Adapting to the "Open World": The Utility of Hybrid Hierarchical Reinforcement Learning and Symbolic Planning

Pierrick Lorang[1,2], Helmut Horvath[3], Tobias Kietreiber[3],
Patrik Zips[2], Clemens Heitzinger[3], and Matthias Scheutz[1]

***Abstract*— Open-world robotic tasks such as autonomous driving pose significant challenges to robot control due to unknown and unpredictable events that disrupt task performance. Neural network-based reinforcement learning (RL) techniques (like DQN, PPO, SAC, etc.) struggle to adapt in large domains and suffer from catastrophic forgetting. Hybrid planning and RL approaches have shown some promise in handling environmental changes but lack efficiency in accommodation speed. To address this limitation, we propose an enhanced hybrid system with a nested hierarchical action abstraction that can utilize previously acquired skills to effectively tackle unexpected novelties. We show that it can adapt faster and generalize better compared to state-of-the-art RL and hybrid approaches, significantly improving robustness when multiple environmental changes occur at the same time.**

I. Introduction

As robot applications expand into the "open world", away from constrained "closed-world" applications where designers get to control much of the operating environment, a pressing challenge to be addressed is rapid adaptation to unforeseen changes in robot's task environment that the agent couldn't anticipate due to lack of prior knowledge —we will refer to such changes as "novelties" (e.g., see [1]).

While some novelties may not impact an agent's performance, others could lead to task failure if ignored. A popular method for addressing such changes in robotics is reinforcement learning (RL), especially when combined with deep neural networks as they can approximate functions in large continuous domains [2]. However, neural networks lack mechanisms to restrict their training to relevant data in such large domains, and they suffer from a performance decline when acquiring new concepts sequentially [3], [4], a phenomenon known as *catastrophic forgetting* [5]. Integrating RL-based approaches with symbolic constraints and reasoning capabilities has recently emerged a promising way to address this stability-plasticity dilemma in neural systems [6]–[9]. Yet, these *hybrid approaches*, while better than pure RL learners, still show slow adaptation to changes, even in moderately sized discrete domains. To be applicable in robotic tasks, these methods need to be applied to continuous domains and shown to be able to adapt quickly to unforeseen conditions.

[1]Tufts University, Human-Robot Interaction Laboratory, Medford, MA, USA first.last@tufts.edu

[2]AIT Austrian Institute of Technology GmbH, Center for Vision, Automation & Control, Vienna, Austria first.last@ait.ac.at

[3]TU Wien, Institute of Information Systems Engineering, Faculty of Informatics, TU Wien, Vienna, Austria first.last@tuwien.ac.at

We propose a hybrid hierarchical RL and planning approach that is able to overcome catastrophic forgetting and swiftly adapt to novelties, outperforming current neural and hybrid methodologies. Enhanced robustness is demonstrated, highlighting zero-shot learning potential in select scenarios.

Our novel contributions include:

- (1) A novel hybrid approach integrating a nested hierarchical action abstraction system into a neural RL learner, allowing the agent to quickly build new *skills* from previously acquired policies and solidifying its overall robustness to changes in complex open-world environments (see Fig. 2).
- (2) The application of hybrid RL planning in a large continuous domain and the utilization of plan operators for transfer learning between policies and better curriculum learning to avoid catastrophic forgetting.
- (3) A comprehensive performance assessment of our approach in the realistic continuous self-driving CARLA [10] simulation and comparison with a standard RL and a hybrid agent on continual novelty accommodation scenarios (see Fig. 1).

II. Related Work

Challenges of open-world learning have been recently explored using variations of deep reinforcement learning and hybrid learning-planning approaches [11], [12]. These approaches, proposed as potential solutions for non-stationary learning, often face challenges like catastrophic forgetting [13] and difficulty adapting to sudden changes in the task environment, leading to permanent task failure. Moreover, tackling such changes in continuous open-ended environments is still not addressed. For example, [7]–[9], [14] develop integrated layered architectures to tackle non-stationarity in open-world environments, but assume simple discrete domains. [15] and [16], on the other hand, present a neurosymbolic approach for open-world novelty accommodation, in which they learn a world model that uses imagination to help its policy adapt to novelties, but they do not inject novelties continually, and the domains are assumed to be discrete. Some recent work in open-ended self-driving car environments developed algorithms for identifying known entities and unknown objects (e.g., [17], [18]) but they are restricted to perception and do not address execution. There is currently no system that demonstrates *the utility of hybrid hierarchical RL planning approaches in continuous robotic domains*.

