Robust Instruction Compliance in Cooperative Multi-Agent Reinforcement Learning

Wo Wei Lin
Department of Computer Sciences
Northeastern University
United States
lin.wo@northeastern.edu

Ethan Rathbun
Department of Computer Sciences
Northeastern University
United States
rathbun.e@northeastern.edu

Enrico Marchesini
Department of Computer Science
University of Verona
Italy
enrico.marchesini@univr.it

Xiang Zhi Tan
Department of Computer Sciences
Northeastern University
United States
zhi.tan@northeastern.edu

Abstract

Multi-agent reinforcement learning (MARL) in real-world use cases may need to adapt to external natural language instructions that interrupt ongoing behavior and conflict with long-horizon objectives. However, conditioning rewards on instructions introduces a fundamental failure mode as Bellman updates couple value estimates across instruction contexts, leading to inconsistent values when instructions interrupt macro-actions. We propose *Macro-Action Value Correction for Instruction Compliance* (MAVIC), which fixes Bellman backups at instruction boundaries by correcting the incoming instruction objective and restoring the continuation value under the current one. Unlike reward shaping, MAVIC modifies the bootstrapping target itself, enabling consistent value estimation under stochastic instruction switching within a unified policy. We provide theoretical analysis, an actor-critic implementation, and show that MAVIC achieves high instruction compliance while preserving base task performance in increasingly complex cooperative multi-agent environments.

1 Introduction

Robots operating in partially observable real-world environments must often contend with dynamic natural language instructions from humans. These instructions arrive unpredictably, interrupt ongoing behavior, and may conflict with long-horizon objectives. For example, a robot executing a multi-step cooking task may receive commands such as "bring me the tomato," requiring immediate interruption and re-planning to complete the instructions. In other cases, an instruction such as "don't use the left cutting board" alters the constraints and lead to a different optimal policy. These challenges are amplified in cooperative multi-agent settings, where agents must maintain coordinated behavior while adapting to human inputs. As multi-agent systems are increasingly deployed in human-centered environments, handling such interrupting instructions is a fundamental requirement.

Various approaches have been developed to attempt to address this setting, but each with its own limitation: (i) Symbolic planning methods based on predefined operators (e.g., PDDL, LTL [Tellex et al., 2021, Liu et al., 2023b]) lack flexibility under uncertainty; (ii) Vision-language models (VLMs) enable instruction following [Shi et al., 2025], but are computationally expensive, usually limited to single-agent settings, and struggle with long-horizon coordination. In contrast, multi-agent reinforcement learning (MARL) with macro-actions provides a scalable framework for long-horizon

cooperation, but assumes uninterrupted execution and cannot handle dynamically arriving instructions [Amato et al., 2014, Marchesini et al., 2026b]. A natural solution is to condition rewards on these instructions by rewarding them. However, we show that this introduces a fundamental failure mode as Bellman updates couple value estimates across instruction contexts. When instructions interrupt macro-actions, value functions bootstrap from states governed by different objectives, causing updates to mix incompatible returns. This "cross-contamination" leads agents to ignore instructions or degrade base task performance, revealing a fundamental limitation of MARL under interrupting objectives.

To address this, we introduce *Macro-Action Value Correction for Instruction Compliance* (MAVIC), an interruption-aware value estimation method for macro-action-based MARL. MAVIC corrects Bellman backups at instruction boundaries by subtracting the contribution of the incoming instruction objective and replacing it with the continuation value under the current task objective. This enforces conditional value consistency, ensuring each update reflects a single instruction context despite stochastic switching. Crucially, MAVIC modifies the bootstrapping target itself, yielding value estimates consistent with uninterrupted execution under each instruction while maintaining a unified policy. We provide a theoretical analysis showing that MAVIC decouples value estimation across instruction classes, making the joint problem equivalent to optimizing each instruction-conditioned objective. Based on this theory, we derive a practical actor-critic implementation that integrates with existing macro-action MARL methods and uses a lightweight frozen language encoder. Empirically, MAVIC achieves high instruction compliance while preserving base task performance across multiple cooperative multi-agent environments, eliminating the trade-off observed in existing methods. These results highlight Bellman consistency as a key requirement for robust instruction following in MARL. Our contributions can be summarized as follows:

- Identifying cross-contamination in Bellman updates under instruction interruptions as a key failure mode in macro-action MARL.
- Proposing MAVIC, a value correction mechanism that enforces per-instruction consistency via modified bootstrapping.
- Providing a theoretical analysis showing that MAVIC decouples instruction-conditioned value functions while preserving a unified optimal policy.
- Empirically demonstrating that MAVIC eliminates the trade-off between instruction compliance and task performance across multiple domains.

Enabling interruption-aware learning is a critical step toward deploying multi-agent systems that can reliably integrate human input in complex real-world environments [Yu et al., 2023, Kim et al., 2024, Marchesini et al., 2026a].

2 Preliminaries

We consider a cooperative MARL setting under partial observability, commonly modeled as a decentralized partially observable Markov decision process (Dec-POMDP) [Omidshafiei et al., 2015], which generalizes the single-agent POMDP [Kaelbling et al., 1998]. In many real-world domains, tasks are long-horizon and require temporally extended actions, making learning over 1-step (primitive) actions intractable. Hence, we model the problem as a macro-action Dec-POMDP (MacDec-POMDP) [Amato et al.], where agents select temporally extended actions. A MacDec-POMDP is defined as $\mathcal{M} = \langle I, S, A, M, \Omega, \zeta, T, R, O, Z, \gamma \rangle$ [Amato et al., 2014] (see Table 1).

Table 1: MacDec-POMDP definition where the subscript i denotes individual agent components.

Primitive components & functions		
Symbol	*Definition*	*Description*
S	Specified by task	State space
I	Specified by task	Set of agents
A	$A = \times_{i \in I} A_i$	Joint action space
Ω	$\Omega = \times_{i \in I} \Omega_i$	Joint observation space
H_Ω	$\times_{i \in I} (\Omega_i \times A_i)^{\mathbb{H}}$	Joint history space
γ	$\gamma \in [0, 1]$	Discount factor
$\mathbb{H}$	$\mathbb{H} \in \mathbb{Z}$	Task horizon
R	$R : S \times A \times S \to \mathbb{R}$	Reward function
T	$T : S \times A \to \Delta(S)$	Transition function
O	$O : S \times A \times S \to \Delta(\Omega)$	Observation function
Macro components & functions		
M	$M = \times_{i \in I} \langle \mathcal{I}_{i,j}, \pi_{i,j}, \beta_{i,j} \rangle$	Joint macro-action space
ζ	$\zeta = \times_{i \in I} \zeta_i$	Joint macro-observation space
H_ζ	$\times_{i \in I} (\zeta_i \times M_i)^{\mathbb{H}}$	Joint macro-history space
Z	$Z : S \times A \times S \to \Delta(\zeta)$	Macro observation function

Each agent i stores its local observation Ω_i in its history $H_{\zeta,i}$ and selects macro-actions from a set M_i instead of primitive actions by following a high-level policy $\Psi_i : H_{\zeta,i} \to \Delta M_i$ that maps each agent's local macro-observation history to a distribution over macro-actions. A macro-action $m_{i,j} = \langle \mathcal{I}_{i,j}, \pi_{i,j}, \beta_{i,j} \rangle$ consists of an initiation set $\mathcal{I}_{i,j} \subseteq S$, a low-level policy $\pi_{i,j} : H_\Omega \to A_i$ defined over primitive actions (i.e., common 1-step actions) that execute the actual macro-action behavior lasting for τ steps, and a termination function $\beta_{i,j} : H_\Omega \to [0, 1]$. During execution, macro-actions have unknown duration a priori and unfold over multiple time steps according to $\pi_{i,j}$ until termination, at which point the agents receive a macro-observation $z_i \in \zeta_i$ and select their next macro-actions. The objective is to learn a joint high-level policy $\vec{\Psi} = \times_{i \in I} \Psi_i$ that maximizes the expected discounted return over a finite horizon, starting from an initial state $s_{(0)}$:

$$\max_{\vec{\Psi}} \mathbb{E}_{s_{(0)} \sim \mathcal{M}}[V^{\vec{\Psi}}(s_{(0)})] = \mathbb{E}_{s_{(0)} \sim \mathcal{M}} \left[\sum_{t=0}^{\mathbb{H}-1} \gamma^t R\big(s_{(t)}, \vec{a}_{(t)}\big) \mid s_{(0)}, \vec{\pi}, \vec{\Psi}, H_\zeta \right] \tag{1}$$

3 Macro-Action Value Cancellation for Instruction Compliance

We model human instructions in MacDec-POMDPs as elements of a discrete set of instruction classes C, where each class $c \in C$ corresponds to a set of semantically equivalent natural language commands (e.g., "get me a tomato" and "please grab a tomato" belong to the same class). We include a null instruction class $c_\emptyset = \{\text{“ ”}\}$ indicating the absence of instructions.

To incorporate instructions into the MacDec-POMDP framework, we augment the state and observation spaces with the current instruction. Given a base MacDec-POMDP $\mathcal{M}$, we define an instruction-augmented process $\mathcal{M}_C = \langle I, S \times C, A, M, \Omega \times C, \zeta \times C, T', R', O', Z', \gamma \rangle$, where the instructions modulate the reward function. In detail, each instruction class $c \in C$ induces a reward function R_c, and the modified reward is given by $R'((s, c), a, (s', c')) = R_c(s, a, s')$. Instruction dynamics are governed by a transition function T' that jointly evolves the environment state and instruction as $T'((s, c), a, (s', c')) = T(s, a, s') \cdot P(c' \mid c, h, a)$, where $P(c' \mid c, h, a)$ models the transition probability of instructions based on the primitive history h. Observations are augmented by appending the current instruction, and we force macro-action termination with instruction transitions: when a new instruction is received or an instruction terminates, the currently executing macro-action is interrupted. This coupling enables agents to respond immediately to instructions changes. Finally, we define the learning objective as optimizing expected return under each instruction class to allow agents to seamlessly and optimally solve different human instruction tasks:

$$\max_{\vec{\Psi}} \mathbb{E}_{s_{(0)} \sim \mathcal{M}_C}[V^{\vec{\Psi}}(s_{(0)})] = \mathbb{E}_{s_{(0)} \sim \mathcal{M}_C} \left[\sum_{t=0}^{\mathbb{H}-1} \gamma^t R_c\big(s_{(t)}, \vec{a}_{(t)}\big) \mid s_{(0)}, \vec{\pi}, \vec{\Psi}, H_\zeta \right], \tag{2}$$

This prevents the system from biasing its behavior in anticipation of certain instructions or from ignoring some instructions that would be otherwise suboptimal under different, more common instructions.

3.1 Coupling Value Estimates Across Instruction Classes

The naive approach to follow instructions in macro-action MARL is to directly optimize the objective in Equation 2 by conditioning rewards on the current instruction. However, this naive formulation introduces an unintended coupling between instruction classes, leading to suboptimal learning and potentially degraded performance.[Marchesini et al., 2025]. For example, consider a simplified setting with two instruction classes: a null instruction $c_\emptyset$ and a single active instruction c. Even when evaluating the value of a policy in a non-instruction state $(s, c_\emptyset)$, the expected return depends on the probability of transitioning into an instruction state and the policy's performance under that instruction. Concretely, denoting by β the probability of receiving an instruction, the value function takes the form:

$$V^{\vec{\Psi}}((s, c_\emptyset)) = \mathbb{E}\left[R_{c_\emptyset} + \gamma \left(\beta\, V^{\vec{\Psi}}((s', c)) + (1 - \beta)\, V^{\vec{\Psi}}((s', c_\emptyset)) \right) \right], \tag{3}$$

This highlights that the value in non-instruction states is recursively coupled with its instructions value, and optimizing performance for one instruction class can adversely affect performance. Agents may learn to ignore instructions that are suboptimal with respect to the base task, or alter their base behavior in anticipation of future instructions, leading to degraded overall performance. We refer

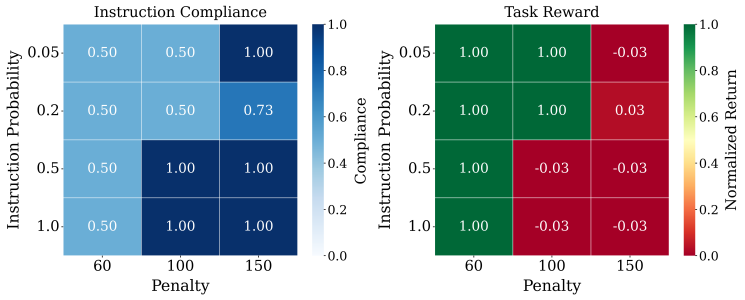


Figure 1: Demonstration of reward cross-contamination in the Box Pushing environment. With a naive reward implementation there is no penalty/probability setting which achieves both high compliance and high task reward.

to this phenomenon as *cross-contamination* of value estimates across instruction classes. Figure 1 demonstrates cross-contamination empirically in the Box Pushing environment (described in Section 4) with a single instruction class "do not push any boxes" resulting in a fixed penalty if the instruction is ignored. Figure 2 additional illustration of this failure mode (top red row), highlighting how cross-contamination propagates through the Bellman objective used in standard MARL methods.

