SCALABLE MULTI-AGENT AUTONOMOUS LEARNING IN COMPLEX UNPREDICTABLE ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

This research introduces a novel multi-agent self-learning solution for large and complex tasks in dynamic and unpredictable environments where large groups of homogeneous agents coordinate to achieve collective goals. Using a novel iterative two-phase multi-agent reinforcement learning approach, agents continuously learn and evolve in performing the task. In phase one, agents collaboratively determine an effective global task distribution based on the current state of the task and assign the most suitable agent to each activity. In phase two, the selected agent refines activity execution using a shared policy from a policy bank, built from collective past experiences. Merging agent trajectories across similar agents using a novel shared experience learning mechanism enables continuous adaptation, while iterating through these two phases significantly reduces coordination overhead. This novel approach was tested with an exemplary test system comprising drones, with results including real-world scenarios in domains like forest firefighting. This approach performed well by evolving autonomously in new environments with a large number of agents. In adapting quickly to new and changing environments, this versatile approach provides a highly scalable foundation for many other applications tackling dynamic and hard-to-optimize domains that are not possible today.

1 INTRODUCTION

Many real-world problems are quite big and complex requiring many agents with different capabilities to effectively tackle them. Autonomous multi-agent applications like delivery systems, warehouse robots, and drone shows work in mostly deterministic and constrained environments. However, there are many complicated dynamic environments such as forest fire-fighting, disaster relief, urban fire, and medical rescue operations where each episode is unique and ridden with unpredictable challenges. Today's MARL algorithms fail to address the enormity and complexity of these tasks (Rashid et al., 2018) (Yu et al., 2022).

We propose a novel two-phase iterative approach of refocus and refine to enable groups of homogeneous agents with different capabilities to autonomously learn to perform huge, unpredictable, fast-changing tasks. Phase One: refocus determines the best way to target the task, phase two: refine uses the collective intelligence of the group for each agent to best perform its task, and iteratively repeating this leads to continuous evolution. This opens the possibility of complementing pure reinforcement learning with adjunct strategies, including domain intelligence or human-in-the-loop (HIL), to expedite learning. It realigns learning to focus on the most relevant portion of the state-space and gives agents autonomy to improvise while significantly reducing the coordination effort across numerous agents. Using shared experience across homogeneous agents with a shared population policy-bank, this result-oriented learning is highly scalable. We demonstrate this approach via an exemplary test system comprising drones fighting forest fires.

2 RELATED WORK

Recent progress in multi-agent reinforcement learning (MARL) has enabled significant achievements in complex environments, yet scaling up to large, dynamic, and unpredictable tasks remains challenging. Scalability issues arise due to exponential growth in state space and agent interactions along with multi-agent variance and multi-observation variance (Hopkins, 2024). With partial observability,

non-stationarity, and dynamic environments these significantly hinder stable learning (Wei et al., 2024; Liang et al., 2025) underscoring the need for improved frameworks that can handle large-scale multi-agent coordination more efficiently.

One approach to manage large problems is to adopt hierarchical reinforcement learning (HRL). Hierarchical RL techniques reduce dimensionality by decomposing tasks into subtasks governed by high-level policies (Dietterich, 2000; Levy et al., 2019; Nachum et al., 2019). These methods define high-level policies that operate over temporally extended actions or subtasks, thereby pruning the search space. However, reliance on static decompositions or domain knowledge, limits their applicability when tasks evolve significantly over time (e.g., rapidly shifting operational zones).

Dividing large tasks into subtasks and assigning them to homogeneous agents is combinatorial and non-trivial, often leading to overlapping roles or inefficient exploration (Martins et al., 2025a; Zheng et al., 2018). Recent studies use self-organization or hierarchical structures for role negotiation (Li et al., 2021), but current methods struggle with real-time, unpredictable changes in agent tasks and group sizes.

Repetitive subtasks (e.g., scouting or delivery) can be addressed through a policy bank of pre-optimized solutions (Teh et al., 2017; Rusu et al., 2016), enabling faster adaptation. Joint experience-sharing—via parameter, memory, or replay sharing—further improves learning efficiency (Gupta et al., 2017; Rashid et al., 2018). Nonetheless, scaling these techniques to truly massive and fluid domains remains a key research challenge. Here we address the large fast changing state-space aided by a task-specific means to decompose an activity assignment and use policy bank to address many types of activities that are still commonplace for the huge tasks, and learn these policies through shared experiences of homogeneous agents.

3 Proposed Approach

3.1 Task Decomposition, Assignment, and Execution policy

Consider a dynamic task $\mathcal{W}(t)$ that evolves over time t. The task is performed by a set of N agents partitioned into G homogeneous groups, such that $\mathcal{A} = \bigcup_{g=1}^{G} \mathcal{A}_g$. Each group $\mathcal{A}_g$ consists of agents with identical capabilities, meaning minimum set of capabilities $\mathcal{C}_g = \mathcal{C}(a_{gi})$ for all $a_{gi} \in \mathcal{A}_g$. The task $\mathcal{W}(t)$ is composed of M_t activities, where $\mathcal{W}(t) = \{w_{1t}, w_{2t}, \ldots, w_{M_t t}\}$.

Each activity w_{jt} has an associated complexity level $c(w_{jt})$ and requires capabilities $\mathcal{C}(w_{jt})$. The activities change over time, appearing and disappearing based on the task's evolving state. Each activity w_{jt} has an associated *relevance duration* $(t_j^{\text{start}}, t_j^{\text{end}})$ such that the activity exists within the time window $t_j^{\text{start}} \leq t \leq t_j^{\text{end}}$. Activities dynamically emerge and vanish depending on task conditions. The presence of an activity is determined by function $\Psi(\mathcal{W}(t), t)$, such that w_{jt} where $1 \leq j \leq M_t$ exists at time t if $\Psi(\mathcal{W}(t), t) = 1$. The task state function $\Phi(\mathcal{W}(t))$ describes the current status of the task and influences which activities are required.

Since agents in a group share capabilities, task decomposition ensures that there are multiple similar activities to fully utilize homogeneous agents. A decomposition function $\mathcal{D}$ partitions the task into activities that match group capabilities, i.e., $\mathcal{D}(\mathcal{W}(t)) = \bigcup_{g=1}^{G} \mathcal{W}_g(t)$, where $\mathcal{W}_g(t)$ is the subset of activities assigned to $\mathcal{A}_g$. Each activity $w_{jt} \in \mathcal{W}_g(t)$ must satisfy $\mathcal{C}(w_{jt}) \subseteq \mathcal{C}_g$. The decomposition process aims to generate enough similar activities such that $|\mathcal{W}_g(t)| \geq |\mathcal{A}_g|$, ensuring full agent utilization.

Each agent $a_{gi} \in \mathcal{A}_g$ is assigned an activity from $\mathcal{W}_g(t)$. The binary assignment matrix $X_t \in \{0,1\}^{N \times M_t}$ is defined such that $x_{ij} = 1$ if agent a_i is assigned to activity w_{jt}, otherwise $x_{ij} = 0$. This assignment of an agent to an activity can be optimized in many ways depending on the overall goal of executing the task. This optimization directly impacts the efficacy of performing the task and therefore, we formalize this framework here to be able to explore this issue in the subsequent sections. For example, if the goal is to perform the task so as to minimize the execution cost, the agent assignment must minimize $\sum_{i=1}^{N} \sum_{j=1}^{M_t} C(a_i, w_{jt}) x_{ij}$, where $C(a_i, w_{jt})$ is the execution cost of agent a_i working on activity w_{jt}. In general, it must do so while satisfying constraints $\sum_{i=1}^{N} x_{ij} \geq r(w_{jt})$, where $r(w_{jt})$ is the minimum number of agents required to execute activity w_{jt}.

