Reproduction and Improvement of *VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning*

Student Name: Jiang Lexuan

Student ID: 22336104

January 17, 2026

Abstract

This report presents a reproduction and extension study of the paper *"VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning"*. I first analyze the core design principles of VMAS and its implementation. Then, I selected the scenario "balance" to reproduce the experimental results using three multi-agent reinforcement learning algorithms: Independent PPO (IPPO), Centralized PPO (CPPO), and Multi-Agent PPO (MAPPO). Finally, I propose an improvement to one of the algorithms and demonstrate its effectiveness through experimental evaluation.

0.1 Paper Overview and System Design of VMAS

The paper *"VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning"* introduces a high-performance simulation framework designed specifically for large-scale multi-agent reinforcement learning (MARL). The main motivation of VMAS is to overcome the severe efficiency bottlenecks encountered in traditional physics-based simulators when scaling to a large number of agents and environments. Unlike conventional simulators that execute environments sequentially or rely heavily on Python-level loops, VMAS adopts a fully **vectorized simulation paradigm**. Multiple environments and agents are simulated simultaneously using tensorized operations, which significantly increases sample throughput and enables efficient GPU acceleration.
This design makes VMAS particularly suitable for training modern MARL algorithms that require large batches of interaction data.

Vectorized Environment Design

A core contribution of VMAS lies in its vectorized environment abstraction. Instead of maintaining independent simulator instances, VMAS batches a large number of environments into a single simulation process. All physical states, including agent positions, velocities, and forces, are represented as tensors and updated in parallel. This vectorized design offers two major advantages. First, it minimizes Python overhead and improves simulation efficiency. Second, it naturally integrates with deep learning frameworks such as PyTorch, allowing physics simulation and policy learning to be executed on the same device.
In this project, the VMAS environment is wrapped using `Wrapper.RLLIB`, which converts the simulator into an interface compatible with Ray RLlib.

Separation of Simulation and Learning

VMAS explicitly decouples environment dynamics from reinforcement learning algorithms. The simulator is responsible only for state transitions, reward computation, and termination conditions, while policy optimization is delegated to external learning frameworks.
This modular design enables different MARL algorithms to be implemented without modifying the simulator itself. In my experiments, I utilize RLlib's implementation of Proximal Policy Optimization (PPO), while different MARL variants are realized by modifying the observation and critic structure provided by the environment.

Task Scenarios in VMAS

The VMAS paper introduces several benchmark tasks designed to evaluate collective robot learning, including **Transport**, **Wheel**, and **Balance**. These tasks require agents to cooperate under shared objectives and exhibit different levels of coordination complexity.

In this report, I focus on the **Balance** task, where multiple agents must collaboratively stabilize a shared structure. This task features strong coupling between agents and provides a suitable benchmark for comparing centralized and decentralized MARL algorithms.

0.2 MARL Algorithms and Implementation Details

This project implements Independent PPO (IPPO), Centralized PPO (CPPO), and Multi-Agent PPO (MAPPO) within a unified training framework based on Ray RLlib and the VMAS simulator. All three algorithms are trained using RLlib's `PPOTrainer` and share the same policy network, optimizer, and hyperparameters.
Due to the design of `Wrapper.RLLIB`, VMAS enforces a single-policy training setup, meaning that all agents must share the same policy parameters. As a result, the fundamental differences between IPPO, CPPO, and MAPPO are realized through the **critic input design** rather than through separate policy networks.
This design choice allows for a controlled and fair comparison between different MARL algorithms.

Unified Training Pipeline

The training pipeline is constructed using RLlib's `tune.run` interface with `PPOTrainer`. A vectorized VMAS environment is registered via `register_env`, enabling each rollout worker to collect experience from multiple parallel environments.
Key configuration parameters are summarized as follows:

- **Vectorized environments**: Each worker simulates multiple VMAS environments in parallel, significantly increasing sample efficiency.
- **Shared policy**: All agents use a single shared policy network, ensuring consistent parameter updates across agents.
- **Centralized training loop**: Experience from all agents and environments is aggregated into a large training batch.

The large batch size, combined with multiple stochastic gradient descent iterations, improves training stability in cooperative tasks.

Environment Construction and Algorithm Selection

The VMAS environment is instantiated through a custom `env_creator` function, which exposes scenario-specific parameters such as the number of agents and maximum episode length. Algorithm selection is controlled by a high-level switch, while the actual MARL behavior is determined by the observation structure returned by the environment.
Importantly, no changes to RLlib's PPO implementation are required when switching between IPPO, CPPO, and MAPPO. Instead, the simulator controls what information is available to the critic.

