CASH: Capability-Aware Shared Hypernetworks for Flexible Heterogeneous Multi-Robot Coordination

Kevin Fu*, **Shalin Anand Jain***, **Pierce Howell, Harish Ravichandar**
Georgia Institute of Technology, Atlanta, USA

Abstract: Recent advances have enabled heterogeneous multi-robot teams to learn complex and effective coordination skills. However, existing neural architectures that support heterogeneous teaming tend to force a trade-off between expressivity and efficiency. Shared-parameter designs prioritize sample efficiency by enabling a single network to be shared across all or a pre-specified subset of robots (via input augmentations), but tend to limit behavioral diversity. In contrast, recent designs employ a separate policy for each robot, enabling greater diversity and expressivity at the cost of efficiency and generalization. Our key insight is that such tradeoffs can be avoided by viewing these design choices as ends of a broad spectrum. Inspired by recent work in transfer and meta learning, and building on prior work in multi-robot task allocation, we propose Capability-Aware Shared Hypernetworks (CASH), a *soft weight sharing* architecture that uses hypernetworks to efficiently learn a *flexible* shared policy that dynamically adapts to each robot post training. By explicitly encoding the impact of robot capabilities (e.g., speed and payload) on collective behavior, CASH enables *zero-shot generalization* to unseen robots or team compositions. Our experiments involve multiple heterogeneous tasks, three learning paradigms (imitation learning, value-based, and policy-gradient RL), and SOTA multi-robot simulation (JaxMARL) and hardware (Robotarium) platforms. Across all conditions, we find that CASH generates appropriately-diverse behaviors and consistently outperforms baseline architectures in terms of performance and sample efficiency during both training and zero-shot generalization, all with 60%-80% fewer learnable parameters.

Keywords: Multi-Robot Learning, Heterogeneous Teams, Parameter Sharing

1 Introduction

Imagine multiple fire departments gathering multiple heterogeneous robots into a team to suppress a wildfire. To be effective, the robots must readily adopt diverse roles and reason about how their individual and collective capabilities (e.g., speed, water-capacity, and sensing radius) interact with the environment (e.g., fire intensities and locations). Further, since the team composition is often unknown until runtime and robot capabilities could deteriorate, we need flexible strategies that zero-shot generalize to different team compositions and robot capabilities.

We address the challenge of learning *flexible* and *diverse* coordination strategies that enable *heterogeneous* robots to adapt their behaviors based on their individual and collective *capabilities*. While learning a separate policy for each robot will enable diverse behaviors [1], they cannot generalize to unseen robots and require significantly more learnable parameters. As such, we aim to encode diverse strategies within a *shared yet flexible* policy, without sacrificing generalization or efficiency.

We propose a novel neural architecture named Capability-Aware Shared Hypernetworks (CASH) to enable flexible and diverse coordination (see Fig. 1). CASH's encoder helps learn *robot-agnostic* coordination strategies that are shared across all robots regardless of their capabilities. Its Hyper Adapter uses a hypernetwork [2] to determine the weights of its Adaptive Decoder *on-the-fly* based

*Equal Contribution
Project Website: https://star-lab.cc.gatech.edu/papers/fu-jain-CASH-CoRL/

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

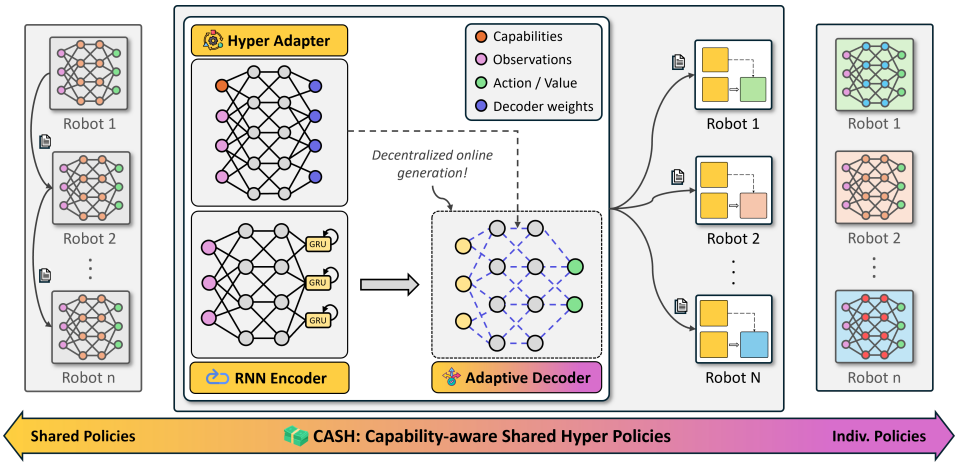

Figure 1: We introduce Capability-Aware Shared Hypernetworks (CASH) (*middle*), a novel class of *soft parameter sharing* architectures that establishes and spans the broad spectrum between shared (*left*) and individualized (*right*) parameter designs. CASH enables effective decentralized heterogeneous teaming, generalization to unseen robots, diverse behaviors, and greater learning efficiency.

on the current context (i.e, observations) and the robots' capabilities. Our use of hypernetworks in CASH helps establish *soft weight sharing* architectures within the context of heterogeneous coordination, and encode *robot-* and *context-specific* strategies within a flexible shared architecture. CASH allows some *policy* parameters to vary across robots, while all *learnable* parameters are shared.

CASH offers several practical benefits. First, CASH improves learning efficiency by significantly reducing the number of learnable parameters and sharing them across heterogeneous robots. Second, CASH enables zero-shot generalization to robots and team compositions unseen during training. Third, CASH can *automatically* learn appropriate levels of behavioral diversity. Fourth, CASH can adapt to *online changes* in robots capabilities. Fifth, CASH can be utilized within any learning paradigm (e.g., imitation learning or RL).

We evaluated CASH on four heterogeneous coordination tasks (wildfire suppression, mining, material transport, and predator-prey) with three separate learning paradigms (imitation learning, value-based RL, and policy-based RL). Our experiments involved a state-of-the-art MARL framework (JaxMARL [3]) and an established hardware testbed for multi-robot systems (Robotarium [4]). Our results demonstrate that CASH consistently outperforms existing (independent and shared) architectures in terms of sample efficiency, and zero-shot generalization to new robots, new team compositions, and larger teams. Further, we show that these benefits extend to physical deployments, where CASH can adapt to online changes to robot capabilities. Notably, CASH offers such improvements while using 60% to 80% fewer learnable parameters than existing architectures.

2 Related Works

Architectures for Heterogeneous Coordination: Shared-parameter architectures – where all agents share a single set of learnable parameters – improve cooperation, learning efficiency, and scalability [5, 6]. However, sharing parameters can prohibit diverse agent behaviors and unique roles. While this limitation is often immaterial for homogeneous teams (e.g., formation control [7] and path planning [8, 9]), it can critically limit heterogeneous teams. To extend shared-parameter architectures to heterogeneous teams, prior work has simply appended unique information about each agent to the input [10, 11]; typically, a *unique ID* assigned to each agent. While such ID-based designs tend to improve efficiency and scalability, recent work has demonstrated that they often fail to learn sufficient behavioral diversity and are not robust to noisy environments [12, 1].

In contrast, *individualized policies* [1, 13] (also referred to as independent or heterogeneous policies) dedicate a separate policy for each agent with no parameter sharing to enable robust and diverse behaviors [14]. Naturally, this improvement comes at the cost of training efficiency due to signifi-

cantly more learnable parameters and limited data reuse. Further, these approaches do not allow for zero-shot generalization to robots not seen in training.

Prior work has also explored a hybrid approach known as *selective* parameter sharing in which shared parameters are constrained to groups of agents that share a certain characteristic (e.g., robot type [13], action space [15], inferred role [12]) with no sharing between groups. These approaches are more efficient than individual policy architectures and generate diverse behaviors unlike shared-parameter architectures. However, they too cannot generalize to new robots and assume a small number of known robot types.

We view shared and individualized-parameter architectures as two ends of a broad spectrum with selective sharing falling somewhere in the middle. Within this context, CASH establishes a new class of *soft weight sharing* architectures that can *adaptively* span this spectrum to simultaneously achieve sample efficiency and behavioral diversity while also generalizing to unseen robots.

Hypernetworks in multi-agent learning: The most notable use of hypernetworks in multi-agent learning is QMIX [16], a popular off-policy MARL algorithm. QMIX uses hypernetworks to mix individual value estimates to improve decentralized coordination via centralized training. In contrast, our novelty is in using hypernetworks to flexibly determine parameters within *individual* robots' policy or value networks based on their capabilities and local observations, enabling a single architecture to encode diverse and adaptive behaviors. To the best of our knowledge, there is only one other parallel work that employs hypernetworks within multi-agent policies [17] which conditions the hypernetwork on agent-specific IDs to produce diverse policies while retaining parameter efficiency. Unlike this work, our design allows for generalization to unseen robots since we condition the hypernetwork on robot capabilities and observations and not on assigned or learned IDs.

3 Problem Formulation and Objectives

We model our problem as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [18] defined by a tuple $(\mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{O}, O, \mathcal{R}, \mathcal{T})$, where $\mathcal{D} = \{1, ..., n\}$ is the set of robots in the team, $\mathcal{S}$ is the set of global states, $\mathcal{A} = \times_{i \in \mathcal{D}} \mathcal{A}_i$ is the joint action space, $\mathcal{O} = \times_{i \in \mathcal{D}} \mathcal{O}_i$ is the joint observation space, $O = P(o|s)$ is the observation function, where o is the joint observation for all robots $\{o_1, ..., o_n\}$, $\mathcal{R}$ is the reward function, and $\mathcal{T} = P(s'|s, a)$ is the transition function.

At each timestep t, every robot receives an observation o_i^t from the joint observation $o^t \sim O(\cdot|s^t)$. Each robot then acts according to their decentralized policy $a_i^t \sim \pi_i(o_i^t)$, forming a joint action $a^t = \{a_1, ..., a_n\}$. The optimal solution to the Dec-POMDP with a fully cooperative team is the set of policies $\{\pi_1^*, ..., \pi_n^*\}$ that maximize the team's expected reward $\mathbb{E}[\sum_{t=0}^{T} r_t]$.

