Evaluating Adaptation Performance of Hierarchical Deep Reinforcement Learning

Neale Van Stralen*, Seung Hyun Kim*, Huy T. Tran, Girish Chowdhary

University of Illinois at Urbana-Champaign, IL, USA

***Abstract*— Deep Reinforcement Learning has been used to exploit specific environments, but has difficulty transferring learned policies to new situations. This issue poses a problem for practical applications of Reinforcement Learning, as real-world scenarios may introduce unexpected differences that drastically reduce policy performance. We propose the use of differentiated sub-policies governed by a hierarchical controller to support adaptation in such scenarios. We also introduce a confidence-based training process for the hierarchical controller which improves training stability and convergence times. We evaluate these methods in a new Capture the Flag environment designed to explore adaptation in autonomous multi-agent settings.**

I. Introduction

Deep Reinforcement Learning (DRL) is a promising solution to problems that require optimal decision policies in stochastic and highly dynamic environments. Here, we focus on DRL for autonomous decision-making in adversarial and multi-agent settings. State-of-the-art DRL algorithms have been shown to develop robust policies able to maximize performance in complex games such as StarCraft [1] and Quake [2]. However, many challenges remain which limit the applicability of DRL to real-world multi-agent applications like distributed control systems [3], automated traffic control [4], resource management, search and rescue missions, and robotics [5]. One critical challenge is the ability to handle situations that are different from those experienced during training; i.e., the ability to adapt trained policies to new environments with different dynamics.

DRL can theoretically develop a generalizable policy able to handle new environment dynamics, following exhaustive exploration of the state space. However, such exploration is infeasible for adversarial and multi-agent settings, where the action space grows exponentially with the number of agents; these conditions often result in an optimal policy that is unstable under changes to environment dynamics. One approach to addressing this challenge is to implement a hierarchical reinforcement learning (HRL) architecture that decomposes the task into sub-policies and trains a hierarchical controller (HC) to dynamically assign those sub-policies to agents over time. This architecture could then be efficiently adapted to the new environment, by only updating the HC while keeping lower-level sub-policies fixed.

This paper presents an HRL architecture and training method which uses diverse behavioral sub-policies to adapt policies to environment changes. We assess the ability of a policy to adapt by measuring the performance drop following changes to the environment, the time required to adapt to those changes, and the performance level reached after adaptation. We compare our method to alternative DRL solutions using a Capture the Flag environment that allows for the introduction of changes to adversary capabilities (see Figure 1). The main contributions of the paper are as follows:

- demonstration of an HRL architecture able to adapt to environmental changes,
- a confidence-based training method able to stabilize training in HRL.

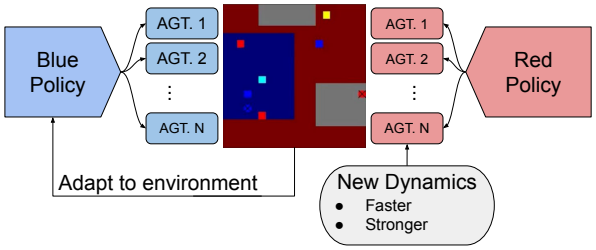


Fig. 1: Adversarial and multi-agent environment where two teams play a game of Capture the Flag. New dynamics can be added to the environment to test adaptation.

II. Background and Related Work

A. Reinforcement Learning

We model our task environment as a Markov Decision Process (MDP), defined by states ($\mathcal{S}$), actions ($\mathcal{A}$), transitions ($\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$), and rewards ($r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$). A policy ($\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$) selects an action at each state, with the environment returning the next state with an immediate reward. The goal of an MDP is to create an optimal policy such that the expected future discounted reward, $\mathbb{E}^{\pi}[\sum_{t=0}^{\inf} \gamma^t r_{t+1}]$, is maximized. DRL uses a deep neural network to model a policy parameterized by θ, where past trajectories and rewards are used to iteratively update θ.