This ensures each activity is assigned sufficient agents. Moreover, $\sum_{j=1}^{M_t} x_{ij} \leq \kappa(a_i)$, where $\kappa(a_i)$ is the maximum number of activities than agent a_i can handle at a given time, thereby ensuring to limit the agent's workload. A simplified representation of this agent assignment is a function $\mathcal{S}(w_{jt})$ that determines the set of agents executing w_{jt}, such that $\mathcal{S}(w_{jt}) = \{a_i \in \mathcal{A} | x_{ij} = 1\}$.

Various operational concerns such as business, technical, and logistics may determine a task decomposition and activity assignment to agents for many complex real-world tasks. Additionally, for the task to be optimized, in addition to an effective agent assignment, it is necessary to also ensure that each agent a_i optimizes its execution of its assigned activity w_{jt}. A task performance metric is given by $J(\mathcal{W}(t), X_t, \Pi)$, where $\mathcal{W}(t)$ represents task activities at time t, X_t is the agent-assignment matrix, and Π denotes execution policies. The goal is to meet all operational concerns and also to continuously adapt $\mathcal{W}(t)$, X_t, and Π such that J improves over time. $\mathcal{W}^*(t)$ and X_t^* are comprehensive when operational concerns are met and when $\sum_{i=1}^{N} x_{ij} \geq r(w_{jt})$ and $\sum_{j=1}^{M_t} x_{ij} \leq \kappa(a_i)$.

Task decomposition to match agent capabilities and generate balanced activities is a combinatorial problem that is often NP-hard. MARL algorithms struggle with such problems, particularly at scale, as shown in Gu et al. (2020); Martins et al. (2025b). They rely on local rewards, perform poorly in discrete combinatorial spaces, and converge slowly in dynamic environments. By Bellman's principle of optimality, if task decomposition and assignment are suboptimal as with MARL, overall task performance cannot be optimal with MARL.

3.2 Two-Phase Approach

Given MARL's limitations in optimally decomposing and assigning tasks in dynamic environments, we introduce a two-phase approach, as illustrated in Figure 1. This iterative process enables continuous adaptation to evolving tasks while ensuring that homogeneous agents execute activities using the most effective policies available.

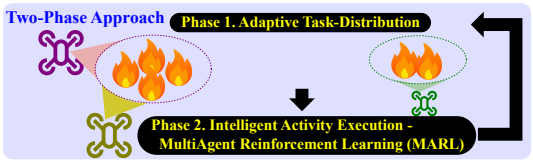


Figure 1: Two-phase approach - iterating between Phase-1 refocusing agent task distribution and Phase-2 executing activity with best policy from the policy bank and shared experience merging of optimal trajectories leads to continuous learning.

3.2.1 Phase One - Task Decomposition and Assignment

In phase one, each agent helps obtain information from their environment and share it with a task distributor. The task distributor decomposes the task in its current state into activities and distributes these activities to the most suitable agents. This allows segmenting a massive state space for a huge task into smaller-scale activities that agents can handle as discussed in the last section. During phase-1 we optimize task decomposition $W^*(t)$ with an optimal assignment X_t^* for all possible tasks $W(t)$ by matching the agent capabilities with the activities. This opens the possibility of complementing pure reinforcement learning with adjunct strategies by leveraging AI-driven task decomposition and assignment methods. Additionally, it becomes feasible to use domain-centric, oracle-centric, human-in-the-loop (HIL), some other learning-based approach, or some combination of all of the above approaches to aid in determining what is the best way for the group of agents to tackle the current state of the task.

At each time-step t, given the current task state $\Phi(W(t))$, we seek an optimal task decomposition $W^*(t)$ and assignment matrix X_t^*. As this is repeated regularly, the system adapts to the dynamic nature of the task and refocuses agents to operate on the currently most relevant aspects of the task. Unlike a pure MARL approach where each agent would learn to tackle a vast task state-space, phase one could refocus the agents to attend a specific narrow state-space of an activity that is most likely to make an immediate and significant contribution to the overall task. By decomposing the large task

environment into small activities that an agent can execute, it becomes amenable to optimization at the local level by the most appropriate reinforcement learning algorithms known for the activity.

3.2.2 PHASE TWO - POLICY EXECUTION AND LEARNING

Each homogeneous agent $a_i \in \mathcal{A}_g$ selects the best-known policy for its assigned activity w_{jt} from a policy bank. Agents execute activities using RL/MARL algorithms such as PPO and record trajectories. A merge operation refines the policy based on the best experiences and stores the updated policy back into the policy bank.

Each agent $a_i \in \mathcal{A}_g$ executes its assigned activity w_{jt} using a policy from a policy bank $\mathcal{B}_g$, where $\pi_{w_{jt}} = \Pi(w_{jt}) \in \mathcal{B}_g$. The policy $\pi_{w_{jt}}$ is selected based on the similarity between the assigned activity and previously encountered activities. At each time step τ, an agent $a_i \in \mathcal{A}_g$ selects an action $a_\tau \sim \pi_{w_{jt}}(h_\tau)$. The policy $\pi_{w_{jt}} : H \times A \to [0, 1]$, where H represents histories and A represents actions, ensures consistency in decision-making across the homogeneous agents in the group. Since agents operate under similar conditions, they can share experiences to collectively refine $\pi_{w_{jt}}$.

Consider $\mathcal{A}_g$ agents executing activity w_{jt}, each following initial policy $\pi_{w_{jt}}$, with expected policy performance $J(\pi_{w_{jt}}) = \mathbb{E}_{\zeta \sim \pi_{w_{jt}}}[R(\zeta)]$, where $R(\zeta)$ is the expected return over trajectory ζ. Each agent a_i collects experience $\mathcal{E}_{a_i} = \{(o_\tau, a_\tau, r_\tau, o_{\tau+1}) \mid \tau = 0, \ldots, T_\zeta\}$ for T_ζ trajectory samples, with POMDP observations $o \in O$, actions $a \in A$, reward $r \in \mathbb{R}$, and time-step τ. The policy improvement in $\pi_{w_{jt}}$ after k updates for experience distribution $\mathcal{E}$ is given by $J(\pi^{(k)}_{w_{jt}}) = J(\pi^{(k-1)}_{w_{jt}}) + \alpha \mathbb{E}_{(o,a)\sim\mathcal{E}}[\nabla J(\pi_{w_{jt}})]$. For individual learning, $\mathcal{E} = \mathcal{E}_{a_i}$. With a merge strategy $\mathcal{M}$, shared learning aggregates experience as $\mathcal{E} = \mathcal{M}(\mathcal{E}_{a_1}, \mathcal{E}_{a_2}, \ldots, \mathcal{E}_{a_{|\mathcal{A}_g|}})$.

Proposition 1. [Convergence Acceleration via Merged Learning] If p homogeneous agents merge the top and bottom n % of the combined trajectories, the policy learns $2pn$ times faster than for a single agent learning using all its trajectories.

Lemma 1.1. [Policy Update through Experience Merging] Updating policy $\pi_{w_{jk}}$ through experience merging with best and worst $n\%$ trajectories ζ across all homogeneous agents ensures improvement in expected task performance: $\mathbb{E}[J(W^*(t), X^*_t, \Pi'_{w_{jt}})] \geq \mathbb{E}[J(W^*(t), X^*_t, \Pi_{w_{jt}})]$

Thus, homogeneous agents can collectively refine a single policy by pooling experiences, leading to faster and more stable learning. During execution, each agent collects experience tuples $\mathcal{E}_{a_i} = \{(o_\tau, a_\tau, r_\tau, o_{\tau+1})\}$. A merge operation refines the policy based on the best-performing trajectories: $\pi'_{w_{jt}} = \mathcal{M}(\pi_{w_{jt}}, \mathcal{E}_{\text{best}})$, where $\mathcal{M}$ integrates high and low reward trajectories into the stored policy. The updated policy replaces the existing one in the policy bank: $\mathcal{B}_g[w_{jt}] \leftarrow \pi'_{w_{jt}}$. This ensures groups of **homogeneous** agents continually refine and reuse the best available policies for task execution under partial observability.