We specifically focus on teams where the robots have heterogeneous capabilities (e.g., speed and sensing radius) [19, 11]; such nominal capabilities as we define them can be readily obtained from robot specifications or sensors. To enable a richer specification of robot heterogeneity, we use $C^t = \{c_1^t, ..., c_n^t\}$ to denote the team's capabilities, where $c_i^t \in \mathbb{R}^m$ represents the capabilities of ith robot at time t. Therefore, we modify the Dec-POMDP definition by giving each policy $\pi_i(\cdot)$ access to the team's collective capabilities, yielding $a_i^t \sim \pi_i(o_i^t, C^t)$.

Finally, our objective is to design a single shared-parameter policy architecture ($\pi = \pi_i, \forall i$) that can produce diverse behaviors and generalize to unseen robots by reasoning about robot capabilities. See Sec. 2 for a discussion of existing policy designs.

4 CASH: Capability-Aware Shared Hypernetworks

In this section, we introduce CASH, a new class of shared-parameter architectures for flexible and generalizable heterogeneous coordination. CASH contains three primary modules (see Fig. 1).

RNN Encoder: In line with contemporary designs [20, 16], CASH encodes local observations using a gated recurrent network (GRU) [21] to handle longer time-horizons and partial observability. The latent embedding generated by this encoder is passed as input to the Adaptive Decoder.

Adaptive Decoder: Following standard practice, we implement our decoder as either a single layer or a two-layer MLP depending on the learning paradigm (see Appendix C for details). Note that only the *structure* of our decoder is identical across robots. Unlike prior designs, CASH allows the decoder's *parameters* (determined by the Hyper Adapter) to be unique to each robot and context.

Hyper Adapter: To allow the Adaptive Decoder to flexibly adapt to each robot and its context, we generate its weights using a hypernetwork-based Adapter module. To represent the current context, we condition the Hyper Adapter on the ego-robot's capabilities, the team's capabilities, and the ego observation. For additional implementation details, see Appendix B and C.1.

By using a hypernetwork and sharing *all learnable parameters* across robots, CASH retains the typical benefits of both full parameter sharing (sample efficiency and generalization to unseen robots) and individualized parameters (diverse and effective heterogeneous coordination). Further, CASH supports *decentralization* as it relies only on local observations and capability information.

We employ hypernetworks [2] since they tackle two common limitations of standard neural networks: i) lack of ability to adapt knowledge between different data contexts, and ii) performance drops due to distributional shifts. Our use of hypernetworks is inspired by the fact that they can produce neural network parameters that allow a target network (e.g. robot policy) to dynamically adapt to new contexts (e.g. tasks, robot capabilities, etc.). Hypernetworks have been shown to encode policies for multiple tasks within a single architecture [22, 23, 24], improve learning efficiency by improving gradient estimation in Q-learning [25], and achieve better performance with significantly fewer learnable parameters in meta-learning [26, 27]. As our experiments reveal, CASH translates these benefits of hypernetworks to heterogeneous multi-robot coordination (see Sec. 5.1 and 5.2).

Formally, at each timestep t, given the ith robot's observations o_i^t, the RNN Encoder $f_\psi(\cdot)$ produces a latent embedding $z_i^t = f_\psi(o_i^t)$. Next, the Hyper Adapter $h_\phi(\cdot)$ generates the parameters θ_i^t of the ith robot's Adaptive Decoder: $\theta_i^t = h_\phi(o_i^t, c_i^t, C_{/i}^t)$, where o_i^t is the local observation, c_i^t denotes the ego-robot's capabilities, and $C_{/i}^t = \{c_j^t | j \neq i\}$ is teammate capabilities. Note that both $f_\psi(\cdot)$ and $h_\phi(\cdot)$ lack the subscript i as they are parameter-shared across robots. Finally, the Adaptive Decoder $g_{\theta_i^t}(\cdot)$ produces final actions (or value estimates) for each robot $a_i^t = g_{\theta_i^t}(z_i^t)$ based on the latent embedding z_i^t. Though the Hyper Adapter's parameters ϕ are shared across robots, the Adaptive Decoder's parameters θ_i^t are unique to each robot. This can be seen as *soft parameter sharing* [28] applied to multi-robot teams. Moreover, since all θ_i^t are generated at each timestep, CASH can dynamically adapt to online changes to robot capabilities.

5 Experimental Evaluation

We evaluated and compared CASH against baseline architectures using two established experimental platforms: JaxMARL [3] (Sec. 5.1) and the Robotarium [4] (Sec. 5.2). All plots are smoothed by downsampling the original mean and standard deviation over all seeds, then taking a rolling average.

5.1 Experiments on JaxMARL

Here, we investigate how CASH i) compares to individual policy designs, ii) compares to other shared-parameter designs, and iii) generalizes to unseen team compositions and robot capabilities.

Architecture Variants: We compared CASH against three baseline architectures that reflect SOTA practices to handle heterogeneity. To ensure fairness, all baselines and CASH share a commonly used design: an RNN encoder followed by an MLP decoder [16, 20] (see Appendix C).

- INDV: Each robot employs a separate architecture without sharing parameters (e.g., [1]). This baseline questions if our shared-parameter approach trades performance for efficiency.
- RNN-IMP: The standard RNN-based architecture that is *implicitly* conditioned on capabilities by way of processing observations. Indeed, some capabilities (e.g. speed) can be inferred solely from observations over time with memory-enabled architectures (e.g., GRU). This baseline questions the need for explicit conditioning on capabilities.

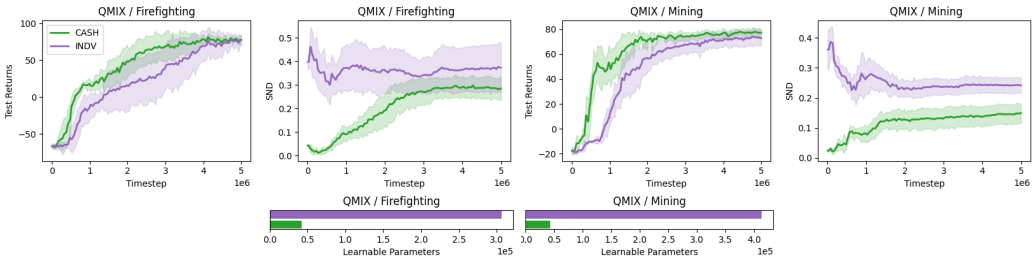

Figure 2: CASH is more sample efficient than individualized policies (see returns) and learns more-effective levels of diversity (see SND), while using drastically fewer learnable parameters (bottom).

- RNN-EXP: A modification of the standard RNN architecture that *explicitly* considers capabilities by appending them to observations. This baseline emulates prior designs that append agent-specific information [10, 11] and questions the need for hypernetworks.
- CASH: Our proposed architecture also explicitly considers capability information, but leverages a hypernetwork to dynamically adapt the action decoder to the robot and context (Sec. 4).

Learning Paradigms: We train CASH with three standard approaches for multi-agent learning, QMIX, MAPPO, and DAgger (representing off-policy MARL, on-policy MARL, and imitation learning, respectively), to demonstrate that CASH is effective regardless of training algorithm. During training, we assume centralized access to the joint observations and actions of all robots, though each robot policy is decentralized (i.e. operates only on local information o_i^t and easily updated capability information C^t). This is a common multi-robot learning assumption known as centralized training, decentralized execution (CTDE) [29, 30].

For MAPPO and QMIX, we adapt JaxMARL's [3] existing implementations to include capabilities. We contribute an implementation of DAgger for JaxMARL based on the existing MARL implementations. All results involving QMIX and MAPPO are over 10 random seeds; all DAgger results are over 3 random seeds. For specific training/architecture details, see Appendix C and our code implementation.

JaxMARL Tasks: We evaluate our method on two heterogeneous cooperative tasks. Both environments are implemented with JaxMARL's variant of the Multi-Agent Particle Environment [3, 30]. For additional experiments on larger team sizes, please see Appendix G.2.

`Firefighting`: A team of three robots spawns in a central depot and aim to put out two fires. Robots have varying speed and water capacity, and fires spawn in random positions with varying sizes. Task success is defined as both fires being successfully extinguished within the time limit.

`Mining`: A team of four robots must mine resources from two deposit zones and bring them to a dropoff zone until a quota for each resource is reached. Robots vary in their carrying capacity for each resource. The team succeeds if both quotas are met within a time limit. This task is similar to the HMT task in a prior study on capability-awareness in multi-agent teams [11].

Metrics: i) *Training returns* (↑) is the sum of all rewards collected during training episodes, ii) *Success rate* (↑), where success is defined for each task as above, and iii) *SND* (↕) quantifies the behavioral heterogeneity of independent policies [31]. We slightly modify SND to accommodate shared-parameter policies conditioned on capabilities (see Appendix A).

Training and Testing Teams: We standardize the teams sampled during training and evaluation across CASH and all baselines as follows (see Appendix D for more details):

Training teams (in-distribution): We sample a fixed number of training teams from a pool of robots whose capabilities uniformly cover a set range.

Unseen teams (out-of-distribution): We generate unseen team compositions with out-of-distribution robots by sampling some capabilities from ranges outside the training ranges.

Below, we organize our discussion into key observations and experiments.

Algorithm	Architecture	Firefighting In-Distribution Success Rate (↑)	Firefighting Out-of-Distribution Success Rate (↑)	Firefighting Out-of-Distribution SND	Mining In-Distribution Success Rate (↑)	Mining Out-of-Distribution Success Rate (↑)	Mining Out-of-Distribution SND
QMIX	RNN-IMP	0.54 ± 0.28	0.49 ± 0.22	0.00 ± 0.00	0.79 ± 0.08	0.63 ± 0.12	0.00 ± 0.00
	RNN-EXP	$\mathbf{0.97 \pm 0.05}$	0.56 ± 0.26	0.15 ± 0.07	0.98 ± 0.03	0.83 ± 0.15	0.07 ± 0.00
	CASH	0.96 ± 0.10	$\mathbf{0.67 \pm 0.09}$	0.28 ± 0.03	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.88 \pm 0.12}$	0.16 ± 0.01
MAPPO	RNN-IMP	0.21 ± 0.20	0.15 ± 0.14	0.00 ± 0.00	0.61 ± 0.24	0.43 ± 0.17	0.00 ± 0.00
	RNN-EXP	$\mathbf{0.71 \pm 0.30}$	0.43 ± 0.27	0.64 ± 0.03	$\mathbf{1.00 \pm 0.00}$	0.74 ± 0.11	0.18 ± 0.03
	CASH	$\mathbf{0.71 \pm 0.43}$	$\mathbf{0.68 \pm 0.17}$	0.58 ± 0.03	0.98 ± 0.04	$\mathbf{0.83 \pm 0.12}$	0.22 ± 0.06
DAgger	RNN-IMP	0.08 ± 0.14	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.07 ± 0.08	0.00 ± 0.00
	RNN-EXP	0.17 ± 0.14	0.08 ± 0.14	0.02 ± 0.00	0.35 ± 0.10	0.30 ± 0.10	0.03 ± 0.00
	CASH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.92 \pm 0.14}$	0.08 ± 0.01	$\mathbf{0.88 \pm 0.11}$	$\mathbf{0.83 \pm 0.10}$	0.06 ± 0.00

Table 1: CASH tends to achieve the highest success rates across all JaxMARL tasks and learning paradigms, particularly when evaluated on teams with out-of-distribution robot capabilities.