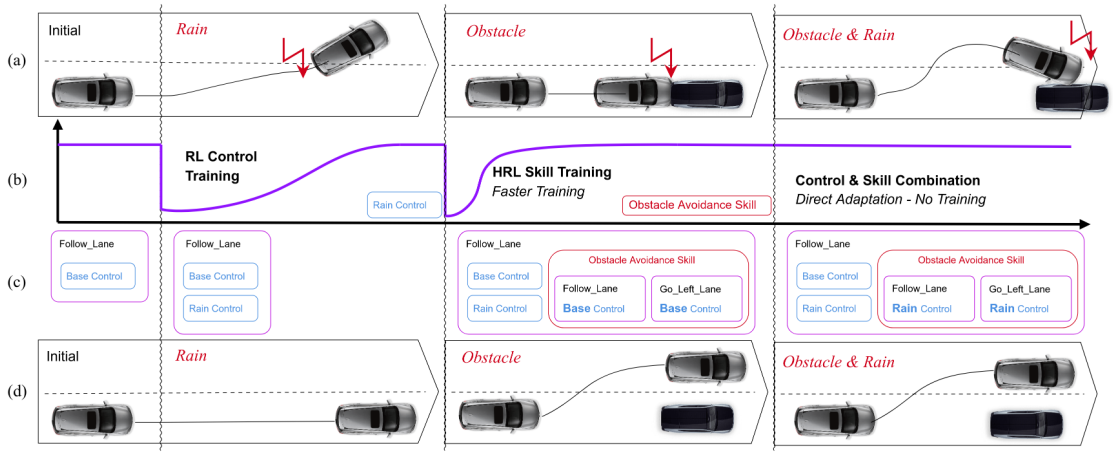

Fig. 1: Adaptive response to environmental changes. (a) In open-world settings, alterations can lead to reduced task performance or failure for state-of-the-art RL algorithms. (b) Our approach employs hierarchical RL to utilize learned controls and skills, enabling continual improvement in adapting to new challenges. (c) Our method effectively responds to introduced novelties, in some cases, without even requiring more training. (d) This adaptability is realized through a Hybrid Planning and Learning framework, integrating a nested hierarchical learner which generates and links new execution protocols to failing planning operators.

III. Preliminaries

We define domain representations for the planning layer and separately for the execution layer (Fig. 2), denoting $\mathcal{S}$ as the discrete state space used by the planning level and $\tilde{S}$ as the continuous sub-symbolic state space used by the execution level. We briefly review basic RL and planning concepts before introducing our proposed approach.

Reinforcement Learning (RL). A Markov Decision Process (MDP) denoted as $M = \langle \tilde{\mathcal{S}}, \mathcal{A}, R, \tau, \gamma \rangle$, consists of an infinite set of sub-symbolic states $\tilde{\mathcal{S}}$ and actions $\mathcal{A}$, a transition function τ that determines the probabilities of transitioning from one state to another given an action, and a function $R(\tilde{s}, a, \tilde{s}')$ that assigns rewards to each transition. An RL algorithm aims to maximize the expected sum of future discounted rewards, denoted as G_t, at a specific time t, with discount factor $\gamma \in [0, 1)$. The value of a state-action pair under a policy π, denoted as $Q^\pi(\tilde{s}, a)$, represents the expected return when starting in state $\tilde{s}$, performing action a, and then following policy π. A generalization of the MDP is the semi-Markov decision process (SMDP) in which the amount of time between one decision and the next is a random variable, either real- or integer-valued.

Hierarchical Reinforcement Learning (HRL). In hierarchical RL, we consider "closed-loop partial policies" designed for specific parts of the problem. For MDPs, this extension adds the set of options [19] to the admissible actions, each of which can itself invoke other options, thus allowing a hierarchical specification of an overall policy. The original one-step actions, now called the "primitive actions", may or may not remain admissible. Extensions along these general lines result in decision processes modeled as SMDPs, where the waiting time in a state now corresponds to the duration of the selected option. If t is the waiting time in state $\widetilde{s}$ upon execution of option a, then a takes t steps to complete when initiated in $\widetilde{s}$, where the distribution of the random variable t depends on the policies and termination conditions of all lower-level actions comprising a.