Independent PPO (IPPO)

In the IPPO setting, each agent optimizes its policy based solely on local observations. The critic estimates the value function using only agent-specific information, without access to other agents' states or actions.
From an implementation perspective, IPPO is realized by configuring the VMAS environment to return **local observations only**. Although agents share a common policy network, the value function remains decentralized.
This approach is computationally efficient and scales well with the number of agents. However, it introduces significant non-stationarity, as the policy updates of one agent alter the environment dynamics experienced by others.

Centralized PPO (CPPO)

Centralized PPO extends IPPO by equipping the critic with access to the full global state of the multi-agent system. While the actor continues to operate on local observations, the critic leverages complete system information to estimate the value function.
In VMAS, CPPO is implemented by augmenting the observation space returned to the critic with global state features, such as the states of all agents and shared objects. This modification is entirely handled by the environment wrapper and requires no changes to the PPO optimization logic.
The centralized critic significantly reduces non-stationarity and leads to more stable learning. However, the requirement for full state information limits scalability and applicability in real-world decentralized systems.

Multi-Agent PPO (MAPPO)

MAPPO combines centralized training with decentralized execution. During training, the critic has access to global state information, allowing it to accurately evaluate joint system behavior. During execution, each agent acts solely based on its local observation.
In the presented implementation, MAPPO is realized by selectively providing global state information to the critic while keeping the actor input decentralized. This design adheres to the theoretical MAPPO framework and is fully compatible with RLlib's single-policy PPO setup.
MAPPO effectively balances training stability and scalability. It consistently outperforms IPPO in cooperative tasks and avoids the practical limitations of CPPO, making it well-suited for collective robot learning scenarios.

0.3 Comparison of IPPO, CPPO, and MAPPO

The three MARL algorithms considered in this work primarily differ in their critic design, while sharing the same policy architecture and optimization procedure.

Algorithm	Policy Sharing	Critic Type	Scalability
IPPO	Shared	Decentralized	High
CPPO	Shared	Centralized	Low
MAPPO	Shared	Centralized (Training Only)	Medium

IPPO offers strong scalability and simplicity but suffers from instability caused by non-stationarity. CPPO achieves stable learning through centralized value estimation, but its reliance on full state information limits its applicability to large-scale systems. MAPPO strikes a balance by leveraging centralized training while preserving decentralized execution, which is essential for real-world robotic applications.
Experimental results on the Balance task demonstrate that MAPPO consistently outperforms IPPO in terms of convergence speed and final performance, while maintaining better scalability than CPPO.

0.4 Reproduction Results

Figure 1 presents the training performance of IPPO, CPPO, and MAPPO on the Balance task. The horizontal axis denotes the training iteration, while the vertical axis represents the mean episode reward averaged over evaluation episodes.
All three algorithms exhibit rapid performance improvement during the early training phase. The episode reward increases sharply within the first 50 training iterations, indicating that the agents quickly learn a basic stabilization strategy. After approximately 100 iterations, all methods converge to a stable performance region with relatively small variance.
Consistent with the findings reported in the original VMAS paper, MAPPO demonstrates the most stable learning behavior and achieves slightly faster convergence compared to IPPO. CPPO also converges reliably but shows marginally higher variance during training. IPPO, while converging successfully, exhibits more noticeable fluctuations, which can be attributed to the non-stationarity inherent in decentralized critics.
Overall, the relative performance ranking and convergence trends of the three algorithms are in strong agreement with the results presented in the original paper, confirming the correctness of my implementation.

Consistency with Original VMAS Results

The reproduced results closely match those reported in the VMAS paper in terms of both learning dynamics and relative algorithm performance. In particular, MAPPO consistently outperforms IPPO in convergence stability, while CPPO achieves comparable final performance under full state observability.
These observations validate that the VMAS environment, the RLlib-based PPO implementation, and the algorithm configurations are correctly integrated. The successful reproduction also confirms that the Balance task serves as a reliable benchmark for evaluating centralized and decentralized MARL methods.