CASH improves sample efficiency and enables generalization without sacrificing performance. To investigate the impact of soft parameter sharing on performance and behavioral diversity, we compared CASH (soft parameter sharing) against the INDV (individualized parameters) baseline. Since INDV assumes that the robot team doesn't change, we trained each seed on a *single* unique multi-robot team throughout training.

We find that CASH is more sample-efficient and performs marginally better than INDV in both tasks, as seen by the return curves in Fig. 2, despite having a fraction of the learnable parameters. The SND plots in Fig. 2 show CASH has lower behavioral diversity than INDV. However, since this lower diversity doesn't result in worse task performance, we conclude that CASH is learning the *appropriate* level of behavioral diversity while INDV produces *superfluous* behavioral diversity that doesn't improve performance. Critically, unlike INDV, CASH can generalize to unseen robots.

In the experiments reported below, we compare CASH's ability to generalize against the two shared-parameter baselines (RNN-IMP and RNN-EXP) on two heterogeneous coordination tasks across three learning paradigms. Note that we cannot compare against INDV since independent parameter designs do not generalize to new robots.

CASH improves parameter and sample efficiency for shared-parameter methods. As seen in Fig 3, CASH is consistently the most sample-efficient method and achieves the highest returns while using 60-80% fewer parameters. CASH's improved efficiency can be attributed to its hypernetwork better modeling the influence of robot capabilities on the team's decisions. Note that the relative benefits of CASH are exaggerated for imitation learning, in which each architecture receives two to three orders of magnitude fewer samples than in the RL settings. This suggests that CASH can better handle data-scarce regimes compared to baselines. These trends hold for task-specific metrics as well (see Appendix G).

CASH improves generalization to unseen team compositions and capabilities. We next evaluated CASH's capacity for zero-shot generalization to unseen team compositions and capabilities, again comparing against the two shared-parameter baselines (RNN-IMP and RNN-EXP) across all conditions (as INDV cannot generalize to new robots). We report success rates and SND across

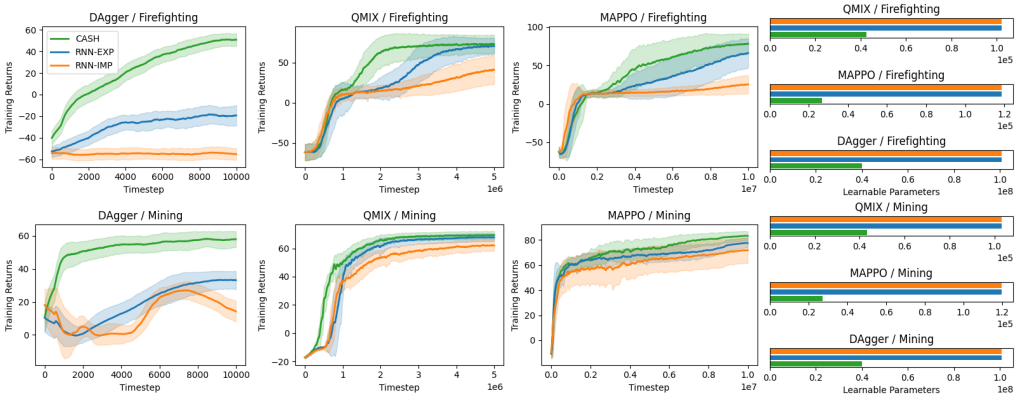

Figure 3: Across two tasks and three learning paradigms, CASH is consistently more sample efficient and yields better returns than the baselines despite using 60%-80% fewer learnable parameters.

conditions in Table 1. Out-of-distribution success rates are from evaluation on unseen test teams (see Sec. 5.1), while in-distribution success rates are from training teams.

While all architectures perform worse when generalizing to new teams (as expected), CASH exhibits the lowest drop in performance as a result of its superior capability grounding. Notably, when trained with QMIX or MAPPO, RNN-EXP and CASH result in similar success rates on training teams. However, within the same task and learning paradigm, CASH vastly outperforms RNN-EXP on unseen teams. These findings show that the baselines can learn effective strategies to cope with heterogeneity between agents in training teams, but struggle to generalize them to unseen teams. By contrast, CASH learns generalizable strategies that exploit the relationship between robot capability and desired behavior associated with each task.

We also note that CASH tends to produce greater behavioral diversity as measured by SND. This increased diversity partially likely explains CASH's superior generalization to unseen capabilities – effective generalization may require policies to generate different behaviors when robot capabilities change, even if observations remain the same. Such differentiation would not be possible for a policy with very limited behavioral diversity. We hypothesize that behavioral diversity might be a necessary but not sufficient condition for successful generalization since greater diversity doesn't always correlate with better performance (e.g., see comparison against INDV. in Sec. 5.1).

5.2 Experiments on the Robotarium

To validate the applicability of CASH to real multi-robot systems, we evaluate it on the Robotarium [4], a publicly available multi-robot physical testbed and simulator that provides realistic robot dynamics and barrier certificates. We again evaluate CASH against the two shared-parameter baselines (RNN-IMP and RNN-EXP). We train all baselines with QMIX using the MARBLER platform [32], which bridges the Robotarium's simulator with existing MARL implementations in EPyMARL [33] and provides several multi-robot tasks to evaluate policies (see Appendix C for training details).

Tasks: We evaluate on two established heterogeneous multi-robot tasks from MARBLER [32]. For additional experiments on these tasks with larger team sizes, see Appendix F.1.

`Material Transport (MT)`: A team of four robots must unload materials from two zones (green circle and rectangle) into a dropoff zone (purple). The team is rewarded for amount of material loaded and unloaded, penalized for collisions, and penalized each timestep the loading zones have remaining material. Robots vary in their speed and carrying capacity.

`Predator Capture Prey (PCP)`: Two sensing robots and two capture robots must collaborate to capture prey (circular markers). Sensing and capturing prey is rewarded while collisions and time-to-capture are penalized. Robots vary in their sensing and capture radii.

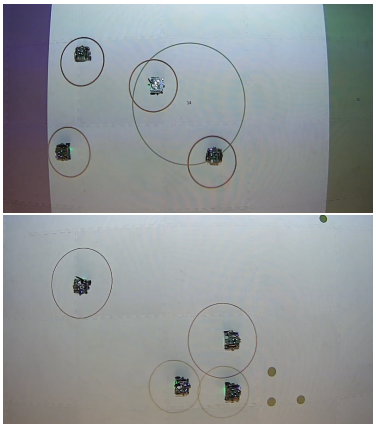

Figure 4: Snapshots of physical deployment on two Robotarium tasks: `MT` (top), `PCP` (bottom).

Online Adaptation: Additionally, to investigate how CASH handles *online* changes to robot capabilities, we create two special scenarios within the `MT` and `PCP` tasks: i) `Failure`, where halfway through the episode, a robot is selected randomly and its capability (speed in `MT`, sensing/capture radius in `PCP`) is decreased by 75% and ii) `Battery Drain`, where at each step, the capabilities of all robots (speed in `MT`, sensing/capture radius in `PCP`) are decayed by a discount factor. Exact capability ranges and deployment teams for these tasks are detailed in Appendix D.

Metrics: We report the following metrics for all tasks, averaged across evaluation episodes: i) *Reward* (↑), the sum of rewards collected per episode, ii) *Makespan* (↓), the episode length, and iii) *Collisions* (↓), the number of collisions per episode. Simulation metrics are averaged across 3 seeds,

			Robotarium (sim - 3 seeds)			*Robotarium (real - best seed)*		
Task	Architecture	# Parameters (↓)	Reward (↑)	Makespan (↓)	Collisions (↓)	Reward (↑)	Makespan (↓)	Collisions (↓)
MT	RNN-IMP	401K	2.32 ± 11.86	63.85 ± 12.41	0.03 ± 0.17	21.52 ± 1.57	54.20 ± 3.92	$\mathbf{0.00 \pm 0.00}$
	RNN-EXP	401K	12.94 ± 8.21	53.59 ± 13.10	0.02 ± 0.13	23.12 ± 2.96	50.20 ± 7.39	$\mathbf{0.00 \pm 0.00}$
	CASH	**162K**	$\mathbf{14.84 \pm 7.91}$	$\mathbf{49.79 \pm 12.87}$	$\mathbf{0.01 \pm 0.12}$	$\mathbf{23.84 \pm 3.72}$	$\mathbf{48.40 \pm 9.31}$	$\mathbf{0.00 \pm 0.00}$
PCP	RNN-IMP	402K	64.65 ± 31.923	79.72 ± 6.29	0.03 ± 0.18	69.40 ± 32.65	$\mathbf{81.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
	RNN-EXP	402K	61.05 ± 30.54	79.60 ± 6.75	0.03 ± 0.17	31.00 ± 33.12	$\mathbf{81.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
	CASH	**164K**	$\mathbf{89.21 \pm 33.54}$	$\mathbf{76.86 \pm 9.28}$	$\mathbf{0.02 \pm 0.13}$	$\mathbf{96.60 \pm 18.49}$	$\mathbf{81.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$

Table 2: CASH achieves the highest reward, lowest makespan, and fewest collisions on hardware.