Hybrid Planning and Learning. We consider a domain description $\sigma = \langle \mathcal{E}, \mathcal{F}, \mathcal{S}, \mathcal{O} \rangle$, where $\mathcal{E}$ is a set of known entities in the environment, $\mathcal{F}$ is a set of known predicates (with their negations) used to describe the relations between entities, $\mathcal{S}$ is the set of environmental states that consists of grounded predicates, and $\mathcal{O}$ denotes the set of known operators o_i such that ψ_{o_i} and ω_{o_i} denote the preconditions and effects of o_i respectively. Following [8], we consider an Integrated Planning Task (IPT) $\mathcal{T} = \langle T, M, d, e \rangle$, that combines the planning task with the lower-level MDP to ground symbolic predicates, specify goals symbolically, and implement action execution from a higher level (operators) to lower level (policies). An IPT consists of a STRIPS task [20] $T = \langle \mathcal{E}, \mathcal{F}, \mathcal{O}, s_0, s_g \rangle$ (s_0 and s_g being the initial state and goal state respectively), an MDP $M = \langle \widetilde{\mathcal{S}}, \mathcal{A}, R, \tau, \gamma \rangle$ (which describes the low-level properties of the environment). We also define the goal set $\widetilde{\mathcal{S}}_g$ as the set of subsymbolic states that satisfy the effects ω_{o_i} of the failed operator o_i.

The planning domain σ is related to the MDP, across the state and action spaces, through two functions (see Fig. 2). The first function is the ***detector function*** $d : \widetilde{\mathcal{S}} \to \mathcal{S}$, where $\widetilde{\mathcal{S}}$ is the set of sub-symbolic states in M. The detector function d maps $\widetilde{\mathcal{S}}$ onto the high-level planning states $\mathcal{S}$ and is therefore surjective: multiple sub-symbolic states can be mapped to the same symbolic state, and each such state has at least one associated sub-symbolic state. The second function is the ***executor function*** $e : \mathcal{O} \to \mathcal{X}$, which maps each planning operator $o \in \mathcal{O}$ to a set of *executors* in $\mathcal{X}$. An executor is defined as $x = \langle I_x, \pi_x, \beta_x \rangle$, where $I_x \subseteq \mathcal{S}$ is the initiation set, denoting the set of symbolic states where the action executor x is available for execution, $\pi_x : \widetilde{\mathcal{S}} \to \mathcal{A}$ is a policy leading to a termination state in $\widetilde{\mathcal{S}}_g$, and $\beta_x(\tilde{s}) \in \{\text{False}, \text{True}\}$ is the termination indicator which

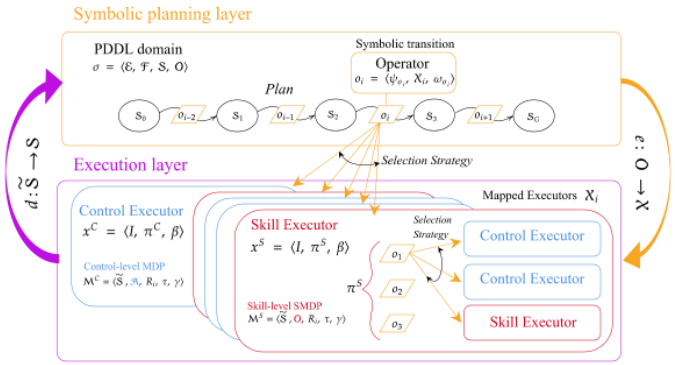

Fig. 2: The hybrid architecture for adapting to novelties and preventing catastrophic forgetting (see text for details).

returns True if the exploration using π_x can be terminated at $\tilde{s}$ and False otherwise.

Problem Formulation. A hybrid agent A starts with some prior knowledge $\mathcal{K}$ described as a planning domain, an initial set of executors in $\mathcal{X}$, a detector function d and an executor function e. If $\mathcal{K}$ is sufficient to solve the planning task T, the agent will find a plan $\mathcal{P}$ for the IPT $\mathcal{T}$ and execute it. If during plan execution the expected effects of any operator are not achieved, because of the presence of *novelty* with respect to the agent's current knowledge, execution is aborted, and the agent must attempt to recover from this failure by finding a *Stretch-IPT* $\tilde{\mathcal{T}} = \langle T', M', d', e' \rangle$ (where $T' = \langle \mathcal{E}', \mathcal{F}', \mathcal{O}', s_o, s_g \rangle$) for which a solution exists in the lower-level MDP M and map new executors onto new operators (we do not deal with new mappings of entities or fluents here).