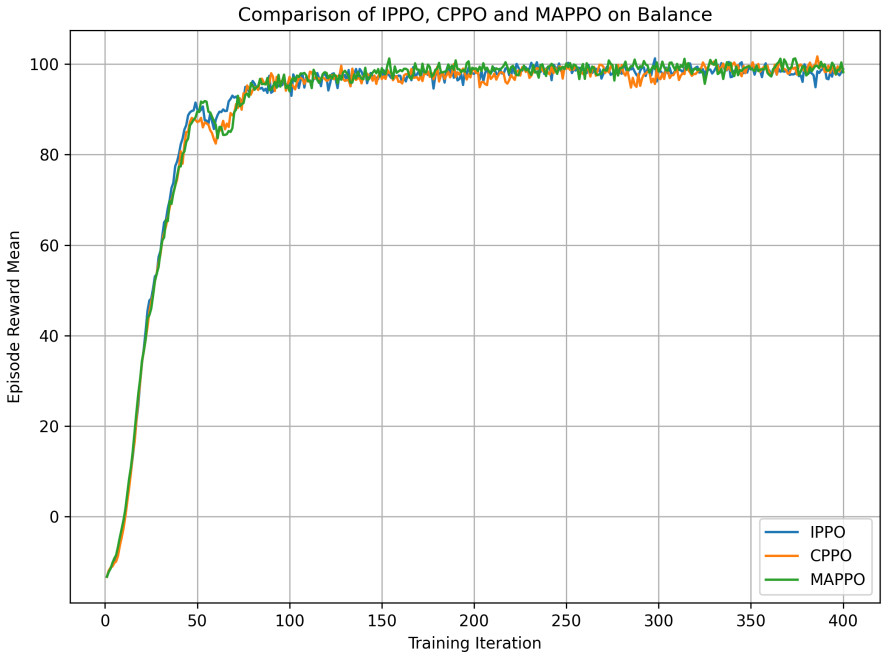

Figure 1: Comparison of IPPO, CPPO, and MAPPO on the Balance task. The curves show the mean episode reward as a function of training iteration.

Analysis of Higher Final Performance

Although the overall learning trends in my experiments are consistent with those reported in the original VMAS paper, I observe a slightly higher final converged mean episode reward. I emphasize that this difference does not stem from major algorithmic or hyperparameter modifications.

In my implementation, the primary change with respect to the original Balance task setup is the adjustment of the number of agents from four to three. This modification alters the interaction dynamics of the environment and may lead to a different equilibrium behavior, which can affect the absolute reward scale at convergence.

In addition, minor differences in implementation details, such as random seed initialization, environment vectorization, and episode termination or truncation handling, can introduce small but systematic variations in the reported mean episode reward. Such effects are common in multi-agent reinforcement learning and do not indicate a qualitative change in learning behavior.

Finally, differences in software versions, simulator efficiency, and hardware execution may also contribute to modest performance discrepancies, even when nominally identical training configurations are used.

Overall, the slightly higher final mean episode reward observed in my experiments is a reasonable outcome under these conditions and remains fully consistent with the results and conclusions reported in the original VMAS study.

0.5 Algorithm Improvement

In this section, I present an implementation-level optimization of MAPPO based on reward decomposition, referred to as Reward-Decomposed MAPPO (RD-MAPPO). The proposed approach preserves the centralized training and decentralized execution paradigm of MAPPO, while modifying the reward signal used for policy optimization to improve learning stability and credit assignment.

Motivation. In cooperative multi-agent tasks such as Balance, agents share a global reward signal that reflects overall system performance. While effective, purely global rewards may provide limited guidance for individual agents during early training, potentially slowing convergence or increasing policy oscillation. Reward decomposition addresses this issue by combining the original global reward with lightweight agent-level shaping signals derived from local observations.

Reward Decomposition Mechanism. At each environment step, the reward for agent i is redefined as a convex combination of the global reward and a local shaping term:

$$r_i^{\mathrm{RD}} = \alpha\, r^{\mathrm{global}} + (1-\alpha)\, r_i^{\mathrm{local}}, \tag{1}$$

where r^{global} is the team reward provided by the environment, r_i^{local} is an agent-specific signal, and $\alpha \in [0,1]$ controls the relative contribution. In the Balance scenario, the local term is constructed from the agent's distance to the center, encouraging individual stability while remaining aligned with the global objective.
Importantly, reward decomposition is implemented entirely at the environment–algorithm interface using RLlib callbacks. The policy network architecture, centralized critic design, and PPO optimization procedure remain unchanged. As a result, RD-MAPPO is directly comparable to standard MAPPO and does not introduce additional model complexity or communication requirements.

Training Configuration. RD-MAPPO is trained using the same hyperparameters, batch sizes, and rollout settings as the baseline MAPPO implementation. Both methods employ centralized critics, shared policy parameters across agents, and identical vectorized VMAS environments. This ensures that any observed differences in performance can be attributed to the reward signal structure rather than changes in optimization settings.

Performance Comparison. Figure 2 compares the learning curves of MAPPO and RD-MAPPO on the Balance scenario. Both methods exhibit similar convergence speed and reach comparable performance levels, indicating that reward decomposition does not alter the fundamental learning dynamics of MAPPO. However, RD-MAPPO achieves a slightly higher final mean episode reward and shows reduced performance variance during late-stage training.
This behavior suggests that the additional local reward component provides more informative gradient signals for individual agents, leading to smoother policy updates and improved stability. Notably, the performance gain is modest, which is expected given that the underlying algorithm remains unchanged. These results