		MT			PCP		
Dynamic Change	Architecture	Reward (↑)	Makespan (↓)	Collisions (↓)	Reward (↑)	Makespan (↓)	Collisions (↓)
Battery Drain	RNN-IMP	-0.39 ± 11.98	66.79 ± 9.43	0.03 ± 0.18	61.09 ± 31.51	79.93 ± 5.97	0.03 ± 0.18
	RNN-EXP	9.18 ± 8.55	60.01 ± 12.42	0.02 ± 0.13	57.02 ± 29.90	79.82 ± 6.53	$\mathbf{0.03 \pm 0.17}$
	CASH	$\mathbf{11.26 \pm 7.91}$	$\mathbf{56.87 \pm 13.29}$	$\mathbf{0.01 \pm 0.12}$	$\mathbf{82.56 \pm 34.25}$	$\mathbf{77.75 \pm 8.85}$	$\mathbf{0.03 \pm 0.17}$
Failure	RNN-IMP	0.11 ± 11.72	65.73 ± 11.16	0.03 ± 0.17	56.28 ± 30.53	79.95 ± 6.02	0.03 ± 0.18
	RNN-EXP	9.84 ± 9.22	57.77 ± 13.69	$\mathbf{0.02 \pm 0.12}$	51.10 ± 29.45	79.95 ± 6.56	0.03 ± 0.16
	CASH	$\mathbf{11.74 \pm 8.88}$	$\mathbf{54.77 \pm 14.34}$	$\mathbf{0.02 \pm 0.12}$	$\mathbf{69.89 \pm 34.21}$	$\mathbf{78.95 \pm 7.66}$	$\mathbf{0.02 \pm 0.15}$

Table 3: CASH outperforms baselines when adapting to online changes in capabilities.

500 episodes per seed. The highest performing model weights for each baseline are selected for physical robot deployment and metrics are averaged across 5 episodes.

CASH's benefits extend to real multi-robot teams. We find that CASH continues to outperform baselines in training despite having significantly fewer parameters, as seen in Tab. 2. In the MT scenario, we observed that CASH is better at reasoning about the speeds and capacities of robots, resulting in smoother and more direct trajectories between the material loading zones and dropoff zone. Meanwhile, in PCP, CASH more consistently learns effective role assignment strategies than baselines. See our supplemental video for full rollouts and more detailed analysis.

For each architecture and task, we selected the highest-performing model across training seeds and physically deployed it on the Robotarium (Fig. 4). We find that CASH effectively coordinates the physical robots and outperforms even the best-performing seeds of the baseline architectures (Tab. 2). The rewards are typically higher when physically deployed than in simulation since the Robotarium's use of barrier certificates prevents collisions that would have otherwise occurred.

CASH adapts to online changes to robot capabilities. We investigated how CASH handles online changes to robot capabilities by taking the trained policies for each task and zero-shot deploying them in the Failure and Battery Drain scenarios. As expected, we observe a drop in task performance across all baselines in Tab. 3. However, CASH consistently outperforms the baselines when dealing with the challenging unplanned changes to robot capabilities and observations during policy execution. This is likely due to CASH's better capability grounding that allows it to dynamically adjust the robots' behaviors even during a rollout, a critical feature to enable robustness in learned heterogeneous teaming.

Qualitatively, we notice that CASH better adapts to the changes in robot speed in MT with robots still being effectively allocated and continuing to pick-up and unload material. In contrast, the inefficient navigation strategies of RNN-IMP and RNN-EXP are heavily impacted by the degraded speed, further hindering their performance. Similarly, in the PCP scenarios, CASH's role-assignment strategy better generalizes to the diminished sensing and capturing capability, while the lack of role assignment in RNN-IMP and RNN-EXP prevents the robots from efficiently localizing prey.

6 Conclusion

We proposed a new architecture named *Capability-Aware Shared Hypernetworks (CASH)* that introduces soft parameter sharing in heterogeneous multi-robot learning, establishing a new middle ground between shared and individualized parameter approaches. CASH is deployable on decentralized teams and supports imitation, value-based, and policy-based RL.

As our experiments on both simulated and real multi-robot platforms reveal, CASH matches the performance of independent-parameter designs while enabling zero-shot generalization and improving sample efficiency. Furthermore, CASH outperforms existing shared-parameter designs in terms of diversity, sample- and parameter-efficiency, zero-shot generalization, and online adaptation.

7 Limitations

Though our work represents an exciting step towards flexible and generalizable heterogeneous multi-robot coordination using parsimonious policy architectures, it has some limitations.

One limitation is that our experiments assume lossless communication between neighboring robots. We simulate true partial observability by directly appending the three nearest teammates' relative positions to the observations of each robot. While reasonable in smaller-scale controlled settings, this assumption might limit the applicability of CASH to larger-scale multi-robot systems. Future work could relax this assumption by integrating lossy communication modules into the training process, e.g. by incorporating a GNN [11, 1].

A constraint of CASH is that it cannot handle entirely new capabilities dimensions during inference. This is a natural limitation of most neural network learning paradigms with the exception of large reasoning models that leverage massive amounts of data and tokenization, which could potentially handle new capabilities during inference time. Furthermore, it is fair to assume for most single-domain policies that the capability dimensions are the same for training and inference.

Another limitation is that the JaxMARL tasks which we evaluated CASH on are simpler than the most complex tasks in heterogeneous multi-agent coordination. For instance, our `Firefighting` task can be seen as a simplified version of a firefighting simulator like FireCommander [13]. However, our results show that despite the simplicity of these tasks, they have the crucial property that better grounding of heterogeneous capabilities tends to yield better performance, as evidenced by RNN-EXP outperforming RNN-IMP and CASH outperforming RNN-EXP. In our opinion, this is the most important characteristic relevant to heterogeneous robots, and abstracting away other details (e.g. more realistic communication, continuous action spaces, sensing and actuation noise, more dynamic environments) allowed us to tease out how policies can more effectively capture the relationship between robot capabilities and behaviors. Further, our experiments in the Robotarium show that the benefits of CASH observed in simulation hold for real multi-robot teams as well.

Acknowledgments

The authors would like to thank the reviewers for their detailed and constructive feedback to help strengthen the paper. We would also like to thank the Robotarium by Georgia Tech for it's accessibility to multi-robot hardware for real-robot experimentation and testing. This work was supported in part by the Army Research Lab under Grant W911NF17-2-0181 (DCIST CRA).

References

[1] M. Bettini, A. Shankar, and A. Prorok. Heterogeneous multi-robot reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2023.

[2] D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[3] A. Rutherford, B. Ellis, M. Gallici, J. Cook, A. Lupu, G. Ingvarsson Juto, T. Willi, R. Hammond, A. Khan, C. Schroeder de Witt, et al. Jaxmarl: Multi-agent rl environments and algorithms in jax. *Advances in Neural Information Processing Systems*, 2024.

[4] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt. The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems. *IEEE Control Systems Magazine*, 2020.

[5] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[6] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems*, 2017.

[7] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar. Graph policy gradients for large scale robot control. In *Proceedings of the Conference on Robot Learning*, 2020.

[8] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 2019.

[9] Q. Li, F. Gama, A. Ribeiro, and A. Prorok. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[10] J. K. Terry, N. Grammel, S. Son, B. Black, and A. Agrawal. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020.

[11] P. Howell, M. Rudolph, R. J. Torbati, K. Fu, and H. Ravichandar. Generalization of heterogeneous multi-robot policies via awareness and communication of capabilities. In *Proceedings of The 7th Conference on Robot Learning*, 2023.

[12] F. Christianos, G. Papoudakis, M. A. Rahman, and S. V. Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[13] E. Seraj, Z. Wang, R. Paleja, D. Martin, M. Sklar, A. Patel, and M. Gombolay. Learning efficient diverse communication for cooperative heterogeneous teaming. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022.

[14] C. Li, T. Wang, C. Wu, Q. Zhao, J. Yang, and C. Zhang. Celebrating diversity in shared multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

[15] C. Wakilpoor, P. J. Martin, C. Rebhuhn, and A. Vu. Heterogeneous multi-agent reinforcement learning for unknown environment mapping. *arXiv preprint arXiv:2010.02663*, 2020.

[16] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 2020.

[17] K. ab Abebe Tessera, A. Rahman, and S. V. Albrecht. Hypermarl: Adaptive hypernetworks for multi-agent rl. *arXiv preprint arXiv:2412.04233*, 2025.

[18] F. A. Oliehoek and C. Amato. *A Concise Introduction to Decentralized POMDPs*. Springer, 2016.

[19] H. Ravichandar, K. Shaw, and S. Chernova. Strata: unified framework for task assignments in large teams of heterogeneous agents. *Autonomous Agents and Multi-Agent Systems*, 2020.

[20] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. WU. The surprising effectiveness of ppo in cooperative multi-agent games. In *Advances in Neural Information Processing Systems*, 2022.

[21] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[22] Y. Huang, K. Xie, H. Bharadhwaj, and F. Shkurti. Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[23] S. Rezaei-Shoshtari, C. Morissette, F. R. Hogan, G. Dudek, and D. Meger. Hypernetworks for zero-shot transfer in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[24] J. von Oswald, C. Henning, B. F. Grewe, and J. Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

[25] E. Sarafian, S. Keynan, and S. Kraus. Recomposing the reinforcement learning building blocks with hypernetworks. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[26] T. Galanti and L. Wolf. On the modularity of hypernetworks. In *Advances in Neural Information Processing Systems*, 2020.

[27] J. Beck, M. T. Jackson, R. Vuorio, and S. Whiteson. Hypernetworks in meta-reinforcement learning. In *Conference on Robot Learning*, 2023.

[28] V. K. Chauhan, J. Zhou, P. Lu, S. Molaei, and D. A. Clifton. A brief review of hypernetworks in deep learning. *Artificial Intelligence Review*, 2024.

[29] L. Kraemer and B. Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 2016.

[30] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems*, 2017.

[31] M. Bettini, A. Shankar, and A. Prorok. System neural diversity: Measuring behavioral heterogeneity in multi-agent learning. *arXiv preprint arXiv:2305.02128*, 2024.

[32] R. J. Torbati, S. Lohiya, S. Singh, M. S. Nigam, and H. Ravichandar. Marbler: An open platform for standardized evaluation of multi-robot reinforcement learning algorithms. In *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2023.

[33] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021.

[34] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[35] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, 2023.

[36] M. Gallici, M. Fellows, B. Ellis, B. Pou, I. Masmitja, J. N. Foerster, and M. Martin. Simplifying deep temporal difference learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[37] J. Beck, R. Vuorio, Z. Xiong, and S. Whiteson. Recurrent hypernetworks are surprisingly strong in meta-rl. In *Advances in Neural Information Processing Systems*, 2023.