We optimize our policies using the proximal policy optimization (PPO) algorithm, an actor-critic method, because it has a monotonic update nature [6]. PPO uses a total loss to update the policy, calculated as,

$$L_{\text{total}} = L^{\text{CLIP}} + \beta_1 L^{\text{Critic}} + \beta_2 L^{\text{Entropy}}. \quad (1)$$

* These authors contributed equally to the paper.
{nealeav2,skim449,huytran1,girishc}@illinois.edu

Here, L^{CLIP} is a clipped surrogate objective used to guarantee stability in training, calculated as,

$$L^{\text{CLIP}} = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right], \tag{2}$$

where $\hat{A}$ is the advantage function and ϵ is a clipping hyperparameter. The advantage function is calculated as,

$$\hat{A} = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l r_{t+1} \tag{3}$$

$$= r_t + \gamma V(s_{t+1}) - V(s_t). \tag{4}$$

L^{Critic} is a critic loss used to update the value function $V(s_t)$, calculated as [7],

$$L^{\text{Critic}} = \left((r_t + \gamma V(s_{t+1})) - V(s_t) \right)^2. \tag{5}$$

L^{Entropy} is an entropy loss used to improve exploration and stability [8], calculated as,

$$L^{\text{Entropy}} = \sum_i p_i \log p_i. \tag{6}$$

While PPO has shown the ability to handle complex tasks, the method still suffers when adapting to new environment dynamics.

B. Adaptation

Two main lines of work have addressed adaptation in RL: transfer learning and one-shot learning. In instances of a well-defined target environment, transfer learning methods aim to correlate information between the source and target environments. One common method is manifold alignment, in which features of the source and target environment are correlated to serve as an intermediate between the trained policy and the target environment [9], [10]. Another method, progressive networks, adds layers to the trained policy so that features from the source environment can be extracted and applied to the target [11]. While successful, these methods are often computationally expensive, as they require significant exploration of the target environment.

One-shot learning extends transfer learning to consider transfer to unknown environments. These methods focus on enhancing the online training process with sample reuse and augmentation or controlled periods of rapid training. A common method is meta-learning, where a policy is trained to maximize its ability to adapt with limited samples from the target environment, resulting in an ability to update a policy on-line [12], [13], [14], [15]. These methods show promise, but often show drops in short-term performance following environment changes and struggle with target environments that are drastically different from those seen during training.

C. Hierarchical Reinforcement Learning

We propose a new approach to adaptation in RL focused on HRL architectures. HRL has been used successfully in temporally abstracted games such as Atari [16], walking environments [17], robotic path planning [18], and for high-level coordination in multi-agent scenarios [19], [20]. HRL is based on the framework proposed by [21], in which a policy is abstracted using Markovian options. A Markovian option is a $(\mathcal{I}_\omega, \pi_\omega, \beta_\omega)$ tuple, where $\mathcal{I}_\omega$ is an initiation condition, π_ω is a sub-policy, and β_ω is a termination condition. Sets of these options are created where each option has a unique sub-policy π_ω representing a distinct mapping of how to traverse the MDP; these sub-policies can then be composed together based on initiations and terminations to create an overall control policy. HRL creates an overall control policy by using an HC to control the initiations and terminations of these sub-policies. Classically, this framework has been used to help solve complex environments by using options to create temporal abstractions that decompose the MDP space [22]. We aim to extend this framework to help with adaptation by creating higher-level strategic abstractions (i.e., sub-policies) that can be exploited by an HC during adaptation.

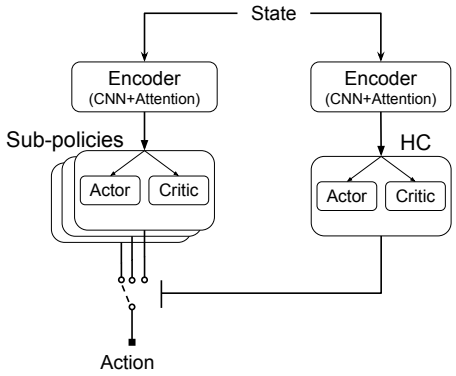


Fig. 2: HRL architecture where an HC selects which sub-policy each agent should use at a given state.