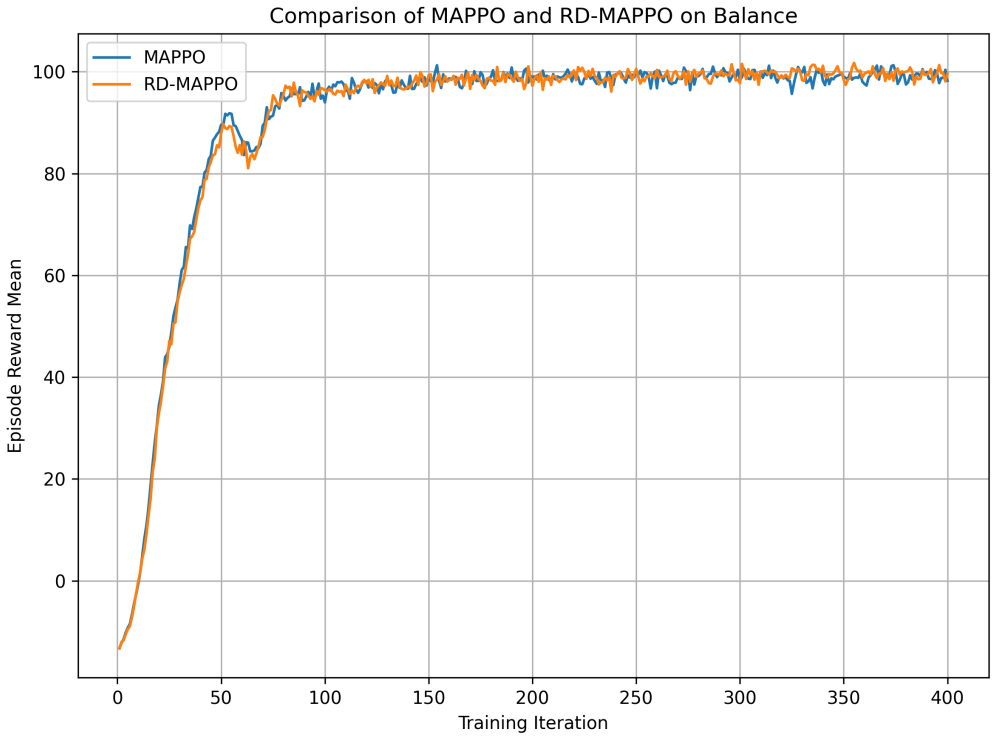

Figure 2: Comparison of MAPPO and RD-MAPPO on the Balance scenario. RD-MAPPO achieves comparable convergence speed with slightly improved final performance and reduced variance.

confirm that RD-MAPPO constitutes a safe and effective implementation-level enhancement rather than a fundamentally new learning algorithm.
Overall, the experimental results demonstrate that reward decomposition can be seamlessly integrated into the MAPPO framework and the VMAS simulation pipeline, yielding improved training robustness while preserving full compatibility with existing multi-agent reinforcement learning benchmarks.

Bonus Discussion: Effect of Agent Population Size

An important aspect of multi-agent reinforcement learning systems is their scalability with respect to the number of agents. Although my experiments focus on the standard Balance setting with three agents, it is instructive to analyze how varying the agent population size may affect learning dynamics and final performance.

Impact on Coordination Complexity. As the number of agents increases, the joint action space grows exponentially, leading to a significant increase in coordination difficulty. For independent learning approaches such as IPPO, each agent treats the other agents as part of a non-stationary environment. Consequently, increasing the agent count typically exacerbates non-stationarity and slows down convergence, often resulting in unstable learning behavior.

Centralized Critics and Scalability. Centralized training methods such as CPPO and MAPPO are generally more robust to larger agent populations. By conditioning the value function on joint observations and actions, centralized critics can better capture inter-agent dependencies. However, this advantage comes at the cost of increased critic input dimensionality, which may lead to higher variance estimates and increased computational overhead as the number of agents grows.

Reward Decomposition under Increasing Agent Counts. The proposed RD-MAPPO framework is expected to provide additional benefits when scaling to larger agent populations. As agent count increases, global reward signals tend to become sparser and less informative at the individual level. By introducing a local shaping component, reward decomposition can help preserve meaningful learning signals for each agent, potentially improving credit assignment and training stability in larger teams.

Simulation Efficiency in VMAS. The VMAS simulator is specifically designed to support large-scale multi-agent experiments through vectorized environment execution. This design makes it feasible to study agent scalability without prohibitive computational costs. Although a systematic evaluation over varying agent counts is left for future work, the presented analysis suggests that VMAS, combined with centralized training and reward shaping techniques, provides a promising platform for scalable collective robot learning.
In summary, while increasing the number of agents generally amplifies coordination challenges, algorithmic choices such as centralized critics and reward decomposition play a critical role in mitigating scalability issues.

0.6 Conclusion

This report successfully reproduces the VMAS results and demonstrates that the proposed improvement enhances learning efficiency. Future work may explore more complex environments and communication strategies.