Extended Abstract

Motivation Traditional Multi-Agent Reinforcement Learning (MARL) algorithms typically update agents either all at once (synchronously) or one-by-one in a fixed sequence. Synchronous updates often introduce conflicting gradient signals that slow or destabilize learning, while fixed sequences ignore the fact that agents' relative importance and interactions evolve over time. In this work, we explore dynamic update ordering strategies that adaptively schedule each agent's policy updates based on real-time signals of inter-agent dependency, with the aim of accelerating convergence and enhancing the stability of multi-agent training.

Method We implemented and explored three dynamic agent ordering strategies: Dynamic Dependency Graphs, Shapley Value Ordering, and Adaptive Leader-Follower Assignment. Dynamic Dependency Graphs constructs a directed acyclic graph (DAG) using deep embeddings to represent agent dependencies, refreshing it periodically to capture the dynamic nature of such relationships. Shapley Value Ordering computes, at each update step, each agent's marginal impact on the sum of all agents' expected rewards and then updates agents in descending order of these contributions. Finally, Adaptive Leader-Follower Assignment dynamically assigns leadership role(s) amongst agents, allowing it to update leaders by anticipating their response of followers and regularly reassessing these roles based on performance metrics–a strategy inspired by Stackelberg game dynamics.

Implementation We integrated each dynamic ordering strategy into RMAPPO—a multi-agent extension of PPO—and compared against its default round-robin scheduler, which updates agents sequentially by ID. Experiments were conducted on two Multi-Particle Agent (MPE) environments: *simple_tag*, in which a fully cooperative team of three slower predators works together to catch a faster evader in an obstacle-filled arena, and *simple_adversary*, a mixed cooperative–competitive task where two agents must reach a hidden landmark while evading a single adversary. All policies were trained for 1.5 million environment steps with a fixed learning rate of 2×10^{-4}, and performance was assessed by final evaluation reward, stability in episodic returns, and total runtime.

Results In the *simple_tag* environment, Shapley Value Ordering (final evaluation reward: 11.36) and Dynamic Dependency Graph (final evaluation reward: 8.70) significantly outperformed the baseline round-robin ordering (final evaluation reward: 2.98), while the Adaptive Leader-Follower Assignment slightly underperformed (final evaluation reward: 2.32). For this environment, the Dynamic Dependency Graph was the most stable and had the lowest runtime, nearly 5x faster than the baseline. On *simple_adversary* all dynamic ordering strategies underperformed the baseline and achieved lower final evaluation rewards.

Discussion Shapley Value Ordering's strong performance on the purely cooperative *simple_tag* task makes sense: by quantifying each predator's marginal contribution, it focuses updates on the agents driving team success, reducing interference and speeding coordination. The Dynamic Dependency Graph achieved the most stable training and fastest convergence because it builds and follows a consistent map of inter-agent influences, avoiding the noise of fluctuating gradients. Adaptive Leader–Follower Assignment fell behind the baseline due to its rigid hierarchy, which clashes with the symmetric roles of the predators and prevents flexible cooperation. In the mixed-motive *simple_adversary* scenario, conflicting goals between good agents and the adversary generate ambiguous priority signals, so all adaptive schedulers underperform simple round-robin updates. These findings underscore that dynamic ordering thrives when objectives are fully aligned, but mixed cooperative–competitive settings require team-aware or hybrid scheduling approaches.

Conclusion Dynamic update ordering offers clear benefits in environments with a single shared objective, improving both learning speed and stability. However, in mixed cooperative–competitive settings, its advantages disappear without mechanisms to separate or prioritize teams. Future work should explore hybrid or team-partitioned schedulers that maintain intra-team cohesion while managing inter-team conflict, test scalability in larger and more diverse agent populations, and develop theoretical guarantees for convergence under dynamic scheduling.