3.2 Value Decoupling for Instruction-Conditioned MARL

Our key idea is to correct Bellman updates at instruction transitions so that value estimates remain consistent within each instruction class (Figure 2, bottom green). When an instruction change occurs, the contribution of the incoming instruction is replaced with the continuation value under the current instruction, $V((h', c))$. This ensures that the update reflects only the return that would have been obtained had the instruction remained unchanged. Consequently, each value function is trained on consistent targets within its own instruction context, eliminating cross-contamination with a unified policy. In contrast, the naive Bellman update bootstraps as $V((h', c'))$, causing value estimates to depend on outcomes from different instruction contexts. MAVIC removes this dependence, decoupling value estimation across instructions while maintaining a shared policy.

Without correction, Bellman updates mix returns across instructions, violating per-instruction value consistency; MAVIC restores this property through value correction. The following theoretical results consider $(\mathcal{M}_C)$ under the value-corrected reward.

Lemma 1. *The value of a policy Ψ in $\mathcal{M}_C$ given an instruction class $c \in C$ is equivalent to its value in the instruction-specific problem $\mathcal{M}_c$ for all $s \in S$.*

To enforce this property, we define a dynamic reward R' (Table 2) that augments R_c at instruction transitions. When $c = c'$, the reward is unchanged. When $c \neq c'$, the correction subtracts the bootstrap from the incoming instruction and replaces it with the continuation value under the current

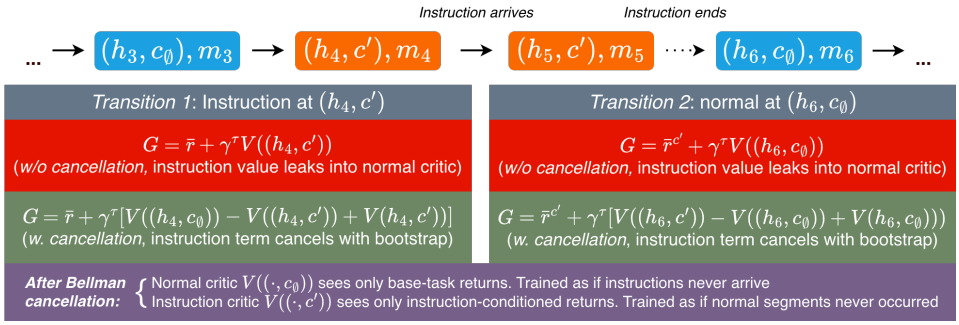


Figure 2: Illustration of value cross-contamination and MAVIC correction. Top (red): standard Bellman updates bootstrap from the value under a new instruction after interruption, causing value estimates to mix across instruction contexts. Bottom (green): MAVIC corrects this contribution and restores the continuation value under the original instruction, ensuring consistent value updates.

instruction. This ensures that future instruction transitions do not influence the value of the current instruction, eliminating cross-contamination.

Table 2: Dynamic reward function $R'((s,c),\vec{a},(s',c'))$

Transition	Reward
$c = c'$	$R_c(s,\vec{a},s')$
$c \neq c'$	$R_c(s,\vec{a},s') - \gamma V^{\vec{\Psi}}((s',c')) + \gamma V^{\vec{\Psi}}((s',c))$

Relation to reward shaping. While the correction may resemble potential-based shaping [Ng et al., 1999], it is fundamentally different. Potential-based methods add a fixed term $\gamma\Phi(s') - \Phi(s)$ and preserve the Bellman operator. In contrast, MAVIC introduces a *policy-dependent* correction that directly modifies the bootstrapping target. It therefore cannot be expressed as a fixed potential function and instead enforces consistency at the level of Bellman updates under stochastic instruction switching. This distinction is critical under interruptions: when an instruction arrives mid-execution, the naive update bootstraps from a different instruction context, introducing bias and training on truncated behaviors. MAVIC instead bootstraps from the intended continuation, preserving consistency with the original macro-action.

Theorem 1. *If a policy Ψ^* is optimal in $\mathcal{M}_C$, then it is also optimal in each instruction-specific problem $\mathcal{M}_c$ for all $c \in C, s \in S$.*

This decoupling arises solely from the modified Bellman updates, without restricting transitions or policies. More broadly, the result applies to settings with stochastic objective switching and temporally extended actions beyond language-conditioned MARL, which is an exciting direction for future investigation.

3.3 MAVIC: Learning with Value Correction

The value-corrected reward formulation (Table 2) provides a principled solution, but requires approximating instruction-conditioned value functions under partial observability and natural language inputs. We address this with a history-conditioned actor-critic architecture and a training procedure that applies value correction at instruction transitions, which can be implemented on top of existing macro-action actor-critic methods such as the asynchronous framework of Xiao et al. [2022].

Figure 3 illustrates the MAVIC architecture. Each agent maintains an actor network Ψ_{θ_i} (where θ_i parametrizes agent's i policy) that selects macro-actions conditioned on its macro-observation history and the current instruction.

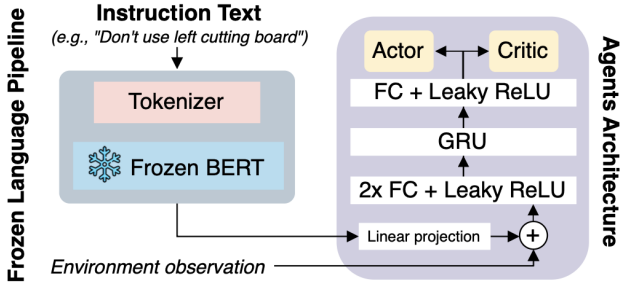


Figure 3: The actor and critic are modified to take the instructions given as natural language as input. This is done through embedding each instruction with a frozen BERT model [Devlin et al., 2019] which is then concatenated with the observation later on.

Natural language instructions are encoded using a frozen language encoder (BERT [Devlin et al., 2019]), and the resulting embedding is projected down and concatenated with the agent's history before being processed by the network. To support value correction, each agent maintains instruction-conditioned critics $V_{\phi_i}((h,c))$ (parametrized by ϕ_i) that estimate the value of a history h under instruction class c. In practice, these critics share parameters but are conditioned on the instruction embedding, enabling generalization across different phrasings of the same instruction. This design replaces state-based value functions $V((s,c))$ of Theorem 1 with history-conditioned approximations $V((h,c))$, which are standard in partially observable settings and serve as sufficient statistics for decision making [Baisero and Amato, 2022].

Algorithm. The MAVIC training procedure is summarized in Algorithm 1. During data collection (lines 3-9), agents execute macro-actions until either a macro-action terminates or a new instruction

Algorithm 1 MAVIC Training Template

Require: $\mathcal{M}$, instruction classes C, frozen language encoder $\mathcal{E}$

1: Initialize actors Ψ_{θ_i}, critics V_{ϕ_i}, replay buffer $\mathcal{D}$
2: **for** episode $= 1, 2, \ldots$ **do**
3: Sample $s_0 \sim \mathcal{M}_C$, set $c_0 = c_\emptyset$, sample $\vec{m} \sim \vec{\Psi}(\cdot \mid h_0, \mathcal{E}(\text{“ ”}))$
4: **while** episode not terminated **do**
5: Execute primitives from $\vec{m}$ until any macro-action terminates; append to $\mathcal{D}$
6: With prob. β_c: terminate all $\vec{m}$, sample $c' \sim \beta_c$, $t \sim c'$
7: Sample $\vec{m} \sim \vec{\Psi}(\cdot \mid h, \mathcal{E}(t))$
8: **end while**
9: **end for**
10: **for** training epoch $= 1, 2, \ldots$ **do**
11: Sample batch $\{(h_k, c_k, h_{k+1}, c_{k+1}, \bar{r}_k, \vec{m}_k)\} \sim \mathcal{D}$
12: **if** $c_k \neq c_{k+1}$ **then**
13: $\bar{r}_k \leftarrow \bar{r}_k + \gamma^{\tau_m}[V_\phi((h_{k+1}, c_k)) - V_\phi((h_{k+1}, c_{k+1}))]$
14: **end if**
15: Compute G via Eq. 5; update Ψ_{θ_i}, V_{ϕ_i} via provided actor-critic gradients
16: **end for**

arrives (line 5). When a macro-action terminates naturally, only the corresponding agent selects a new macro-action. Instruction transitions are sampled according to β_c (line 6), and when a new instruction is received, all active macro-actions are terminated and new macro-actions are selected conditioned on the updated instruction (line 7). Transitions at time t are stored in a replay buffer as tuples $(h_t, c_t, h_{t+1}, c_{t+1}, \bar{r}_t, \vec{m}_t)$, capturing both environment evolution and instruction changes.

During training (lines 10-15), the key step is the application of value correction. For each sampled transition, if an instruction change occurs ($c_t \neq c_{t+1}$), we apply the correction (line 12) where τ_m is the macro-timestep:

$$\bar{r}_t \leftarrow \bar{r}_t + \gamma^{\tau_m}\left[V_\phi((h_{t+1}, c_t)) - V_\phi((h_{t+1}, c_{t+1}))\right], \tag{4}$$

which removes the contribution of the incoming instruction and replaces it with the continuation value under the current instruction. Using this corrected reward, we compute the return G over macro-action segments (line 14):

$$G = \begin{cases} \bar{r}^c + \gamma^{\tau_m} V_c & \text{if } c = c', \\ \bar{r}^c + \gamma^{\tau_m}(V_c - V_{c'}) + \gamma^{\tau_m} V'_c & \text{if } c \neq c'. \end{cases} \tag{5}$$

This value correction mechanism provides a practical realization of Theorem 1. By removing the contribution of future instruction states from each update, the corrected return ensures that the value function is trained as if the instruction class c remained fixed. Consequently, each instruction-conditioned value function is optimized independently, allowing the learned policy to recover per-instruction optimal behavior while remaining unified across all instruction classes.

To implement MAVIC within an actor-critic framework, we derive a macro-action policy gradient that is consistent with the value-corrected objective, ensuring that policy updates respect the decoupled structure established in Theorem 1. The resulting gradient for agent i is given by:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\left[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i, c)\left(\bar{r}_i^c + \gamma^{\tau_{m_i}} V^{\vec{\Psi}}((h'_i, c')) - V^{\vec{\Psi}}((h_i, c))\right)\right]. \tag{6}$$

The full derivation is provided in Appendix A. This gradient mirrors the standard macro-action policy gradient, with $\bar{r}_i^c$ denoting the corrected macro-reward from Equation 5; therefore the complete target is conditioned only on the current instruction context. Then, each critic is trained by minimizing the temporal-difference error $\mathbb{E}\left[(G - V_\phi((h, c)))^2\right]$, and the actor is updated using the policy gradient method with the corrected returns (line 15). As illustrated in Figure 2, this procedure ensures that Bellman updates no longer bootstrap across instruction boundaries, preventing value leakage and enabling stable learning under temporally interleaved instructions.