Proposition 2. [Two-Phase Task Optimization] Let $J(W(t), X_t, \mathcal{B}_g)$ be the task performance function, where $\mathcal{B}_g$ is the policy bank. The iterative execution of phase one and phase two ensures the task policy converges to an optimal solution as the iterations progress if

1. Task decomposition and assignment are comprehensive: ($W^*(t)$ and X^*_t), and
2. Policy update through experience merging ensures improvement in expected task performance: $\mathbb{E}[J(W^*(t), X^*_t, \Pi'_{w_{jt}})] \geq \mathbb{E}[J(W^*(t), X^*_t, \Pi_{w_{jt}})]$

Theorem 1 (Task Learning)**.** If there is a dynamic task $\mathcal{W}(t)$ decomposed and assigned comprehensively as $(W^*(t), X^*_t)$ as described in section 3.1, the task $\mathcal{W}(t)$ can be effectively distributed and learned among agents $a_i \in \mathcal{A}$.

Algorithm 1 demonstrates the two-phase approach where agents obtain activity from task distributor, perform the activity using operateAgent procedure where they operate using a reinforcement learning algorithm suitable for the optimal policy for the activity, and collect their experiences in D_i. The agents use a merge strategy to update the policy using Algorithm 2. The updateSharedLearning procedure updates the policy based on the reinforcement learning algorithm used by the agent.

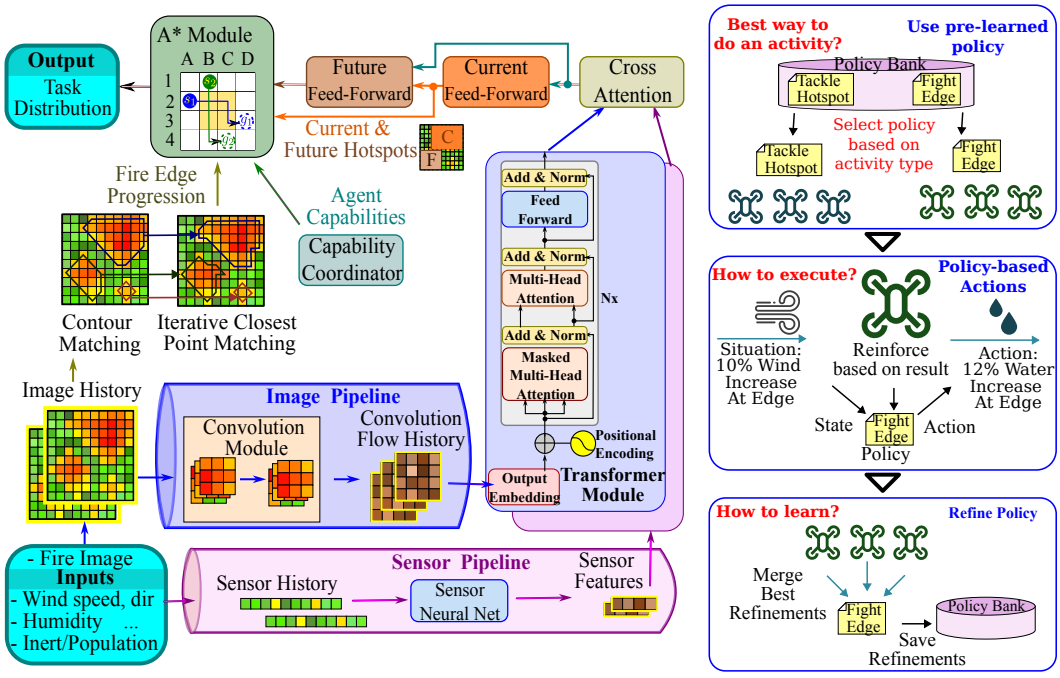
Algorithm 1 Population policy MARL for agent a_i

1: Initialize populations $\Pi^0(b)$ for all agent activity-types $b \in B$
2: **for** each task iteration $k = 1, 2, 3, \cdots$ **do**
3: Obtain activity assignment t_i from task distributor.
4: Select optimal policy for $b = type(t_i)$ as $\Pi^k(b)$ from population.
5: $\Pi_i^k(b) = \Pi^k(b)$
6: $D_i =$ operateAgent(a_i, $\Pi_i^k(b)$)
7: Prune D_i using merge strategy
8: policyMerge(D_i, $\Pi_i^k(b)$, a_i)
9: **end for**

Algorithm 2 Merging learned policies - policyMerge

Require: D_i set of trajectories $(h_i^t, a_i^t, r_i^t, h_i^{t+1})$, $\Pi_i^k(b)$ policy, a_i agent identity
1: $D_{shared} = \bigcup_{i \in I, type(t_i)=b} D_i$
2: Await potentially contributing agents $i \in I$ with $type(t_i) = b$
3: $\Pi^k(b)$= updateSharedLearning($\Pi^k(b)$, D_{shared})
4: save $\Pi^k(b)$ to population.

3.3 Exemplary System



(a) Phase one - task distribution

(b) Phase two - policy learning

Figure 2: Exemplary two-phase approach for forest fire-fighting - standard Phase-1 task distribution complemented with forest fire fighting pipelines. Phase-2 activity execution with optimal policy selection followed by shared experience learning.

This approach was tested with an exemplary system as shown in Figures 2a and 2b. It works with a large number of simulated drones that can operate alongside a few actual replicas of real-world autonomous drones. Figure 2a shows phase one task distribution for a forest fire-fighting system used to showcase the implementation, testing, and results discussed here. Similar implementations handle task distribution for flood control and synthetic domains. Here, the standard phase-1 task distribution

A^* module and capability coordinator are complemented with fire-fighting specific pipelines to expedite learning.

Each drone takes in inputs of critical fire-fighting components like fire image, wind speed and direction, location, humidity, temperature, vegetation type, and population. It detects fire-spread locations and hotspots. An edge progression module detects fire boundary progression since last time-step. These components are fed into a convolution-transformer pipeline to detect current and predict future hotspots and their intensity. A task distributor collects the boundary and hot-spot information along with drone capabilities and uses heuristics-based A^* planner to divide task and assign agents to an activity. Some exemplary activities include fight-edge and fight-hotspot of different sizes and intensities as shown in Figure 2b.

Figure 2b shows phase two where homogeneous agents $a_i \in \mathcal{A}_g$ select best policy $\pi_{w_{jt}}$ from policy bank for its activity w_{jt}. Each agent a_i performs its activity w_{jt} using RL/MARL algorithms based on PPO in Schulman et al. (2017), Actor-Critic in Konda & Tsitsiklis (2000), and DQN in Mnih et al. (2013) for $\pi_{w_{jt}}$ and gather their experience as in Algorithms 1 and merge their experiences as in Algorithm 2. Their shared experience evolves the system and the two-phase approach allows adapting to the dynamics of forest-fires in an effective manner.

4 Results and Discussion

4.1 Experimental Setup

This system was tested with the exemplary forest-fighting system disclosed in the last section. The simulation allows testing a large number of drones in a variety of simulated environments based on real fires, and testing with actual drones shows how the system can operate in real world. A detailed description of the experimentation is disclosed in Appendix A.3Experiment Details.

Simulating wildfires is an active research area, with many accurate ways to model the fire and fire extinguishing. We used the WRF-Fire modeling guidelines in Coen et al. (2013) to determine the spread of wildfires based on factors like fuel and weather, and used Hansen (2012) to determine water extinguisher efficacy based on the spray angle, duration, and power along with vegetation type. A custom simulator was created using these modeling guidelines to test our approach for fighting forest fires. A fleet of three custom-built drones that can coexist with more than 3000 simulated drones was used. The drones were built using a PixHawk with an Ardupilot flight controller, a LASER to emulate a fire extinguisher, and an onboard Raspberry Pi for autonomous operation in coordination with an on-ground custom ground controller integrated with the simulator.