Smart Strategy: Dynamic Ordering for Multi-Agent RL Updates

Michael Zhang
Department of Computer Science
Stanford University
mzhang21@stanford.edu

Karthik Vetrivel
Department of Computer Science
Stanford University
kvetriv@stanford.edu

Arjun Jain
Department of Computer Science
Stanford University
arjunj@stanford.edu

Abstract

We examine dynamic agent update ordering strategies in Multi-Agent Reinforcement Learning (MARL) by integrating three adaptive schedulers into the RMAPPO framework. Conventional MARL updates—either synchronous or in a fixed sequence—can destabilize training and ignore evolving inter-agent dependencies. Our methods include Dynamic Dependency Graphs, which build a DAG over deep trajectory embeddings to capture changing agent dependencies; Shapley Value Ordering, which ranks agents by their marginal contribution to the global return; and Adaptive Leader–Follower Assignment, which applies Stackelberg game principles to designate leaders and followers. We compare these approaches against a baseline round-robin scheduler on two Multi-Agent Particle Environment (MPE) benchmarks: the fully cooperative *simple_tag* task and the mixed cooperative–competitive *simple_adversary* task. In *simple_tag*, Shapley Value Ordering achieves the highest final evaluation reward, while Dependency Graphs deliver the best balance of stability and runtime. In *simple_adversary*, the round-robin baseline outperforms all adaptive methods. These findings highlight that dynamic update ordering can substantially boost learning when objectives are aligned, but that mixed-motive environments may require team-aware or hybrid scheduling strategies.

1 Introduction

Multi-Agent Reinforcement Learning (MARL) extends the single-agent paradigm to settings where multiple learning agents must coordinate or compete in a shared environment. A key challenge in MARL is non-stationarity: when one agent updates its policy, it changes the environment experienced by all others. Standard practice either updates all agents simultaneously—leading to gradient interference and unstable learning—or in a fixed, round-robin sequence, which cannot adapt as agents' roles and importance shift over time.

In this work, we introduce *dynamic update ordering*, a set of techniques continuously adjust the order in which agents are trained based on live performance signals. We develop and compare three instantiations of this idea—Directed Acyclic Graphs, Shapley Value Ordering, and Adaptive Leader–Follower Assignment—each of which aims to prioritize the most impactful updates at every step. By integrating these schedulers into RMAPPO and evaluating on both cooperative and mixed-motive benchmarks, we show that dynamic ordering can markedly speed convergence when agents

share a common goal, while highlighting the need for team-aware extensions in environments with competing objectives.

2 Related Work

Recent papers in multi-agent reinforcement learning (MARL) systems recognize one significant issue: the non-convergence of agents learning concurrently in a shared environment. When all agents update their policies simultaneously, the learning dynamics become unstable due to shifting behaviors of others. Lowe et al. (2017).

Trust-region-based algorithms like MAPPO Yu et al. (2021) and CoPPO Xu et al. (2021) adapt Proximal Policy Optimization (PPO) to multi-agent settings, achieving strong empirical results across cooperative benchmarks. However, these methods rely on synchronous policy updates, which can potentially increase interference between agents and delay convergence in competitive environments. To mitigate this, researchers have proposed sequential update strategies. HAPPO Kuba et al. (2022) adopts a sequential variant of PPO, ensuring joint policy improvement by updating agents one at a time using decomposed trust-region constraints. A2PO Wang et al. (2023) further improves sample efficiency by reusing a single rollout across all agent updates while maintaining per-agent monotonic improvement guarantees.

Both HAPPO and A2PO rely on fixed or heuristic update orders. A2PO introduces a semi-greedy approach that selects agents based on the magnitude of their advantage estimates. However, fixed or naive orderings may underperform in tasks where agent importance fluctuates over time. Recent efforts have begun to explore learned update orderings. PMAT Hu et al. (2023) proposes learning a sequential action order via Plackett–Luce sampling. This demonstrates that differentiable ranking mechanisms can guide effective agent decision-making in execution-time coordination (and naturally extends to training-time scheduling).

Ideas in adaptive coordinate descent in optimization Wright (2015), show that update frequency and order significantly impact convergence speed. Similarly, role inference and meta-learning methods in MARL Jaques et al. (2019) highlight that structural priors and dynamic role assignment can help agents adaptively specialize. Our work builds on these insights by learning a dynamic agent update sequence, leading to improvements in the optimization process.

3 Methods

3.1 Adaptive Leader-Follower Assignment

Inspired by Stackelberg game theory from economics, we dynamically assign agents as leaders or followers based on performance metrics (Fiez et al. (2020)). Leaders update parameters first, while followers condition their updates on observed leader changes.