Convergence and approximation scope MAVIC inherits the convergence assumptions and limitations of the underlying asynchronous macro-action actor-critic framework [Xiao et al., 2022]. The correction itself introduces no additional cross-instruction bootstrap error: because the same critic estimate appears in the cancellation and in the ordinary bootstrap, the incoming-context term cancels exactly as shown in Equation 5. Hence, whenever the underlying actor-critic converges for an instruction-specific objective, MAVIC reconstructs that same learning target at instruction boundaries; Theorem 1 then establishes the corresponding optimality-preservation property across instruction classes. With nonlinear function approximation, we do not claim a stronger global convergence guarantee than that of the underlying actor-critic method.

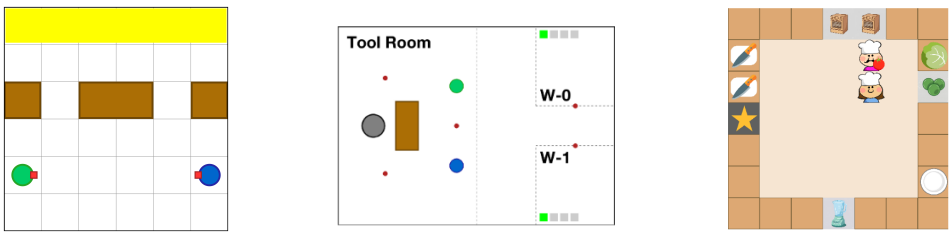
Limitations. MAVIC relies on accurate estimation of instruction-conditioned value functions, which can be challenging under partial observability and sparse instructions, potentially increasing variance in the correction terms. It also assumes that instruction embeddings generalize across semantically equivalent commands, which may degrade under diverse or ambiguous language. Nonetheless, MAVIC focuses on control-level consistency, not language grounding. Finally, while MAVIC mitigates interference between instruction classes, it does not explicitly address exploration over rare or imbalanced instructions. These limitations primarily reflect opportunities for improving value estimation, representation learning, and exploration in instruction-conditioned MARL.

4 Experiments

Environments. We evaluate the performance of MAVIC with the box pushing (BP), warehouse tool delivery (WH) [Xiao et al., 2022], and the overcooked (OC) [Wu et al., 2021] domains, which are standard benchmarks in the MARL macro-action literature. Macro-actions are predefined in both domains and the primitive actions are part of the set of macro-actions. In the following, we briefly describe the environments, referring to Appendix E and the original works for more details:

BP: Two robot agents must coordinate to reach and push a central box to a target location. Instructions include: "Go to small boxes" and "Don't push the box."
WH: Three heterogeneous robot agents (two mobile, one arm) assist three human workers by delivering tools whose required order may change based on instructions. Humans move between work and break stations. Instructions include: "Get me tool 0-3."
OC: Three agents (and a human) collaborate in a grid 7×7 to prepare dishes that require cutting, blending, baking and delivery. Agents receive rewards for correct actions and penalties for delays or mistakes. Instructions include constraints ("Don't use the left cutting board"), suboptimal requests ("Get me the lettuce"), and direct commands ("Move left"). Other instructions will be noted in Appendix E.2

Experiment setup and baselines. We evaluate MAVIC on top of the macro-action actor-critic methods Mac-IAC and Mac-CAC, implemented using the original codebase of [Yu et al., 2023], with hyperparameters selected via grid search (Appendix E.3). We compare MAVIC implementations against two instruction-aware baselines: (i) a naive approach directly using instruction task rewards with no value correction, similar to the top row of Figure 2; (ii) a "switch" training paradigm that gathers training data from each instruction class independently, side stepping the cross-contamination issue. Both baselines highlight different challenges in learning to optimize our task-aware objective. The naive baseline struggles with reward cross-contamination, as we discussed in Section 3.1, and our switch training baseline illuminates the difficulty of exploring unique instruction contexts and reducing training instability even when cross contamination is eliminated. Lastly, all methods are



(a) BP (b) WH (c) OC

Figure 4: Overview of the tasks. BP is Boxpushing, WH is Warehouse, and OC is Overcooked.

Table 3: Comparison of Base and Compliance metrics across different algorithms and environments. Best performing values for each metric are highlighted in bold.

Algorithm performance across environments												
Environment	**OC**				**BP**				**WH**			
Metric	Base	σ	Compl.	σ	Base	σ	Compl.	σ	Base	σ	Compl.	σ
MAVIC	122.28	0.56	99%	1.3%	288.32	4.1	99%	0.14%	438.90	15.238	93%	2.0%
Naive	3.62	9.98	93%	3.3%	270.60	5.68	5%	5.0%	359.12	63.62	92%	1.3%
Switch	104.00	49.46	99%	0.1%	267.85	3.90	6%	7.9%	454.49	30.11	86%	21%
Vanilla Mac-IAC	152.31	22.00	N/A	N/A	290.42	0.01	N/A	N/A	473.65	13.35	N/A	N/A

compared against vanilla Mac-IAC, trained and evaluated without instructions, to act as an upper bound on base task performance.

Metrics We evaluate all methods across two main metrics: base task performance and instruction compliance. Base task performance is measured without instructions being given in the environment, allowing for a clean and accurate measure of how well each method learns to solve the original environment task. Compliance is measured within a live instruction environment, meaning instructions are given online at random intervals during execution. We compute compliance as the ratio of instructions followed divided by the total number of instructions given. For details on how instruction compliance is measured for each environment and instruction pair, see Appendix E.2. Unless otherwise noted, results correspond to the average return over a 10-episode window, aggregated across 5 independent training seeds per method with shaded regions indicating 95% bootstrapped confidence intervals.

4.1 Empirical Evaluation

In Table 3 we compare the performance of MAVIC against the naive and switch baselines across our three evaluation environments. Our results show that MAVIC is the only method consistently capable of achieving both high levels of base task performance and instruction compliance. Specifically, our results for the naive baseline gives further proof of the cross contamination problem we discussed in Section 3.1 as the method either demonstrates a significant drop in base task performance, in the case of OC and WH, or a complete lack of task compliance, as in BP.

Additionally, our switch baseline also struggles in this setting despite being trained independently between instruction and base task instances, showing either lower levels of compliance than MAVIC, in the case of BP and WH, or a drop in base task performance, as in OC. This is a likely result of unstable training induced by per-episode task switching, which is further evidenced by the high standard deviation values for the method seen in OC and WH. Furthermore, since the agents remain within a single task context for each episode under the switch training paradigm, they are less likely to explore a diverse set of initial instruction contexts. For instance, if the agents start with the instruction "don't move", they will never explore other states outside the initial state set once they learn the instruction.

In contrast, MAVIC trains directly within the instruction interruption environment, meaning it can learn how to dynamically switch between tasks while simultaneously prevent cross-task reward contamination. As a result, MAVIC is a more principled approach for training instruction-aware agents by enabling in-context training across all instructions. Additional interruption-recovery, switch-frequency, critic-noise, transition-noise, and reward-shaping analyses are reported in Appendix D.1.

Is value correction necessary? To isolate the contribution of value correction, we compare MAVIC against a zero-bootstrap variant that removes the incoming instruction's bootstrap at instruction boundaries without restoring the continuation value under the original instruction. Table 4 shows that both methods achieve comparable base-task returns in WH. However, their performance under online interruptions differs substantially: zero bootstrap achieves only 4.4% episode completion, compared to 96.9% for MAVIC, and its post-switch return drops from 264.4 to -17. These results demonstrate that eliminating cross-instruction value contamination alone is insufficient. Restoring the continuation value is crucial for preserving coordinated behavior across instruction boundaries.

4.2 Environment Specific Case Studies

Overcooked OC is notable for having high levels of compliance across all methods while seeing the most significant drops in base task performance in the naive and switch baselines. Specifically, under

restrictive instructions like "don't use the left cutting board" or "don't get the tomato" the agents receive a large negative reward (e.g., -50) for accessing the restricted resource. This greatly biases returns received by agents trained with naive rewards, resulting in them avoiding the tomato and left cutting board even when the instructions are inactive. Since these resources are necessary for base task completion, the agents never solve the base task.

We see a similar drop in base task performance for the switch baseline in OC, though not as severe, along with a high level of variance across training seeds. This variance is caused by the switch baseline failing to learn the base task across a number of training seeds. In particular, we noticed a similar failure mode as observed in the naive baseline, the agents learn to always follow restrictive instructions even when they are not given. It is possible that this issue could be solved by adjusting how often the method switches between environment task types, however this meta optimization is completely unnecessary with MAVIC which shows much higher levels of stability across all environments and instructions.

Boxpushing The BP environment has proven to be difficult for all methods to comply with the instruction as the environment is more difficult to solve compared to WH and OC. On BP, complying with "push the small boxes" requires both agents to abandon the dominant +300 big-box reward which is different from OC and WH where compliance is usually a local sub-task swap. Naive and Switch cannot resolve the reward ambiguity and collapse to 5% compliance focusing on task performance, treating the instruction input as noise. MAVIC, however, is able to separate task and instruction cleaner reaching 99% compliance while preserving Vanilla's benign return (288 vs 290).

Warehouse tool delivery Figure 3 shows the results in the WH domain and confirm our previous takeaways. When instructions only reorder existing rewards (e.g., by changing the tool delivery order in WH) without introducing additional shaping terms, MAVIC and the other baselines perform well. In this setting, instruction and base-task objectives remain aligned, so value coupling does not significantly impact learning. However, changing the tool delivery order in the middle of the episode makes the task harder, and MAVIC successfully delivers the new tool correctly with low variance while the other methods struggle. Despite similar rewards, the baselines have a tradeoff by having higher variance and lower compliance. These results highlight that value correction is most critical when instruction-conditioned rewards alter the underlying value structure, indicating that *MAVIC is necessary in tasks where instruction signals induce conflicting value estimates.*

Out-of-Distribution Instructions. Furthermore, we investigate MAVIC's performance on out of distribution instructions not observed during training. We first evaluated MAVIC on instructions semantically similar to those seen during training, but still out of distribution. We show our results in Table 5 where we observe MAVIC's ability to generalize across novel instructions. Next, we probe the behavior of MAVIC under instructions that are far out of distribution, representing irrelevant or infeasible requests (e.g., "fly," "spin around the room"). In Figure 5 we observe that every infeasible instruction results in a single, shared macro-action being chosen with high probability. This uniformity of behavior is reflective of the relatively limited language coverage the agents were trained on, as all instructions are mapped to identical behaviors. This emphasizes the need for calibrated language representations and a diversity of language instructions to ensure better generalization.

5 Related Work

MAVIC relates to prior work on learning under multiple competing objectives. While typically studied in adversarial contexts, recent work in backdoor attacks uses reward constructions to induce agents to optimize temporally interleaved behaviors within a single policy [Rathbun et al., 2024]. MAVIC is inspired by this perspective but differs fundamentally in objective: rather than embedding hidden behaviors, we address a structural inconsistency in value estimation arising from stochastic instruction switching, and correct it at the level of Bellman updates.

A large body of work studies language-guided decision making using VLMs and LLMs [Huang et al., 2022, Liu et al., 2023a, Stone et al., 2023, Kim et al., 2024]. These approaches often employ

Table 4: Zero-bootstrap ablation in Warehouse.

Method	Base return	Recovery	Completion	Post-switch return	Waiting
Zero bootstrap	420.1 ± 1.4	87.8%	$4.4 \pm 6.3\%$	-17 ± 23	230.3 ± 62.6
MAVIC	423.4 ± 28.7	100%	$96.9 \pm 4.4\%$	264.4 ± 46.3	130 ± 24.6

Table 5: Instruction Compliance for Out-of-Distribution Commands

OOD instruction	Intent	Compliance	σ
"the delivery is mine"	deliver	**99.9%**	0.4%
"i've got the right cutting board"	knife-1	**99.9%**	0.5%
"i will get plate 1"	plate-1	99.0%	**4.0%**

hierarchical decomposition, where high-level language reasoning generates subgoals executed by low-level controllers [Shi et al., 2025]. While effective for instruction following, such methods are computationally expensive and struggle with long-horizon coordination, particularly in multi-agent settings. In contrast, MAVIC operates at the control level, treating instructions as stochastic conditioning signals and avoiding expensive language inference at test time.

