tau}_t^a\big), \tag{4}$$

where $\mathbf{u}^i$ is a learnable agent-identity embedding and $\boldsymbol{\tau}_t^a$ is a learnable action-time embedding. The resulting $\mathbf{h}_t^i$ denotes the state-conditioned action representation of agent i at timestep t. This conditioning allows each action representation to capture state-dependent dynamical effects without jointly re-encoding the shared state for every agent action, thereby reducing redundant state encoding.

Latent Joint-Action Effects. Although the state-conditioned representations capture the transition effect of each individual action, directly retaining all N agent representations still leads to a representation size that grows with the number of agents and preserves substantial redundancy among correlated action effects. We therefore aggregate these representations into a fixed number K of latent joint-action effects, each latent component aggregates information from dynamically related agent actions Inspired by the assignment mechanism of Slot Attention (Locatello et al., 2020), LJAM introduces K learnable latent queries to construct a structured representation of simultaneous agent actions, aiming to reduce redundant information in the raw joint action while preserving dynamics-relevant action information. Each query $\mathbf{q}_t^k$ is conditioned on the current state context and serves as a learnable reference for extracting one component of the latent joint-action representation. For clarity, we describe a single attention head below, while the implementation uses multi-head attention. The compatibility between action representation i and latent query k is computed as

$$\ell_{ik} = \frac{\left(\mathbf{W}_q \,\mathrm{LN}(\mathbf{q}_t^k)\right)^\top \left(\mathbf{W}_k \,\mathrm{LN}(\mathbf{h}_t^i)\right)}{\sqrt{d}}, \tag{5}$$

where $\mathbf{W}_q$ and $\mathbf{W}_k$ are learnable query and key projection matrices, respectively, and d denotes the feature dimension of each attention head.

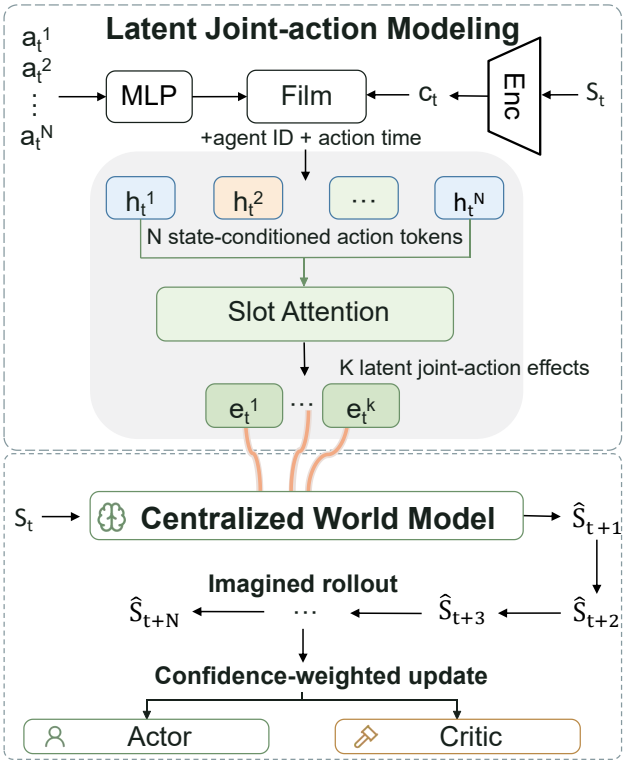


Figure 2: Overview of LJAM.

We normalize the scores over the K latent queries and subsequently renormalize them across agent representations:

$$\alpha_{ik} = \frac{\exp(\ell_{ik})}{\sum_{k'=1}^{K} \exp(\ell_{ik'})}, \qquad w_{ik} = \frac{\alpha_{ik}}{\sum_{j=1}^{N} \alpha_{jk} + \epsilon}, \tag{6}$$

where ϵ is a small constant for numerical stability.

Each latent query then aggregates information from the state-conditioned action representations $\{\mathbf{h}_t^i\}_{i=1}^N$:

$$\mathbf{e}_t^k = \mathrm{FFN}\left(\mathbf{q}_t^k + \mathbf{W}_o \sum_{i=1}^{N} w_{ik} \mathbf{W}_v \,\mathrm{LN}(\mathbf{h}_t^i)\right), \tag{7}$$

where $\mathbf{W}_v$ is the learnable value projection matrix and $\mathbf{W}_o$ is the output projection applied to the aggregated representation. The residual feed-forward refinement is absorbed into $\mathrm{FFN}(\cdot)$ for notational simplicity.

The resulting latent action effects $\mathbf{E}_t = \{\mathbf{e}_t^k\}_{k=1}^K$ form a structured latent representation of the joint action and are subsequently used for centralized dynamics prediction.

4.2 CENTRALIZED WORLD MODEL LEARNING

Given the latent action effects $\mathbf{E}_t = \{\mathbf{e}_t^k\}_{k=1}^K$, we use the resulting latent joint-action representation to condition the centralized world model for dynamics prediction. Instead of directly conditioning on the raw joint action, the transition model integrates $\mathbf{s}_t$ with $\mathbf{E}_t$ to predict the next global state.

Specifically, the global state is first projected into the model feature space and then fused with the latent action effects through two successive fusion blocks. Each fusion block consists of a multi-head cross-attention layer followed by a feed-forward network, where the state representation serves as the query and the latent action effects serve as the keys and values:

$$\mathrm{Fusion}(\mathbf{s}_t, \mathbf{E}_t) = \mathrm{FFN}\left(\mathrm{CrossAttn}\left(\mathbf{s}_t, \mathbf{E}_t, \mathbf{E}_t\right)\right). \tag{8}$$

In practice, residual connections and LayerNorm are applied within each fusion block. Two such blocks are stacked sequentially, and the resulting state–action representation is mapped to the next global state through an MLP:

$$\hat{\mathbf{s}}_{t+1} = \mathrm{MLP}_s\left(\mathrm{Fusion}\left(\mathrm{Fusion}(\mathbf{s}_t, \mathbf{E}_t), \mathbf{E}_t\right)\right). \tag{9}$$

The transition model is trained using multi-step closed-loop prediction. At each prediction step, the predicted global state is recursively fed back into the transition model together with the corresponding action sequence. The dynamics objective is defined as

$$\mathcal{L}_{\mathrm{dyn}} = \frac{1}{T}\sum_{j=1}^{T} \|\hat{\mathbf{s}}_{t+j} - \mathbf{s}_{t+j}\|_2^2, \tag{10}$$

where T denotes the transition prediction horizon.