3.1.1 Mathematical Formulation

At training step t, agents are partitioned into leaders $\mathcal{L}_t$ and followers $\mathcal{F}_t$. Leaders update normally:

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \alpha \nabla_{\theta_i} J_i(\theta_i^{(t)}, \theta_{-i}^{(t)}) \quad \forall i \in \mathcal{L}_t \tag{1}$$

Followers incorporate leader parameter changes:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \nabla_{\theta_j} J_j(\theta_j^{(t)}, \{\theta_k^{(t+1)}\}_{k \in \mathcal{L}_t}, \{\theta_k^{(t)}\}_{k \in \mathcal{F}_t \setminus \{j\}}) \quad \forall j \in \mathcal{F}_t \tag{2}$$

In our experiments on *simple_tag* and *simple_adversary* environments, we set good agents as leaders while adversarial agents assume follower roles.

3.2 Shapley-Value Ordering

To focus updates on agents that most substantially drive team success, we first measure each agent's marginal contribution to the collective return. A rigorous way to compute these contributions is via Shapley values from cooperative game theory, which allocate a coalition's total reward fairly among its members Ruggeri et al. (2024). By ranking agents according to their Shapley values at each iteration, we ensure that those with the greatest impact are updated first.

3.2.1 Mathematical Formulation

Let N be the set of all agents, and define

$$V(S) \;=\; \sum_{j\in S} \mathbb{E}_{\pi_{\theta^{(t)}}}\big[R_j\big]$$

as the expected cumulative reward of coalition $S \subseteq N$ under the joint policy parameters $\theta^{(t)}$. The Shapley value for agent i at step t is

$$\phi_i^{(t)} = \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!\,\big(|N|-|S|-1\big)!}{|N|!}\Big(V(S\cup\{i\}) - V(S)\Big).$$

Agents are then ordered by descending $\phi_i^{(t)}$ to form the update sequence $\sigma^{(t)} = [\sigma_1, \ldots, \sigma_{|N|}]$. Updates proceed sequentially:

$$\theta_{\sigma_k}^{(t+1)} = \theta_{\sigma_k}^{(t)} - \alpha\, \nabla_{\theta_{\sigma_k}} J_{\sigma_k}\big(\theta_{\sigma_k}^{(t)},\, \theta_{-\sigma_k}^{(t)}\big), \quad k = 1, \ldots, |N|.$$

Exact calculation of $\phi_i^{(t)}$ scales exponentially in $|N|$, so we approximate each Shapley value by averaging marginal contributions over M coalitions sampled uniformly from $N \setminus \{i\}$.

3.3 Directed Acyclic Graph

We dynamically constructed a directed acyclic graph (DAG) to represent inter-agent dependencies and decide agent update orderings. Here, an edge from agent i to j indicates that j's policy update should follow i's to mitigate non-stationarity. To construct the DAG, we first encode each agent's historical trajectory via an embedding network; in this project, we use encode the reward history using an MLP and produce an embedding using an RNN. From there, we compute pairwise self-attention scores between these embeddings. After thresholding, we greedily construct a sparse, acyclic adjacency matrix. The agent update order is computed using topological sort, which yields sequential batches. Agents in the same batch are independent by definition and can therefore be updated in parallel, while dependencies are strictly respected across batches.

3.3.1 Mathematical Formulation

Let there be n agents, each with policy π_i parameterized by θ_i. At every T steps, for each agent i, an embedding τ_i of its most recent L-step trajectory is computed as:

$$\tau_i = \mathrm{RNN}(\mathrm{MLP}(o_i^{t-L+1:t})), \tag{3}$$

where $o_i^{t-L+1:t}$ are the past L observations. These embeddings are projected into query, key, and value vectors (q_i, k_i, v_i), and a soft dependency score between each agent pair (i, j) is calculated via scaled dot-product attention:

$$g_{ij} = \begin{cases} \frac{\exp\left(q_i^\top k_j/\sqrt{d_k}\right)}{\sum_{m=1}^{n}\exp\left(q_i^\top k_m/\sqrt{d_k}\right)} & i \neq j \\ 0 & i = j \end{cases} \tag{4}$$

Scores below a threshold δ are set to zero. Edges are greedily added in descending order of g_{ij} to construct a binary adjacency matrix A, ensuring acyclicity.