Outside of LLM-based approaches, prior work incorporates language, safety requirements or human desiderata into RL via reward shaping, temporal logic constraints, or policy conditioning [Shah et al., 2018, Marzari and Marchesini, 2026, Aydeniz et al., 2024, Liu et al., 2023b, Holk et al., 2024, Marzari et al., 2025, Mayoral-Macau et al., 2025, Shi et al., 2024, Cui et al., 2023]. These methods typically assume that standard Bellman updates remain valid when instructions modify rewards or objectives. However, as we show, this assumption breaks down when instructions arrive asynchronously and interrupt temporally extended actions, leading to cross-contamination in value estimates. MAVIC differs fundamentally by modifying the Bellman backup itself, rather than the reward or policy alone, to enforce consistency under stochastic instruction switching in multi-agent settings.

Standard multi-task and goal-conditioned RL methods learn policies across multiple tasks or goals, including partially observable multi-agent settings [Omidshafiei et al., 2017, Chen et al.]. These formulations generally treat the active task or goal as fixed over a rollout or training instance rather than as an objective that can change mid-trajectory and interrupt a temporally extended action. Our "Switch" baseline is a natural adaptation of this independent-context paradigm. MAVIC instead targets the Bellman-boundary inconsistency created by within-episode objective changes and can therefore be viewed as complementary to task/goal conditioning rather than a replacement for it.

6 Conclusion

We introduced MAVIC, a value correction method for following dynamic human instructions in asynchronous cooperative settings. We showed that naively conditioning rewards on instructions induces coupling between value estimates and proposed a value-corrected reward mechanism that prevents instruction-following returns from contaminating base task learning. Combined with an interruption-aware macro-action formulation, MAVIC enables consistent value estimation even under forced early termination. This yields a unified instruction-conditioned policy that generalizes across instruction phrasings via a frozen language encoder. Empirically, MAVIC achieves strong instruction compliance while preserving task performance across increasingly complex environments, outperforming MARL baselines. Additional appendix analyses show robust recovery under interruptions, value-estimation noise, and stochastic transitions.

These results highlight value-level decoupling as a fundamental mechanism for enabling MARL to robustly adapt to dynamic human input in real-world environments. Extending MAVIC to larger teams, non-predefined macro-actions, and open-ended instruction grounding remains future work.

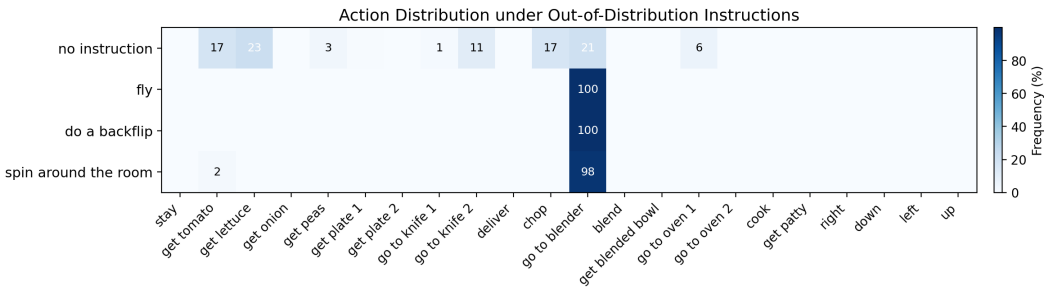


Figure 5: Macro-action frequencies under no instruction and infeasible out-of-distribution ones.

Acknowledgments and Disclosure of Funding

The authors thank the reviewers for their insightful and constructive feedback, which has substantially improved the quality of this work. We also thank everyone from PARCs lab for feedback and also Shuo Liu, Andrea Baisero, Rupali Bhati, Xueguang Lyu, and everyone from LLPR Lab for helpful discussions and feedback throughout the project.

This work was supported in part by the Italian Ministry of University and Research (MUR) through the Italian Science Fund (FIS 3), project *BeTrusty* (Grant FIS-2024-05614).

This work was also supported in part by Northeastern University startup funds.

References

Christopher Amato, George D Konidaris, and Leslie P Kaelbling. Planning with Macro-Actions in Decentralized POMDPs.

Christopher Amato, George D. Konidaris, Gabriel Cruz, Christopher A. Maynor, Jonathan P. How, and Leslie P. Kaelbling. Planning for Decentralized Control of Multiple Robots Under Uncertainty, February 2014. URL http://arxiv.org/abs/1402.2871. arXiv:1402.2871 [cs].

Ayhan Alp Aydeniz, Enrico Marchesini, Christopher Amato, and Kagan Tumer. Entropy seeking constrained multiagent reinforcement learning. In *23rd International Conference on Autonomous Agents and Multiagent Systems*, pages 2141–2143, 2024.

Andrea Baisero and Christopher Amato. Unbiased Asymmetric Reinforcement Learning under Partial Observability, January 2022. URL http://arxiv.org/abs/2105.11674. arXiv:2105.11674 [cs].

Jiayu Chen, Yuanxin Zhang, Yuanfan Xu, Huimin Ma, Huazhong Yang, Jiaming Song, Yu Wang, and Yi Wu. Variational Automatic Curriculum Learning for Sparse-Reward Cooperative Multi-Agent Problems.

Yuchen Cui, Siddharth Karamcheti, Raj Palleti, Nidhya Shivakumar, Percy Liang, and Dorsa Sadigh. No, to the Right: Online Language Corrections for Robotic Manipulation via Shared Autonomy. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '23, pages 93–101, New York, NY, USA, March 2023. Association for Computing Machinery. ISBN 978-1-4503-9964-7. doi: 10.1145/3568162.3578623. URL https://dl.acm.org/doi/10.1145/3568162.3578623.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. URL http://arxiv.org/abs/1810.04805. arXiv:1810.04805 [cs].

Simon Holk, Daniel Marta, and Iolanda Leite. PREDILECT: Preferences Delineated with Zero-Shot Language-based Reasoning in Reinforcement Learning. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '24, pages 259–268, New York, NY, USA, March 2024. Association for Computing Machinery. ISBN 979-8-4007-0322-5. doi: 10.1145/3610977.3634970. URL https://dl.acm.org/doi/10.1145/3610977.3634970.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents, March 2022. URL http://arxiv.org/abs/2201.07207. arXiv:2201.07207 [cs].

Whiyoung Jung, Sunghoon Hong, Deunsol Yoon, Kanghoon Lee, and Woohyung Lim. Agent-Centric Actor-Critic for Asynchronous Multi-Agent Reinforcement Learning.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, May 1998. ISSN 00043702. doi: 10.1016/S0004-3702(98)00023-X. URL https://linkinghub.elsevier.com/retrieve/pii/S000437029800023X.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An Open-Source Vision-Language-Action Model, September 2024. URL http://arxiv.org/abs/2406.09246. arXiv:2406.09246 [cs].

Huihan Liu, Alice Chen, Yuke Zhu, Adith Swaminathan, Andrey Kolobov, and Ching-An Cheng. Interactive Robot Learning from Verbal Correction, October 2023a. URL http://arxiv.org/abs/2310.17555. arXiv:2310.17555 [cs].

Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Grounding Complex Natural Language Commands for Temporal Tasks in Unseen Environments, October 2023b. URL http://arxiv.org/abs/2302.11649. arXiv:2302.11649 [cs].

Enrico Marchesini, Andrea Baisero, Rupali Bhati, and Christopher Amato. On stateful value factorization in multi-agent reinforcement learning. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, page 1445–1453, 2025. ISBN 9798400714269.

Enrico Marchesini, Eva Boguslawski, Alessandro Leite, Christopher Amato, Dussartrem Matthieu, Marc Schoenauer, Benjamin Donnot, and Priya L. Donti. MARL2grid-TR: A multi-agent RL benchmark in power grid operations. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=mpAMH1OyMO.

Enrico Marchesini, Wo Wei Lin, Priya Donti, and Christopher Amato. Asynchronous value factorization for multi-robot cooperative reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026b.

Luca Marzari and Enrico Marchesini. Probabilistic verification of recurrent neural networks for single and multi-agent reinforcement learning. In *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-26*, pages 262–270, 2026.

Luca Marzari, Priya L. Donti, Changliu Liu, and Enrico Marchesini. Improving policy optimization via ϵ-retrain. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2025*, pages 1464–1472, 2025. doi: 10.5555/3709347.3743780.

Arnau Mayoral-Macau, Manel Rodriguez-Soto, Enrico Marchesini, Martí Sanchez-Fibla, Maite López-Sanchez, Juan Antonio Rodríguez-Aguilar, and Alessandro Farinelli. An approximate embedding for designing ethical reinforcement learning environments. In *ECAI 2025*, pages 2458–2465, 2025. doi: 10.3233/FAIA251093.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, page 278–287, 1999. ISBN 1558606122.

Shayegan Omidshafiei, Ali-akbar Agha-mohammadi, Christopher Amato, and Jonathan P. How. Decentralized Control of Partially Observable Markov Decision Processes using Belief Space Macro-actions, February 2015. URL http://arxiv.org/abs/1502.06030. arXiv:1502.06030 [cs].

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability, July 2017. URL http://arxiv.org/abs/1703.06182. arXiv:1703.06182 [cs].

Ethan Rathbun, Christopher Amato, and Alina Oprea. SleeperNets: Universal Backdoor Poisoning Attacks Against Reinforcement Learning Agents, May 2024. URL http://arxiv.org/abs/2405.20539. arXiv:2405.20539.

Pararth Shah, Marek Fiser, Aleksandra Faust, J. Chase Kew, and Dilek Hakkani-Tur. FollowNet: Robot Navigation by Following Natural Language Directions with Deep Reinforcement Learning, May 2018. URL http://arxiv.org/abs/1805.06150. arXiv:1805.06150 [cs].

Lucy Xiaoyang Shi, Zheyuan Hu, Tony Z. Zhao, Archit Sharma, Karl Pertsch, Jianlan Luo, Sergey Levine, and Chelsea Finn. Yell At Your Robot: Improving On-the-Fly from Language Corrections, March 2024. URL http://arxiv.org/abs/2403.12910. arXiv:2403.12910 [cs].

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi Robot: Open-Ended Instruction Following with Hierarchical Vision-Language-Action Models. June 2025. URL https://openreview.net/forum?id=lNVHg9npif.

Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Sean Kirmani, Brianna Zitkovich, Fei Xia, Chelsea Finn, and Karol Hausman. Open-World Object Manipulation using Pre-trained Vision-Language Models, October 2023. URL http://arxiv.org/abs/2303.00905. arXiv:2303.00905 [cs].

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, November 1999. MIT Press.

Stefanie Tellex, Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, and Lawson LS Wong. Interpreting human-robot instructions, August 2021. URL https://patents.google.com/patent/US11086938B2/en.

Zifan Wu, Chao Yu, Deheng Ye, Junge Zhang, haiyin piao, and Hankz Hankui Zhuo. Coordinated Proximal Policy Optimization. In *Advances in Neural Information Processing Systems*, volume 34, pages 26437–26448. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/de73998802680548b916f1947ffbad76-Abstract.html.

Yuchen Xiao, Weihao Tan, and Christopher Amato. Asynchronous Actor-Critic for Multi-Agent Reinforcement Learning, October 2022. URL http://arxiv.org/abs/2209.10113. arXiv:2209.10113 [cs].

Chao Yu, Xinyi Yang, Jiaxuan Gao, Jiayu Chen, Yunfei Li, Jijia Liu, Yunfei Xiang, Ruixin Huang, Huazhong Yang, Yi Wu, and Yu Wang. Asynchronous Multi-Agent Reinforcement Learning for Efficient Real-Time Multi-Robot Cooperative Exploration, April 2023. URL http://arxiv.org/abs/2301.03398. arXiv:2301.03398 [cs].