IV. Nested Hierarchical Framework

Our framework (Fig. 2) consists of two core algorithms 1 and 2 in addition to the planning and RL algorithms. The agent uses a symbolic planner to generate a plan, i..e, a sequence of operator invocations, for which the executor function e finds the set of mapped control or skill executors which are then ordered by the *prioritization strategy* based on their effectiveness in the current environmental context. A control executor uses primitive actions to operate in the lower MDP, while a skill executor uses grounded operators to act in a discrete-time semi-MDP atop the control MDP, acting as a HRL policy that chooses between "options" to be executed for a finite number of steps. These "options" can either be control executors, or other (nested) skill executors. Note that in our framework, the executor definition is similar to that of "options" defined by [21], except that we allow for the direct derivation of the initiation and termination conditions from the agent's observations or knowledge. For an executor $x = \langle I_x, \pi_x, \beta_x \rangle$, π_x (for π_x^c or π_x^s) is the RL policy which explores and learns the missing information by acting in the transformed environment, I_x is the initiation set continually abstracted from observation in this environment (using the detector function d), and β_x is derived from the agent's knowledge. To build I_x, we use the detector function d to abstract symbolic information of the subsymbolic states from which the executor found a path to its goal, and β_x is computed from the grounded predicates of the failed operator's expected effects (using the plan $\mathcal{P}$). A unique reward function R_i is associated with each operator o_i to learn the new associated executors, $x^s_{new} = \langle I_x, \pi^s_x, \beta_x \rangle$ if it is a skill and $x^c_{new} = \langle I_x, \pi^c_x, \beta_x \rangle$ if it is a control.

Algorithmic description. The agent attempts to generate a plan from its current state using its domain knowledge (Alg. 1 line 1) and if successful, gathers the set of all executors $\mathcal{X}_i$ for each operator o_i in the plan, prioritizing them (line 5) based on a selection strategy (cf. below) and executing them (line 7) until one succeeds (execution is deemed successful if the agent reached a plannable state in S_g from which a plan $\mathcal{P}$ to the task goal exists). If all executors in $\mathcal{X}_i$ fail or if $\mathcal{X}_i$ is empty, the agent enters *Recovery mode* (lines 13–24) through which it learns either a new skill, in a skill-SMDP $M^s = \langle \widetilde{\mathcal{S}}, \mathcal{O}, R_i, \tau, \gamma \rangle$, or a new control, in a control-MDP $M^c = \langle \widetilde{\mathcal{S}}, \mathcal{A}, R_i, \tau, \gamma \rangle$. The decision which MDP to consider (line 14) is based on whether the change has local or global effects which the agent can determine by attempting to execute each of its executors (if none work, the effect is global). The agent then instantiates a *Stretch-IPT* $\tilde{\mathcal{T}}'$ (lines 17 and 20), fed to Alg. 2 *Adapt* (line 21), along with the failed sub-state and the operator's expected effects. A successfully learned executor x is added to the failed operator's $\mathcal{X}_i$.

Alg. 2 *Adapt* initiates both I and β from respectively the grounded predicates of $\widetilde{s_f}$, the failed sub-symbolic state, and ω_{o_i}, the expected effects of o_i (lines 1–2). The off-policy RL algorithm (in this work SAC or PPO) is initialized using the given hyper-parameters (line 3) for training (lines 5–9) on the task problem and returning the executor on success and *failure* otherwise. Note that the agent uses the symbolic planner in two ways: to build the trajectory to reach the global task goal (i.e., Alg. 1, line 1) and repeatedly in the RL learning process (i.e., in ***Train***, Alg. 2, line 6) whenever the agent reaches a sub-symbolic state $\widetilde{s} \in \widetilde{\mathcal{S}}_g$[1] to verify whether $d(\widetilde{s})$ is a plannable state[2]. For the latter, the planner is called to ensure that a path exists from the state reachable by the policy to the task goal. Hence, plannable states are part of the MDP's (or SMDP) termination conditions, which is why ***Train*** receives the stretch IPT, including T', the domain required to compute the plannable states. Once the agent learns the new executor x_{new}, the agent executes it (Alg. 1, line 24) to resolve the impasse. If the agent fails to learn a solution executor, the framework returns *failure*.

Executor Selection Strategy. Whenever the agent selects an operator, it chooses between mapped executors by first checking the availability of each executor (using I) and then prioritizes them based on how well they fit the current environment. If no executor is available, no I_x, $x \in \mathcal{X}_i$ matches the current symbolic observation, or all available

[1] β is used to check whether the reached sub-symbolic state is in $\widetilde{\mathcal{S}}_g$.

[2] We do not call the planner at every time step while learning, but only when the agent reaches a state in $\widetilde{\mathcal{S}}_g$.