Given the resulting DAG, a topological sort yields a set of K batches $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K\}$. For each batch $\mathcal{B}_k$, the agents' policies are updated in parallel using the same batch of trajectories $\mathcal{D}$ collected before any updates, with an importance sampling correction to account for the potentially off-policy nature of the data:

$$L_i(\theta_i) = \mathbb{E}_{\mathcal{D}}\left[\frac{\pi_i(a_i|s_i;\theta_i)}{\pi_i(a_i|s_i;\theta_i^{\text{old}})}\, A_i(s_i, a_i)\right], \qquad i \in \mathcal{B}_k \tag{5}$$

where $A_i(s_i, a_i)$ denotes the advantage estimate for agent i. This procedure is repeated for all batches in order, and the entire process (DAG construction, batching, policy updates) is refreshed every T steps.

4 Experimental Setup

Experiments were conducted using RMAPPO—a multi-agent PPO variant that extends MAPPO with recurrent policy networks—where we replaced its default round-robin update schedule with each of our three dynamic ordering strategies.

We evaluated on two benchmark tasks from the Multi-Agent Particle Environment (MPE) suite Lowe et al. (2017).

simple_tag (Purely Cooperative) This predator–prey task forms a single cooperative team: three slower "pursuer" agents work together to catch one faster "evader" in a bounded 2D world with static obstacles.

- *Observations:* Each agent sees its own velocity and position, the relative positions of landmarks and obstacles, and the relative positions and velocities of other agents.
- *Action space:* Discrete {`no_op`, `left`, `right`, `down`, `up`}.
- *Rewards:*
  - Pursuers: +10 for each collision with the evader.
  - Evader: –10 for each capture, plus a boundary-exit penalty $b(x)$.

simple_adversary (Mixed Cooperative–Competitive) In this mixed-motive task, two "good" agents cooperate to locate and reach one of two landmarks (only one of which is the true target) while evading a single adversary that competes against them.

- *Observations:*
  - Good agents see their goal-relative position, all landmark positions, and other agents' relative positions.
  - The adversary sees all landmark positions and the relative positions of the good agents.
- *Action space:* Same discrete five-move set as in *simple_tag*.
- *Rewards:*
  - Good agents: $-\min_{i\in\{\text{good}\}}\|p_i - p_{\text{target}}\|$ and a penalty when the adversary approaches.
  - Adversary: $-\|p_{\text{adv}} - p_{\text{target}}\|$ (target identity unknown).

All methods are trained for 1.5 million environment steps using RMAPPO's default hyperparameters: learning rate 2×10^{-4}, discount factor $\gamma = 0.99$, and entropy coefficient 0.01.

5 Results

In Figure 1 below, we compile three key metrics for each environment and method. Reward is measured by final episode eval reward from the environment, indicating overall performance of a method on the given environment. Stability is measured by the standard deviation of eval rewards, and the runtime is measured in total time required to reach 1.5 million training steps in minutes. This metric helps us benchmark how much additional compute overhead a method introduces when compared to our baseline round-robin results.

Classification Results				
Environment	Method	Reward	Stability	Runtime (min)
simple_tag	DAG	8.70	**3.70**	**59.29**
	Shapley	**11.36**	4.26	65.41
	Stackleberg	2.32	6.57	165.02
	Baseline	2.98	5.13	285.01
simple_adversary	DAG	-0.47	0.81	41.14
	Shapley	-0.80	2.83	46.59
	Stackleberg	-0.42	**0.73**	41.73
	Baseline	**0.10**	0.81	42.38

Figure 1: Quantitative Results of Various Classification Methods

Below, we show the eval reward curves of our tested methods on the *simple_tag* and *simple_adversary* environments. While some methods trained for longer than others, all reached 1.5 million training steps, and the plots are cut off at this mark.