One limitation of HRL is that discounted reward assignment is difficult when switching between sub-policies [19]. A common method for addressing this is to operate the HC at a higher temporal scale where sub-policies are switched at a fixed-step interval. Fixed-step intervals stabilize training by creating longer sub-policy trajectories that are easier for the network to learn from. However, fixed-step training often fails when the selected interval does not temporally align with environment time scales. An alternative method is the option-critic framework, which switches sub-policies based on learned activation and termination functions [22]. These functions create extended sub-policy trajectories that allow sub-policies to be trained end-to-end; however, these sub-policies are typically not globally defined, limiting their value towards adaptation.

III. Methods

A. HRL Architecture

We implement HRL for adaptation by updating the initiations and terminations defining the HC, while keeping the sub-policies themselves fixed. We focus our policy updates on the HC to achieve more drastic changes in explored trajectories than updating a lower-level policy that only controls primitive actions. We propose, and our experiments indeed show, that this approach provides improved adaptation

to new environment dynamics relative to traditional end-to-end DRL.

Our HRL architecture is shown in Figure 2. We model the sub-policies and the HC with separate networks because they operate on different temporal scales, requiring unique spatial and temporal feature encodings of the state. Temporal features are captured by stacking four consecutive frames, which eliminates the need for recurrent elements in our networks. Spatial features are captured using a convolution neural network with a non-local attention block, which can correlate distant features to enhance the understanding of abstracted information, such as tactical position and team coordination [23]. We use a shared feature encoder to allow the sub-policies to share features for improved encoding density. Our networks terminate with two fully-connected layers to create the actor and critic used for training with PPO.

We train a policy for the source environment in two phases, by separately training the sub-policies and the HC. The first phase trains each sub-policy independently. We train these sub-policies using heuristically defined sub-rewards for this effort, as discussed in Section V-B. We leave the exploration of alternative methods for defining and training sub-policies, such as unsupervised discovery, as future work. The second phase then fixes these trained sub-policies and uses them as a basis for training the HC. During adaptation, the trained sub-policies and HC are directly applied to the target environment and only the HC is updated online. We use confidence-based training for the HC to select sub-policy usage, which is used the source and target environments. All network updates are made using the PPO algorithm.

B. Sub-policies

We place two assumptions on the sub-policies used within our architecture. First, we assume that the sub-policies overlap and are distinct in the MDP, which allows the HC to switch between distinct sub-policies to enable adaptation as described earlier. We satisfy this assumption by independently training sub-policies with different engineered rewards. Training sub-policies in parallel (as done in many existing HRL implementations) could lead to sub-policies that only span a localized region of the MDP. Second, we assume that optimal adapted policy can be achieved with the initially defined sub-policies since we do not update sub-policies during adaptation. This assumption is difficult to satisfy in many scenarios and we do not propose guarantees for meeting it; we hope to address this point further in future work. We discuss specific sub-policies used in Section V-B.

C. Confidence-based Training

A challenge with classic HRL is reward assignment for the HC. If the HC makes decisions every step during training, advantages from selected sub-policies cannot be distinguished from each other well enough to provide stable updates to the HC. On the other hand, if the HC makes decisions every n steps, it may provide better reward assignment, but with the potential for poor convergence due to mis-matched temporal scales and poor sample inefficiency.

for *Episode* **do**
  Initialize Switching Threshold;
  while *Episode Running* **do**
    Evaluate internal confidence metric **if** *Confidence* > *Threshold* **then**
      Switch sub-policy;
      Set new threshold;
    else
      Modify threshold;
    end
    Compute Segmented Reward for HC;
  end
  Update Controller based on Segmented Trajectory;
end