Algorithm 1 ***Plan&Execute*** $(\mathcal{T})$

Require: $\mathcal{T} = \langle T, \mathcal{M}, d, e \rangle$ ▷ Integrated Planning Task
Require: $T = \langle \mathcal{E}, \mathcal{F}, \mathcal{O}, s_o, s_g \rangle$
$\mathcal{P}$ = **Plan**($\mathcal{T}$, d($\widetilde{s}$)) ▷ $\mathcal{P} = \langle o_1, o_2, ..., o_{|\mathcal{P}|} \rangle$
if $\mathcal{P}$={} **then return** *failure* ▷ Abort
for $o_i \in \mathcal{P}$ **do**
  $\mathcal{X}_i \leftarrow e(o_i)$
  for x in ***Prioritized***$(\mathcal{X}_i)$ **do** ▷ *Selection Strategy* Priorization
    $s \leftarrow d(\widetilde{s})$
    Success = **Execute**(x)
    if *Success* **then**
      if $s \notin I_x$ **then**
        $I_x \leftarrow I_x \cup s$ ▷ Add the initiation substate to I_x
      break for
  if ¬*Success* **then** ▷ If all failed, proceed to *Recovery*
    $\widetilde{s_f} \leftarrow$ *failure sub-symbolic state*
    Novelty-type $\leftarrow$ ***Evaluate failure()*** ▷ Test in the environment
    if *Novelty-type* is *Local* **then**
      Build Skill-level SMDP $\mathcal{M}^s$
      $\tilde{\mathcal{T}} = \langle T, \mathcal{M}^s, d, e \rangle$ ▷ Stretch Skill-IPT
    else if *Novelty-type* is *Global*
      Build Control-level MDP $\mathcal{M}^c$
      $\tilde{\mathcal{T}} = \langle T, \mathcal{M}^c, d, e \rangle$ ▷ Stretch Control-IPT
    $x_{new} \leftarrow$ ***Adapt***$(\tilde{\mathcal{T}}, \omega_{o_i}, \widetilde{s_f})$
    if x_{new} **is *failure* then return** *failure* ▷ Abort
    $\mathcal{X}_i \leftarrow \mathcal{X}_i \cup \{x_{new}\}$
    Execute(x_{new}) ▷ Else, Execute Learned Executor

executors already failed, the agent still tries the remaining executors in $\mathcal{X}_i$. If one of them is successful, the agent will add the failure state to its initiation set so that it can be directly selected on another trial. If none worked, the agent enters *recovery mode*.

Learning a Control Executor with RL. If the novelty is *global*, the agent must learn new controls for each operator in the control-level MDP M^c, as done by classical Hybrid approaches. In that case, the novelty affects the entirety of the environment; all states are considered failing ($\widetilde{s_f}$) and the initial state is sampled randomly from all sub-states that map—through d—onto every known symbolic state. These states are all added to the I set of each new executor, and their termination condition β is individually computed by utilizing the planner to predict the expected effects of the operator when starting from $d(\widetilde{s_f})$.

Learning a Skill Executor with HRL. If the novelty is *local*, only one operator at a time might fail. In such instances, it is conceivable that a solution to overcome the novelty already exists within the repertoire of known operators. However, the agent might not be aware of this solution because the symbolic representation is not detailed enough or/and because the detector function, which describes the low level observations in this symbolic representation, might be incapable of extracting the relevant information from sensors. The idea is then to learn this solution from experience using a HRL protocol in the discrete-time skill-level Semi-MDP (SMDP) M^s. The difference compared to a control-level MDP, aside from the discrete n-steps execution, lies in the action space of the skill-SMDP which consists of all known operators. Upon selection of an operator in the skill-level SMDP, the agent will search for the set of mapped executors, and will afterwards leverage the selection strategy

Algorithm 2 ***Adapt*** $(\tilde{\mathcal{T}}, \omega_{o_i}, \widetilde{s_f}) \rightarrow x_{new}$

Require: an off-policy RL algorithm **A**
Require: hyper-parameters H
Require: N_{eps} ▷ Number of episodes
Require: η ▷ The success rate evaluation threshold
Require: $\tilde{\mathcal{T}}$ ▷ For finding plannable states
Require: ω_{o_i} ▷ For episodes termination
Require: $\widetilde{s_f}$ ▷ For episodes reset
I: initiation set ▷ Initialized with $d(\widetilde{s_f})$
$\beta(\tilde{s})$: termination indicator ▷ Computed from ω_{o_i}
initialize **A** using H
initialize π_x^{new} ▷ Transfer from existing policy: *Selection Strategy*
for N_{eps} *episodes* **do**
  $\pi_x^{new} \leftarrow$ ***Train***$(\pi_x^{new}, \tilde{\mathcal{T}}, \widetilde{s_f}, \beta)$
  if *success*$(\pi_x^{new}, \tilde{\mathcal{T}}) > \eta$ **then**
    $x_{new} \leftarrow \langle I_x, \pi_x^{new}, \beta_x \rangle$ **return** x_{new} ▷ Executor Discovered
return *failure*

to queue and execute the executors in this set for n steps (we set n to 10). Following such protocol, a skill executor invokes either a control executor or an other nested skill executors.