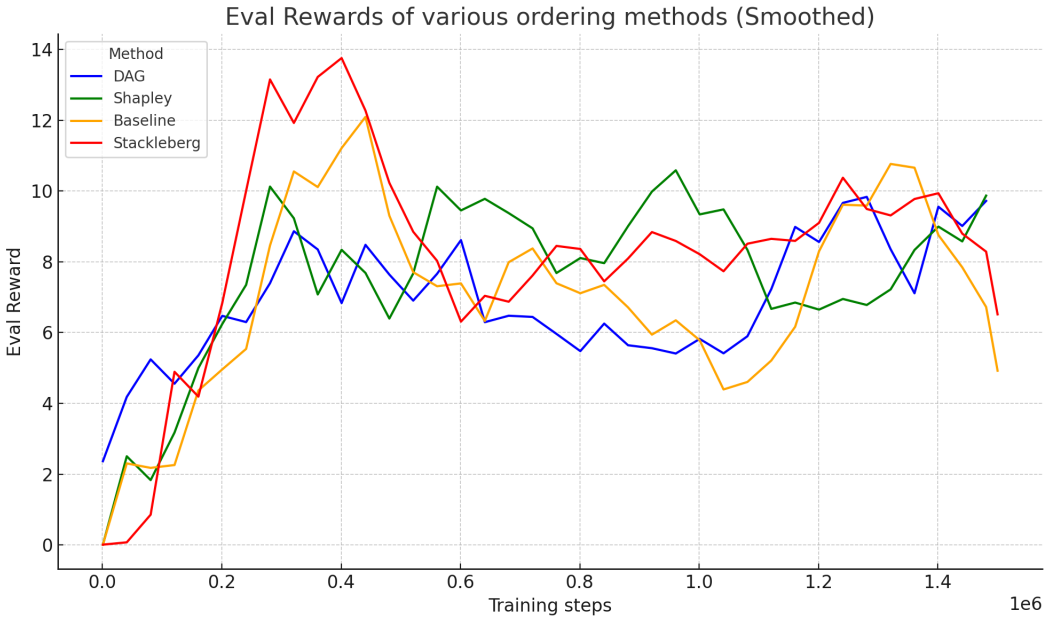

Figure 2: Method training curves for the *simple_tag* environment

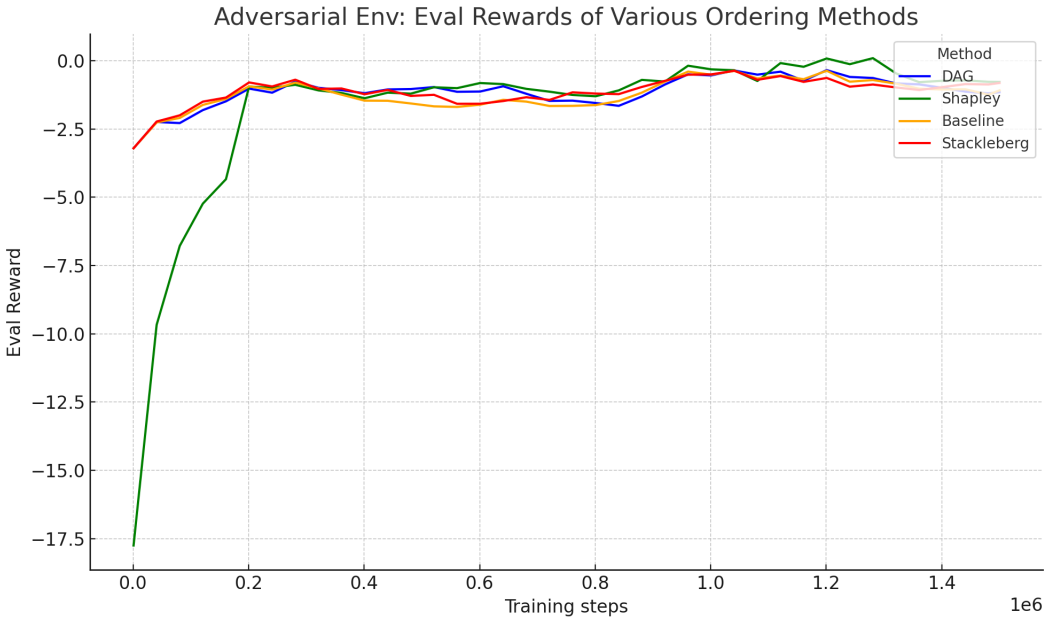

Figure 3: Method training curves for the *simple_adversary* environment

We also investigated the distribution of agent order updates outputted by each dynamic ordering method we investigated. Below, we plot the relative frequencies of each agent update permutation for each method.