Algorithm 1: HC Confidence Based Training

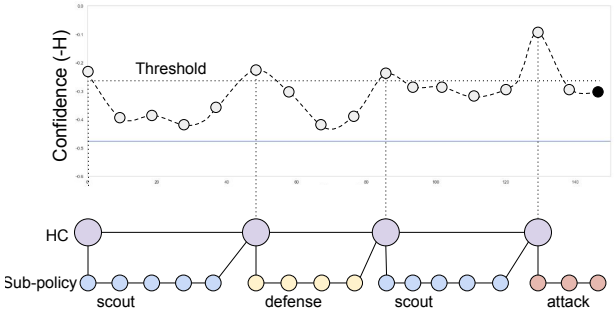
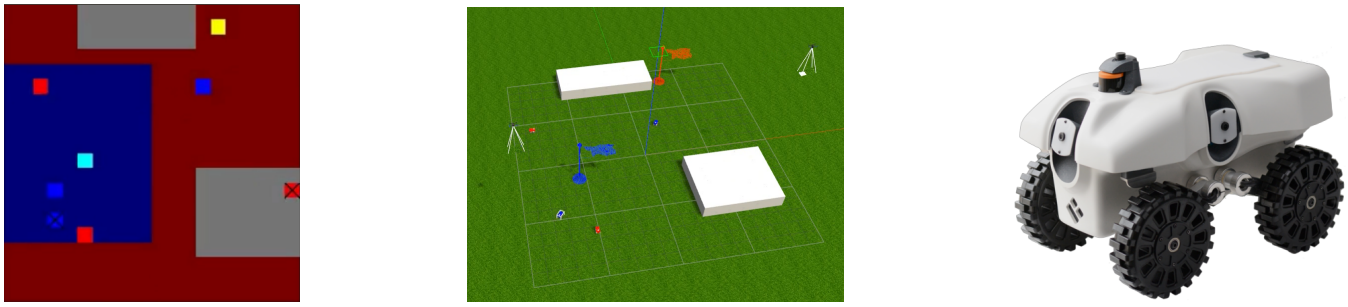


Fig. 3: An example implementation of confidence-based training. The HC begins by assigning the "scout" sub-policy to an agent, which then implements that sub-policy for several steps. The HC is not allowed to assign a new sub-policy to the agent until the confidence threshold has been reached, at which point it assigns the "defense" sub-policy, etc.

We propose a confidence-based training method for improved training of the HC, summarized in Algorithm 1. Our algorithm uses an intrinsic confidence measure and associated switching rules to control when the HC is allowed to switch sub-policies during training. This approach allows for longer trajectories to be sampled early in training when confidence is low in the HC policy, resulting in more stable updates. However, the network will switch more frequently later in training when confidence is high in the HC, resulting in more exploitation of the environment. We calculate the confidence measure as the information entropy of the HC's policy output. We then define a constant confidence threshold to limit when the HC is allowed to switch sub-policies. We focus on a constant threshold for this effort, but note that dynamic thresholds can be easily implemented. An notional example of our method is shown in Figure 3.

Our method is simplistic, but supports HRL adaptation because confidence drastically decreases when the network experiences new scenarios (thus promoting exploration), but then gradually increases given more samples (thus promoting exploitation). A possible extension of the method is to



(a) (b) (c)

Fig. 4: (a) shows the 2D CtF Python simulation, where squares represent ground agents, squares with an "x" represent UAS, light blue and yellow squares represent flags, blue and red areas represent team territories, and grey areas represent boundaries. Only ground agents were considered in this effort. (b) shows a 3D Gazebo-ROS implementation of the game for HITL testing. (c) shows a ground robot available for use within the CtF pipeline.

incorporate metrics that directly capture uncertainty in the environment into the confidence measure, for example by implementing anomaly detection techniques or comparing expected and actual rewards.

IV. Environment

We evaluated our methods in a Capture the Flag (CtF) grid-world environment, where two teams of multiple agents compete against each other to win a game by either capturing the opposing team's flag or eliminating all of their agents. We chose this environment because it enables us to test control policies in an adversarial and multi-agent setting with the ability to easily inject changes to environment dynamics that require policy adaptation. Furthermore, our environment supports the transition of these methods from simulation to real systems by providing a pipeline that goes from Python-based simulation to Gazebo-ROS hardware-in-the-loop simulation (HITL) to real robots and unmanned aerial systems (UAS), as shown in Figure 4. This pipeline can be used to demonstrate and test a wide range of autonomy technologies, including object detection, localization, control, communication, and planning.

The simulation is performed in two main phases: movement and interaction. The movement phase takes inputs from both teams and updates agent positions. The interaction phase determines when the flag has been captured or agents have been eliminated. Agents are eliminated in a stochastic manner, where the elimination probability is based on an agent's strength factor, which team's territory it is in, and how many teammates are nearby. CtF enables the injection of three types of changes to the environment (i.e., adaptation events):

- Changing transition and observation dynamics,
- Changing agent capabilities (e.g., speed and strength),
- Changing the control policy of a team or its agents.