[38] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[39] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[40] T. Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

A Evaluating Behavioral Diversity with SND

To quantify the behavioral heterogeneity of the learned policies, we use the System Neural Diversity metric (SND) [31]. While SND was introduced to evaluate the heterogeneity of independent policies trained without parameter sharing, we modify SND to evaluate the heterogeneity of parameter-shared policies conditioned on heterogeneous capabilities. Specifically, given an observation, we append each agent's capability to the observation and find the average pairwise distances between the policy outputs when conditioned on differing capabilities. We define pairwise distance as

$$d(i,j) = \frac{1}{|\mathcal{O}|} \sum_{o_t \in \mathcal{O}} TVD(\pi_\theta(o_t||c^i), \pi_\theta(o_t||c^j))$$

where $\mathcal{O}$ is a set of observations obtained from several rollouts, c^i is the capability vector of agent i, and c^j is the capability vector of agent j. These pairwise distances are aggregated into a distance matrix, and the average of the upper triangular portion of this matrix is the final SND value. For value-based methods, individual networks output estimates of value per each action an agent can take. We interpret these Q-Values as a categorical distribution by taking a softmax over the values, and use Total Variational Distance when computing pairwise distances. Similarly, for discrete stochastic policies, we use Total Variational Distance between the categorical distributions over actions predicted by the policies.

B Layer normalization is crucial to CASH

To show the importance of adding layer normalization (LayerNorm) [34] when training hypernetworks for multi-agent coordination problems, we present an ablation over the inclusion of LayerNorm in CASH's Hyper Adapter in Figure 5. Across all tasks and learning algorithms, the inclusion of LayerNorm improves training stability and decreases variance between seeds, which aligns with recent works suggesting the benefit of LayerNorm for improving training stability of deeper networks in RL [35, 36]. In all cases except one (MAPPO / Mining), it results in a significant improvement on the final converged returns. DAgger is a notable outlier in terms of performance gap with and without LayerNorm; we see that when training with DAgger, the performance of CASH without LayerNorm in `Mining` plummets around timestep 4000 and never recovers, while in `Firefighting`, CASH without LayerNorm never demonstrates better than random performance.

Taken together, these results show that layer normalization improves training stability for hypernetworks, and that this improved stability is crucial for our architecture. We note that adding layer normalization to hypernetworks was suggested by the original Hypernetworks paper [2] to improve gradient flow, which our findings corroborate. Surprisingly, prior work on using hypernetworks in meta-RL for generalization to new tasks [27, 37] does not include LayerNorm in the hypernetwork, even though we found it to be critically important for CASH. We speculate that these works are able to achieve success despite omitting normalization from their hypernetworks because the input to their hypernetworks is a strong pretrained encoder. By contrast, our hypernetwork is conditioned directly on task observations and team capabilities, without the benefit of an encoder for preprocessing. We leave a more thorough investigation of normalization schemes for training deep hypernetworks to future work.

C Additional Training Details

C.1 Hyper-Adapter

Our experiments revealed that a deeper four-layer hypernetwork is better for our use case. However, consistent with prior work using hypernetworks in RL [27], we found it difficult to optimize a standard four-layer hypernetwork, both in reinforcement and imitation learning regimes. We believe this is due to the combination of instabilities inherent in both the learning paradigms as well as hypernetworks themselves. However, we found that simply adding layer normalization (LayerNorm) [34] before each ReLU activation in our hypernetwork stabilizes learning and greatly improves performance (see Appendix B for more details).

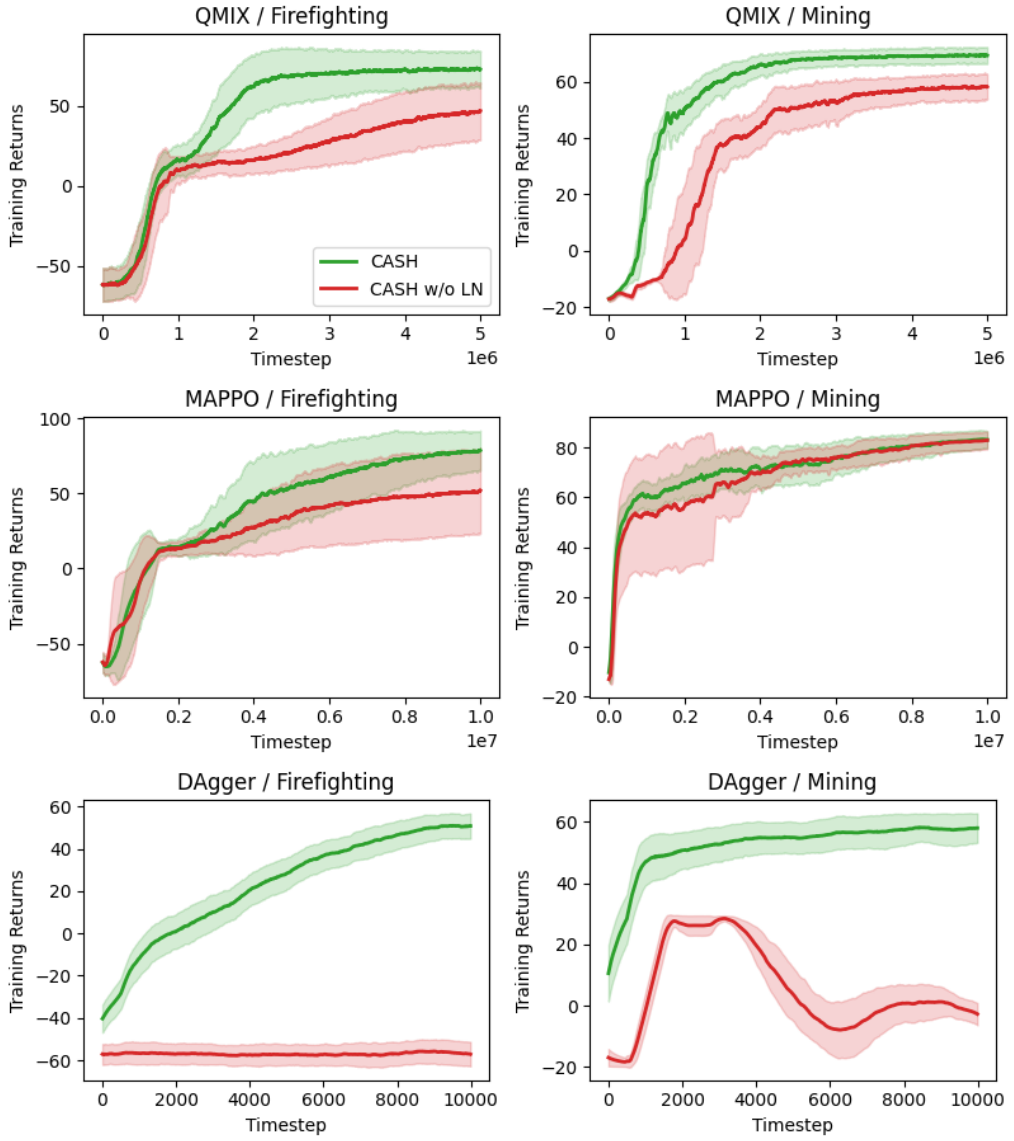

Figure 5: This figure shows the impact on training returns as a result of removing Layer Normalization from the Hyper Adapter of CASH. The results are presented for three learning paradigms across two simulation tasks. It is evident that LayerNorm is a crucial component in stabilizing the training of the hypernetwork within CASH.

Hyperparameter	Value
Total timesteps	10e6
Learning rate	2e-3
Anneal learning rate	True
Update epochs	4
Number of minibatches	4
Gamma	0.99
GAE lambda	0.95
Clip epsilon	0.2
Scale clip epsilon	False
Entropy coefficient	0.01
Value function coefficient	0.5
Max gradient norm	0.5
Optimizer	Adam [38]

Table 4: Hyperparameters for MAPPO

Hyperparameter	Value
Total timesteps	10e6
Learning rate	0.005
Learning rate linear decay	True
Buffer size	5000
Buffer batch size	32
Epsilon start	1.0
Epsilon finish	0.05
Epsilon anneal time	100000
Mixer embedding dim	32
Mixer hypernetwork hidden dim	64
Mixer initialization scale	0.00001
Max gradient norm	25
Target update interval	200
Optimizer	AdamW [39]
Epsilon Adam	0.001
Weight decay Adam	0.00001
TD lambda loss	True
TD lambda	0.6
Gamma	0.9

Table 5: Hyperparameters for QMix (JaxMARL)

For CASH, we implemented the Hyper Adapter as two separate hypernetworks, one which generates the weights of the target linear layer and the other which generates the biases. We found that initializing both hypernetwork with orthogonal weights, zero-bias was conducive to good task success, which is common in other works. However, we noticed a slight performance increase when setting the *scale* of those weights to be 0 for the bias-generating hypernet and 0.2 for weight-generating hypernet. We could not find a reference for this odd initialization scheme in the literature.

C.2 MAPPO

All networks are trained with the following hyperparameters in Table 4. We ablated over four widths (32, 64, 128, 256) of RNN hidden dimension with the `RNN-IMP` architecture, and found 128 to be the best-performing variant as measured by test return. Based on this, we then designed a `CASH` architecture of lower parameter count than the baselines by selecting an RNN width of 32 and a hypernetwork width of 16.

C.3 QMIX - JAXMARL

All networks are trained with the following hyperparameters in Table 5. We ablated over four widths (32, 64, 128, 256) of RNN hidden dimension with the `RNN-IMP` architecture, and found 128 to be the best-performing variant as measured by test return. Based on this, we then designed a `CASH`

Hyperparameter	Value
Total timesteps	4e6
Learning rate	0.0005
Learning rate linear decay	True
Buffer size	5000
Buffer batch size	32
Epsilon start	1.0
Epsilon finish	0.05
Epsilon anneal time	50000
Mixer embedding dim	32
Mixer hypernetwork hidden dim	64
Max gradient norm	10
Target update interval	200
Optimizer	RMSprop [40]
Epsilon RMSProp	0.00001
Gamma	0.9

Table 6: Hyperparameters for QMix (EPyMARL)

Hyperparameter	Value
Expert buffer size	10000
Initial expert trajectories	1000
Iterations	10
Trajectories per iteration	1000
Learning rate	1e-4
Learning rate linear decay	False
Beta	1.0
Beta linear decay	True
Updates per iteration	100
Update batch size	64
Max gradient norm	1
Optimizer	AdamW [39]
Epsilon Adam	0.001
Weight decay Adam	0

Table 7: Hyperparameters for DAgger

architecture of lower parameter count than the baselines by selecting an RNN width of 64 and a hypernetwork width of 32.