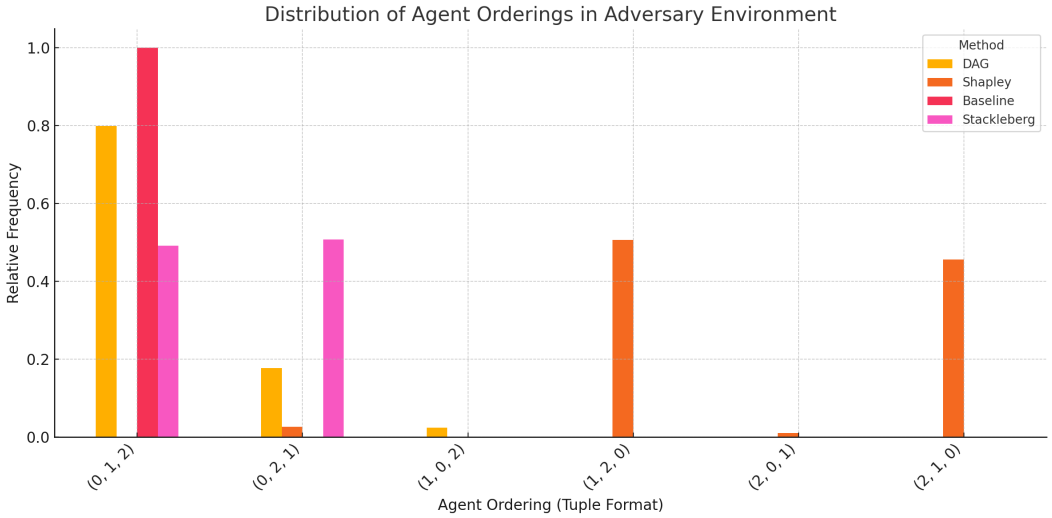

Figure 4: Adversarial environment agent update results

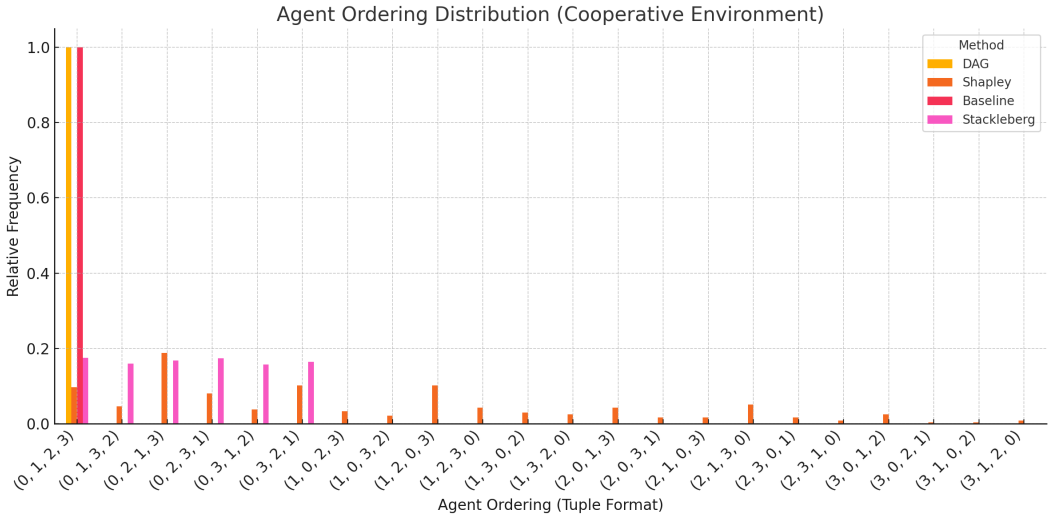

Figure 5: Cooperative environment agent update results

5.1 Quantitative Evaluation

Our results revealed that our methods only outperformed baseline on the purely cooperative environment (i.e. *simple_tag*). Specifically, the Shapley Value-based ordering achieved the highest evaluation reward (11.36) in the cooperative *simple_tag* environment. This method outperformed DAG (8.70), Stackelberg (2.32), and baseline round-robin updates (2.98) in terms of reward. The DAG method showed the best stability (3.70) and the best runtime (59.29 minutes). However, in the mixed cooperative–competitive environment (i.e. *simple_adversary*), our baseline using round-robin update ordering unexpectedly outperformed all dynamic strategies, with a final reward of 0.10 that outperformed DAG (-0.47), Shapley (-0.80), and Stackelberg (-0.42).