The POMDP reward function R_a used by agents is based on change in fire intensity $\frac{\Delta I}{I}$ and fire-area $\frac{\Delta A}{A}$ as a result of an agent action. $R_a = \alpha \cdot min\left(\frac{\Delta I}{I}, k_1\right) + \beta \cdot min\left(\frac{\Delta A}{A}, k_2\right)$ where, factors α and β control the weightage of changes in intensity and fire-area on the resulting reward. The experiments used by default $\alpha = 2500, \beta = 3500, k_1 = 0.02, k_2 = 0.02$ to balance effects of both intensity and area with slight overweight for area change as smaller area offers better opportunities to contain and fight with fewer high capacity drones.

Both public datasets such as Singla et al. (2020); Fantineh (2023); Nguyen et al. (2024); Center (2025) and synthetic datasets using fire models were used for testing, to test specific aspects of the system for different fire scenarios. Fire was simulated with multi-colored fabric that can be moved along the ground simulating different fire positions and intensities of a fire dataset sample. On-board drone CNN trained for this fabric fire simulation effectively helped simulate many fire scenarios. A fire unit represents a normalized unit area of full fire on the ground. Three groups of drones with capability types small, medium, and large having speeds of 4x, 2x, and 1x and fire extinguisher capacities of 10 liter, 50 liter, and 100 liter resp. were used with varying density and fleet composition per fire unit.

A baseline of firefighters from the public datasets was used to evaluate the overall fire containment performance using fire containment time and extinguisher resources. The fire containment performance of 3,000 simulated agents - comprising a drone fleet with small:medium:large size ratios of 50:35:15 and equipped with water-based extinguishers - is compared to that of real firefighters. The evaluation focuses on improvements in containment time and efficiency of fire-extinguishing resource usage. This was tested for fires of different sizes and hotspots. To ensure repeatability and consistency in performance, multiple trials were conducted to measure percentage improvements in

time and resource usage across fires of different fires. Specifically, medium fires with 10 hotspots and large fires with 30 hotspots were tested, each using 10 different random seeds.

The evaluation return for T time-steps is computed as the cumulative returns during multiple trial episodes, using the greedy policy after training it for T time-steps. The effect of individual components and algorithms to learning the policy is evaluated by comparing evaluation return across configurations, as it isolates the learning dynamics of the training phase.

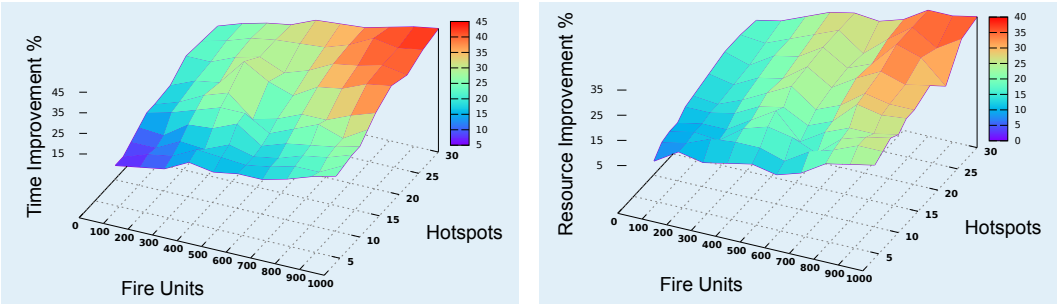
An ablation study of phase one components was done with a transformer, edge progression, and A* distributor to evaluate the efficacy of phase one and its components. The transformer was replaced by a no-transformer component predicting the location and fire intensity using image analysis based on fire colors. The A* component was replaced by a rules-based distribution method, and the edge-progression component was replaced by a fire-edge contour detector along with the mean intensity along each contour.

Algorithms used for two-phase population policy-bank based learning are evaluated, including on-policy PPO, Actor-Critic and off-policy DQN - and compared against traditional MARL versions of these algorithms with 25 agents, including MAPPO as in Yu et al. (2022), an A2C alternative of MAPPO, and QMIX as in Rashid et al. (2018). The scalability of this approach was examined by conducting a test where a hotspot of the same size was assigned to each available agent and recording the total area fought in fixed duration of 2000 time-steps.

The impact of trajectory merging based on shared experiences was analyzed in terms of the fire containment time improvement, while maintaining the same level of resource usage as under the fire-fighter baseline. Three trajectory merge strategies tested include Best-N, Hybrid-N, and Weighted-N trajectory merging. Their impact was evaluated using ANOVA test for statistical significance. Trajectories from similar homogeneous agents were ranked based on reward and used for shared experience learning. The Best-N strategy merges the top N trajectories, hybrid-N merges the top and bottom N trajectories and weighted N merges trajectories by repeating them multiple times based on their weights computed by their top and bottom ranks.

Note that we explored many standard benchmarks that exist for traditional MARL algorithms, such as the SMAC benchmark as in Samvelyan et al. (2019) that focuses on zero-sum competitive games or games with a limited number of agents. These benchmarks did not allow evaluating the many aspects of our system for cooperative tasks with high scalability. Therefore, it was necessary to test this system with an exemplary firefighting system involving coordination between large number of agents to cooperatively accomplish a complex, unpredictable and fast changing task like fighting forest fires.

4.2 Comparisons and Analysis

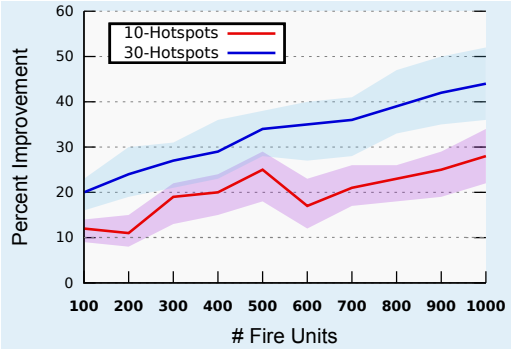


(a) Containment time

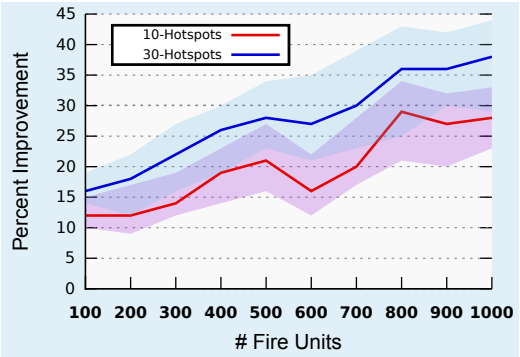
(b) Containment resources

Figure 3: Containment performance

Figure 3 shows the fire containment time and resource improvement of our approach over the baseline system. In Figure 3a, regardless of the number of units and hotspots, our approach outperforms the baseline by over 15% and exceeds 40% for a large number of units and hotspots. In Figure 3b as fire units and hotspots increase, our approach outperforms the baseline in resource consumption. As the number of fire units and hotspots increase, optimizations along burning edges and hotspots increase,



(a) Containment time



(b) Containment resources

Figure 4: Containment by hotspots

greatly reducing the containment time and fire-extinguishing resource usage. Figure 4a and 4b further support this observation, showing greater improvements with more hotspots and larger fire-sizes as bigger tasks offer more scope for optimizations.