Imagined Rollout. Following DIMA (Zhang et al., 2025c), we adopt a VQ-based state-to-observation decoder (van den Oord et al., 2017) to recover agent-wise local observations from the predicted global state. Specifically, the predicted state is first projected by an MLP into a sequence of latent tokens, which are quantized using a shared VQ codebook and subsequently decoded by another MLP into the local observations of all agents:

$$\left\{\hat{\mathbf{o}}_{t+1}^i\right\}_{i=1}^N = D_{\mathrm{obs}}\left(\hat{\mathbf{s}}_{t+1}\right). \tag{11}$$

The reconstructed local observations are then provided to the decentralized actors to generate the next joint action. In parallel, a separate reward and continuation model predicts the reward and continuation probability required for imagined rollouts. Given the global state and joint action, the model produces

$$(\hat{r}_t, \hat{p}_t) = F_{\mathrm{rew}}\left(\mathbf{s}_t, \mathbf{a}_t\right), \tag{12}$$

where $\hat{r}_t$ denotes the predicted reward and $\hat{p}_t$ the predicted continuation probability. We implement F_{rew} as a Transformer that models the temporal sequence of states and joint actions. Together, the transition model, state-to-observation decoder, and reward/continuation model generate multi-step imagined trajectories for subsequent actor and critic optimization.

4.3 CONFIDENCE-WEIGHTED IMAGINED POLICY LEARNING

For policy optimization, we adopt a MAPPO-style actor–critic framework under centralized training and decentralized execution, with decentralized actors and a centralized critic. Policy learning from imagined trajectories is inherently affected by world-model errors, which accumulate over rollout horizons and make some imagined transitions substantially less reliable than others. Treating all imagined samples equally can therefore propagate model bias into both actor and critic updates. To mitigate this effect, we estimate the prediction error of each imagined transition and use it to weight its contribution to policy optimization. During world-model training, the prediction-error target at step h is defined as

$$y_{t,h} = \log\left(1 + \frac{1}{D_s}\|\hat{\mathbf{s}}_{t+h} - \mathbf{s}_{t+h}\|_2^2\right), \tag{13}$$

where D_s denotes the dimension of the global state. An auxiliary error head predicts $\hat{y}_{t,h}$ from the transition features and is optimized by

$$\mathcal{L}_{\mathrm{err}} = \frac{1}{T}\sum_{h=1}^{T}\left(\hat{y}_{t,h} - y_{t,h}\right)^2. \tag{14}$$

The features and error targets are detached when training the error head, such that this auxiliary objective does not modify the transition representation.

During imagination, ground-truth future states are unavailable. We therefore use the predicted error to construct a confidence weight. Since prediction errors naturally vary with rollout depth, we maintain horizon-specific running statistics and normalize the predicted error as

$$z_{t,h} = \frac{\hat{y}_{t,h} - \mu_h}{\sqrt{\sigma_h^2 + \epsilon}}. \tag{15}$$

The confidence weight is then computed as

$$c_{t,h} = \mathrm{clip}\left(\exp\left(-\alpha z_{t,h}\right), c_{\min}, c_{\max}\right), \tag{16}$$

where μ_h and σ_h^2 are maintained using exponential moving averages. Lower predicted errors relative to the corresponding rollout horizon lead to larger confidence weights, while transitions with relatively high predicted errors are downweighted.

Finally, the confidence weights are applied to both actor and critic objectives using a normalized weighted mean:

$$\mathcal{L}_{\text{actor}} = \frac{\sum_{t,h} c_{t,h} \mathcal{L}_{\text{actor}}^{t,h}}{\sum_{t,h} c_{t,h}}, \qquad \mathcal{L}_{\text{critic}} = \frac{\sum_{t,h} c_{t,h} \mathcal{L}_{\text{critic}}^{t,h}}{\sum_{t,h} c_{t,h}}. \tag{17}$$

This weighting mechanism reduces the contribution of imagined transitions with relatively high predicted errors during policy optimization.

5 Experiments

Following the experimental setup of DIMA, we evaluate LJAM on challenging multi-agent continuous-control benchmarks to investigate whether latent joint-action modeling improves world-model learning and downstream policy optimization. We compare LJAM against representative model-free MARL algorithms, including MAPPO, HAPPO, and HASAC (Yu et al., 2022; Kuba et al., 2022; Liu et al., 2024), as well as recent model-based multi-agent world-modeling methods, including MARIE and DIMA (Zhang et al., 2025b;c). Experiments are conducted on MAMuJoCo and Bi-DexHands, covering cooperative locomotion and high-dimensional dexterous manipulation tasks (Schroeder de Witt et al., 2020; Chen et al., 2024). All reported results are averaged over three random seeds, with 95% bootstrap confidence intervals.

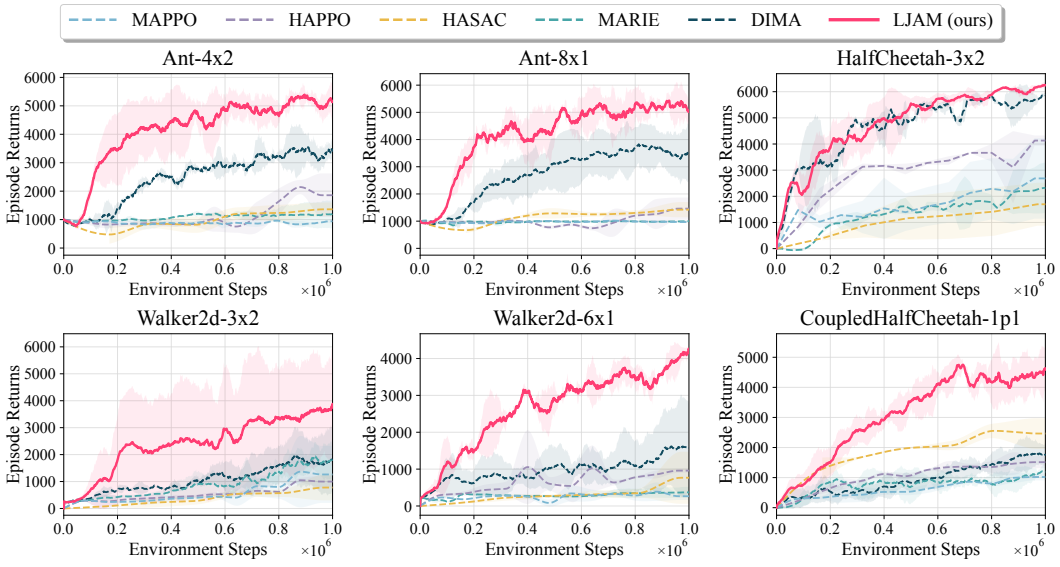


Figure 3: Curves of averaged episode returns for all methods in MAMuJoCo.