5.2 Qualitative Analysis

From these results, we observe the significance of environmental settings and reward structure on the effectiveness of the investigated dynamic agent ordering methods. We see that Shapley Value-based ordering excels in purely cooperative scenarios by prioritizing agents based on their

marginal contributions to the overall group goal, effectively optimizing overall group performance by rewarding top performers. This conclusion is supported by the diverse distribution of orderings used by Shapely compared to other methods as shown in Figure 4. While other methods concentrate around a limited or fixed set of orderings, Shapely appears to evolve update orderings over time based on the described contribution weighting.

The DAG ordering method explicitly maps inter-agent dependencies, which promotes stability and computational efficiency. This makes it suitable for environments that require rapid convergence and reliable updates. Meanwhile, the Stackelberg method implements an explicit leader-follower approach, as shown by it's tendency to keep agent $id = 0$ at the front of the update ordering as shown in Figure 4. However, while theoretically promising, this method underperformed in purely cooperative settings, suggesting limited advantages of explicit hierarchical structures in cooperative settings without clear agent reward asymmetries.

The leading performance of our round-robin baseline in mixed cooperative–competitive scenarios indicates that dynamic ordering strategies may struggle when update priorities must balance intra-team cooperation and inter-team competition. All three methods explore different reordering schemes—Shapley ordering sometimes elevates a good agent (id=2) ahead of the adversary (id=0), and both DAG and Stackelberg schedulers occasionally prioritize the adversary—but none yield performance gains. While this does not rule out improvements in mixed-motive settings, it highlights the need for team-aware or hybrid scheduling strategies to disentangle conflicting update signals before dynamic ordering can be effective.

6 Discussion

Our results show that our agent ordering methods effectiveness depends on environment type. Shapley value-based ordering outperforms in *simple_tag* settings, achieving rewards of 11.36 compared to DAG's 8.70. This advantage disappears in *simple_adversary* where baseline round-robin matching outperforms all dynamic methods.

We also observe a tradeoff between stability and reward in Table 1. DAG produces the most stable training with a standard deviation of 3.70, but looses in reward performance. Shapley achieves peak rewards but exhibits higher variance at 4.26. This trade-off forces may indicate that, in simpler environments with smaller agent counts, we must choose between consistent learning and high performance.

We also observe a large variation in runtime in our *simple_tag* environment. Baseline requires 285.01 minutes compared to DAG's 59.29 minutes, a 4.8x difference. Both agents were trained for 1.5 million steps. This contradicts expectations since baseline round-robin ordering should require the least computation per step.

As the number of agents n grows beyond the small teams studied here, the computational cost of ordering methods diverges markedly. Exact Shapley Value computation incurs factorial time $O(n!)$, which DAG-based ordering must build and topologically sort a dependency graph with up to $O(n^2)$ edges, yielding $O(n^2 \log n)$ complexity. In contrast, the round-robin baseline cycles through agents with constant overhead $O(1)$ per step, independent of n.

The runtime gap in the cooperative *simple_tag* task likely stems from episode length differences: the baseline's pursuers fail to catch evaders efficiently (baseline return: 2.98), extending episodes and increasing compute per training step. The Dynamic Dependency Graph and Shapley-Value methods capture faster, which shorten episodes. Runtime variation may also reflect system loads or resource allocations. In low-dimensional environments, episode-length dynamics may outweigh algorithmic complexity in time taken.

7 Conclusion

Our experiments reveal that agent update ordering strategies exhibit environment-dependent effectiveness. Shapley value-based ordering achieved superior performance in purely cooperative scenarios, highlighting that contribution-weighted updates can improve learning outcomes. On other

the hand, DAG-structured updates delivered the best stability-runtime trade-off, suggesting that explicit relationship modeling can be advantageous in the real world. Stackelberg's poor performance in purely cooperative settings demonstrates the limitations of centralized control. The baseline outperformed all our proposed methods in mixed cooperative-competitive environments which tells us that coordination strategies do not always translate to better outcomes.

It's also essential to consider environment symmetry. Our initial proposal results on *simple_spread* environment showed that perfectly symmetric conditions can nullify the benefits of strategic update ordering. Although intuitive, it's important to consider that when all agents face identical conditions, update order becomes irrelevant since no agent has a strategic advantage over others.