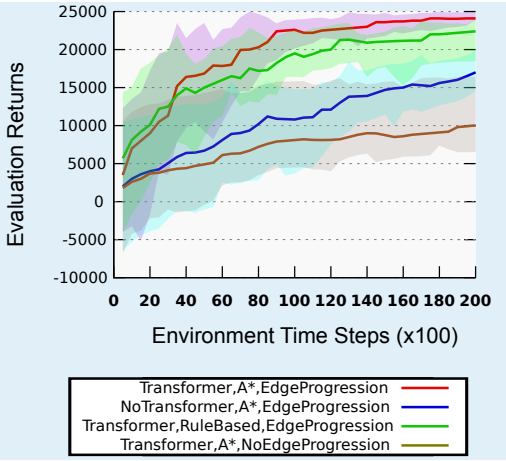


Figure 5: Task distribution methods

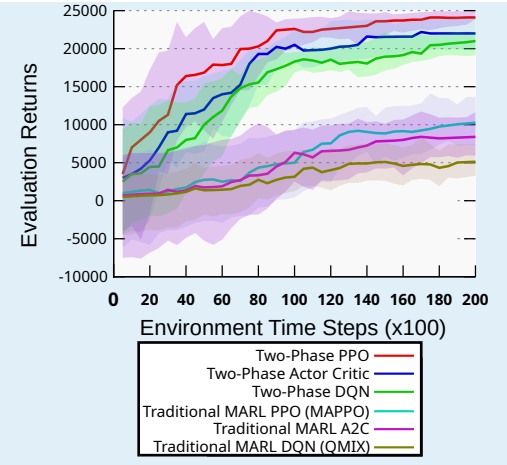


Figure 6: Two-phase learning algorithms

Figure 5 shows ablation study results with transformer, A^*, and edge progression providing the best performance, as edge progression helps detect edges, transformers predict hotspots, and A^* best distributes when edges, hotspots, and agent capabilities are available. This also shows that task-specific combinatorial optimization quickly offers good performance. The transformer allows predicting future hotspots and edge-progression allows accurate edge tracking, which are crucial in staying ahead of the fire spread by timely positioning and spraying extinguisher. A* allows using heuristics based on danger quotient which allows assigning high capability drones to areas that are prone to maximum fire spread.

Figure 6 shows phase two algorithms significantly outperform traditional MARL algorithms. Phase One refocuses training to relevant activities and phase two uses the best known policies to efficiently perform and using shared learning quickly optimizes those policies. PPO clipping the loss function to limit updates performs better than other on and off-policy algorithms. Traditional MARL algorithms take too long to learn and cannot optimize well on its own for such complex tasks. Additionally, the 95% confidence interval for traditional MARL algorithms is much wider as the non-determinism in joint observations and drastically varying joint actions lead to drastically different rewards and much different policy learning. Phase one in the two-phase approach drastically mitigates these issues by

pruning the state-space, which leads to a much faster rise in the early stages of learning and overall much higher evaluation returns.

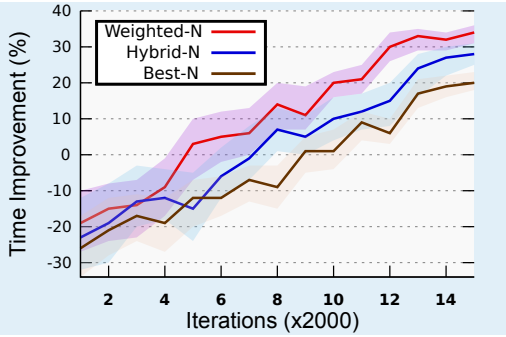


Figure 7: Shared experience learning

Table 1: Trajectory merge summary

Purpose	Stability KL Divergence	Adaptation KL Divergence	Adaptation Iterations
Weighted-N	0.0181	0.0681	8
Hybrid-N	0.0323	0.0776	11
Best-N	0.0206	0.0998	13

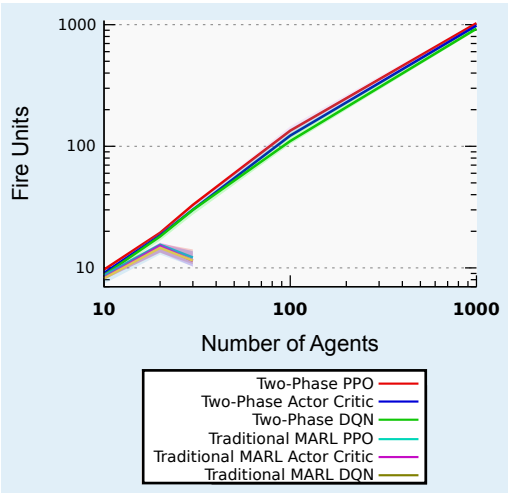


Figure 8: Multi-agent scalability

Figure 8 shows two-phase algorithms significantly outperform traditional MARL algorithms as the number of agents increases. It was not possible to run tests with more than 30 MARL agents as the coordination effort significantly increases and the state-space becomes exponential due to many agents. However, the two-phase approach circumvents this issue allowing for a very large number of agents. This is done by first distributing activities to the best agents capable to handle such activity and then the agents working on those assigned activities, typically either independently or in smaller groups. With groups of homogeneous agents, it becomes possible to use shared experience learning using population policy-bank, making it feasible to learn handling very large fires as shown in the two-phase algorithms in this figure.

Figure 7 shows shared learning performance using three prominent strategies. Weighted-N strategy performed the best reaching an average 34% improvement. The ANOVA test yielded a F-statistic of 13.92 and p-value of $0.0000221 < 0.05$, indicating a statistically significant difference between the means of the three merge trajectory strategies contributing to a significant time improvement. The table 1 shows a low KL divergence for Weighted-N strategy indicating high stability. Furthermost, with a low adaptation KL divergence the Weighted-N strategy is resilient to adversarial environment changes and its low adaptation iteration signifies quick adaptation to diverse new conditions.

These results show that the two-phase multi-agent approach is very effective and scalable in performing large, unpredictable tasks using groups of homogeneous agents.

5 Conclusion

In this paper, we presented a novel approach to effectively learn how to best perform a dynamic task with multiple groups of homogeneous agents in complex environments. The novel two-phase refocus, refine, repeat approach where phase one evaluates how to best assign the agents to accomplish the task, and phase two refines the performance of the task by using the collective intelligence of the agents to learn an optimal RL policy performs well for such tasks. We demonstrated this approach works quite well with an exemplary system where a large number of drones learn to fight forest fires and tested it using both simulations and with actual drones. This approach can be used in many other applications including fighting fires in urban settings, providing medical assistance in urban settings, and many disaster relief scenarios.

Reproducibility Statement

This paper contributes an approach for performing complex tasks with a large number of agents. We fully described our proposed approach in section 3 with additional details in A.2 Exemplary Phase Two Algorithms on algorithm implementations. Theorem 1, Prepositions 1 and 2, and Lemma 1.1 give a theoretical basis and are proven in Appendix A.1 Two Phase Approach Proofs . Section 4 and Appendix A.3 Experiment Details gives details on obtaining the results. This constitutes complete details on reproducing the work presented in this paper.

References

National Interagency Fire Center. National interagency fire center data portal, 2025.

Janice L. Coen, Marques Cameron, John Michalakes, Edward G. Patton, Philip J. Riggan, and Kara M. Yedinak. Wrf-fire: Coupled weather–wildland fire modeling with the weather research and forecasting model. *Journal of Applied Meteorology and Climatology*, 52(1):16–38, January 2013. ISSN 1558-8432. doi: 10.1175/jamc-d-12-023.1. URL http://dx.doi.org/10.1175/JAMC-D-12-023.1.

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Fantineh. Next day wildfire spread. Kaggle, 2023. URL https://www.kaggle.com/datasets/fantineh/next-day-wildfire-spread.

Yifan Gu, Qi Sun, and Xinye Cai. Multiagent reinforcement learning for combinatorial optimization. In *International Conference on Neural Computing for Advanced Applications*, pp. 23–34. Springer, 2020.

Jayesh K. Gupta, Maxim Egorov, and Mykel J. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS) Workshop*, pp. 66–83, 2017.

Richard Hansen. Estimating the amount of water required to extinguish wildfires under different conditions and in various fuel types. *International Journal of Wildland Fire*, 21(5):525, 2012. doi: 10.1071/wf11022.

Bryce Alexander Hopkins. Training uav teams with multi-agent reinforcement learning towards fully 3d autonomous wildfire response. Master's thesis, Clemson University, 2024.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning, 2017. URL https://arxiv.org/abs/1711.00832.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Shuo Li, Chen Wang, Yuan Zhu, and Hongcheng He. Hierarchical multi-agent reinforcement learning for multi-domain task allocation. *IEEE Access*, 9:121580–121593, 2021.