C.4 QMIX - EPyMARL

All networks are trained with the following hyperparameters in Table 6. We ablated over two widths (128, 256) of RNN hidden dimension with the `RNN-IMP` architecture, and found 256 to be the best-performing variant as measured by test return. Based on this, we then designed a `CASH` architecture of lower parameter count than the baselines by selecting a RNN width of 128 and a hypernetwork width of 64.

C.5 DAgger

All networks are trained with the following hyperparameters in Table 7. We ablated over four widths (512, 1024, 2048, 4096) of RNN hidden dimension with the `RNN-IMP` architecture, and found 4096 to be the best-performing variant as measured by test return. We were unable to include larger width RNNs due to memory constraints. Based on this, we then designed a `CASH` architecture of lower parameter count than the baselines by selecting an RNN width of 2048 and a hypernetwork width of 1024.

D Capability Sampling Details

D.1 Firefighting

In Table 8 and Table 9 we detail our randomly selected train and test teams for `Firefighting`. More details on how we chose these teams and how they were used can be found in the main body of Chapter 3. For context, in `Firefighting` the fires range from 0.2-0.3 strength and the training distribution of agents ranged between 0.1-0.3 firefighting capacity and 1-3 acceleration. For testing teams, we sample agents from out of bounds in both radius and acceleration, but match other agents to ensure task feasibility.

Agent	(radius, acceleration) of agent
0	(0.3, 1)
1	(0.2, 2)
2	(0.1, 3)
3	(0.1, 3)
4	(0.2, 2)

Table 8: `Firefighting` training robots.

Team	(radius, accel) for each agent in team
0	(0.09, 3.43), (0.21, 2.94), (0.42, 0.75)
1	(0.09, 3.41), (0.21, 3.00), (0.48, 0.63)
2	(0.05, 3.46), (0.25, 2.76), (0.44, 0.60)
3	(0.08, 3.23), (0.23, 2.80), (0.50, 0.61)
4	(0.09, 3.14), (0.21, 1.16), (0.47, 0.86)
5	(0.06, 3.45), (0.24, 2.08), (0.46, 0.76)
6	(0.08, 3.06), (0.22, 1.08), (0.48, 0.56)
7	(0.07, 3.04), (0.23, 2.37), (0.45, 0.56)
8	(0.09, 3.36), (0.21, 2.20), (0.49, 0.64)
9	(0.09, 3.26), (0.21, 2.59), (0.47, 0.64)

Table 9: `Firefighting` testing teams.

D.2 Mining

In `Mining` we have a training range between 0-0.5 for each capacity, with each agent having a total capacity across the two materials that sums to 0.5. During testing the random sampling is altered such that the total capacity of each agent sums to 1.0, to present an out of distribution challenge. Exact hyperparameters can be found in 10 and 11.

Team	(1st material capacity, 2nd material capacity) for each agent of team
0	(0.1, 0.4), (0.2, 0.3), (0.3, 0.2), (0.5, 0.0)
1	(0.2, 0.3), (0.3, 0.2), (0.4, 0.1), (0.5, 0.0)
2	(0.0, 0.5), (0.1, 0.4), (0.3, 0.2), (0.5, 0.0)
3	(0.0, 0.5), (0.1, 0.4), (0.2, 0.3), (0.4, 0.1)
4	(0.0, 0.5), (0.1, 0.4), (0.2, 0.3), (0.5, 0.0)
5	(0.1, 0.4), (0.3, 0.2), (0.4, 0.1), (0.5, 0.0)
6	(0.0, 0.5), (0.2, 0.3), (0.4, 0.1), (0.5, 0.0)
7	(0.0, 0.5), (0.2, 0.3), (0.3, 0.2), (0.5, 0.0)
8	(0.0, 0.5), (0.1, 0.4), (0.4, 0.1), (0.5, 0.0)
9	(0.1, 0.4), (0.2, 0.3), (0.4, 0.1), (0.5, 0.0)

Table 10: Mining training teams.

D.3 Material Transport

In `Material Transport` we construct teams of 4 agents consisting of,

- 2 *fast* agents with *low* capacity.

Team	(1st material capacity, 2nd material capacity) for each agent of team
0	(0.72, 0.28), (0.98, 0.02), (0.17, 0.83), (0.12, 0.13)
1	(0.63, 0.37), (0.88, 0.12), (0.56, 0.44), (0.17, 0.08)
2	(0.04, 0.96), (0.24, 0.76), (0.29, 0.71), (0.25, 0.)
3	(0.01, 0.99), (0.56, 0.44), (0.18, 0.82), (0.05, 0.2)
4	(0.26, 0.74), (0.65, 0.35), (0.95, 0.05), (0.16, 0.09)
5	(0.37, 0.63), (0.52, 0.48), (0.77, 0.23), (0.04, 0.21)
6	(0.41, 0.59), (0.96, 0.04), (0.12, 0.88), (0.05, 0.2)
7	(0.86, 0.14), (0.97, 0.03), (0.38, 0.62), (0.12, 0.13)
8	(0.68, 0.32), (0.87, 0.13), (0.73, 0.27), (0.15, 0.1)
9	(0.4, 0.6), (0.41, 0.59), (0.54, 0.46), (0.02, 0.23)

Table 11: Mining testing teams.

- 2 *slow* with *high* capacity.

The values for each are uniformly sampled from the sets in Table 12. We deploy the trained policies on the five teams in Table 13, where the values for each capability were randomly sampled from the training capability sets.

fast	0.40, 0.45, 0.50, 0.55, 0.60
slow	0.10, 0.15, 0.20, 0.25, 0.30
high	14, 16, 18, 20, 22
low	4, 6, 8, 10, 12

Table 12: Material transport capability sets.

Team	(speed, capacity) for each robot
0	(0.25, 20), (0.25, 20), (0.4, 6), (0.4, 6)
1	(0.30, 22), (0.30, 22), (0.55, 6), (0.55, 6)
2	(0.20, 20), (0.20, 20), (0.55, 6), (0.55, 6)
3	(0.30, 16), (0.30, 16), (0.50, 12), (0.50, 12)
4	(0.10, 14), (0.10, 14), (0.60, 12), (0.60, 12)

Table 13: Material transport deployment teams.

D.4 Predator Capture Prey

In `Predator Capture Prey` we construct teams of 4 agents consisting of 2 `capture` agents and 2 `sensing` agents. The capabilities for each are uniformly sampled from the sets in Table 14. We deploy the trained policies on the five teams in Table 15, where the values for each capability were randomly sampled from the training capability sets.

E Environment Implementations

We extended JaxMARL to include our two custom environments as well as a training script which implements DAgger. We use the implemented MARBLER Material Transport and Predator Capture Prey tasks, and extend them to sample from capability sets.

Please refer to our code for additional information on reward schemes, observation spaces, etc.

F Additional MARBLER Results

We report the training curves for `Material Transport` and `Predator Capture Prey` in Fig. 6. The plots are smoothed by downsampling the original mean and standard deviation over all seeds and then taking a rolling average.

capture radii	0.10, 0.125, 0.15, 0.175, .20
sensing radii	0.20, 0.25, 0.30, 0.35, 0.40

Table 14: Predator capture prey capability sets.

Team	(capture radius, sensing radius) for each robot
0	(0.15, 0.00), (0.15, 0.00), (0.00, 0.20), (0.00, 0.20)
1	(0.15, 0.00), (0.15, 0.00), (0.00, 0.35), (0.00, 0.35)
2	(0.125, 0.00), (0.125, 0.00), (0.00, 0.30), (0.00, 0.30)
3	(0.20, 0.00), (0.20, 0.00), (0.00, 0.40), (0.00, 0.40)
4	(0.10, 0.00), (0.10, 0.00), (0.00, 0.35), (0.00, 0.35)

Table 15: Predator capture prey deployment teams.

F.1 Team Size Generalization

We leverage the decentralized execution of CASH's policy design to zero-shot deploy the policies trained on 4-robot teams on teams of 12 robots. We scale both environments by doubling the amount of material or the number of prey, and robots are restricted to observing their three nearest neighbors.

Scenario	Architecture	MT Task Completion Rate (↑)	PCP Task Completion Rate (↑)
Zero-Shot (4 → 12)	RNN-IMP	0.45 ± 0.39	0.67 ± 0.21
	RNN-EXP	0.32 ± 0.39	0.58 ± 0.25
	CASH	$\mathbf{0.67 \pm 0.40}$	$\mathbf{0.68 \pm 0.21}$

Table 16: Zero-shot decentralized deployment to 12 robots over 50 episodes across 3 random seeds. Task completion is the percentage of prey captured or material dropped off.

CASH can be readily decentralized and deployed on larger robot teams. As seen in Table 16, CASH policies can be zero-shot deployed on larger teams. CASH performs either similarly to (in PCP) or better than (in MT) the baselines in when deployed on teams of 12 robots. The modest improvement over RNN-IMP in PCP be explained by the fact that increasing the density of predators and prey decreases the need for tight coordination among the predators.

G Additional JaxMARL Results

G.1 Comparison against ID-based methods

To further contextualize the INDV results, we compare against shared-parameter ID-based methods where a unique one-hot ID is appended to the observation space of each agent. ID-based methods represent a shared parameter baseline that cannot generalize to unseen robots. The experiment setup is identical to that in Section 5.1, but includes results from an agent ID baseline labeled RNN-ID. We plot the returns and SND for the Firefighting and Mining tasks in Figure 9. Expectedly, the shared encoder does improve sample efficiency when compared to individualized policies, and RNN-ID learns a lower level of behavioral diversity.

G.2 Training on Larger Teams

We investigate training CASH, RNN-EXP, and RNN-IMP on 12 agent teams in Mining and Firefighting. We restrict robots in both scenarios to only observe their three nearest neighbors to model decentralized execution.