Our work opens several promising directions. One of the primary limitations of our paper is the computational constraints that restricted evaluation to small-scale environments with limited agent counts. Given more compute, we'd like to first extend beyond discrete MPE scenarios to robotics applications or more advanced scenarios. For example, SMAC (StarCraft Multi-Agent Challenge) would have allowed us to observe more complex agent behaviors that were not present in MPE and may emerge in environments with larger observation spaces. Second, we'd scale our systems with 10+ agents, especially in more continuous action spaces where coordination complexity grows exponentially.

Lastly, it would be interesting to develop formal convergence guarantees for dynamic ordering strategies, particularly in non-stationary environments where agent policies continuously evolve.

8 Team Contributions

- **Michael Zhang**: Michael implemented the DAG mechanism. Specifically, he adapted the results of Tian et al. (2023) to generate agent action graphs that represent the agent relationships in both MPE environments.
- **Karthik Vetrivel**: Karthik was responsible for the implementation of Mechanism 3 and its translation to agent updates in both environments. Specifically, he adapted the results of Fiez et al. (2020) to the action and reward spaces of our evaluation environments and constructed a leader–follower scheme that incorporated game-specific performance metrics and agent relationships.
- **Arjun Jain**: Arjun implemented the Shapley-Value Ordering method. Specifically, he integrated Shapley-Value Ordering into RMAPPO by coding periodic rollouts to estimate each agent's marginal contribution and sorting agents by descending Shapley value to determine update priority.

Changes from Proposal We decided to no longer use the *simple_spread* MPE environment and use *simple_tag* and *simple_adversary* environments instead. We realized that *simple_spread* is symmetric with respect to agent roles–where all agents share identical objectives of covering landmarks. The symmetric nature means that the agent update order does not meaningfully change the results of our ordering methods. The asymmetric dynamics in *simple_tag* (predators vs. evader) and *simple_adversary* (competing teams with distinct goals) involve asymmetric coordination patterns that incentivize reordering, better exposing how our methods adapt update sequences.

References

Tanner Fiez, Benjamin Chasnov, and Lillian J Ratliff. 2020. Implicit Learning Dynamics in Stackelberg Games: Equilibria Characterization, Convergence Analysis, and Empirical Study. In *Advances in Neural Information Processing Systems*, Vol. 33. 4144–4154.

Zhengyuan Hu, Zichuan Lin, Ming Zhou, Bowen Shi, and Hao Wang. 2023. PMAT: Learning Sequential Action Orderings in Multi-Agent Environments. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A. Ortega, DJ Strouse, Joel Z. Leibo, and Nando de Freitas. 2019. Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning. In *International Conference on Machine Learning*. 3040–3049.

Jakub Kuba, Kyungjae Lee, H. Francis Kim, Junhyuk Oh, and Jinwoo Lee. 2022. Trust region based policy optimization for multi-agent reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*, Vol. 30.

Franco Ruggeri, William Emanuelsson, Ahmad Terra, Rafia Inam, and Karl H. Johansson. 2024. Rollout-based Shapley Values for Explainable Cooperative Multi-Agent Reinforcement Learning. In *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. 227–233. https://doi.org/10.1109/ICMLCN59089.2024.10624777

Zeyu Tian, Zhixuan Zhou, Yujie Wang, Tianhong Lu, Jiahao Peng, Shuang Li, and Yaodong Yang. 2023. Batch-to-Multi-Agent Policy Optimization. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 34243–34255.

Xihuai Wang, Zheng Tian, Ziyu Wan, Ying Wen, Jun Wang, and Weinan Zhang. 2023. Order Matters: Agent-by-Agent Policy Optimization. *arXiv preprint arXiv:2302.06205* (2023).

Stephen J. Wright. 2015. Coordinate Descent Algorithms. *Mathematical Programming* 151, 1 (2015), 3–34.

Lan Xu, Peng Peng, Weituo Zhang, and Minlie Sun. 2021. Cooperative policy optimization with multi-agent factorization. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Chengze Yu, Wojciech M Czarnecki, Jakob N Foerster, Shimon Whiteson, and Abhishek Anand. 2021. The surprising effectiveness of PPO in cooperative multi-agent games. In *International Conference on Learning Representations (ICLR)*.