Jiaxin Liang, Haotian Miao, Kai Li, Jianheng Tan, Xi Wang, Rui Luo, and Yueqiu Jiang. A review of multi-agent reinforcement learning algorithms. *Electronics*, 14(4), 2025. ISSN 2079-9292. doi: 10.3390/electronics14040820. URL https://www.mdpi.com/2079-9292/14/4/820.

Miguel S. E. Martins, João M. C. Sousa, and Susana Vieira. A systematic review on reinforcement learning for industrial combinatorial optimization problems. *Applied Sciences*, 15(3), 2025a. ISSN 2076-3417. doi: 10.3390/app15031211. URL https://www.mdpi.com/2076-3417/15/3/1211.

Miguel SE Martins, João Sousa, and Susana Vieira. A systematic review on reinforcement learning for industrial combinatorial optimization problems. *Applied Sciences (2076-3417)*, 15(3), 2025b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Does hierarchy help? Bidirectional data flow for hierarchical reinforcement learning. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, pp. 5508–5518, 2019.

Dung Nguyen, Erin J. Belval, Yu Wei, Karen C. Short, and David E. Calkin. Dataset of united states incident management situation reports from 2007 to 2021. *Scientific Data*, 11(1), Jan 2024. doi: 10.1038/s41597-023-02876-8.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 4292–4301, 2018.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. In *Proceedings of the 33rd International Conference on Machine Learning (ICML) Workshop*, 2016.

Mikayel Samvelyan, Tabish Rashid, Christian S. Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Fred Rudin, Chia-man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, pp. 2186–2196, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Samriddhi Singla, Tina Diao, Ayan Mukhopadhyay, Ahmed Eldawy, Ross Shachter, and Mykel Kochenderfer. Wildfiredb: A spatio-temporal dataset combining wildfire occurrence with relevant covariates. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.

Yee Whye Teh, Victor Bapst, Wojciech M. Czarnecki, Razvan Pascanu, Raia Hadsell, and Nicolas Heess. Distral: Robust multitask reinforcement learning. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, pp. 4499–4509, 2017.

XiaoLong Wei, WenPeng Cui, XiangLin Huang, LiFang Yang, XiaoQi Geng, ZhuLin Tao, and Yan Zhai. Hierarchical rnns with graph policy and attention for drone swarm. *Journal of Computational Design and Engineering*, 11(2):314–326, 03 2024. ISSN 2288-5048. doi: 10.1093/jcde/qwae031. URL `https://doi.org/10.1093/jcde/qwae031`.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

Le Zheng, Jiaxu Yang, Han Cai, Ming Zhou, Wensheng Zhang, and Jun Wang. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, pp. 8222–8229, 2018.

A Appendix

A.1 Two Phase Approach Proofs

A.1.1 Proposition 1

Proposition (Convergence Acceleration via Merged Learning)**.** If p homogeneous agents merge the top and bottom n % of the combined trajectories, the policy learns $2pn$ times faster than for a single agent learning using all its trajectories.

Proof. Since p agents are homogeneous, all trajectories $\mathcal{Z} = \{\zeta_1, \zeta_2, ...\zeta_n\}$ are interchangeable. Therefore, if we were originally getting s trajectories for an agent, we are now getting sp trajectories which are all interchangeable. However, we are only choosing $2n$ % trajectories to train the policy, resulting in $2nsp$ total trajectories. This means if the original training took t time-steps, the new training only takes $t/2pn$ timesteps, which is $2pn$ times faster than a single agent. □

A.1.2 Lemma 1.1

Lemma (Policy Update through Experience Merging)**.** Updating policy $\pi_{w_{jk}}$ through experience merging with best and worst n% trajectories ζ across all homogeneous agents ensures improvement in expected task performance: $\mathbb{E}[J(W^*(t), X_t^*, \Pi'_{w_{jt}})] \geq \mathbb{E}[J(W^*(t), X_t^*, \Pi_{w_{jt}})]$

Proof. The policy gradient theorem states $\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[R(\tau)\nabla_\theta \log p_\theta(\tau)\right]$. Since we select the policies with n% highest and lowest returns, let there be indicator function $\mathbf{1}_{sel}(\tau)$, which is 1 if τ is in the top or bottom n%. Since $E_{\tau\sim p_\theta}\mathbf{1} = 2n$, the expectation of the estimator is: $\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[\mathbf{1}_{\text{sel}}(\tau)\frac{R(\tau)}{2n}\nabla_\theta \log p_\theta(\tau)\right]$. With $J \leftarrow J + \alpha\nabla_\theta J(\theta)$, it results in improvement of J by $\alpha\nabla_\theta J(\theta)$. Thus the $2n$ samples result in expected improvement of $\frac{\alpha}{2n}\nabla_\theta J(\theta)$. □

A.1.3 Proposition 2

Proposition (Two-Phase Task Optimization)**.** Let $J(W(t), X_t, \mathcal{B}_g)$ be the task performance function, where $\mathcal{B}_g$ is the policy bank. The iterative execution of phase one and phase two ensures the task policy converges to an optimal solution as the iterations progress if

1. Task decomposition and assignment are comprehensive: $(W^*(t)$ and $X_t^*)$, and
2. Policy update through experience merging ensures improvement in expected task performance: $\mathbb{E}[J(W^*(t), X_t^*, \Pi'_{w_{jt}})] \geq \mathbb{E}[J(W^*(t), X_t^*, \Pi_{w_{jt}})]$

Proof. Since all homogeneous agents $a_i \in \mathcal{A}$ use a particular policy $\pi_{w_{jt}}$, all trajectories $\mathcal{Z} = \{\zeta_1, \zeta_2, ...\zeta_n\}$ are interchangeable and therefore can be treated equivalently. According to Lemma 1.1, if the top and bottom n% policies from each $\zeta_i \in \mathcal{Z}$ are merged, the $\mathbb{E}(J(W^*(t), X_t^*, \Pi'_{w_{jt}}))$ increases and therefore, the policy improves. Since the two-phase process is continuously repeated, the $\mathbb{E}(J(W^*(t), X_t^*, \Pi'_{w_{jt}}))$ continually improves. □

A.1.4 Theorem 1

Theorem (Task Learning)**.** If there is a dynamic task $\mathcal{W}(t)$ decomposed and assigned comprehensively as $(W^*(t), X_t^*)$ as described in section 3.1, the task $\mathcal{W}(t)$ can be effectively distributed and learned among agents $a_i \in \mathcal{A}$.

Proof. Since the task $\mathcal{W}(t)$ is decomposed and assigned comprehensively i.e. $(W^*(t), X_t^*)$, the constraints $\sum_{i=1}^{N} x_{ij} \geq r(w_{jt})$ and $\sum_{j=1}^{M_t} x_{ij} \leq \kappa(a_i)$ hold true. This ensures that each activity is adequately assigned enough agents and that the agents are not overworked and are capable to work on their assigned activity.

Each activity has an assigned policy from the policy bank and at least $r(w_{jt})$ agents have trajectories for the activity. By proposition 1, a policy with r agents taking the top and bottom n % of trajectories

learns $2rn$ times faster than a single agent. So if $r > \frac{1}{2n}$, distributing learning across multiple agents as done in phase two leads to faster learning. Since $r \geq 1$ and $r > \frac{1}{2n}$, by proposition 2 the system continuously learns by merging the policies, allowing for continual evolution and optimal learning. □

A.2 Exemplary Phase Two Algorithms

An *operateAgent* procedure allows a homogeneous agent $a_i \in \mathcal{A}_g$ to execute an activity $\pi_{w_{jt}}$. This procedure is invoked from Algorithm 1. A variety of single-agent on-policy algorithms like ones based on PPO and Actor-Critic and off-policy algorithms like ones based on DQN can be used for operateAgent procedure that executes activity $\pi_{w_{jt}}$. A sample algorithm based on PPO (Schulman et al., 2017) is shown by algorithm 3

For the forest fire fighting with drones application, the state is the current representation of the drone and its relation with the fire, including location, fire intensity, wind speed, wind direction, humidity, distance of nearest settlement, distance to body of water, etc. The action is the discrete actions the drones can do, including moving in a certain direction, spraying water in a certain direction or intensity, or creating a controlled fire. Since the number of states and actions is reasonable for each particular policy due to the distributed approach, it is able to learn it in a reasonable timeframe.