G.2.1 Scaled Environments

For Firefighting, we increased the strength of fires to range from 0.3-0.4, and the distribution of agents ranged between 0.05-0.2 firefighting capacity and 1-3 acceleration. We sample testing teams as described in Appendix D. Exact training and testing teams are listed in Tables 19 and 20.

For Mining, the material quotas scale with the number of agents; see our provided code for exact implementations. We retain the same sampling strategy for constructing training (in-distribution) and testing (out-of-distribution) teams as described in Appendix D. Exact teams are listed in Tables 21 and 22.

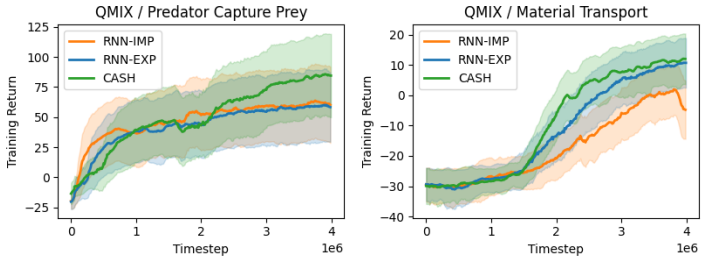

Figure 6: MARBLER training curves for `Predator Capture Prey` and `Material Transport`.

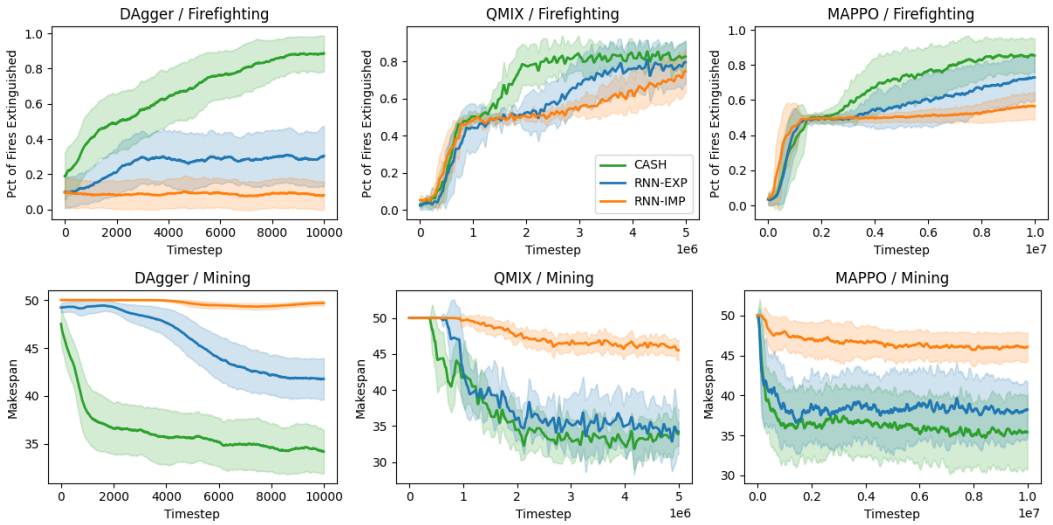

Figure 7: Task performance metrics across two JaxMARL tasks and three learning paradigms as evaluated on out-of-distribution robot capabilities and unseen team compositions. Percentage of Fires Extinguished (↑, top row) is for Firefighting. Makespan (↓, bottom row) is for Mining. These metrics provide additional context beyond the success rates in Table 1.

G.2.2 Training Details

We train all baselines with QMIX and MAPPO in both environments. We choose the hidden width for RNN-EXP and RNN-IMP by ablating over widths and learning rates for RNN-IMP and selecting the best performing hidden width and learning rate combination. For QMIX, we use hidden widths (64, 128, 256) and learning rates (0.005,0.003,0.001). For MAPPO, we use hidden widths (64, 128, 256) and learning rates (0.002, 0.001, 0.0005, 0.0003). We then construct CASH using a smaller hidden width and perform the same learning rate ablation. We report the exact widths and learning rates of each baseline for each algorithm and environment in Table 17. For CASH, we report the (hidden width, hypernetwork hidden width) respectively.

G.2.3 Results

We find that CASH consistently outperforms baselines in-distribution when training on large teams. Consistent with section 5.1, we find that CASH tends to exhibit more diverse behaviors as well. When generalizing to out-of-distribution teams, we find that CASH outperforms baselines across all conditions except on MAPPO/`Mining`, where surprisingly RNN-IMP. Future work can investigate under what conditions CASH is less effective in learning for larger team sizes. Likely, curriculum approaches commonly employed in multi-agent scenarios could be useful in learning more robust strategies with large teams.

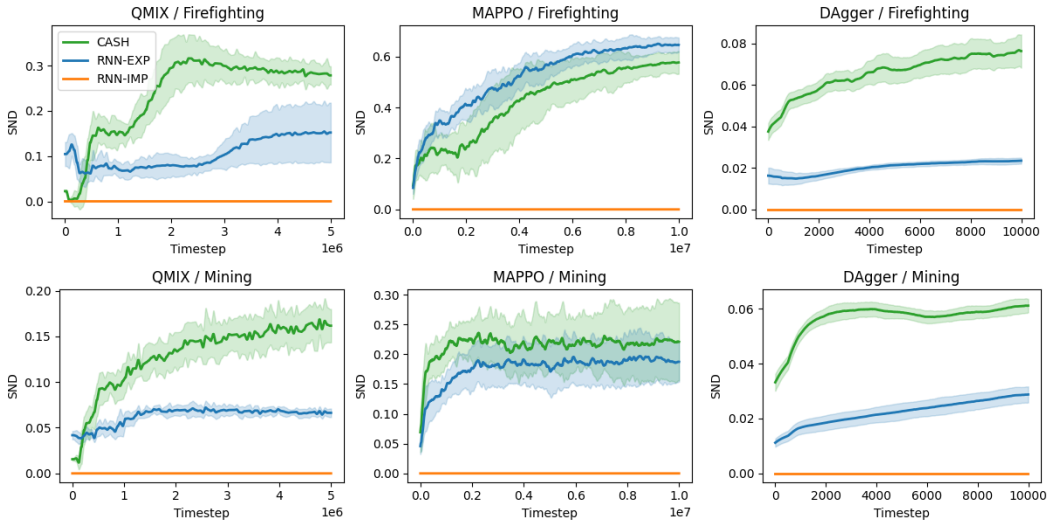

Figure 8: Behavioral diversity (↕) across two JaxMARL tasks and three learning paradigms. In all cases but one (MAPPO/`Firefighting`), CASH learns more diverse behaviors than the baseline approaches. Qualitative analysis of MAPPO/`Firefighting` suggests that RNN-EXP is able to learn diverse behaviors, but the corresponding success rate in Table 1 suggests these behaviors are unproductive for task success. Conversely, CASH is able to learn an appropriate level of heterogeneity for strong task success.

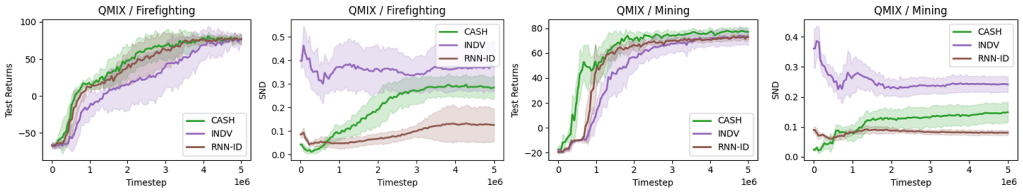

Figure 9: ID-based methods are more sample efficient than individualized policies (see returns), but learn a lower level of behavioral diversity (see SND). CASH matches or outperforms ID-based methods in both sample efficiency and behavioral diversity.

Environment	Algorithm	Architecture	Width	Learning Rate
`Firefighting` (12-agent)	QMIX	CASH	(64, 32)	0.003
		RNN-EXP	128	0.003
		RNN-IMP	128	0.003
	MAPPO	CASH	(64, 32)	0.0003
		RNN-EXP	256	0.001
		RNN-IMP	256	0.001
`Mining` (12-agent)	QMIX	CASH	(128, 64)	0.003
		RNN-EXP	256	0.003
		RNN-IMP	256	0.003
	MAPPO	CASH	(64, 32)	0.0005
		RNN-EXP	256	0.0005
		RNN-IMP	256	0.0005

Table 17: Final hidden widths and learning rates used for 12 agent `Firefighting` and `Mining`.

Algorithm / Environment	Architecture	# Parameters (↓)	In-Distribution Success Rate (↑)	Out-of-Distribution Success Rate (↑)	Out-of-Distribution SND
QMIX / `Firefighting` (12-agent)	RNN-IMP	103K	0.17 ± 0.30	0.08 ± 0.08	0.00 ± 0.00
	RNN-EXP	103K	0.50 ± 0.58	0.09 ± 0.11	0.08 ± 0.04
	CASH	**43K**	$\mathbf{0.97 \pm 0.02}$	$\mathbf{0.33 \pm 0.19}$	0.17 ± 0.03
MAPPO / `Firefighting` (12-agent)	RNN-IMP	468K	0.15 ± 0.10	0.31 ± 0.06	0.00 ± 0.00
	RNN-EXP	468K	0.75 ± 0.23	0.29 ± 0.04	0.31 ± 0.02
	CASH	**170K**	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.54 \pm 0.10}$	0.36 ± 0.02
QMIX / `Mining` (12-agent)	RNN-IMP	403K	0.91 ± 0.05	0.89 ± 0.11	0.00 ± 0.00
	RNN-EXP	403K	0.93 ± 0.07	0.87 ± 0.22	0.08 ± 0.01
	CASH	**165K**	$\mathbf{0.94 \pm 0.05}$	$\mathbf{0.94 \pm 0.10}$	0.17 ± 0.01
MAPPO / `Mining` (12-agent)	RNN-IMP	469K	0.81 ± 0.06	$\mathbf{0.90 \pm 0.10}$	0.00 ± 0.00
	RNN-EXP	469K	0.96 ± 0.07	0.88 ± 0.11	0.09 ± 0.03
	CASH	**170K**	$\mathbf{0.98 \pm 0.04}$	0.71 ± 0.34	0.12 ± 0.03

Table 18: CASH tends to achieve the highest success rates across learning paradigms on the 12-agent JaxMARL tasks. CASH also seems to retain the generalization benefits observed in Table 1, though we acknowledge MAPPO / `Mining` is a notable exception.