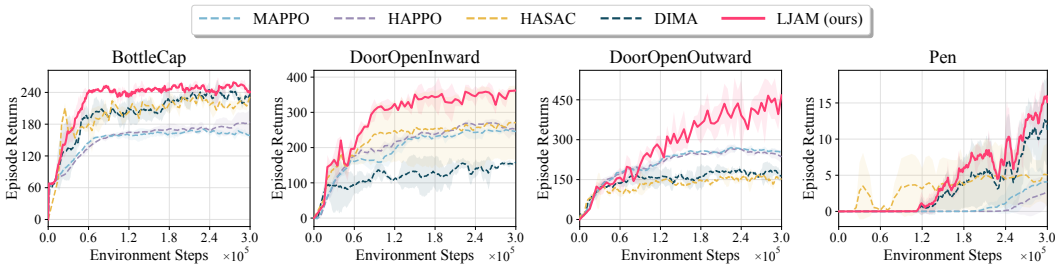


Figure 4: Curves of averaged episode returns for all methods in Bi-DexHands.

5.1 Results

Figures 3 and 4 report results averaged over three random seeds, with shaded regions indicating the corresponding confidence intervals. LJAM consistently outperforms the compared baselines across MAMuJoCo and Bi-DexHands. The gains are particularly evident on the Ant, Walker2d, and door-opening tasks, where LJAM achieves both faster learning and higher final returns. The advantage is more pronounced on tasks with stronger inter-agent coupling, suggesting that latent joint-action modeling is especially beneficial when raw joint actions contain more correlated and redundant information. Overall, these results indicate that modeling joint actions in a latent space provides a more effective conditioning representation for centralized world-model prediction.

5.2 Ablation and Analysis

In this subsection, we analyze LJAM by addressing four questions: (i) Is the slot-based formulation effective for constructing latent joint-action effects? (ii) Does confidence weighting improve policy learning from imagined trajectories? (iii) Does latent joint-action conditioning improve long-horizon dynamics prediction? and (iv) Does the learned latent joint-action representation preserve dynamics-relevant information?

(i) Is the slot-based formulation effective for constructing latent joint-action effects? Figure 5(a) evaluates how latent joint-action effects are constructed on Ant-4x2. We compare the slot-based module with three alternatives: standard cross-attention, an MLP-based mapping, and directly feeding joint-action tokens to the transition model without latent aggregation. All three alternatives perform worse than the slot-based formulation, supporting the benefit of structured latent aggregation. Figure 5(d) provides further insight: standard cross-attention yields highly similar attention patterns across the two latent representations, whereas the slot-based formulation produces more differentiated assignments across latent action effects, indicating a more structured decomposition of joint-action information for dynamics prediction.

(ii) Does confidence weighting improve policy learning from imagined trajectories? Figure 5(b) shows that removing confidence weighting leads to lower returns and larger training fluctuations, while the full model learns more consistently. This indicates that predictive-error-aware confidence weighting improves the stability and effectiveness of imagined policy updates. As an additional sensitivity analysis, Fig. 5(c) varies the number of latent action effects K. LJAM remains robust across the tested values, with $K = 2$ achieving slightly better overall performance and therefore used in the main experiments.

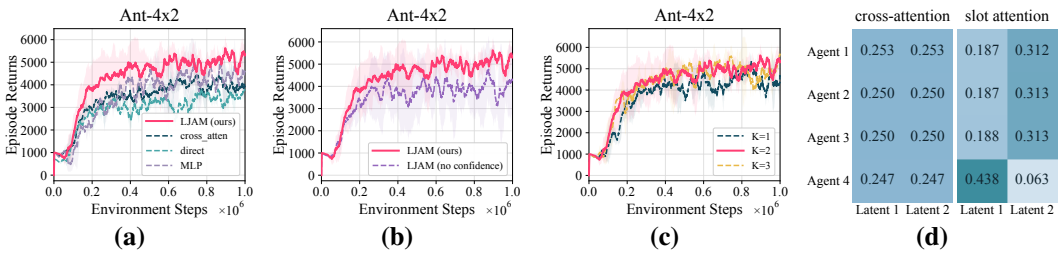


Figure 5: Ablation studies and attention visualization on Ant-4x2.

(iii) Does latent joint-action conditioning improve long-horizon dynamics prediction? We evaluate 15-step autoregressive prediction on CoupledHalfCheetah-1p1 using 96 shared trajectory windows, with 32 collected from each of the LJAM, DIMA, and MARIE policies. Each rollout is initialized with three ground-truth context frames and then uses the recorded future action sequence, recursively feeding model predictions back as input without future ground-truth observations. Prediction error is measured by Euclidean distance in the decoded joint observation space. Each method is evaluated with three trained checkpoints; curves report the mean with checkpoint-wise sample standard deviation. As shown in Table 1 and Figure 6, LJAM achieves the lowest prediction error

Table 1: Long-horizon prediction error (↓) at different horizons.

Method	$h = 1$	$h = 3$	$h = 6$	$h = 9$	$h = 12$	$h = 15$
LJAM (ours)	**1.52**	**1.80**	**2.02**	**2.15**	**2.20**	**2.18**
DIMA	2.50	2.72	2.90	3.00	3.10	3.15
MARIE	3.85	4.30	4.50	4.60	4.55	5.00

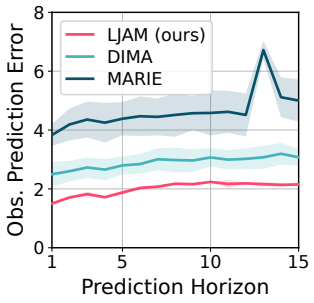


Figure 6: Observation prediction error over rollout horizons. Lower is better.

across all horizons and exhibits slower error accumulation. MARIE shows a noticeable error increase near $h = 13$ because its context reaches the positional encoding limit, after which its native cache-refresh mechanism reinitializes the KV cache using predicted observation tokens. Thus, later MARIE errors reflect both autoregressive error accumulation and truncated historical context.

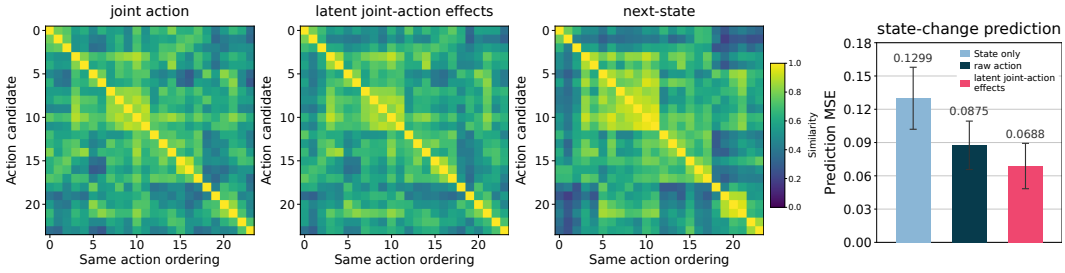


Figure 7: Joint-action representations and state-change prediction. The three heatmaps show pairwise similarities of raw joint actions, latent joint-action effects, and next-state effects, using the same action ordering. The bar chart compares state-change prediction MSE across input representations; error bars indicate standard deviation across held-out episodes.