A Theorems

A.1 Macro-Action-Based Policy Gradient Theorem under R'

We adapt the macro-action-based policy gradient theorem MacDec-POMDP $\mathcal{M}_C$ with modified reward R'. The Bellman equation for the value of a history-based policy $\vec{\Psi}$ under R' takes the form:

$$V^{\vec{\Psi}}((h,c)) = \sum_{m} \vec{\Psi}(m \mid h,c)\, Q^{\vec{\Psi}}((h,c),m) \tag{7}$$

$$Q^{\vec{\Psi}}((h,c),m) = \bar{r}^{\,c}(h,m) + \sum_{(h',c')} P((h',c') \mid (h,c),m)\, V^{\vec{\Psi}}((h',c')) \tag{8}$$

$$\bar{r}^{\,c}(h,m) = \mathbb{E}_{\tau_m \sim \beta_m, s_{t_m}|h}\left[\sum_{t=t_m}^{t_m+\tau_m-1} \gamma^{\,t-t_m} R'((s,c),\vec{a},(s',c'))\right] \tag{9}$$

and $P((h',c') \mid (h,c),m)$ is the joint transition over histories and instruction classes induced by T'.

Now the standard policy gradient derivation [Sutton et al., 1999]:

$$\nabla_\theta V^{\vec{\Psi}_\theta}((h,c)) = \nabla_\theta \left[\sum_{m} \vec{\Psi}_\theta(m \mid h,c)\, Q^{\vec{\Psi}_\theta}((h,c),m)\right] \tag{10}$$

$$= \sum_{m} \Big[\nabla_\theta \vec{\Psi}_\theta(m \mid h,c)\, Q^{\vec{\Psi}_\theta}((h,c),m) + \vec{\Psi}_\theta(m \mid h,c) \nabla_\theta Q^{\vec{\Psi}_\theta}((h,c),m)\Big] \tag{11}$$

$$= \sum_{m} \Big[\nabla_\theta \vec{\Psi}_\theta(m \mid h,c)\, Q^{\vec{\Psi}_\theta}((h,c),m) + \vec{\Psi}_\theta(m \mid h,c) \sum_{(h',c')} P((h',c') \mid (h,c),m) \nabla_\theta V^{\vec{\Psi}_\theta}((h',c'))\Big] \tag{12}$$

where $(\hat{h},\hat{c})$ ranges over instruction-conditioned histories reachable from (h,c) in k macro-action steps under $\vec{\Psi}_\theta$, and $P\big((h,c) \to (\hat{h},\hat{c}), k, \vec{\Psi}_\theta\big)$ is the corresponding k-step transition probability. By repeated unrolling:

$$\nabla_\theta V^{\vec{\Psi}_\theta}((h,c)) = \sum_{(\hat{h},\hat{c})} \sum_{k=0}^{\infty} P\big((h,c) \to (\hat{h},\hat{c}), k, \vec{\Psi}_\theta\big) \sum_{m} \nabla_\theta \vec{\Psi}_\theta(m \mid \hat{h},\hat{c})\, Q^{\vec{\Psi}_\theta}((\hat{h},\hat{c}),m) \tag{13}$$

where $P\big((h,c) \to (\hat{h},\hat{c}), k, \vec{\Psi}_\theta\big)$ is the probability of reaching $(\hat{h},\hat{c})$ from (h,c). The policy gradient is then:

$$\nabla_\theta J(\theta) = \mathbb{E}_{(h,c)\sim\rho^{\vec{\Psi}_\theta}, m\sim\vec{\Psi}_\theta}\left[\nabla_\theta \log \vec{\Psi}_\theta(m \mid h,c)\, Q^{\vec{\Psi}_\theta}((h,c),m)\right] \tag{14}$$

where $\rho^{\vec{\Psi}_\theta}$ is the discounted state-distribution induced by $\vec{\Psi}_\theta$ over instruction histories.

In our actor-critic implementation, we replace $Q^{\vec{\Psi}_\theta}$ with a learned advantage estimate using a history-value baseline, giving the policy gradient used during training:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\left[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i,c)\left(\tilde{r}_i^{\,c} + \gamma^{\tau_{m_i}} V^{\vec{\Psi}}((h_i',c')) - V^{\vec{\Psi}}((h_i,c))\right)\right] \tag{15}$$

where $\tilde{r}_i^{\,c}$ is the cancellation-corrected macro-reward, so the advantage uses the complete corrected target from Equation 5.

A.2 Proof of Theorem 1

In this section we provide a formal proof of Theorem 1. We will begin by proving an intermediate lemma.

Proof. **Lemma 1** *The value of a policy Ψ in $\mathcal{M}_C$ given an instruction class $c \in C$ is equivalent to its value in the instruction specific problem $\mathcal{M}_c$ for all $s \in S$.*

Let $\mathcal{M}_C$ be some arbitrary MacDec-POMDP with instruction classes C and reward design specified in Section 3.2. Next, without loss of generality, let $c \in C$ be some instruction class in C and let $\mathcal{M}_c$ denote a single-task MacDec-POMDP defined as:

$$\mathcal{M}_c = \langle I, S \times \{c\}, A, M, \Omega \times \{c\}, \zeta \times \{c\}, T', R', O', Z', \gamma \rangle \tag{16}$$

where $T'((s,c), \vec{a}, (s',c)) = T(s, \vec{a}, s')$ and $R'((s,c), \vec{a}, (s',c)) = R_c(s, \vec{a}, s')$ for all $s \in S$. Since $\mathcal{M}_c$ only includes one instruction class and only ever computes rewards over R_c, solving $\mathcal{M}_c$ requires optimizing the instruction class c alone. Next, let $\vec{\Psi}$ be an arbitrary instruction policy $\vec{\Psi} : (\mathcal{T} \times H_\zeta) \to \Delta M$. We will now prove that the value of $\vec{\Psi}$ in $\mathcal{M}_C$ is equivalent to its value in $\mathcal{M}_c$ for all $h \in H_\zeta$ and $t \in c$. First, recall that the value of $\vec{\Psi}$ in $\mathcal{M}_c$ is the following given an arbitrary $s \in S$ and history $\vec{h} \sim H_\zeta$ sampled given $\mathcal{M}_c$ and $\vec{\Psi}$:

$$V_c^{\vec{\Psi}}(\vec{h}) = \sum_{\vec{m} \in M} \vec{\Psi}(\vec{m}|\vec{h}) \sum_{\vec{h}' \in H_\zeta,\ s' \in S} P(\vec{h}', s'|\vec{h}, \vec{m}, s) \Big[R_c(s, \vec{a}, s') + \gamma^\tau V_c^{\vec{\Psi}}(\vec{h}') \Big] \tag{17}$$

Next, lets expand the value of $\vec{\Psi}$ in the multi-task problem $\mathcal{M}_C$ given an arbitrary (s,c) for $s \in S$ and history $\vec{h} \sim H_\zeta$ sampled given $\mathcal{M}_C$ and $\vec{\Psi}$:

$$\begin{aligned} V^{\vec{\Psi}}((\vec{h},c)) = &\sum_{\vec{m} \in M} \vec{\Psi}(\vec{m}|(\vec{h},c)) \sum_{\vec{h}' \in H_\zeta,\ s' \in S} P((\vec{h}',c), s'|(\vec{h},c), \vec{m}, s) \Big[R_c(s, \vec{a}, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) \Big] \\ &+ \sum_{c' \in C \setminus c} P((\vec{h}',c'), s'|(\vec{h},c'), \vec{m}, s) \Big[R'((s,c), \vec{a}, (s',c')) + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c')) \Big] \end{aligned}$$

Next we expand the definition further based upon our definition of R' from Table 2:

$$\begin{aligned} &= \sum_{\vec{m} \in M} \vec{\Psi}(\vec{m}|(\vec{h},c)) \sum_{\vec{h}' \in H_\zeta,\ s' \in S} P((\vec{h}',c), s'|(\vec{h},c), \vec{m}, s) \Big[R_c(s, \vec{a}, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) \Big] \\ &+ \sum_{c' \in C \setminus c} P((\vec{h}',c'), s'|(\vec{h},c'), \vec{m}, s) \Big[(R_c(s, \vec{a}, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) - \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c'))) + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c')) \Big] \\ &= \sum_{\vec{m} \in M} \vec{\Psi}(\vec{m}|(\vec{h},c)) \sum_{\vec{h}' \in H_\zeta,\ s' \in S} P((\vec{h}',c), s'|(\vec{h},c), \vec{m}, s) \Big[R_c(s, \vec{a}, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) \Big] \\ &+ \sum_{c' \in C \setminus c} P((\vec{h}',c'), s'|(\vec{h},c'), \vec{m}, s) \Big[R_c(s, a, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) \Big] \end{aligned}$$

Notably, the second term, which is a sum over $c' \in C \setminus c$ is now equivalent to the first term, just scaled by a different probability, therefore we can combine them as follows:

$$\begin{aligned} &= \sum_{\vec{m} \in M} \vec{\Psi}(\vec{m}|(\vec{h},c)) \sum_{\vec{h}' \in H_\zeta,\ s' \in S} P(\vec{h}', s'|\vec{h}, \vec{m}, s) \sum_{c' \in C} P(c'|c, h, \vec{M}) \Big[R_c(s, \vec{a}, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) \Big] \\ &= \sum_{\vec{m} \in M} \vec{\Psi}(\vec{m}|(\vec{h},c)) \sum_{\vec{h}' \in H_\zeta,\ s' \in S} P(\vec{h}', s'|\vec{h}, \vec{m}, s) \Big[R_c(s, \vec{a}, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) \Big] \sum_{c' \in C} P(c'|c, h, \vec{M}) \\ &= \sum_{\vec{m} \in M} \vec{\Psi}(\vec{m}|(\vec{h},c)) \sum_{\vec{h}' \in H_\zeta,\ s' \in S} P(\vec{h}', s'|\vec{h}, \vec{m}, s) \Big[R_c(s, \vec{a}, s') + \gamma^\tau V^{\vec{\Psi}}((\vec{h}',c)) \Big] \\ &= V_c^{\vec{\Psi}}(\vec{h}) \end{aligned}$$

as a result, we have shown that $V^{\vec{\Psi}}((\vec{h},c))$ does not depend on any other instruction histories or instruction states (h',c'), (s',c') for $c' \neq c$. The value only depends on rewards from R_c and future values within c. Therefore, we have proven Lemma 1

□

Now that we have proven Lemma 1, we will proceed with proving our final result in Theorem 1.

Proof. **Theorem 1.** *If a policy Ψ^* is optimal in $\mathcal{M}_C$ then it also is optimal in each instruction-specific problem $\mathcal{M}_c$ for all $c \in C, s \in S$.*

We will proceed with a proof by contradiction.

Let Ψ^* be an optimal policy in the multi-instruction problem $\mathcal{M}_C$, but assume Ψ^* is not optimal in an instruction specific problem $\mathcal{M}_c$ for some $c \in C$.

This implies that $\exists\ \Psi'$ and $h \in H_\zeta$ such that $V_c^{\psi'}(h) > V_c^{\psi^*}(h)$ in $\mathcal{M}_c$.

Therefore, from Lemma 1 we know the following to be true in the multi-task problem $\mathcal{M}_C$: $V^{\psi'}((h,c)) > V^{\psi^*}((h,c))$

However, this implies that Ψ^* is not optimal in $\mathcal{M}_C$ which is a contradiction.

Therefore, if Ψ^* is optimal in $\mathcal{M}_C$ it must also be optimal in the task specific problem $\mathcal{M}_c$ for all $s \in S$ and $c \in C$. □

B State-Value Calculation

The value cancellation principle also applies to the action-value function Q during macro-action interruptions. With our unified state notation, transitions between the base task and different instruction classes follow a consistent formulation. When agent i is executing macro-action m_i under a current instruction class c (which may be the base task) and a new instruction c' arrives at primitive step $T_{\text{inst}} < \tau_{m_i}$, forcing early termination, the action-value function becomes:

$$Q_i((h_i,c),m_i;T_{\text{inst}}) = \sum_{t=0}^{T_{\text{inst}}-1} \gamma^t r^c_{(t_{m_i}+t)} + \gamma^{T_{\text{inst}}}\left[V^{\vec{\Psi}}((h'_i,c)) - V^{\vec{\Psi}}((h'_i,c'))\right] \tag{18}$$

where (h'_i, c') is the post-interruption history and new instruction class, and $r^c_{(t_{m_i}+t)}$ is the primitive reward at each step from the start of the macro-action t_{m_i} under the original instruction c.