We focused on changes to enemy agent capabilities for this effort, namely speed and strength.

V. Experiments

The main objective of our experiments was to compare the adaptation performance of our HRL architecture to alternative solutions. We compared three methods for creating optimal control policies in our environment: our HRL architecture trained with confidence-based training for the HC (confidence HC), our HRL architecture trained with fixed-step updates for the HC (fixed HC), and a baseline non-hierarchical PPO algorithm (PPO) [6]. We introduced changes to the environment by varying the strength and speed of red team agents after initial training, while keeping the capabilities of blue team agents fixed. We then compared the ability of each method to adapt to these new target environments. The following sections describe details of our experiments.

A. Training

We used the PPO algorithm to train all networks used in our experiments, using the same hyperparameters throughout for consistency. The fixed step parameter for the fixed HC method was selected through experimentation within the source environment. As discussed in Section III-A, we trained our HRL architectures in two phases, one for the sub-policies and one for the HC. The sub-policies were trained until they converged to steady-state performance in their respective partial game settings; more details about sub-policy training are provided in Section V-B. Once the sub-policies converged, the HC was trained in the source environment until it converged to steady-state performance. The baseline PPO was also trained in the source environment until it converged to steady-state performance. All methods generally reached stead-steady performance around episode 150,000. After fully training the HC and baseline PPO, enemy capabilities were changed to create a target environment for the HC and baseline PPO to adapt to. We considered eight target environments, where enemy agents were given some combination of faster or slower speed and weaker or stronger strength. We ran four replicates for each source and target environment and calculated mean performance metrics from those replicates. Most replicates showed win rates within a couple percent of the mean.

We trained policies to control the blue team. During training, the red team policy was randomly selected from a set of heuristically defined policies to support generalization

	# Blue	# Red	Red Policy	Rewards
Source and Target	4	4	All	Capture flag (+1) Flag captured (-1) Eliminate enemy (+0.25)
Attack	2	4	Patrol	Eliminate enemies (+1)
Scout	1	2	Patrol	Capture flag (+1)
Defend	1	4	A^*	Defend flag (+1)

TABLE I: Different game settings and rewards used for policy creation. The "Source and Target" row defines game settings and rewards used for training overall policies used to evaluate adaptation. The "Attack", "Scout", and "Defend" rows define partial game settings and sub-rewards used for training sub-policies.

of blue team policies. We considered random, patrol (randomly navigate within own territory), and A^* (search for enemy's flag using A^*) policies for the red team. Policies were trained in game maps randomly selected from a set of 2000 symmetrical maps (i.e., maps providing no advantage to either team). Table I summarizes the game settings and rewards used for training policies in the source and target environments.

B. HRL Sub-policies

We created sub-policies for our HRL methods by heuristically defining three intrinsic sub-rewards and independently training corresponding sub-policies to optimize each of those sub-rewards. We defined these sub-rewards such that some combination of them could be used to represent the overall task of winning the game. We focused on attack, scout, and defend sub-rewards for this effort. We trained each sub-policy in a partial game setting with randomized maps (including asymmetric ones), using only patrol and A^* red team policies to play against. Table I summarizes the partial game settings and sub-rewards used for training sub-policies.

C. Evaluating Adaptation

A common metric for evaluating adaptation performance is the steady-state performance after n samples from the target environment. This metric captures an important aspect of adaptation, but does not fully consider the transient dynamics of the adaptation process [24]. Instead, we propose a set of three metrics to more thoroughly evaluation the adaptation capabilities of a given control policy. We define δ as the immediate performance drop after a change to the target environment, which captures the initial robustness of the policy. We define σ as the steady-state performance after adapting to the target environment, which captures the long-term ability of the policy to adapt. Finally, we define τ as recovery time, calculated as the number of training episodes required to bring the win rate to within 2% of the σ, which captures the ability for the policy to quickly adapt. These metrics are visualized in Figure 5.