This nested structure has two important advantages when it comes to learning: for one, the action space of operators is often significantly smaller than that of executors. And second, while learning a new skill executor for a given operator, the agent can possibly execute another skill that was already trained for this operator, yet simply in different conditions. This latter system provides the agent with the ability to leverage analogies between scenarios, which in certain circumstances would act as a guidance to learn faster. For instance, consider a car that has learned a skill for bypassing an *obstacle* on the road using control level executors such as change_lane_left and right. Our method not only transfers weights of this skill policy to training a new one, e.g., driving with *obstacle & black ice*, but it is also able to invoke that skill executor (as an action) during training to guide the agent into choosing what control fits best given the observation.

V. Experiments & Results

A. Experimental Setup

The experiments were conducted in the CARLA [10] driving simulation.[3] We evaluated three agents on a pre-defined *navigation task* measuring navigation performance from one fixed departure location to 20 fixed arrival locations, comparing our approach to the hybrid RapidLearn [7] and a Soft Actor-Critic [22] (SAC, implementation provided by [23]) reinforcement learning agent. We chose SAC as empirical observations have indicated some robustness of maximum entropy algorithms (like SAC) to environmental disturbances [24], [25]. We used SAC to learn control policies and PPO to learn skill policies[4]. We averaged our results over five different random seeds.

The two hybrid agents use the symbolic metricFF [26] planner with initial symbolic domain knowledge in PDDL

[3]The code is available at `https://drive.google.com/file/d/1P8hu3bDMuwh9Od0aJtDnA16_tYeYgeNQ/view?usp=sharing`

[4]PPO was chosen for convenience (no discrete SAC exists in [23] yet).

Fig. 3: **Top:** Agent Performance evaluated by success rate on novelty injection before training: *higher values indicate better performance*. **Middle:** Agent Performance after training: *higher values indicate better performance*. **Bottom:** Training duration per novelty for each agent evaluated by the number of training steps: *lower values indicate faster training*.

[27], [28] which includes essential operators (*follow_lane*, *turn_right*, *turn_left*), *change_lane_left* and *change_lane_right* for comprehensive navigation, using a designated detector function d[5]. A function e maps existing operators to RL-learned executors that were trained in the initial domain. The RL learners' observation space is based on a 2D LiDAR-like sensor, emitting beams at $\frac{75}{360}$°-angles, supplemented by additional sensors providing driving-related information such as velocity, throttle, steering angle, and road curvature.

All agents were trained on the navigation task, with the RL baseline training for 14M steps and the Hybrid agents for 1M steps/operator (5M total) allowing them to achieve a similar success rate on the *navigation task* of around 85%.[6] The RL agent initially underperformed compared to Hybrid approaches, despite concerted efforts to enhance its policy. However, the primary objective of these experiments is to assess each agent's capability to maintain initial performance levels despite encountering novelties.

Training comprised 2M steps for *global* and 700k for *local* novelties, set empirically for fair comparison. A reward threshold, determined empirically for all methods, halted training upon convergence. All agents were subjected to a similar reward function based on road offset throughout the experiment, from initial training to scenario completion.

We used ***six*** sequential scenarios with either a single or multiple novelties in the following order: (1) ***Deflated Tire:*** One of the tires are deflated and has no friction with the road. (2) ***Black Ice:*** The road becomes very slippery, and friction on all wheels is reduced. (3) ***Deflated Tire & Rain:*** Besides the tire deflating, rain makes the road slightly more slippery. (4) ***Obstacle:*** Static cars appear on some portion of the road. They block some lanes making the follow_lane operator fail. (5) ***Obstacle & Black Ice:*** Besides static cars, black ice makes the road slippery. Maneuvering the car around obstacles becomes more difficult. (6) ***Obstacle & Mist:*** Mist reduces the LiDAR's maximum detection distance making obstacle detection more difficult. Note that the first three are global novelties, they require the agent to learn new control strategies, while the last three are local novelties. Our hierarchical approach capitalizes on prior controls and skills to adapt to local novelties which do not necessarily require to learn new controls from scratch. In scenarios where only global novelties occur, our agent would perform similarly as a typical Hybrid agent would, as it would not be able to utilize this prior knowledge.

To test the agents' ability to adapt to novelties, we evaluated them *at novelty injection* to measure how much they were impacted by the change, and *after training* on each novelty to measure their accommodation level.