Symmetrically, when agent i is executing m_i under an active instruction c and the instruction ends at step T_{inst}, returning the agent to the base task (denoted by instruction class c'), the exact same formulation applies:

$$Q_i((h_i,c),m_i;T_{\text{inst}}) = \sum_{t=0}^{T_{\text{inst}}-1} \gamma^t r^c_{(t_{m_i}+t)} + \gamma^{T_{\text{inst}}}\left[V^{\vec{\Psi}}((h'_i,c)) - V^{\vec{\Psi}}((h'_i,c'))\right] \tag{19}$$

since the unified value function $V^{\vec{\Psi}}((h,c))$ intrinsically evaluates states across all modes.

Under our assumption that no further instruction interrupts an active instruction, the instruction-mode case simplifies to:

$$Q_i((h_i,c),m_i;T_{\text{inst}}) = \sum_{t=0}^{T_{\text{inst}}-1} \gamma^t r^c_{(t_{m_i}+t)} \tag{20}$$

since during the instruction mode, no further interruption occurs and a new macro-action in normal mode is sampled only after the instruction ends.

In both cases, the value cancellation at the interruption boundary follows the same principle as the macro-action-level formulation: the value under the arriving mode (c') is subtracted, and the value under the departing mode (c) is added, ensuring that partial returns accumulated before the interruption do not contaminate the values in the opposing mode. For example, if an instruction arrives at the third primitive step, the return includes the original-mode rewards $r_0^c + \gamma r_1^c + \gamma^2 r_2^c$ accumulated prior to interruption, followed by the bootstrapped value-difference correction from that point forward.

When no interruption occurs (i.e., $T_{\text{inst}} = \tau_{m_i}$), these reduce to the standard macro-action returns:

$$Q_i((h_i, c), m_i) = \bar{r}_i^c + \gamma^{\tau_{m_i}} V^{\vec{\Psi}}((h_i', c)) \tag{21}$$

where $\bar{r}_i^c = \sum_{t=t_{m_i}}^{t_{m_i}+\tau_{m_i}-1} \gamma^{t-t_{m_i}} r_{(t)}^c$ is the standard cumulative reward of the macro-action under instruction class c.

By Theorem 3.3, the policy update under the modified reward R' produces gradients identical to those of the unmodified MacDec-POMDP on each mode. This allows a single shared actor per agent rather than training separate actors for different instruction classes. The actor is updated with the standard policy gradient:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}\left[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i \mid h_i, c)\left(G_t - V^{\vec{\Psi}}((h_i, c))\right)\right] \tag{22}$$

C Asynchronous Execution

Due to the nature of macro-actions having variable durations, agents in a MacDec-POMDP are able to execute asynchronously: each agent terminates and selects new macro-actions independently. For that reason, standard synchronous actor critic methods do not directly handle this as macro-observations are only sampled when an agent completes its macro-action. Recent work has extended actor critics to asynchronous settings [Xiao et al., 2022, Jung et al.]. MAVIC builds on top of this foundation by introducing value correction at instruction boundaries.

D Additional Experiments

D.1 Interruption and Robustness Stress Tests

Interruption recovery diagnostics. To directly measure the coordination cost of online interruptions, we evaluate Warehouse trajectories with approximately 2.7 interruptions per episode. MAVIC recovers the instructed objective after interruptions to either heterogeneous robot type and converts recovery into coordinated task progress: episode completion after Fetch interruptions rises from 27.8% to 96.9%, human waiting is reduced by roughly 80–90 steps, and post-interruption instruction return more than doubles.

Table 6: Warehouse interruption diagnostics (5 seeds; ~ 2.7 interruptions/episode).

Metric	Naive	MAVIC
Recovery after Fetch interruption	$90.0 \pm 10.0\%$	$\mathbf{100.0 \pm 0.0\%}$
Recovery after Turtlebot interruption	$95.6 \pm 3.8\%$	$\mathbf{100.0 \pm 0.0\%}$
Episode completion (Fetch interruption)	$27.8 \pm 28.9\%$	$\mathbf{96.9 \pm 4.4\%}$
Human waiting (Fetch)	227 ± 35.4	$\mathbf{147 \pm 11.8}$
Human waiting (Turtlebot)	223 ± 40.8	$\mathbf{130 \pm 24.6}$
Post-interruption instruction return	108 ± 115.1	$\mathbf{264.4 \pm 46.3}$

Instruction-switch frequency. We vary instruction-switch probability during training. At low switching probability the methods are similar because cross-instruction interference is limited; as switching becomes more frequent, the naive baseline develops substantially higher variance and lower base-task return, whereas MAVIC remains stable. At 100%, both methods have base return -200 because only instruction tasks are performed.

Controlled critic perturbations. To test sensitivity to value-estimation error directly, we inject zero-mean Gaussian noise into the critic estimates used by the cancellation terms. Despite substantial perturbations, MAVIC does not exhibit systematic divergence or amplified instability.

Table 7: Warehouse performance versus instruction-switch frequency (5 seeds).

Switch	Naive base	Naive compl.	MAVIC base	MAVIC compl.
10%	$\mathbf{460.5 \pm 10.3}$	$99.93 \pm 0.09\%$	454.1 ± 5.1	$99.97 \pm 0.06\%$
30%	396.4 ± 15.4	$99.09 \pm 0.35\%$	$\mathbf{439 \pm 26}$	$99.13 \pm 0.43\%$
50%	419.7 ± 20.1	$97.88 \pm 0.76\%$	$\mathbf{433 \pm 1}$	$98.33 \pm 0.75\%$
75%	139.5 ± 323.0	$97.33 \pm 1.19\%$	$\mathbf{408 \pm 10}$	$96.67 \pm 1.24\%$
100%	-200	$96.59 \pm 1.58\%$	-200	$94.84 \pm 1.09\%$

Table 8: MAVIC return under injected critic noise in Warehouse.

Critic noise σ	Mean return
0	428.4 ± 18
10	353.8 ± 18
25	356.7 ± 9
50	366.2 ± 43
75	411.7 ± 18
100	289.4 ± 51

Transition noise. The optional Gaussian transition noise in Warehouse was disabled in the original main experiment. We enable it here to test robustness to stochastic dynamics. MAVIC retains near-complete recovery at every tested noise level, while instruction-active return varies moderately without systematic collapse.

Table 9: MAVIC with Gaussian transition noise in Warehouse.

σ	Overall	Instr.-active	Instr.-free	Recovery	Completion
0.00	432.8 ± 4.5	417.0 ± 3.4	437.1 ± 5.7	$100.0 \pm 0.0\%$	$96.9 \pm 4.4\%$
0.05	447.9 ± 11.0	408.1 ± 6.3	454.1 ± 10.0	$100.0 \pm 0.0\%$	$90.3 \pm 2.0\%$
0.10	438.0 ± 5.0	428.7 ± 5.0	440.5 ± 5.0	$99.3 \pm 1.2\%$	$100.0 \pm 0.0\%$
0.20	437.0 ± 9.3	384.4 ± 63.5	444.0 ± 5.9	$100.0 \pm 0.0\%$	$83.7 \pm 21.4\%$

Reward-shaping scale. We scale the Warehouse instruction-shaping terms while keeping the objective switch itself. MAVIC maintains strong return and recovery even at $0\times$, where there is no compliance bonus or penalty, indicating that its advantage is not an artifact of a large engineered shaping term.

Switch-count stress test. In a larger 10×10 Box Pushing variant, we increase the number of mid-macro-action switches per episode. The gap grows at frequent interruptions: both methods lose absolute return because little recovery time remains, but MAVIC continues attempting the big-box objective when instructions become inactive.

Instruction Compliance. Figure 6 compares the percentage of instruction compliance for the naive baseline and MAVIC across training (early: 50k, mid: 100k, late: 150k episodes). Both methods achieve high compliance throughout training, with the baseline slightly higher in all phases. However, this difference reflects over-prioritization of instruction rewards rather than better instruction handling as shown in Figure 7. Because instruction rewards dominate base-task rewards in Overcooked (refer to E.2), the baseline learns to favor instruction-driven behavior even when no instruction is active, effectively anticipating instructions. This leads to degraded task performance despite high compliance. In contrast, MAVIC achieves similarly high compliance while preserving base-task returns by decoupling instruction-conditioned value estimates. These results highlight that *instruction compliance alone is not sufficient: without value cancellation, it can come at the expense of overall task performance.*

Table 10: Warehouse reward-shaping ablation.

Scale	Pen./search/deliv.	Naive base	MAVIC base	Naive rec.	MAVIC rec.
$0\times$	$0/0/0$	18.4 ± 60.5	$\mathbf{387.9 \pm 25.3}$	$77.9 \pm 9.2\%$	$\mathbf{91.2 \pm 7.8}\%$
$0.25\times$	$-6.25/+2.5/+12.5$	151.3 ± 270.6	$\mathbf{435.7 \pm 6.4}$	$87.3 \pm 15.4\%$	$\mathbf{98.7 \pm 2.3}\%$
$0.5\times$	$-12.5/+5/+25$	-53.1 ± 144.9	$\mathbf{445.0 \pm 1.0}$	$56.0 \pm 48.6\%$	$\mathbf{99.2 \pm 0.7}\%$
$1\times$	$-25/+10/+50$	419.7 ± 20.1	$\mathbf{433.0 \pm 1.0}$	$98.7 \pm 0.6\%$	$\mathbf{100.0 \pm 0.0}\%$

Table 11: Base-task return versus number of instruction switches in 10×10 Box Pushing.

Switches	MAVIC	Naive
0	242 ± 2	242 ± 2
1	240 ± 2	228 ± 3
2	148 ± 101	185 ± 78
4	$\mathbf{112 \pm 103}$	53 ± 57
8	$\mathbf{55 \pm 72}$	23 ± 8

In (a), we describe how in the baseline method (naive), agents learn to anticipate instruction that don't come which causes the overall returns to perform poorly. Figure 7(b,c) further break down performance by instruction type. In (b), agents receive a suboptimal request ("get me the tomato") that introduces a positive shaping bonus tied to a specific ingredient. This induces coordination failure, as agents over-prioritize the instructed item instead of distributing roles across ingredients. The baseline plateaus at low return because this reward bias leaks into non-instruction value targets, distorting the base policy. In contrast, MAVIC maintains near-vanilla performance on non-instruction segments while correctly responding to the instruction, demonstrating effective separation between objectives. In (c), agents receive a prohibitive instruction ("don't use the right cutting board") that also imposes a large negative penalty. The baseline collapses because these penalties contaminate base-task value estimates, leading agents to avoid the cutting board even when the instruction is inactive, preventing task completion. MAVIC localizes this penalty to instruction states, preserving base-task behavior and recovering full performance.