(iv) Does the learned latent joint-action representation preserve dynamics-relevant information? We conduct a controlled intervention experiment on Ant-4x2 using 24 perturbed joint actions from the same environment state. As shown in Fig. 7, action pairs that are substantially different in the raw joint-action space remain well separated in the latent space, indicating that LJAM preserves important distinctions among joint actions while reorganizing them into a more structured representation. Moreover, the latent similarity pattern more closely matches that of the true next-state effects, suggesting that the learned representation emphasizes differences that are more relevant to environment dynamics. This is further supported by the probe results: the latent representation achieves a state-change prediction MSE of 0.0688, compared with 0.0875 for raw joint actions and 0.1299 for state-only inputs. Together, these results suggest that LJAM reduces redundant joint-action structure while preserving information useful for dynamics prediction.

6 Conclusion

We introduced LJAM, which models simultaneous multi-agent actions through latent action effects for centralized world-model prediction, together with a prediction-error-aware confidence mechanism for imagined policy optimization. Experiments on MAMuJoCo and Bi-DexHands show improved policy performance and dynamics prediction, while ablations and representation analyses validate the effectiveness of latent joint-action modeling and confidence weighting.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing, rephrasing, manuscript organization, LaTeX formatting, bibliographic checking, and code development and debugging. We also used these tools to improve the clarity and presentation of technical descriptions and experimental analyses. All AI-assisted text, references, and code were manually reviewed by the authors. AI-assisted code was inspected, tested, and validated before being used in the experiments, and all reported experimental results, analyses, and conclusions were checked against the actual implementation and experimental records. Generative AI tools were not used to fabricate experimental data or results. We take full responsibility for the final content of this work, including text, code, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work focuses on methodological research in multi-agent reinforcement learning and is evaluated entirely in simulated environments. It does not involve human subjects, personal data, or real-world deployment. We are not aware of any immediate ethical concerns arising from the proposed method or experimental setting.

REPRODUCIBILITY STATEMENT

We provide an anonymous implementation of LJAM as supplementary material, including the training and evaluation code used in our experiments. The main paper describes the model architecture, learning objectives, benchmark settings, and evaluation protocols, while additional implementation details, hyperparameters, and ablation configurations are provided in the appendix. The released code contains scripts for reproducing the main experiments on MAMuJoCo and Bi-DexHands, as well as the analyses reported in the paper. Together, these materials are intended to enable reproduction of the reported results.

REFERENCES

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A unified latent action world model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 35101–35113, 2026.

Jake Bruce, Michael D. Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie C. Y. Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments. In *International Conference on Machine Learning*, pp. 4603–4623, 2024.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Learning to act anywhere with task-centric latent actions. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.014.

Xizhou Bu, Jiexi Lyu, Fulei Sun, Ruichen Yang, Zhiqiang Ma, and Wei Li. LAOF: Robust latent action learning with optical flow constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 27334–27344, 2026.

Xiaoyu Chen, Hangxing Wei, Pushi Zhang, Chuheng Zhang, Kaixin Wang, Yanjiang Guo, Rushuai Yang, Yucen Wang, Xinquan Xiao, Li Zhao, Jianyu Chen, and Jiang Bian. villa-X: Enhancing latent action modeling in vision-language-action models. In *International Conference on Learning Representations*, 2026.

Yi Chen, Yuying Ge, Weiliang Tang, Yizhuo Li, Yixiao Ge, Mingyu Ding, Ying Shan, and Xihui Liu. Moto: Latent motion token as the bridging language for learning robot manipulation from videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19752–19763, 2025.

Yuanpei Chen, Yiran Geng, Fangwei Zhong, Jiaming Ji, Jiechuang Jiang, Zongqing Lu, Hao Dong, and Yaodong Yang. Bi-DexHands: Towards human-level bimanual dexterous manipulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):2804–2818, 2024. doi: 10.1109/TPAMI.2023.3339515.

Vladimir Egorov and Alexei Shpilman. Scalable multi-agent model-based reinforcement learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 381–390, 2022.

Shenyuan Gao, Siyuan Zhou, Yilun Du, Jun Zhang, and Chuang Gan. AdaWorld: Learning adaptable world models with latent actions. In *International Conference on Machine Learning*, pp. 18744–18771, 2025.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025. doi: 10.1038/s41586-025-08744-2.

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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=EcGGFkNTxdJ.

Jiarong Liu, Yifan Zhong, Siyi Hu, Haobo Fu, Qiang Fu, Xiaojun Chang, and Yaodong Yang. Maximum entropy heterogeneous-agent reinforcement learning. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=tmqOhBC4a5.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, pp. 11525–11538, 2020.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient action tokenization for vision-language-action models. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.012.

Sandeep Kumar Routray, Hengkai Pan, Unnat Jain, Shikhar Bahl, and Deepak Pathak. ViPRA: Video prediction for robot actions. In *International Conference on Learning Representations*, 2026.

Dominik Schmidt and Minqi Jiang. Learning to act without actions. In *International Conference on Learning Representations*, 2024.

Christian Schroeder de Witt, Bei Peng, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Deep multi-agent reinforcement learning for decentralized continuous cooperative control, 2020. URL https://arxiv.org/abs/2003.06709v1.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Aravind Venugopal, Stephanie Milani, Fei Fang, and Balaraman Ravindran. MABL: Bi-level latent-variable world model for sample-efficient multi-agent reinforcement learning. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 1865–1873, 2024.

Xihuai Wang, Zhicheng Zhang, and Weinan Zhang. Model-based multi-agent reinforcement learning: Recent progress and prospects. *CoRR*, abs/2203.10603, 2022.

Yucen Wang, Fengming Zhang, De-Chuan Zhan, Li Zhao, Kaixin Wang, and Jiang Bian. Co-evolving latent action world models. In *International Conference on Machine Learning*, 2026a. URL https://icml.cc/virtual/2026/poster/62210.

Zizhao Wang, Chang Shi, Jiaheng Hu, Kevin Rohling, Roberto Martín-Martín, Amy Zhang, and Peter Stone. Factored latent action world models. In *International Conference on Machine Learning*, 2026b.

Di Xue, Jing Jiang, Shaowei Zhang, Wenhao Guo, Lei Yuan, Zongzhang Zhang, and Yang Yu. Learning disentangled multi-agent world model for decentralized control. In *International Conference on Machine Learning*, 2026.

Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Se June Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, Lars Liden, Kimin Lee, Jianfeng Gao, Luke Zettlemoyer, Dieter Fox, and Minjoon Seo. Latent action pretraining from videos. In *International Conference on Learning Representations*, 2025.

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 *Advances in Neural Information Processing Systems*, volume 35, pp. 24611–24624, 2022.

Xiaoling Zeng and Qi Zhang. Efficient information sharing for training decentralized multi-agent world models. *Reinforcement Learning Journal*, 6:909–922, 2025.

Chuheng Zhang, Tim Pearce, Pushi Zhang, Kaixin Wang, Xiaoyu Chen, Wei Shen, Li Zhao, and Jiang Bian. What do latent action models actually learn? In *Advances in Neural Information Processing Systems*, volume 38, 2025a.

Tianqiu Zhang, Muyang Lyu, Yufan Zhang, Fang Fang, and Si Wu. DiLA: Disentangled latent action world models. In *International Conference on Machine Learning*, 2026a.

Wancong Zhang, Basile Terver, Artem Zholus, Soham Chitnis, Harsh Sutaria, Mido Assran, Randall Balestriero, Amir Bar, Adrien Bardes, Yann LeCun, and Nicolas Ballas. Hierarchical planning with latent world models, 2026b.

Yang Zhang, Chenjia Bai, Bin Zhao, Junchi Yan, Xiu Li, and Xuelong Li. Decentralized transformers with centralized aggregation are sample-efficient multi-agent world models. *Transactions on Machine Learning Research*, 2025b.

Yang Zhang, Xinran Li, Jianing Ye, Shuang Qiu, Delin Qu, Xiu Li, Chongjie Zhang, and Chenjia Bai. Revisiting multi-agent world modeling from a diffusion-inspired perspective. In *Advances in Neural Information Processing Systems*, volume 38, 2025c.

Zijie Zhao, Honglei Guo, Shengqian Chen, Kaixuan Xu, Bo Jiang, Yuanheng Zhu, and Dongbin Zhao. Empowering multi-robot cooperation via sequential world models. In *International Conference on Learning Representations*, 2026.

A Environments

We evaluate LJAM on two multi-agent continuous-control benchmarks, MAMuJoCo (Schroeder de Witt et al., 2020) and Bi-DexHands (Chen et al., 2024). They cover complementary coordination regimes: cooperative locomotion and high-dimensional dexterous manipulation. Representative environments are shown in Fig. 8.

MAMuJoCo. MAMuJoCo extends MuJoCo locomotion tasks to cooperative multi-agent continuous control by partitioning the actuated joints of a single robotic system among multiple agents. Each agent controls a subset of the joints, while all agents share the task objective and must coordinate to generate coherent whole-body motion. The resulting joint actions are strongly coupled through the shared physical dynamics, making MAMuJoCo a natural testbed for evaluating latent joint-action modeling.



(a) HalfCheetah-2x3 (b) DoorOpenOutward (c) Pen

Figure 8: Representative environments from the evaluated benchmarks: (a) MAMuJoCo HalfCheetah and (b–c) Bi-DexHands manipulation tasks.

Bi-DexHands. Bi-DexHands is a high-dimensional bimanual dexterous manipulation benchmark built in Isaac Gym. It requires coordinated control of two dexterous robotic hands across contact-rich manipulation tasks. Compared with locomotion, Bi-DexHands involves higher-dimensional action spaces and more complex cross-agent interactions arising from finger coordination, object contact, and bimanual manipulation, providing a challenging setting for evaluating the scalability and expressiveness of LJAM.

B Additional Results

B.1 Final Performance Summary

Final Performance Summary. For completeness, we report the final evaluation returns of all compared methods on MAMuJoCo and Bi-DexHands in Table 2. Results are reported as the mean $\pm$ sample standard deviation over three random seeds. The best result for each task is highlighted in light blue, and the second-best result is underlined.

Table 2: Final episode returns across MAMuJoCo and Bi-DexHands tasks. Each entry reports the mean final return and sample standard deviation over three random seeds. The best result is highlighted in light blue, and the second-best result is underlined.

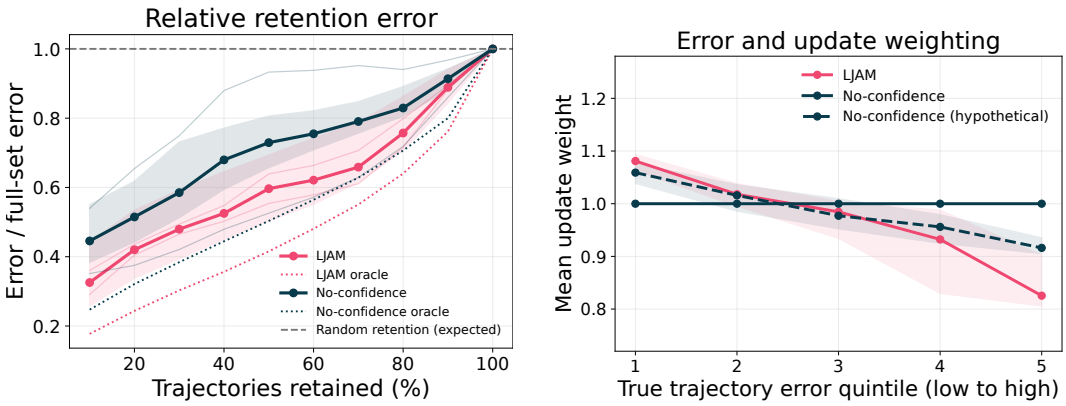
Benchmark	Task	LJAM (ours)	DIMA	MARIE	HAPPO	HASAC	MAPPO
6*MAMuJoCo	Ant-4×2	$5929.93_{\pm 301.64}$	$3489.50_{\pm 135.06}$	$1184.49_{\pm 218.06}$	$1856.66_{\pm 769.23}$	$1362.31_{\pm 237.77}$	$879.62_{\pm 91.40}$
	Ant-8×1	$6037.72_{\pm 42.30}$	$3545.87_{\pm 282.72}$	$999.50_{\pm 5.77}$	$1460.81_{\pm 407.12}$	$1422.31_{\pm 280.86}$	$968.16_{\pm 29.30}$
	HalfCheetah-3×2	$6188.49_{\pm 129.08}$	$6107.75_{\pm 99.02}$	$2338.13_{\pm 977.53}$	$4131.08_{\pm 195.24}$	$1695.85_{\pm 820.99}$	$2684.52_{\pm 1538.60}$
	Walker2d-3×2	$3754.58_{\pm 2569.32}$	$2148.31_{\pm 2052.07}$	$1792.32_{\pm 1261.96}$	$998.03_{\pm 215.57}$	$776.39_{\pm 476.07}$	$1262.85_{\pm 1058.81}$
	Walker2d-6×1	$3059.14_{\pm 1796.39}$	$1649.47_{\pm 1700.87}$	$355.70_{\pm 141.53}$	$958.84_{\pm 59.02}$	$765.81_{\pm 706.39}$	$265.24_{\pm 103.13}$
	CoupledHalfCheetah-1p1	$4365.47_{\pm 2449.76}$	$1785.46_{\pm 810.32}$	$1146.04_{\pm 267.49}$	$1513.53_{\pm 200.47}$	$2464.93_{\pm 511.33}$	$1019.84_{\pm 112.94}$
4*Bi-DexHands	BottleCap	$248.06_{\pm 2.52}$	$238.91_{\pm 27.59}$	—	$183.78_{\pm 21.69}$	$232.54_{\pm 4.86}$	$160.97_{\pm 8.63}$
	DoorOpenInward	$361.96_{\pm 11.85}$	$162.88_{\pm 3.16}$	—	$251.90_{\pm 20.78}$	$182.48_{\pm 2.17}$	$245.72_{\pm 2.76}$
	DoorOpenOutward	$467.03_{\pm 40.21}$	$162.81_{\pm 34.89}$	—	$235.95_{\pm 24.27}$	$154.67_{\pm 16.69}$	$252.27_{\pm 12.90}$
	Pen	$15.89_{\pm 3.21}$	$11.30_{\pm 8.26}$	—	$2.79_{\pm 1.70}$	$5.06_{\pm 4.36}$	$3.94_{\pm 1.09}$