B. Results

Fig. 3 displays all agents' performance. The x-axis shows novelty injection-training pairs, and the y-axis shows navigation success rate. Throughout the sequential injection, the baseline RL agent shows limited flexibility and struggles to adapt to new conditions. Both hybrid agents demonstrate good adaptability across diverse conditions due to two benefits of the hybrid approach: it employs operators to learn specific policies (for specific circumstances) instead of adapting one global policy, and when facing a local novelty, it can reduce the search space and facilitate adaptation to the affected operators. Hybrid approaches consistently

[5]Example in CARLA, d outputs grounded predicates such as $at(car, l_1)$ for sub-symbolic states within a square centered at location l_1.

[6]For the baseline RL agent to learn the navigation task, it must acquire knowledge of the map model corresponding to the PDDL transitions. To assist the baseline agent in effectively learning the high-level task with a minimally altered reward, we introduced additional termination conditions, concluding an episode if the car deviated from the intended path, therefore encoding navigational information via episode length. We leveraged the integrated CARLA navigator system to construct the trajectory from the car's position to a target location. It is important to note that the generated trajectory itself remained hidden to the agent during training.

improve their performance on each introduced novelty, often matching their performance in the initial environment while the baseline RL agent shows catastrophic forgetting. The results also indicate that hybrid approaches exhibit robustness to changes without additional training. For novelties with minimal impact on the agent's performance, such as *Deflated Tire & Rain*, hybrid agents maintain stable performance (90% before, 85% after injection), obviating the need for retraining. This ability to reuse trained executors contributes to the agents' robustness to novel conditions.

While both hybrid agents have similar performance on global novelties, because new low-level control laws need to be learned and prior knowledge cannot be utilized, the performance difference can be seen with local novelties, where our agent significantly outperforms RapidLearn, both in terms of performance and in terms of training time. An exception appears for *Obstacle & Mist* novelty, where RapidLearn slightly outperforms our approach after training. Such exception shows a limitation of learning a skill from existing controls, while refined controls could capture better the mechanism of a novelty leading to improved performance on it. Nevertheless, our method post-training performance sees an overall remarkable boost of nearly 30% when compared to traditional methods like RapidLearn, highlighting the effectiveness of our approach. Additionally, our approach substantially accelerates the training process, reducing the training time by more than tenfold (around 14 times). Even when accounting for the execution of ten environment steps for every HRL training step, it remains a more computationally efficient approach, as execution steps are considerably less computationally intensive than combined execution and training steps.

Finally, our approach also greatly enhances robustness, particularly in scenarios that involve a combination of previously encountered local and global novelties. Notably, our agent achieves 58% success rate when the novelty *Obstacle & Black Ice* is injected on zero-shot learning, i.e., without any training. On such novelty, our agent robustness completely outperforms RapidLearn by a factor of three and RL by a factor of 20. This heightened robustness can be attributed to HRL's capacity of combining skills, such as avoiding obstacles, with lower-level policies, such as ice control policies, when needed.

VI. Discussion

Our results are encouraging as they point to the utility of hierarchical RL as a way to learn behaviors that do not require the acquisition of new low-level control policies. With enough low-level policies in place, the agent will be able to handle a wide variety of novelties better and faster than other approaches. And importantly, our approach intrinsically addresses a shortcoming of planning-based approaches that downward refinability does not always hold in robotic domains, i.e., that details of the underlying continuous state may have important effects on the entirely plan.

However, our approach comes with limitations. It assumes a function to categorize novelties (as local or global), which is an ongoing challenge in open-world adaptation. We simplify by empirically classifying novelties based on their effects on the environment; if multiple operators fail, it is classified as global. This binary classification may not reflect real-world scenarios, impacting our agent adaptation strategy. Additionally, our method does not enhance training efficiency on global novelties compared to RapidLearn-like methods, lacking mechanisms for faster control learning beyond transfer learning. In essence, if a global novelty affects all operators, a new executor version must be learned for each. While this approach surpasses pure RL in such scenarios, there remains significant room for improvement.

Additionally, there are still important challenges to be addressed before the proposed system could be deployed in real-world settings, the most important of which is one closed-world assumption shared by most RL algorithms, i.e., for an RL algorithm to learn a new policy quickly without requiring the physical platform to perform all the actions in the real world, it needs a simulation environment with fidelity sufficient for simulation-trained policies to mostly work on the robot. However, in open-world settings the robot cannot assume that its simulator would reflect all aspects of the real world, otherwise there would be no novelties. But then it is of limited use for training. To address this challenge, the robot would need a way to model at least parts of the novelty to be able to learn a partial policy in simulation, and then continue training the policy in the real world with hopefully only a few trials (which our hierarchical hybrid approach would enable). Whether this strategy can work, will depend on the extent to which the robot can quickly *characterize* the novelty and use the characterization to add it to its simulation. While novelty characterization was outside the scope of this paper, it would also be helpful for the problem of assessing whether the novelty has only local or global effects, relieving the robot from trying out all of its executors to determine whether a new control strategy might be needed. Finally, it would also help with the selection of executors by using its characterization to find the best fitting one (rather than having to determine for each of them their predicted outcomes of the actions they would take in the given state, selecting one action at random, say, and then determining which policies was closest to the real outcome, repeating the process until the best executor is found).