When is Value Correction Necessary? We conduct an ablation study to evaluate the contribution of MAVIC by comparing to a dual actor-critic variant of Mac-IAC (Figure 8). Each agent maintains separate actor-critic pairs for base-task and instruction-conditioned learning. One pair is trained solely on the base reward, while the other is trained only on instruction-conditioned rewards. Despite this explicit separation at the data and network level, the method underperforms MAVIC. This is because value estimates remain coupled at instruction-transition boundaries: returns still bootstrap across modes, introducing bias. These results show that separating data or networks is insufficient as *value correction is required to eliminate cross-contamination at the level of Bellman updates.*

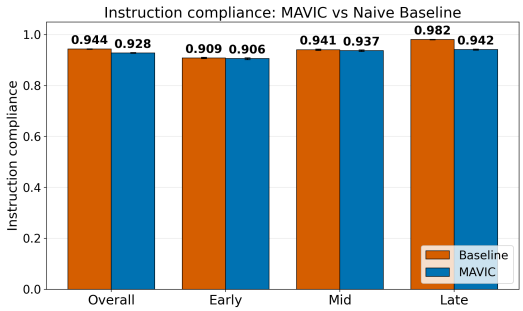
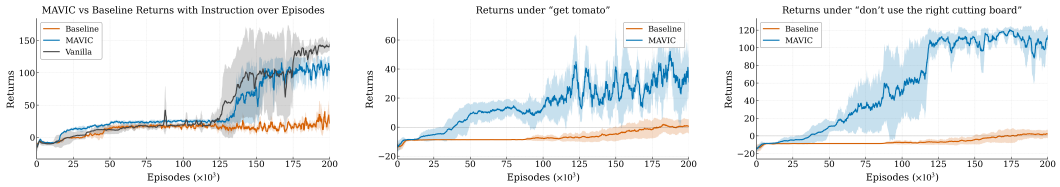


Figure 6: Avg. instruction compliance of the baseline vs MAVIC for all the given instructions across training.



(a) Avg. return for the task with instructions.

(b) Avg. return for "get me the tomato" instruction.

(c) Avg. return for "don't use the right cutting board" instruction.

Figure 7: Empirical evaluation in the OC domain for Mac-IAC and its variants.

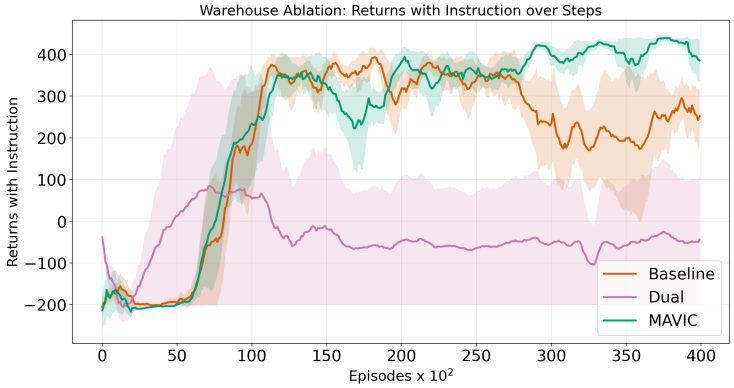


Figure 8: Return comparison when instructions are active with Mac-IAC MAVIC method, Dual-Policy Method, and Baseline Mac-IAC. The Dual Policy method is trained by separating and training using only instruction experiences for one actor and non-instruction experiences for another actor.

E Implementation Details

E.1 Computational Resources

Experiments were run on a research cluster equipped with Xeon E5-2650 CPU nodes and 256 of RAM.

E.2 Environments

E.2.1 Overcooked

Goal. Two or three agents need to learn to cooperatively prepare a lettuce–peas–tomato patty and deliver it to the 'star' counter cell as soon as possible. The challenge is that neither the recipe nor the tool sequence is given to the agents. They must discover that lettuce and tomato need to be chopped, that the chopped lettuce, chopped tomato, and (raw) peas must be combined in the blender, that the resulting blended bowl must be cooked in an oven to form a patty, and that the patty must finally be plated and delivered.

State Space. The environment is a 7×7 grid world containing two agents, one tomato, one lettuce, one bag of peas, two plates, two cutting boards, one blender, two ovens, and one delivery cell. The global state consists of the positions of every agent and item, the chop status of each choppable vegetable (chopped/unchopped/under-chopping with progress), the blender status (empty / filling / blending-progress / blended), and the status of each oven (empty / cooking-progress / cooked).

Primitive-Action Space. Each agent has five primitive actions: *up, down, left, right, stay*. Picking, placing, chopping, blending, and cooking are all triggered by standing next to the corresponding cell and moving against it.

Macro-Action Space. We list the main effect of each macro-action and its termination conditions.

- Five one-step macro-actions identical to the primitives;

- CHOP, cuts a raw vegetable into pieces (three time steps) when the agent stands next to a cutting board with an unchopped vegetable on it; otherwise it does nothing. It terminates when:
  - the vegetable on the cutting board has been chopped;
  - the agent is not next to a cutting board;
  - there is no unchopped vegetable on the board;
  - the agent already holds something in hand.
- GET-LETTUCE, GET-TOMATO, GET-PEAS, navigate the agent to the latest observed position of the ingredient and pick it up if it is there; otherwise the agent moves to check the initial position. Termination conditions:
  - the agent successfully picks up the ingredient;
  - the ingredient is observed to be already held by another agent or by itself;
  - the agent is holding something else;
  - the agent's path is blocked by another agent;
  - the ingredient is found at neither the latest nor the initial location;
  - the ingredient has already been consumed by the blender / blended bowl / patty (Map-D specific) so it is no longer a valid pickup target;
  - the agent has lower priority than another agent contending for the same cell.
- GET-PLATE-1, navigate to the latest observed position of the plate and pick it up; otherwise the agent moves to check the initial position. Termination conditions are analogous to the GET-vegetable macros.
- GO-CUT-BOARD-1/2, navigates the agent to the corresponding cutting board:
  - the agent stops in front of the board and places an in-hand item on it if the board is unoccupied;
  - if a teammate is using the target board, the agent stops next to the teammate;
  - the agent has lower priority than another agent contending for the same cell.
- GO-BLENDER, navigates the agent to the blender. If the agent holds a chopped lettuce, chopped tomato, or peas, the ingredient is placed into the blender on arrival; if the contents are already blended, the agent stops in front of the blender ready to pick up the resulting blended bowl. The macro terminates upon arrival in front of the blender or upon priority-loss with another agent.
- BLEND, runs the blender for five time steps when the agent stands next to it and the blender contains all three required ingredients (chopped lettuce, chopped tomato, and peas). It terminates when:
  - the contents have been fully blended (five steps);
  - the agent is not next to the blender, or the blender cannot blend (missing ingredients), or the contents are already blended.
- GET-BLENDED-BOWL, navigates to and picks up the blended bowl from the blender, with termination conditions analogous to the GET-vegetable macros.
- GO-OVEN-1/2, navigates the agent to the corresponding oven. If the agent holds a blended bowl, the bowl is placed into the oven on arrival; if the oven contains a cooked product, the agent stops in front of it ready to pick up the resulting patty. Termination is analogous to GO-BLENDER.
- COOK, runs the oven for ten time steps when the agent stands next to it and the oven currently contains a blended bowl that is not yet cooked. Termination conditions are analogous to BLEND.
- GET-PATTY, navigates the agent to the cooked oven and picks up the patty, with termination conditions analogous to the GET-vegetable macros.
- DELIVER, navigates the agent to the 'star' delivery cell:
  - the agent places the in-hand item on the cell;
  - if a teammate stands in front of the star cell, the agent stops next to the teammate;
  - the agent has lower priority than another agent contending for the same cell.

Observation Space. Each agent observes a 5×5 window centered on itself, containing the positions and statuses of the entities (including the blender and ovens) that fall inside the window. The initial position of every item is known to all agents.

Dynamics. Transitions are deterministic. To chop a vegetable into pieces an agent must stand next to a cutting board and execute *left* three times; to blend, the agent stands next to the blender and executes *up* five times after all three ingredients (chopped lettuce, chopped tomato, peas) are inside; to cook, the agent stands next to the oven holding the blended bowl and executes *down* ten times. *Only* a cooked patty (and *not* the raw chopped vegetables or the blended bowl) is a valid item to plate. Delivering any wrong entity resets it to its initial position.

Reward. $+10$ for each subtask milestone (chopping a tomato or lettuce, finishing a blend, finishing a cook), $+200$ terminal reward for delivering the lettuce–peas–tomato patty, -5 for delivering any incorrect entity, and -0.1 per time step. Each subtask reward is paid at most once per episode.

Instructions. Whenever a natural-language instruction is active for agent i at a macro-decision step, a per-agent compliance term is added to that agent's reward: 0 when the chosen macro-action satisfies the instruction (i.e. it lies in the instruction's allowed set for positive instructions such as *"get tomato"* / *"chop"* / *"deliver"*, or it lies outside the prohibited set for negative instructions such as *"don't touch the lettuce"* / *"don't use the left oven"* / *"let me do all the chopping"*), and -50 when it violates the instruction. Agents that receive no instruction at step t receive the base task rewards. When "let me do all the chopping is called", it will be in an environment with 3 agents or replaced by human. The third agent will be in charge of chopping.

Natural Language Instruction Set Below is the complete set of natural language instructions used in the Overcooked domain, categorized by their core intent and effect on agent behavior.

Prohibition Instructions *(Low-cost compliance where an alternative exists)*

- **Avoid Right Cutting Board:** "don't use the right cutting board", "avoid the right cutting board", "stay away from the right cutting board", "don't go to the right cutting board", "do not touch the right cutting board", "please don't use the right cutting board", "i'll handle the right cutting board", "let me handle the right cutting board", "skip the right cutting board", "the right cutting board is mine", "don't chop on the right", "use the left knife instead", "no chopping on the right", "right cutting board is off limits", "keep away from the right knife".
- **Avoid Left Cutting Board:** "don't use the left cutting board", "don't use left cutting board", "avoid the left cutting board", "stay away from the left cutting board", "don't go to the left cutting board", "do not touch the left cutting board", "please don't use the left cutting board", "i'll handle the left cutting board", "let me handle the left cutting board", "skip the left cutting board", "the left cutting board is mine", "don't chop on the left", "use the right knife instead", "no chopping on the left", "left cutting board is off limits", "keep away from the left knife".

2. Command Instructions *(Forces a specific action)*

- **Stay/Halt Position:** "stay", "stay still", "don't move", "wait there", "hold position", "i'll handle it", "i've got this", "let me take care of everything", "don't help right now", "stand by", "stop what you're doing", "freeze", "just wait".

3. Suboptimal Instructions *(Compliance imposes a routing or coordination cost)*

- **Avoid Plate 1:** "don't use plate 1", "avoid plate 1", "skip plate 1", "don't take plate 1", "don't get plate 1", "use plate 2 instead", "leave plate 1 alone", "i'll get plate 1", "plate 1 is mine", "don't grab plate 1".
- **Avoid Plate 2:** "don't use plate 2", "avoid plate 2", "skip plate 2", "don't take plate 2", "don't get plate 2", "use plate 1 instead", "leave plate 2 alone", "i'll get plate 2", "plate 2 is mine", "don't grab plate 2".

- **Avoid Left Oven:** “don’t use the left oven”, “don’t go to the left oven”, “avoid left oven”, “avoid the left oven”, “skip the left oven”, “use the right oven instead”, “stay away from the left oven”, “left oven is off limits”.
- **Avoid Right Oven:** “don’t use the right oven”, “don’t go to the right oven”, “avoid right oven”, “avoid the right oven”, “skip the right oven”, “use the left oven instead”, “stay away from the right oven”, “right oven is off limits”.
- **Partner Delivers:** “i will deliver”, “i’ll deliver”, “i will deliver it myself”, “let me deliver”, “i’ve got the delivery”, “leave the delivery to me”, “don’t worry about delivering”, “don’t deliver”, “skip delivering”, “no delivering”.
- **Partner Chops:** “let me do all the chopping”, “i’ll do the chopping”, “leave the chopping to me”, “don’t chop anything”, “i’ll handle all the chopping”, “you don’t need to chop”, “stop chopping”, “i’ve got the cutting boards”.

Episode Termination. The episode ends when the agents successfully deliver the lettuce–peas–tomato patty, or when the maximum horizon of 200 time steps is reached.

E.2.2 Warehouse

Goal. Two or three simulated heterogeneous robots — two Turtlebots and one Fetch robot — must cooperate to serve two human workers as quickly as possible. Each human follows an assembly task whose intermediate steps each require a specific tool to be delivered before the human can advance. The Turtlebots transport tools between the tool room and the work area; the Fetch robot searches for and hands tools over to a Turtlebot waiting at the tool-room table. None of the agents directly observes the global state, so they must learn to coordinate through their local observations and the timing of macro-action terminations.