VI. Results and Discussion

Table II shows mean values for our adaptation metrics in our eight target environments. Overall, we see that our confidence HC method generally shows better immediate performance drop results than the fixed HC and baseline

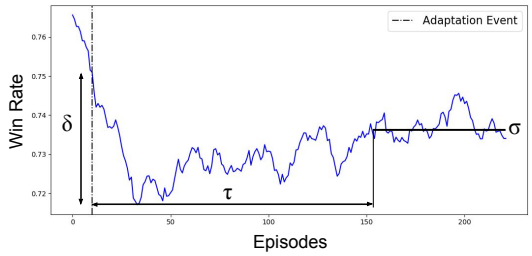


Fig. 5: Metrics used to evaluate adaptation performance of a policy. δ is the immediate performance drop, τ is the recovery time, and σ is the steady-state performance.

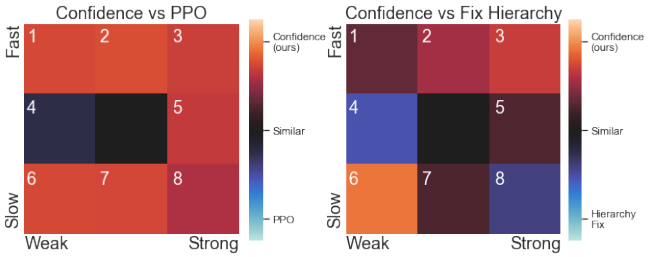


Fig. 6: Heat map comparison of immediate performance drop between methods for the eight target environments. The x-axis defines the strength of enemy agents, while the y-axis defines their speed. Colors show how much better one method was than the other for the target environment, with respect to δ.

PPO methods. Regarding recovery time, the confidence HC and baseline PPO methods show the best results for a similar number of target environments. However, despite this similarity, we also see that when the confidence HC shows a shorter recovery time than the baseline PPO, it tends to be significantly shorter. Alternatively, when the baseline PPO shows the shortest recovery time, the confidence HC result is only slightly worse. Finally, we see that the steady-state performance of the baseline PPO is generally higher than both HRL methods, though these differences are relatively small compared to differences seen in the other adaptation metrics. This result is likely due to the fact that we did not update our sub-policies during adaptation and indicates that our sub-policies were not optimally defined for these target environments. Enabling sub-policy adaptation may address these differences by enabling the HC to reach a globally optimal policy for these target environments; however, enabling sub-policy adaptation may also lead to sub-policy convergence and limit adaptation performance to future environment changes (i.e., it may lead to overfitting for to the target environment).

Figure 6 provides a more direct comparison of the immediate performance drop results for our confidence HC method relative to the baseline PPO and fixed HC methods. Here, we clearly see that the confidence HC method provides a smaller or similar immediate performance drop compared to the baseline PPO in most target environments. Relative to the fixed HC method, the confidence HC only shows worse performance in immediate performance drop in scenarios

Target Environment	Enemy Capabilities		Confidence HC (ours)			Fixed HC			Baseline PPO		
	Strength	Speed	δ	τ	σ	δ	τ	σ	δ	τ	σ
1	Weak	Fast	**0.045**	**5.0**	0.752	0.116	11.0	0.664	0.134	58.0	**0.775**
2	-	Fast	**0.078**	46.0	0.681	0.135	42.0	0.685	0.157	**30.0**	**0.698**
3	Strong	Fast	**0.255**	**1.0**	**0.580**	0.283	N/A	0.453	0.324	46.0	0.568
4	Weak	-	0.061	N/A	0.838	-0.026	**17.0**	0.823	**-0.055**	27.0	**0.858**
5	Strong	-	-0.003	**29.0**	0.837	**-0.084**	N/A	0.809	0.117	102.0	**0.924**
6	Weak	Slow	0.121	**56.0**	**0.891**	**0.079**	N/A	0.617	0.161	83.0	0.777
7	-	Slow	**-0.047**	92.0	0.809	0.020	N/A	0.754	0.030	**90.0**	**0.879**
8	Strong	Slow	**-0.017**	N/A	0.780	0.078	N/A	0.704	0.182	**106.5**	**0.852**

TABLE II: Comparison of our adaptation metrics for the eight target environments. A τ of "N/A" indicates that the method did not reach a steady-state performance above the initial performance drop. The best performing method is shown in bold.