Algorithm 3 operateAgent: Proximal Policy Optimization (PPO)

Require: Agent identifier a_i, starting policy $\Pi_i^k(b)$
Ensure: Returns a set of trajectories $(h_i^t, a_i^t, r_i^t, h_i^{t+1})$

Initialize actor network π with parameters ϕ
Initialize critic network V with parameters θ
Initialize policy $\pi \leftarrow \Pi_i^k(b)$
Initialize empty trajectories set $\mathcal{D}_i$
for each episode **do**
  for time step $t = 0, 1, 2, \ldots$ **do**
    Observe current state h_i^t
    Sample action $a_i^t \sim \pi(\cdot|h_i^t; \phi)$
    Apply action a_i^t; observe reward r_i^t and next state h_i^{t+1}
    $D_i = D_i \cup \langle h_i^t, a_i^t, r_i^t, h_i^{t+1} \rangle$
    $\pi_\beta(a_i^t|h_i^t) \leftarrow \pi(a_i^t|h_i^t; \phi)$
    for epoch $e = 1, \ldots, N_e$ **do**
      Importance sampling ratio: $\rho(h_i^t, a_i^t) \leftarrow \frac{\pi(a_i^t|h_i^t;\phi)}{\pi_\beta(a_i^t|h_i^t)}$
      N-step: $Adv(h_i^t, a_i^t) = \sum_{\tau=0}^{N-1} \gamma^\tau R(h_i^{t+\tau}, a_i^{t+\tau}, h_i^{t+\tau+1}) + \gamma^N V(h_i^{t+N}) - V(h_i^t)$
      Target: $y_i^t \leftarrow \sum_{\tau=0}^{N-1} \gamma^\tau r_i^{t+\tau} + \gamma^N V(h_i^{t+N})$
      Entropy regularization: $\mathcal{H}(\pi(\cdot|h_i^t;\phi)) = \sum_{a \in \mathcal{A}} \pi(a|h_i^t;\phi) \log \pi(a|h_i^t;\phi)$
      Actor loss: $\mathcal{L}(\phi) \leftarrow -\min \Big[\rho(h_i^t, a_i^t) \cdot Adv(h_i^t, a_i^t),$
      $\text{clip}(\rho(h_i^t, a_i^t), 1-\epsilon, 1+\epsilon)$
      $\cdot Adv(h_i^t, a_i^t) \Big] - \alpha \mathcal{H}(\pi(\cdot|h_i^t;\phi))$
      Critic loss: $\mathcal{L}(\theta) \leftarrow (y_i^t - V(h_i^t;\theta))^2$
      Update parameters ϕ by minimizing actor loss $\mathcal{L}(\phi)$
      Update parameters θ by minimizing critic loss $\mathcal{L}(\theta)$
    end for
  end for
end for
return $\mathcal{D}_i$

A set of trajectories ζ is selected across the group of agents $\mathcal{A}_g$ to merge shared experiences back to the policy $\pi_{w_{jt}}$ before placing it back in the policy bank. The *updateSharedLearning* procedure is invoked by Algorithm 2 to merge shared learning across the agents. A variety of single-agent on-policy algorithms like ones based on PPO and Actor-Critic and off-policy algorithms like ones based

on DQN can be used for *updateSharedLearning* procedure alongside corresponding *operateAgent* procedure. Here, we illustrate an *updateSharedLearning* algorithm based on actor-critic (Konda & Tsitsiklis, 2000) to merge a selection of trajectories $\mathcal{Z} = \{\zeta_1, \cdots, \zeta_n\}$ obtained from homogeneous agents $a_i \in \mathcal{A}_g$ while they executed activity w_{jt} using policy $\pi_{w_{jt}}$.

Algorithm 4 updateSharedLearning: On-Policy Experience Sharing

Require: Shared policy $\Pi^k(b)$, shared experience buffer $\{D_{shared}\}$
Ensure: Updated shared policy $\Pi^k(b)$ with off-policy corrections
Initialize temporary policy $\Pi^k_{\text{temp}}(b) \leftarrow \Pi^k(b)$
for each epoch $e = 1, \ldots, N_e$ **do**
  for each mini-batch of transitions (h^k, a^k, r^k, h^{k+1}) sampled from D_{shared} **do**
    for each agent i **do**
      Importance sampling ratio correcting off-policy updates: $\rho(h_i^k, a_i^k) \leftarrow \frac{\pi(a_i^k|h_i^k;\phi_i)}{\pi_\beta(a_i^k|h_i^k)}$
      N-step Adv: $Adv(h_i^k, a_i^k) = \sum_{\tau=0}^{N-1} \gamma^\tau R(h_i^{k+\tau}, a_i^{k+\tau}, h_i^{k+\tau+1}) + \gamma^N V(h_i^{k+N}; \theta_i) - V(h_i^k; \theta_i)$
      Target: $y_i^k \leftarrow r_i^k + \gamma \max_{a_i' \in A_i} Q(h_i^{k+1}, a_i'; \bar{\theta}_i)$
      Corrected actor loss: $\mathcal{L}(\phi_i) = -\rho(h_i^k, a_i^k) \left(r^k + \gamma V(h^{k+1}; \theta_i) - V(h^k; \theta_i)\right) \log \pi(a_i^k|h_i^k; \phi_i)$
      Critic loss: $\mathcal{L}(\theta_i) \leftarrow \frac{1}{B} \sum_{k=1}^{B} (y_i^k - Q(h_i^k, a_i^k; \theta_i))^2$
      Update actor parameters ϕ_i by minimizing $\mathcal{L}(\phi_i)$
      Update critic parameters θ_i by minimizing $\mathcal{L}(\theta_i)$
    end for
  end for
end for
Update shared policy $\Pi^k(b) \leftarrow \Pi^k_{\text{temp}}(b)$ using aggregated policy updates
return Updated shared policy $\Pi^k(b)$

A.3 Experiment Details

A.3.1 Setup and Parameters

Various aspects of the two-phase approach were tested with experiments using an exemplary forest-fighting system disclosed in section 4.1. A simulated agent was operated using a set of test fire-images and corresponding sensor data for that image. Inputs from many agents are reported to a task distributor that performs the task distribution. A real agent is an actual fire-fighting drone that captures the fire-image using its camera and acquire current sensor data using its on-board sensors to correspond with the captured image. This data is periodically sent to a task distributor to reassign activities to each agent. The test fire-images were input to the image pipeline and the sensor data were input to the sensor pipeline as shown in Figure 2a. The task distribution result assigns a hotspot or an edge to an agent. Such assignment is reported to the agent as an activity assignment. An agent continuously performs its assigned activity as shown in the Figure 2b. A reassignment of a different activity by the task distributor results in the agent preempting its current assigned activity and moving to the new assigned activity. Best and worst trajectories across similar agents performing an activity get used for merging their shared experiences to their shared policy persisted in the policy bank.

A real agent is a Raspberry-Pi based drone exemplary agent that is a X-Configuration Quadcopter UAV with a PixHawk 2.4.8 flight controller driving A2212/KV930 motors with 8038 propellers and SimonK 10A ESC, a GPS M8N and Matek Optical Flow sensor for positioning along with Benewake TFmini Plus LIDAR sensor. Drone captures temperature, humidity, pressure, wind speed, and wind direction using onboard sensors along with image frames using Raspberry Pi Camera and reports them for task distribution by default every 15 seconds. Image resolution defaults to 384 x 384. It communicates directly with a ground control station using onboard WiFi, and falls back to radio telemetry if WiFi is out of range.