Agent	(radius, acceleration) of agent
0	(0.07, 2.31)
1	(0.09, 1.74)
2	(0.07, 2.63)
3	(0.06, 2.84)
4	(0.06, 1.33)
5	(0.10, 2.66)
6	(0.09, 1.89)
7	(0.09, 2.98)
8	(0.15, 2.48)
9	(0.19, 2.13)
10	(0.19, 2.56)
11	(0.16, 1.99)
12	(0.07, 2.31)
13	(0.09, 1.74)
14	(0.07, 2.63)
15	(0.06, 2.84)
16	(0.06, 1.33)
17	(0.10, 2.66)
18	(0.09, 1.89)
19	(0.09, 2.98)
20	(0.15, 2.48)
21	(0.19, 2.13)
22	(0.19, 2.56)
23	(0.16, 1.99)

Table 19: 12 agent `Firefighting` training teams.

Team	(radius, accel) for each agent in team
0	(0.04, 3.49), (0.04, 3.34), (0.04, 3.33), (0.04, 3.01), (0.10, 2.96), (0.11, 1.28), (0.10, 2.97), (0.10, 2.35), (0.23, 0.62), (0.23, 0.81), (0.22, 0.52), (0.25, 0.76)
1	(0.05, 3.00), (0.04, 3.23), (0.03, 3.43), (0.05, 3.35), (0.10, 1.51), (0.11, 1.89), (0.11, 1.56), (0.10, 1.97), (0.22, 0.92), (0.24, 0.97), (0.20, 0.82), (0.22, 0.64)
2	(0.03, 3.04), (0.03, 3.15), (0.04, 3.35), (0.04, 3.03), (0.11, 2.50), (0.11, 2.71), (0.10, 1.78), (0.11, 2.63), (0.20, 0.52), (0.22, 0.74), (0.21, 0.89), (0.22, 0.57)
3	(0.05, 3.25), (0.05, 3.47), (0.04, 3.23), (0.04, 3.02), (0.10, 2.23), (0.10, 2.06), (0.10, 2.19), (0.10, 2.61), (0.21, 0.81), (0.22, 0.76), (0.24, 0.51), (0.24, 0.73)
4	(0.03, 3.06), (0.04, 3.47), (0.03, 3.44), (0.04, 3.45), (0.11, 1.82), (0.11, 2.27), (0.11, 2.76), (0.10, 2.38), (0.21, 0.90), (0.21, 0.64), (0.21, 0.90), (0.22, 0.93)
5	(0.04, 3.11), (0.04, 3.11), (0.04, 3.20), (0.03, 3.33), (0.11, 1.94), (0.10, 1.06), (0.11, 2.10), (0.11, 1.17), (0.21, 0.58), (0.24, 0.93), (0.20, 0.59), (0.21, 0.52)
6	(0.03, 3.35), (0.03, 3.38), (0.04, 3.14), (0.04, 3.22), (0.11, 1.05), (0.11, 1.35), (0.10, 2.15), (0.11, 1.59), (0.23, 0.80), (0.21, 0.94), (0.20, 0.71), (0.21, 0.56)
7	(0.03, 3.14), (0.03, 3.43), (0.03, 3.34), (0.03, 3.31), (0.11, 1.07), (0.11, 1.21), (0.11, 2.74), (0.11, 1.14), (0.21, 0.96), (0.24, 0.98), (0.23, 0.69), (0.23, 0.74)
8	(0.04, 3.02), (0.04, 3.05), (0.05, 3.16), (0.03, 3.14), (0.10, 2.02), (0.10, 1.78), (0.10, 1.24), (0.11, 1.48), (0.21, 0.63), (0.20, 0.85), (0.21, 0.64), (0.22, 0.54)
9	(0.04, 3.32), (0.04, 3.16), (0.04, 3.07), (0.03, 3.11), (0.11, 1.66), (0.10, 1.98), (0.10, 2.79), (0.11, 2.61), (0.24, 0.84), (0.21, 0.78), (0.22, 0.76), (0.21, 0.55)

Table 20: 12 agent `Firefighting` testing teams.

Team	(1st material capacity, 2nd material capacity) for each agent of team
0	(0.00, 0.50), (0.06, 0.44), (0.09, 0.41), (0.18, 0.32), (0.21, 0.29), (0.24, 0.26), (0.26, 0.24), (0.29, 0.21), (0.32, 0.18), (0.35, 0.15), (0.38, 0.12), (0.41, 0.09)
1	(0.00, 0.50), (0.03, 0.47), (0.06, 0.44), (0.09, 0.41), (0.12, 0.38), (0.21, 0.29), (0.29, 0.21), (0.32, 0.18), (0.35, 0.15), (0.44, 0.06), (0.47, 0.03), (0.50, 0.00)
2	(0.00, 0.50), (0.03, 0.47), (0.12, 0.38), (0.18, 0.32), (0.26, 0.24), (0.29, 0.21), (0.32, 0.18), (0.35, 0.15), (0.38, 0.12), (0.41, 0.09), (0.44, 0.06), (0.50, 0.00)
3	(0.00, 0.50), (0.09, 0.41), (0.15, 0.35), (0.18, 0.32), (0.21, 0.29), (0.24, 0.26), (0.26, 0.24), (0.29, 0.21), (0.41, 0.09), (0.44, 0.06), (0.47, 0.03), (0.50, 0.00)
4	(0.00, 0.50), (0.09, 0.41), (0.12, 0.38), (0.15, 0.35), (0.18, 0.32), (0.24, 0.26), (0.32, 0.18), (0.38, 0.12), (0.41, 0.09), (0.44, 0.06), (0.47, 0.03), (0.50, 0.00)
5	(0.06, 0.44), (0.09, 0.41), (0.12, 0.38), (0.15, 0.35), (0.18, 0.32), (0.24, 0.26), (0.26, 0.24), (0.32, 0.18), (0.35, 0.15), (0.38, 0.12), (0.44, 0.06), (0.47, 0.03)
6	(0.00, 0.50), (0.03, 0.47), (0.06, 0.44), (0.15, 0.35), (0.18, 0.32), (0.21, 0.29), (0.26, 0.24), (0.29, 0.21), (0.35, 0.15), (0.41, 0.09), (0.47, 0.03), (0.50, 0.00)
7	(0.06, 0.44), (0.09, 0.41), (0.12, 0.38), (0.15, 0.35), (0.21, 0.29), (0.26, 0.24), (0.29, 0.21), (0.35, 0.15), (0.38, 0.12), (0.41, 0.09), (0.47, 0.03), (0.50, 0.00)
8	(0.00, 0.50), (0.03, 0.47), (0.09, 0.41), (0.12, 0.38), (0.18, 0.32), (0.24, 0.26), (0.26, 0.24), (0.35, 0.15), (0.41, 0.09), (0.44, 0.06), (0.47, 0.03), (0.50, 0.00)
9	(0.00, 0.50), (0.03, 0.47), (0.06, 0.44), (0.09, 0.41), (0.18, 0.32), (0.21, 0.29), (0.26, 0.24), (0.29, 0.21), (0.32, 0.18), (0.35, 0.15), (0.41, 0.09), (0.47, 0.03)

Table 21: 12 agent `Mining` training teams.

Team	(1st material capacity, 2nd material capacity) for each agent of team
0	(0.31, 0.69), (0.75, 0.25), (0.47, 0.53), (0.70, 0.30), (0.61, 0.39), (0.62, 0.38), (0.44, 0.56), (0.43, 0.57), (0.99, 0.01), (0.11, 0.14), (0.10, 0.15), (0.10, 0.15)
1	(0.34, 0.66), (0.40, 0.60), (0.93, 0.07), (0.91, 0.09), (0.77, 0.23), (0.31, 0.69), (0.11, 0.89), (0.73, 0.27), (0.79, 0.21), (0.04, 0.21), (0.06, 0.19), (0.12, 0.13)
2	(0.74, 0.26), (0.51, 0.49), (0.48, 0.52), (0.92, 0.08), (0.06, 0.94), (0.84, 0.16), (0.37, 0.63), (0.87, 0.13), (0.47, 0.53), (0.16, 0.09), (0.02, 0.23), (0.03, 0.22)
3	(0.32, 0.68), (0.85, 0.15), (0.27, 0.73), (0.43, 0.57), (0.23, 0.77), (0.49, 0.51), (0.62, 0.38), (0.57, 0.43), (0.88, 0.12), (0.23, 0.02), (0.23, 0.02), (0.05, 0.20)
4	(0.94, 0.06), (0.98, 0.02), (0.39, 0.61), (0.30, 0.70), (0.96, 0.04), (0.82, 0.18), (0.35, 0.65), (0.90, 0.10), (0.56, 0.44), (0.13, 0.12), (0.16, 0.09), (0.01, 0.24)
5	(0.48, 0.52), (0.70, 0.30), (0.91, 0.09), (0.57, 0.43), (0.87, 0.13), (0.44, 0.56), (0.09, 0.91), (0.84, 0.16), (0.58, 0.42), (0.16, 0.09), (0.19, 0.06), (0.04, 0.21)
6	(0.08, 0.92), (0.66, 0.34), (0.73, 0.27), (0.67, 0.33), (0.33, 0.67), (0.69, 0.31), (0.09, 0.91), (0.18, 0.82), (0.79, 0.21), (0.09, 0.16), (0.12, 0.13), (0.05, 0.20)
7	(0.45, 0.55), (0.20, 0.80), (0.93, 0.07), (0.32, 0.68), (0.63, 0.37), (0.59, 0.41), (0.82, 0.18), (0.56, 0.44), (0.73, 0.27), (0.01, 0.24), (0.02, 0.23), (0.20, 0.05)
8	(0.16, 0.84), (0.55, 0.45), (0.12, 0.88), (0.54, 0.46), (0.70, 0.30), (0.40, 0.60), (0.57, 0.43), (1.00, 0.00), (0.52, 0.48), (0.05, 0.20), (0.06, 0.19), (0.01, 0.24)
9	(0.87, 0.13), (0.14, 0.86), (0.68, 0.32), (0.19, 0.81), (0.52, 0.48), (0.85, 0.15), (0.73, 0.27), (0.88, 0.12), (0.60, 0.40), (0.05, 0.20), (0.05, 0.20), (0.13, 0.12)

Table 22: 12 agent `Mining` testing teams.