VII. Conclusion

We introduced a hybrid hierarchical reinforcement learning and symbolic planning framework that significantly improves open-world adaptability of robots compared to state-of-the-art hybrid and reinforcement learning approaches as we demonstrated with our extensive evaluation: robots can acquire and refine skills in structured and leverage previously learned skills to enhance their performance in new and more complex tasks, resulting superior performance and faster accommodation. The framework also enables more efficient skill interplay, which improves the robustness to variations experienced changes and can even handle unexperienced novelties without additional training in some cases.

REFERENCES

[1] F. Muhammad, V. Sarathy, G. Tatiya, S. Goel, S. Gyawali, M. Guaman, J. Sinapov, and M. Scheutz, "A novelty-centric agent architecture for changing worlds," in *Proceedings of 20th International Conference on Autonomous Agents and Multiagent Systems*, 2021.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[3] X. Qu, Z. Sun, Y. S. Ong, A. Gupta, and P. Wei, "Minimalistic attacks: How little it takes to fool deep reinforcement learning policies," *IEEE Transactions on Cognitive and Developmental Systems*, 2020.

[4] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253519307377

[5] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[6] A. Chen, A. Sharma, S. Levine, and C. Finn, "You only live once: Single-life reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 784–14 797, 2022.

[7] S. Goel, Y. Shukla, V. Sarathy, M. Scheutz, and J. Sinapov, "Rapid-learn: A framework for learning to recover for handling novelties in open-world environments," in *IEEE International Conference on Development and Learning (ICDL), London, UK, September 12-15, 2022*. IEEE, 2022, pp. 1–8. [Online]. Available: https://arxiv.org/pdf/2206.12493

[8] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, and M. Scheutz, "Spotter: Extending symbolic planning operators through targeted reinforcement learning," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2021, pp. 1118–1126. [Online]. Available: https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p1118.pdf

[9] P. Lorang, S. Goel, P. Zips, J. Sinapov, and M. Scheutz, "Speeding-up continual learning through information gains in novel experiences," in *4th Planning and Reinforcement Learning (PRL) Workshop at IJCAI-2022*, 2022.

[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[11] P. W. at ICAPS 2022, "Bridging the gap between ai planning and reinforcement learning (prl @ icaps)," 2022. [Online]. Available: https://prl-theworkshop.github.io/prl2022-icaps/

[12] P. W. at IJCAI 2022, "Bridging the gap between ai planning and reinforcement learning (prl @ ijcai)," 2022. [Online]. Available: https://prl-theworkshop.github.io/prl2022-ijcai/

[13] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[14] F. Yang, D. Lyu, B. Liu, and S. Gustafson, "Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making," 07 2018, pp. 4860–4866.

[15] J. Balloch, Z. Lin, R. Wright, X. Peng, M. Hussain, A. Srinivas, J. Kim, and M. O. Riedl, "Neuro-symbolic world models for adapting to open world novelty," 2023.

[16] S. Doncieux, N. Bredeche, L. L. Goff, B. Girard, A. Coninx, O. Sigaud, M. Khamassi, N. Díaz-Rodríguez, D. Filliat, T. Hospedales, *et al.*, "Dream architecture: a developmental approach to open-ended learning in robotics," *arXiv preprint arXiv:2005.06223*, 2020.

[17] P. R. Vieira, P. D. Félix, and L. Macedo, "Open-world active learning with stacking ensemble for self-driving cars," *arXiv preprint arXiv:2109.06628*, 2021.

[18] K. Joseph, S. Khan, F. S. Khan, and V. N. Balasubramanian, "Towards open world object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5830–5840.

[19] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370299000521

[20] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[23] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[24] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," 2019.

[25] S. Huang, H. Su, J. Zhu, and T. Chen, "Svqn: Sequential variational soft q-learning networks," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1xPh2VtPB

[26] J. Hoffmann, "The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables," *Journal of Artificial Intelligence Research*, vol. 20, pp. 291–341, 2003.

[27] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL–the Planning Domain Definition Language," 1998.

[28] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL–the Planning Domain Definition Language," 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212