Simulated agents are pure software components that ran on multiple servers with 32 core, 128GB RAM, 1TB storage medium end servers. These agents use a pre-captured stream of image and sensor data with around 4800 samples for different fire scenarios. The task distribution uses multiple

16GB vRAM GPUs depending on the number of agents and fire analysis request frequency for each experiment.

Different fire scenarios are simulated based on the actual fire dataset. In a virtual agent, the duration and frequency of spray operation is recorded to determine the effect of the fire extinguisher in changing the fire based on modeling guidelines in Hansen (2012). This change helps in computing the reward for the current action of the agent. When using a real-agent, it is also necessary to precisely recreate a test environment so that the performance of a real-agent can be evaluated in conditions close to that of a real forest fire. Based on an actual fire sample from the fire dataset, a forest-fire mock layout is created on the ground using different fire-colored fabric pieces as shown in 10. The fabric is moved to simulate changes in the fire condition. Images captured by a real-agent is processed using a CNN model that is trained on these forest-fire mock layouts. A point-laser device operated by the real-agent is used to simulate the spraying of fire-extinguisher. The duration and frequency of this laser operation is recorded and using modeling guidelines in Hansen (2012), the effect of the fire extinguisher is determined to guide altering the fire status on ground. In order to streamline results with real and virtual agents, a fire-unit is used to represent one unit of fire. By default one fire-unit maps to one square kilometer of a real forest-fire and this is typically equivalent to one square centimeter of the forest-fire mock layout. Fire unit is used to represent the size of fire for all results in section 4.2.

A.3.2 Containment Performance Study

This experiment evaluated the performance of the two-phase approach against a baseline of actual fire-fighters. It evaluates the improvement of the containment time and fire-extinguisher resources needed to reach that containment against the baseline. A fire was sampled from the dataset and based on its size, groups of homogeneous agents are used to with fleet comprising of 50% small capacity drones, 35% medium capacity drones, and 15% large capacity drones. The drones used a pre-trained population policy-bank. The containment time included the time since the drones are armed to the time the entire fire is extinguished. Moreover, each drone recorded the total amount of fire-extinguisher used and these were compared against the baseline of real fire-fighters. The test was repeated for fires of different size and hotspots. The same test was repeated for multiple trials on samples with 10 and 30 hotspots.

A.3.3 Ablation Study

The task distribution is performed during phase one processing and it can have a profound impact on the overall performance. Since there are multiple components for performing this task distribution, an ablation study was performed to determine the necessary components for optimal task distribution. The transformer was replaced by image-analysis based hotspot detector, the A* component was replaced by a rule-based task assigner, and the edge-progression component was replaced by a contour-based edge processing. A component was swapped out and the evaluation return was recorded to identify which components provide optimal performance.

A.3.4 Two-Phase Algorithms Study

Upon assignment of an activity, each agent loads a policy from the population policy-bank and performs activity steps under the guidance of this policy. The efficacy of this algorithm directly impacts the efficacy of the overall approach and therefore, different algorithms are evaluated to determine which algorithms provide optimal performance. The policies are not pre-trained - the test uses the evaluation returns to as agents learn policies and execute activity steps using these policies. On-policy PPO was evaluated with clipping epsilon of 0.1 with a policy gradient actor and critic models with two layers of 128 nodes. The actor-critic policy also used models with two layers of 128 nodes and DQN used a Q and target networks with two layers of 128 nodes. A shared experience with weighted-N trajectory merging strategy was used to merge experiences of homogeneous agents sharing similar activities. Traditional MARL Algorithms tested include Centralized Training Decentralized Execution Actor-Critic, PPO, and DQN Algorithms. Since traditional MARL algorithms do not perform well, this testing was done using 25 agents to compare the efficacy of two-phase algorithm versus traditional MARL algorithms. The tests were performed for different environment timesteps ranging from 2000 to 20000 time steps.

A.3.5 MULTIAGENT SCALABILITY STUDY

The two-phase algorithms study was further extended to evaluate performance with different number of agents. Each agent was assigned a hotspot spanning a fire-unit and allowed to perform the activity for a total of 2000 time-steps. Upon completion, the amount of fire extinguished across all agents is computed to determine effective total number of fire-units that were collectively extinguished across these agents. The number 2000 time-steps was chosen to allow an agent sufficient time to extinguish a large portion of the fire. It must be noted that since traditional MARL does not scale well beyond around 30 agents, the tests were conducted with only two-phase algorithms beyond 30 agents.

For trajectory merge test as in Table 1, KL divergence shows the difference in the probability distributions. For this paper, it is used to show the improvement of policy refinement through trajectory merging. Stability KL Divergence is the difference in the distributions between the current stable distribution and minor perturbances affecting that stability. Adaptation KL Divergence is the difference in the distributions between the original distribution and a restabilized distribution that has undergone major perturbances such as drastic changes in wind speeds and humidity. Adaptation Iterations is the number of phase-one -> phase-two cycle iterations that it takes to reach the accuracy of the current domain, to see how quickly the system can adapt to different environments.

A.3.6 SHARED EXPERIENCE LEARNING STUDY

This experiment was conducted to study the efficacy of merged experience learning using trajectories from homogeneous agents with similar capabilities performing a similar activity. Unlike the conventional population-based training for policy space response oracle (PSRO) as in Lanctot et al. (2017) for non-cooperative tasks, here, the cooperating agents learn by sharing their experiences upon completion of an activity and the goal is to determine an optimal way to merge the experiences captured in the trajectory of these agents. Trajectories are compared based on a reward for a step in the trajectory. The best-N strategy was tested by selecting only N-best trajectories from the reporting agents, N typically set to one-fourth of total homogeneous agents reporting their trajectories. However, worst experiences also teach what not to do and therefore, a hybrid-N strategy was also tested with best-N and worst-N trajectories. Another variant of hybrid strategy is the weighted-N strategy where the best and worst strategies are given the highest weight among the best-N and worst-N trajectories. More weight causes a trajectory to be repeatedly used that many times for experience learning and each of the N best and worst trajectories are weighted based on their ranking. A policy gets saved in the population policy bank upon shared learning and this policy gets distributed across agents, serving as a critical means to communicate and share experiences across the agents. Therefore, efficacy of shared experience learning forms an important aspect of the two phase learning approach.

A.4 FIRE-FIGHTING WITH DRONES

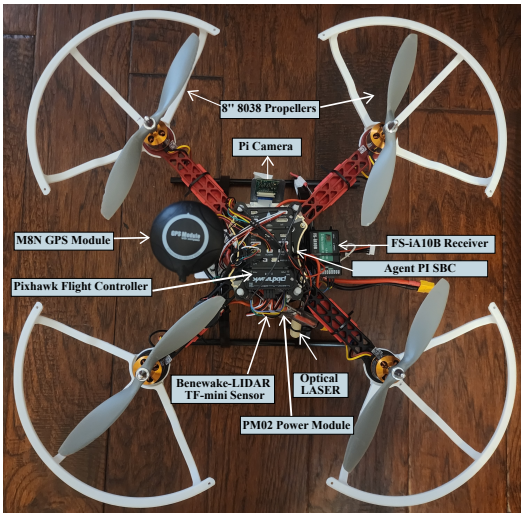


Figure 9: Drone top view

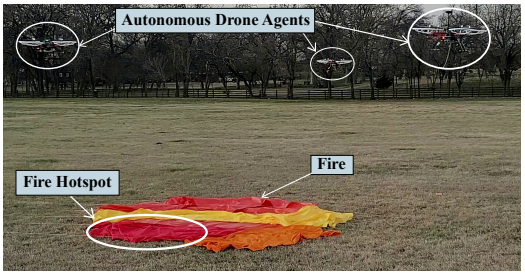


Figure 10: Drones in action