Table 3: Training efficiency on Ant-4×2 using an NVIDIA RTX 4090 GPU. Training time denotes the total wall-clock time required for training, and GPU memory denotes the peak memory usage.

Method	Training Time	GPU Memory
DIMA	43 h	3.15 GB
MARIE	89 h	3.10 GB
LJAM (ours)	**8 h**	**1.90 GB**

Training Efficiency. We further compare the computational efficiency of LJAM with DIMA and MARIE on Ant-4×2. As shown in Table 3, LJAM requires approximately 8 hours of training and

1.90 GB of GPU memory on an NVIDIA RTX 4090, substantially reducing both training time and memory usage compared with the two model-based baselines.



(a) Confidence-based trajectory retention.

(b) Prediction error and update weights.

Figure 9: Analysis of confidence weighting on Ant-v2 with a 4×2 agent partition. (a) Mean prediction error among retained trajectories, normalized by the full-set mean error. Trajectories are ranked using the predicted reliability score. The horizontal dashed line denotes expected random retention, and dotted curves denote oracle rankings based on observed errors. (b) Mean update weights across quintiles of observed trajectory error, ordered from lowest to highest. No-confidence retains the error predictor but applies unit weights; its dashed curve shows hypothetical weights if weighting were enabled. Thick curves average two training seeds. Shaded regions show 95% source-episode bootstrap intervals, conditional on these fixed training seeds.

Confidence-based trajectory reliability. We investigate whether LJAM's confidence signal identifies reliable model predictions and assigns lower weights to inaccurate ones. We freeze the final checkpoints of LJAM and its No-confidence ablation from two training seeds. The ablation retains the error-prediction head and its training, but disables confidence weighting in actor–critic updates. All checkpoints are evaluated on matched real-state histories and recorded action sequences from 24 held-out episodes, yielding 397 trajectory segments and 5,934 valid prediction steps. Predictions are recursively rolled out for up to 15 steps. State prediction errors are normalized using a common per-dimension scale estimated from eight separate reference episodes.

For each trajectory, we average the horizon-wise standardized predicted errors to obtain a reliability score, with lower scores indicating more reliable predictions. Figure 9a reports the error obtained by retaining increasingly large fractions of the highest-ranked trajectories. When retaining the most reliable 20%, LJAM achieves a relative error of 0.420, compared with 0.515 for No-confidence and 1.0 in expectation for random retention. Thus, LJAM's confidence signal identifies a subset whose prediction error is substantially lower than the full-set average. This is a selection effect rather than a reduction in the errors of the underlying predictions.

Error-dependent update weighting. Figure 9b groups trajectories into five quintiles according to their observed prediction errors. For LJAM, the mean weight decreases from 1.081 in the lowest-error quintile to 0.825 in the highest-error quintile, whereas No-confidence applies a constant weight of 1. These weights are reconstructed on the evaluation trajectories using the confidence calibration state stored in each checkpoint; observed test errors are used only for evaluation and grouping. The decreasing trend indicates that LJAM preferentially weights more accurate predictions and down-weights less reliable ones. The hypothetical No-confidence curve further illustrates that retaining an error predictor alone does not apply its reliability signal to the actor–critic objective.

Together, these results support the intended mechanism of confidence weighting: identifying relatively reliable predictions and assigning them greater weight. They do not, by themselves, establish improved policy returns or reduced gradient error. Moreover, the error predictor underestimates the magnitude of long-horizon recursive errors, so useful reliability ranking should not be conflated

Table 4: Optimizer settings. LR, WD, and Clip denote learning rate, weight decay, and gradient-norm clipping threshold, respectively. The transition optimizer also updates the error-prediction head.

Module	Optimizer	LR	WD	ϵ	Clip
Transition	AdamW	10^{-4}	0.01	10^{-8}	1
Reward/continuation	AdamW	10^{-4}	0.01	10^{-8}	10
State decoder/VQ	AdamW	3×10^{-4}	0.01	10^{-8}	10
Actor	Adam	5×10^{-4}	0	10^{-5}	10
Critic	Adam	5×10^{-4}	0	10^{-5}	10

with accurate numerical calibration. The analysis uses fixed recorded actions and two training seeds, rather than reproducing the full distribution of policy-generated imagined rollouts.

C Reproducibility Details

C.1 Baseline Implementations

Model-Free Baselines. We implement MAPPO, HAPPO, and HASAC using the official HARL codebase: `https://github.com/PKU-MARL/HARL`. HARL provides unified implementations of these algorithms together with interfaces for both MAMuJoCo and Bi-DexHands. We use the default configurations provided by the official repository and do not perform additional baseline-specific hyperparameter tuning. Only environment-specific settings, random seeds, and the total number of environment steps are adjusted to match our experimental protocol.

Model-Based Baselines. For MARIE, we use the official implementation released by the authors: `https://github.com/breez3young/MARIE`. We follow the default/recommended configurations provided in the repository for MAMuJoCo experiments, without additional hyperparameter tuning.