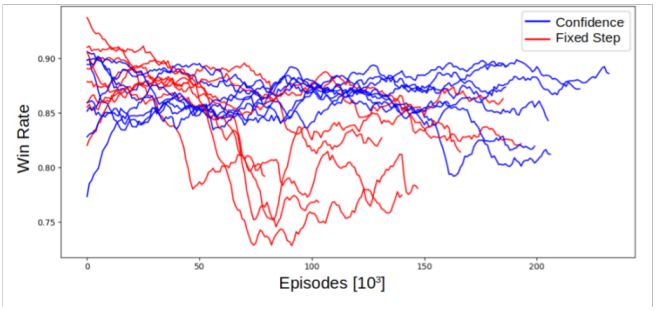


Fig. 7: The transient win-rate of the confidence and fixed HCs replicates in the weak-slow and slow environments. Confidence HC shows more stable convergence than fixed HC.

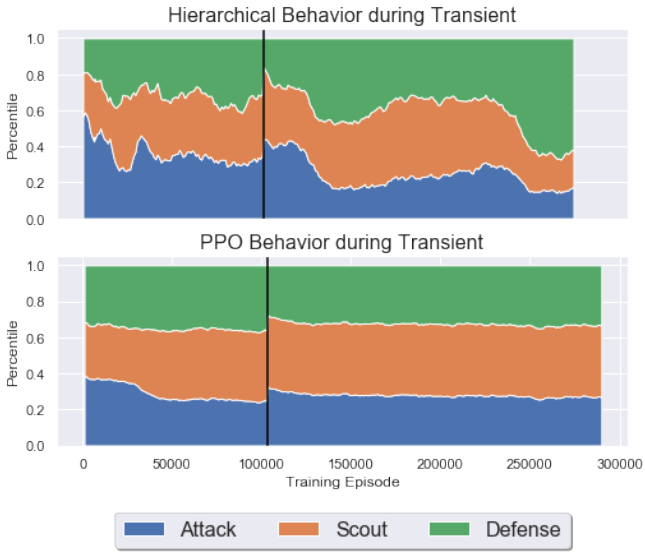


Fig. 8: A comparison of the transient performance of the confidence HC and baseline PPO methods with an adaptation occurring at the black vertical line. The confidence HC behavior (top) shows sub-policy usage. The baseline PPO behavior (bottom) is estimated by calculating the sub-rewards that would have been received from the environment, and determining the fraction corresponding to each sub-policy.

where the environment got easier (i.e., the enemy got slower, weaker, or some combination of the two).

We also compare the convergence performance of our confidence HC method to the fixed HC method, as shown in Figure 7. As previously stated, the fixed HC method requires that the fixed step update parameter is temporally aligned with changes in the game, which is difficult to ensure. These results show that using our confidence-based training method notably reduces the likelihood of failed convergence when training HRL architectures, relative to fixed step training.

Finally, we explore the transient behaviors of our confidence HC method and the baseline PPO to understand how these networks before and during adaptation. Figure 8 shows how often each sub-policy was selected by a confidence HC policy before and during adaptation, and compares this with the distribution of sub-rewards received from actions taken by the baseline PPO. While this is not an direct comparison, we see that the confidence HC method creates significant shifts in the selected sub-policy distribution during adaptation, whereas the baseline PPO maintains near constant behaviors. These differences in transient behaviors may explain how our confidence HC method was often able to outperform the baseline PPO in adaptation, as they suggest the confidence HC policies were more flexible in their ability to change sub-policy usage to better suit target environments.

VII. Conclusion

This paper presents an HRL architecture and confidence-based training method designed to enable adaptation to new dynamics in adversarial and multi-agent settings. Our results show that using specialized sub-policies in a hierarchical structure significantly reduces both the immediate performance drop due to a sudden environment change and the training time required to exploit the new environment (i.e., adapt to it), relative to a baseline PPO network. We also demonstrate that our confidence-based training method enables this HRL architecture to be more stably trained and updated during adaptation than a common fixed step method. One future direction for this work is the unsupervised discovery of optimal sub-policies for adaptation in HRL architectures. Another direction is the demonstration and testing of these adaptive RL techniques on real systems, for example using the CtF pipeline presented in this paper.