State Space. The environment is a $7\,\text{m} \times 5\,\text{m}$ continuous workspace divided by a wall into a tool room (left half) and a work area (right half). The global state consists of the 2D position of each Turtlebot, the 2D position of the Fetch robot, the contents of each Turtlebot’s basket, the queue of tools currently on the Fetch robot’s hand-over spots, the current step index of each human’s assembly task, and the remaining time of each human’s current step.

Macro-Action Space. The two Turtlebots share the same macro-action set; the Fetch robot has a different one. None of the agents has primitive actions exposed at the policy level: every macro-action runs an internal low-level controller that consumes a (state-dependent) number of time steps.

Turtlebot macro-actions.

- GO-WA0, drives the Turtlebot to the waypoint in front of human 0 in the work area. Terminates when the Turtlebot reaches the waypoint; if at arrival the Turtlebot’s basket contains the tool that human 0 currently requests, the tool is automatically handed to the human.
- GO-WA1, identical to GO-WA0 but for human 1.
- GO-TOOL-ROOM, drives the Turtlebot to the waiting waypoint inside the tool room and terminates upon arrival.
- GET-TOOL, drives the Turtlebot to the tool-room table and waits there for the Fetch robot to hand over an object. It terminates when:
  - the Fetch robot finishes a PASS-OBJ action aimed at this Turtlebot (whether or not an object was actually transferred);
  - the waiting time exceeds the maximum `get_tool_wait` (the sum of the look-for-object and pass-object time costs).

Fetch-robot macro-actions.

- WAIT-REQUEST, idles for one time step; always terminates immediately.
- LOOK-FOR-TOOL-i ($i = 0, 1, 2$), searches for and retrieves an instance of tool i from the tool-room shelves. Terminates after a fixed look-for time cost ($T_{\text{look}} = 6$, optionally with

manipulation noise) or immediately if the Fetch robot already holds two pending objects or no tool i remains.

- PASS-OBJ-T0 / PASS-OBJ-T1, hands the front-most pending object to Turtlebot 0 or Turtlebot 1. Terminates after the pass time cost ($T_{\text{pass}} = 4$). The action is *successful* (object is dropped into the Turtlebot's basket) only if the targeted Turtlebot is at the tool-room table and currently executing GET-TOOL; otherwise the object is dropped and the team incurs a drop-object penalty.

Observation Space. Each Turtlebot observes a binary feature vector consisting of (i) a one-hot of its own current discrete location (work-area-0, work-area-1, tool-room waiting, tool-room table, en-route), (ii) the current step index of each of the two humans in the work area (observable only when the Turtlebot is at the corresponding work area), (iii) which tools are in its own basket, and (iv) a 2-bit ready-flag indicating which Turtlebot has an object waiting on the hand-over spot. The Fetch robot observes which tool indices are currently sitting on the hand-over spots, a 2-bit indicator of which Turtlebot is at the tool-room table, and a per-Turtlebot one-hot of the tool currently being requested by the Turtlebot in front of the table (revealed only for Turtlebots physically present).

Dynamics. Turtlebots move continuously at a fixed speed of 0.8 m/s along straight-line paths to the requested waypoint, optionally with Gaussian transition noise. A Turtlebot can only deliver a tool to the human in front of it if the tool currently held in its basket matches the human's current requested tool, and the delivery happens automatically on macro-action arrival. The Fetch robot's manipulation actions take fixed (or noisy) time costs and may fail the hand-over if the Turtlebot is not yet ready. Each human consumes one time step at every step of its assembly chain; once the requested tool has been delivered *and* the per-step time has elapsed, the human advances to the next step.

Reward. $+100$ team reward for each correctly delivered tool that satisfies a human's current request, an additional fixed delay penalty (-20) per delayed delivery (i.e., the requested tool arrives only after the human has already exceeded the per-step time budget), -10 for each dropped object, and -1 per time step.

Instructions. Warehouse instructions take the form *"fetch tool k first"* and apply only to the Fetch robot (Turtlebots receive no shaping for these instructions). While the priority tool k has not yet been delivered, the Fetch robot receives a per-step compliance term added to its reward: $+10$ when its current macro-action is consistent with the priority (i.e. it is not a LOOK-FOR-OBJ-j for any $j \neq k$), and -25 when it executes a LOOK-FOR-OBJ-j with $j \neq k$. In addition, the first tool actually delivered to a human in the episode triggers a one-shot terminal credit on the Fetch robot of $+50$ if the delivered tool matches the priority tool k, or -50 otherwise; once this one-shot credit fires the dense compliance shaping is disabled for the rest of the episode so the Fetch robot is free to fetch the remaining tools to complete the humans' assembly tasks

Episode Termination. The episode terminates when both humans have reached the final step of their assembly tasks (their last requested tool has been delivered), or when the maximal horizon of 200 time steps is reached.

E.2.3 BoxPushing

Goal. Two agents need to learn to cooperatively push boxes from the bottom half of a 6×6 grid world to the top (goal) row as fast as possible. The map contains two small boxes that can each be pushed by a single agent, and one large (double-width) box that can only be moved when both agents push it from below *in synchrony*. Coordinating on the large box yields a much higher payoff than splitting up to push the small boxes, but mis-coordinating wastes time and incurs penalties.

State Space. The environment is a 6×6 discrete grid containing two agents, two small (1×1) boxes, and one large (1×2) box. The global state consists of each agent's (x, y) position and orientation (N/E/S/W), and the (x, y) position of each box. Agents start in the bottom row at $(0.5, 1.5)$ and $(5.5, 1.5)$ both facing east/west; the two small boxes start at $(0.5, 3.5)$ and $(5.5, 3.5)$, and the large box at $(3.0, 3.5)$ spanning columns 2.5 and 3.5.

Macro-Action Space. Agents do not act with primitive grid moves; their policies select among the following macro-actions, each driven by an internal A* low-level controller:

- GO-TO-SMALL-BOX-0 / GO-TO-SMALL-BOX-1, navigates the agent to the waypoint immediately below the corresponding small box, terminating when the agent reaches the waypoint or when its path is blocked by the teammate.
- GO-TO-BIG-BOX-0 / GO-TO-BIG-BOX-1, navigates the agent to one of the two waypoints below the large box (its left and right halves respectively), with the same termination conditions as above.
- PUSH, advances the agent one cell in the direction of its current orientation. It terminates when:
  - the agent reaches the goal row;
  - the agent attempts to push into a wall (the action is rejected and a penalty is incurred);
  - the agent attempts to push the large box alone or from a non-zero (non-north) orientation (the action is rejected and a penalty is incurred);
  - the small box ahead is successfully pushed onto the goal row.

  The large box advances one row only when *both* agents select PUSH simultaneously while standing on the big-box waypoints with northward orientation.
- TURN-LEFT / TURN-RIGHT, rotates the agent by 90° counter-/clockwise; one-step actions, always terminating immediately.
- STAY, does nothing for one time step.

Observation Space. Each agent observes only the cell directly in front of its current orientation, encoded as a 5-bit binary vector with the labels {*small-box, large-box, empty, wall, teammate*}. Agents do *not* observe their own position, the position of the other agent, or the position of any box outside the front cell.

Dynamics. Transitions are deterministic. The two small boxes can each be pushed independently by a single agent moving north into them. The large box can only be moved when both agents simultaneously execute PUSH from the two big-box waypoints, both facing north; in that case the large box and both agents advance one row per shared PUSH. Any single-agent attempt to push the large box, or any push into a wall, is rejected and produces no movement.

Reward. $+10$ for each small box pushed onto the goal row, $+100$ for the large box reaching the goal row, -5 for each invalid push (into a wall or against the large box without coordination), and -0.1 per time step.

Instructions. Whenever a natural-language instruction is active for an agent at a macro-decision step, a per-agent compliance term is added to that agent's reward: 0 when the chosen macro-action satisfies the instruction (i.e. it lies in the instruction's allowed set for positive instructions such as *"go to small box 0"*, *"go to big box spot 1"*, or *"push"*, or it lies outside the prohibited set for negative instructions such as *"don't go to small box 0"*, *"avoid small boxes"*, or *"don't push"*), and -50 when it violates the instruction. Agents that receive no instruction at step t receive no shaping. The shaping is applied once per macro-decision (not once per primitive step) and only to the agent that holds the instruction.

Episode Termination. The episode terminates as soon as any one box reaches the goal row, or when the maximum horizon of 100 time steps is reached.

E.3 Hyperparameters

Table 12: Each network consists of two FC layers with Leaky-ReLU, one GRU layer, and one more FC layer followed by a linear output layer. Mac-IAC is fully decentralized (each agent owns an independent actor and critic of the same widths). Mac-CAC keeps a decentralized actor per agent but uses a single centralized critic that conditions on the joint macro-observation and macro-action.

Domain Method	Box Pushing Mac-CAC	Overcooked Mac-IAC	Warehouse Mac-IAC
Actor (decentralized)			
MLP-1	32	32	32
MLP-2	32	32	32
GRU	32	32	32
MLP-3	32	32	32
Critic (Mac-IAC: decentralized; Mac-CAC: centralized)			
MLP-1	32	32	32
MLP-2	32	32	32
GRU	32	32	32
MLP-3	32	32	32

Table 13: Hyper-parameters used for Mac-IAC in Overcooked Map-D, Task 9 (*lettuce–peas–tomato patty*).

Parameter	Mac-IAC
Training Episodes	400K
Actor learning rate	4e-4
Critic learning rate	5e-3
Episodes per train (`train_freq`)	16
Parallel envs (`n_env`)	16
Target-net update freq (episode)	32
N-step TD	5
ϵ_{start}	1.0
ϵ_{end}	0.10
ϵ_{decay} (episodes)	100K
γ	0.992
Episode horizon	200

Table 14: Hyper-parameters used for Mac-IAC in Warehouse-A (`OSD-D-v7`, 3 agents, 2 humans, 3 tools; human step times $[27, 20, 20, 20]$ for both humans).

Parameter	Mac-IAC
Training Episodes	40K
Actor learning rate	3e-4
Critic learning rate	3e-3
Episodes per train (`train_freq`)	4
Parallel envs (`n_env`)	4
Target-net update freq (episode)	32
N-step TD	5
ϵ_{start}	1.0
ϵ_{end}	0.01
ϵ_{decay} (episodes)	10K
γ	1.0
Episode horizon	200

Table 15: Hyper-parameters used for Mac-CAC in Box Pushing 6×6 (2 agents, episode horizon 100, big-box reward 300).

Parameter	Mac-CAC
Training Episodes	50K
Actor learning rate	5e-4
Critic learning rate	3e-3
Episodes per train (`train_freq`)	32
Parallel envs (`n_env`)	16
Target-net update freq (episode)	32
N-step TD	0
ϵ_{start}	1.0
ϵ_{end}	0.01
ϵ_{decay} (episodes)	4K
γ	0.995
Episode horizon	100

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes the main claims are reflected in the paper in Section 3 and Section 4. Proofs of the theoretical results are in the Appendix.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: There is a limitation section.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The main statements are in Section 3 and proofs are in the appendix.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have a section 4 describing the setup and the architecture and hyperparameters are in the appendix.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: We will release the code upon acceptance.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We have a section 4 describing the setup and the architecture and hyperparameters are in the appendix

   Guidelines:

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We have a section 4 describing the setup and how multiple runs are conducted.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have a section 4 describing the setup.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work has the potential to make RL agents comply more to instructions and we do not see foresee any societal impact. We did not conduct any experiments with human users.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper shows a fundamental issues in instruction following in multi-agent reinforcement learning. Due to the theoretical nature of the work, there is no perceived societal impacts to discuss.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not not believe the model and training paradigm can be misused.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We mentioned the original codebase in which we built MAVIC.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not introduce any new assets.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We do not conduct any experiments with humans.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We do not conduct any experiments with actual humans.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [No]

Justification: We do not use LLMs for the research. Only used for editing.