For DIMA, we use the official implementation: `https://github.com/breez3young/DIMA`. Similarly, we retain the default hyperparameters and training settings provided by the authors, modifying only the environment, agent configuration, random seed, and training horizon when required by the corresponding task.

Excluded Baselines. We do not include MAMBA (Egorov & Shpilman, 2022) in our empirical comparison. The official MAMBA implementation (`https://github.com/jbr-ai-labs/mamba`) supports Flatland and StarCraft environments, whereas our experiments focus on continuous-control benchmarks such as MAMuJoCo and Bi-DexHands. More importantly, the released implementation is designed around discrete actions: its actor and controller use one-hot categorical action distributions and discrete action-availability masks. Adapting MAMBA to our benchmarks would therefore require substantial changes to the policy output distribution, action-sampling pipeline, environment interface, and related rollout components. Such modifications would depart considerably from the released implementation and make it difficult to ensure a faithful comparison with the original method. We therefore restrict our comparison to methods with implementations directly applicable to, or already established on, the continuous-control settings considered in this work.

C.2 Training Hyperparameters

C.2.1 Optimization

Table 4 summarizes the optimizer settings. All Adam and AdamW optimizers use $(\beta_1, \beta_2) = (0.9, 0.999)$. Weight decay for the transition model and the reward/continuation model is applied through parameter groups, with selected bias and normalization parameters excluded. Only the transition optimizer uses a learning-rate warm-up of 100 optimizer updates. The gradient accumulation factor is one.

Table 5: Batch sizes and sequence lengths. The actor/critic batch setting is distinct from the imagined-rollout batch size.

Hyperparameter	Value
State decoder batch size	512
World-model batch size	64
Imagined-rollout batch size	600
Imagination horizon	15
History conditioning length	3
Multi-step transition training length	3
Actor/critic batch setting	30
Number of mini-batches	1

Table 6: Transition-model architecture and training settings.

Hyperparameter	Value
Action encoder	Attention
Hidden dimension	256
History MLP hidden dimension	512
History MLP depth	2
Number of action slots K	2
Slot refinement depth	2
Configured slot attention heads	4
Fusion depth / attention heads	2 / 4
Competitive action slots	Enabled
FiLM / FiLM zero initialization	Enabled / Enabled
Agent identity conditioning	Enabled
Shared action adapter	Enabled
Number of transition prediction heads	1
Bootstrap probability	1.0

C.2.2 Data Collection and Update Scheduling

The replay buffer stores up to 250,000 entries, and training starts once its size reaches 5,000. Transition and reward/continuation updates are scheduled using budgets accumulated from newly collected real environment steps. Each new step adds 0.2 updates to each of these two budgets. Both budgets are initialized to 200, and at most 200 updates per module are performed at a single training trigger. Thus, 200 is an update cap rather than a fixed number of updates at every trigger.

The actor update budget accumulates at a rate of 0.005 updates per new real environment step, with at most five updates per trigger. Actor-budget activation additionally requires the training counter to exceed nine. Table 5 distinguishes batch sizes used by different training components.

C.2.3 Transition Model

The transition model predicts residual state changes using a zero-initialized output head. It is trained with mean squared error over three closed-loop prediction steps. Predicted states are fed back as inputs and detached at each step, so gradients do not propagate through the feedback predictions across steps. The error-prediction auxiliary loss has weight 0.1, whereas the object-state auxiliary loss is disabled for Ant. Table 6 lists the remaining architecture settings.

A bootstrap probability of 1.0 means that random bootstrap subsampling is not used. For the MLP action-encoder ablation, the action encoder is replaced by an MLP; the attention-slot architecture above describes the full model.

C.2.4 State Representation and Auxiliary Networks

The state representation uses vector quantization with 12 observation tokens, a 64-dimensional embedding, and a vocabulary of 256 entries. The VQ commitment-loss coefficient is 10. The state decoder uses the `OPTION1` configuration. The reward/continuation model is a causal Transformer.

Table 7: Actor, critic, and reward/continuation network settings. Depths are reported as configured.

Hyperparameter	Value
Actor hidden dimension / depth	128 / 3
Critic hidden dimension / depth	256 / 2
Reward Transformer embedding dimension	256
Reward Transformer layers	6
Reward Transformer attention heads	4
Reward Transformer dropout	0.1
Reward Transformer attention	Causal

Table 8: Policy and value-learning settings. The epoch and clipping entries report configured hyperparameters, rather than defining the policy-update objective.

Hyperparameter	Value
Policy distribution	Gaussian
Discount factor γ	0.99
Return parameter λ	0.95
Configured policy epochs	5
Configured policy clipping parameter	0.1
Value normalization	Enabled
Huber loss threshold δ	10
Clipped value loss	Enabled
Entropy coefficient	0
Entropy annealing factor	1.0
Controller temperature	1.0
Additional exploration ϵ / noise	0 / 0

Its embedding, residual, and attention dropout rates are all 0.1. Network dimensions are summarized in Table 7.

C.2.5 Policy and Value Learning

The actor uses a Gaussian policy. The discount factor and return parameter are $\gamma = 0.99$ and $\lambda = 0.95$, respectively. Value normalization, Huber value loss with $\delta = 10$, and clipped value loss are enabled. No additional exploration noise or ϵ-greedy exploration is added; action sampling from the Gaussian policy can nevertheless remain stochastic. Table 8 reports the associated settings.

C.2.6 Confidence Weighting

Confidence weighting uses $\alpha = 1.0$ and temperature 1.0, with weights clipped to $[0.75, 1.25]$. It has a warm-up of 50,000 real environment steps, followed by a gradual activation interval of 100,000 real environment steps. Confidence calibration uses an exponential moving average (EMA) with decay 0.99. This EMA belongs to confidence calibration and should not be conflated with any EMA used by the state representation.

Table 9: Confidence-weighting hyperparameters. Step counts refer to real environment steps.

Hyperparameter	Value
Confidence weighting	Enabled
Confidence coefficient α	1.0
Confidence temperature	1.0
Weight clipping interval	$[0.75, 1.25]$
Warm-up steps	50,000
Gradual activation interval	100,000
Calibration EMA decay	0.99

C.3 CONFIDENCE-WEIGHTED ACTOR–CRITIC OPTIMIZATION

We follow the centralized-training decentralized-execution actor–critic framework of DIMA **?** and retain its released implementation of the return computation. Our confidence mechanism modifies the contribution of individual imagined transitions to policy optimization.

Let b index imagined trajectories, $t \in \{0, \dots, H-1\}$ index time steps, and $i \in \{1, \dots, N\}$ index agents. An imagined transition contains a global state $\hat{s}_{b,t}$, local observations $\hat{o}^i_{b,t}$, actions $\hat{a}^i_{b,t}$, a shared reward $\hat{r}_{b,t}$, and a continuation probability $\hat{c}_{b,t}$. We use a parameter-shared decentralized actor π_θ and a centralized critic v_ϕ. The operator $\mathrm{sg}[\cdot]$ denotes stop-gradient.