- CtF Code: https://github.com/raide-project/ctf_public

VIII. Acknowledgements

This work was supported by DARPA Grant FA8750-18-C-0137.

References

[1] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising Experience Replay for Deep

Multi-Agent Reinforcement Learning," in *Proceedings of the 34th International Conference on Machine Learning*, Sydney, Australia, 2017. [Online]. Available: http://arxiv.org/abs/1702.08887

[2] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, "Human-level performance in 3D multiplayer games with population-based reinforcement learning," *Science*, vol. 364, pp. 859–865, 2019.

[3] M. Hussin, N. Asilah Wati Abdul Hamid, and K. A. Kasmiran, "Improving reliability in resource management through adaptive reinforcement learning for distributed systems," *Journal of Parallel and Distributed Computing*, vol. 75, pp. 93–100, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2014.10.001

[4] S. Mikami and Y. Kakazu, "Genetic reinforcement learning for cooperative traffic signal control," *IEEE Conference on Evolutionary Computation - Proceedings*, vol. 1, pp. 223–228, 1994.

[5] M. Zucker and J. A. Bagnell, "Reinforcement planning: RL for optimal planners," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1850–1855, 2012.

[6] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proceedings of the 31st International Conference on Machine Learning*, Lille, France, 2015.

[7] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," pp. 1–14, 2015. [Online]. Available: http://arxiv.org/abs/1506.02438

[8] Z. Ahmed, N. L. Roux, M. Norouzi, and D. Schuurmans, "Understanding the impact of entropy on policy optimization," 2018. [Online]. Available: http://arxiv.org/abs/1811.11214

[9] H. B. Ammar, E. Eaton, P. Ruvolo, and M. E. Taylor, "Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment," *Proceedings of the National Conference on Artificial Intelligence*, vol. 4, pp. 2504–2510, 2015.

[10] G. Joshi and G. Chowdhary, "Cross-Domain Transfer in Reinforcement Learning Using Target Apprentice," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 7525–7532, 2018.

[11] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-Real Robot Learning from Pixels with Progressive Nets," no. CoRL, pp. 1–9, 2016. [Online]. Available: http://arxiv.org/abs/1610.04286

[12] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," *Advances in Neural Information Processing Systems*, pp. 3637–3645, 2016.

[13] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-Learning with Memory-Augmented Neural Networks," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2740–2751, 2016.

[14] D. J. Rezende, S. Mohamed, I. Danihelka, K. Gregor, and D. Wierstra, "One-Shot generalization in deep generative models," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2274–2282, 2016.

[15] S. Ravi and H. Larochelle, "Optimization as a Model for Few-Shot Learning," *ICLR*, pp. 1–11, 2017.

[16] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *Advances in Neural Information Processing Systems*, no. Nips, pp. 3682–3690, 2016.

[17] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[18] B. Marthi, S. Russell, D. Latham, and C. Guestrin, "Concurrent hierarchical reinforcement learning," in *IJCAI International Joint Conference on Artificial Intelligence*, 2005, pp. 779–785.

[19] M. Ghavamzadeh, S. Mahadevan, and R. Makar, "Hierarchical multi-agent reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 197–229, 2006.

[20] Y. Cai, S. X. Yang, and X. Xu, "A combined hierarchical reinforcement learning based approach for multi-robot cooperative target searching in complex unknown environments," *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL*, pp. 52–59, 2013.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in RL," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.

[22] P. L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pp. 1726–1734, 2017.

[23] L. Wang and J. T. Wu, "Characterizing the dynamics underlying global spread of epidemics," *Nature Communications*, vol. 9, p. 218, 2018. [Online]. Available: http://www.nature.com/articles/s41467-017-02344-z

[24] D. D. Woods, "Four concepts for resilience and the implications for the future of resilience engineering," *Reliability Engineering and System Safety*, vol. 141, pp. 5–9, 2015.