Return targets and shared advantages. Let $\bar{V}_{b,t}$ denote the rollout-time critic prediction expressed in the original, unnormalized return scale. Following the return computation in the released DIMA implementation, we set

$$d_{b,t} = \gamma \hat{c}_{b,t}, \tag{18}$$

$$G_{b,H} = \bar{V}_{b,H}, \tag{19}$$

$$G_{b,t} = \hat{r}_{b,t} + \gamma d_{b,t}(1-\lambda)\bar{V}_{b,t+1} + d_{b,t}\lambda G_{b,t+1}, \quad t = H-1, \dots, 0. \tag{20}$$

Here, γ is the discount factor and λ controls the mixing of bootstrapped and recursively accumulated returns. For implementation reproducibility, Eq. equation 20 retains the additional γ factor in the one-step bootstrap term of the released code.

The unnormalized advantage and its standardized version are

$$A_{b,t} = \mathrm{sg}\left[G_{b,t} - \bar{V}_{b,t}\right], \tag{21}$$

$$\widetilde{A}_{b,t} = \mathrm{sg}\left[\frac{A_{b,t} - \mu_A}{\sigma_A + 10^{-4}}\right], \tag{22}$$

where μ_A and σ_A are the unweighted mean and sample standard deviation over all trajectory and time indices in the imagined batch. Since the critic evaluates the shared global state, $\widetilde{A}_{b,t}$ is shared across agents. Neither the return recursion nor advantage normalization uses confidence weights.

Actor surrogate. Let θ_{old} denote the parameters of the behavior policy that generated the imagined batch. For agent i, define

$$\rho^i_{b,t}(\theta) = \frac{\pi_\theta(\hat{a}^i_{b,t} \mid \hat{o}^i_{b,t})}{\pi_{\theta_{\mathrm{old}}}(\hat{a}^i_{b,t} \mid \hat{o}^i_{b,t})}. \tag{23}$$

For continuous actions, each density includes all action coordinates of that agent. The ratio is computed separately for each agent, rather than as a joint ratio across agents.

The per-agent clipped surrogate loss is

$$\begin{aligned}\ell^\pi_{b,t,i}(\theta) = & -\min\Big\{\rho^i_{b,t}(\theta)\widetilde{A}_{b,t}, \\ & \qquad \mathrm{clip}\big(\rho^i_{b,t}(\theta), 1-\epsilon_\pi, 1+\epsilon_\pi\big)\widetilde{A}_{b,t}\Big\} \\ & -\beta_{\mathcal{H}}\mathcal{H}\big[\pi_\theta(\cdot \mid \hat{o}^i_{b,t})\big],\end{aligned} \tag{24}$$

where ϵ_π is the policy clipping threshold and $\beta_{\mathcal{H}}$ is the entropy coefficient.

Critic regression. We use ValueNorm and a clipped Huber regression loss. Let $\mathcal{N}$ denote the running return normalization transform and let $v^{\mathrm{old}}_{b,t}$ denote the stored normalized critic prediction at rollout collection time. For each optimization minibatch, the normalization statistics are updated using its unweighted returns. The normalized regression target and clipped value prediction are

$$y_{b,t} = \mathrm{sg}[\mathcal{N}(G_{b,t})], \tag{25}$$

$$v^{\mathrm{clip}}_{\phi,b,t} = v^{\mathrm{old}}_{b,t} + \mathrm{clip}\big(v_\phi(\hat{s}_{b,t}) - v^{\mathrm{old}}_{b,t}, -\epsilon_v, \epsilon_v\big). \tag{26}$$

The per-transition critic loss is

$$\begin{aligned}\ell^V_{b,t}(\phi) = \max\Big\{ & h_\delta(y_{b,t} - v_\phi(\hat{s}_{b,t})), \\ & h_\delta\Big(y_{b,t} - v^{\mathrm{clip}}_{\phi,b,t}\Big)\Big\},\end{aligned} \tag{27}$$

with

$$h_\delta(e) = \begin{cases} \frac{1}{2}e^2, & |e| \le \delta, \\ \delta\left(|e| - \frac{\delta}{2}\right), & |e| > \delta. \end{cases} \tag{28}$$

Confidence-weighted objectives. Let $w_{b,t}$ be the transition confidence weight defined by our confidence mechanism. We explicitly detach this weight:

$$\bar{w}_{b,t} = \mathrm{sg}[w_{b,t}]. \tag{29}$$

For a minibatch $\mathcal{B}$ of imagined transitions, the actor and critic objectives are

$$\mathcal{L}_\pi(\theta) = \frac{\sum_{(b,t)\in\mathcal{B}} \bar{w}_{b,t} \sum_{i=1}^{N} \ell^{\pi}_{b,t,i}(\theta)}{\max\left\{ N \sum_{(b,t)\in\mathcal{B}} \bar{w}_{b,t}, 10^{-8} \right\}}, \tag{30}$$

$$\mathcal{L}_V(\phi) = \frac{\sum_{(b,t)\in\mathcal{B}} \bar{w}_{b,t} \ell^{V}_{b,t}(\phi)}{\max\left\{ \sum_{(b,t)\in\mathcal{B}} \bar{w}_{b,t}, 10^{-8} \right\}}. \tag{31}$$

The same transition weight is broadcast to all agents in the actor objective and applied once to the centralized critic loss. When entropy regularization is enabled, it receives the same confidence weight as the policy surrogate.

During the optimization epochs for an imagined batch, the confidence weights, return targets, normalized advantages, behavior-policy probabilities, and rollout-time value predictions are fixed. The normalized critic targets additionally reflect the running ValueNorm updates described above. Actor and critic optimization does not backpropagate through the confidence weights or the imagined dynamics. The prediction-error head is trained separately through its world-model objective.

Consequently, confidence weighting changes the relative gradient contributions of imagined transitions without modifying their return targets or using weighted advantage or ValueNorm statistics. Setting $w_{b,t} = 1$ for every transition recovers the corresponding unweighted actor and critic objectives.

Optimization settings. For Ant-v2 4×2, we use $H = 15$, $\gamma = 0.99$, $\lambda = 0.95$, $\epsilon_\pi = \epsilon_v = 0.1$, $\delta = 10$, and $\beta_{\mathcal{H}} = 0$. Each imagined batch contains 600 trajectories and is reused for five optimization epochs, with one minibatch per epoch. The actor and critic are optimized separately using Adam with learning rate 5×10^{-4}, optimizer epsilon 10^{-5}, zero weight decay, and gradient-norm clipping at 10.