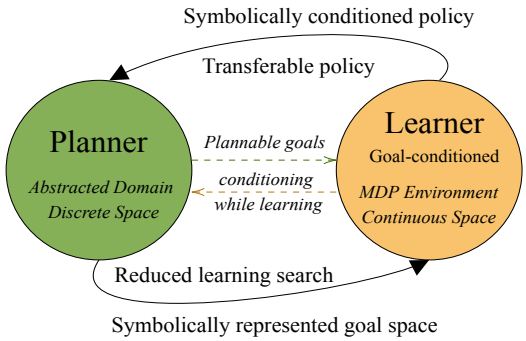
2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

A Framework for Neurosymbolic Goal-Conditioned Continual Learning in Open World Environments

Pierrick Lorang[1,2], Shivam Goel[1], Yash Shukla[1], Patrik Zips[2], and Matthias Scheutz[1]

***Abstract*— In dynamic open-world environments, agents continually face new challenges due to sudden and unpredictable novelties, hindering Task and Motion Planning (TAMP) in autonomous systems. We introduce a novel TAMP architecture that integrates symbolic planning with reinforcement learning to enable autonomous adaptation in such environments, operating without human guidance. Our approach employs symbolic goal representation within a goal-oriented learning framework, coupled with planner-guided goal identification, effectively managing abrupt changes where traditional reinforcement learning, re-planning, and hybrid methods fall short. Through sequential novelty injections in our experiments, we assess our method's adaptability to continual learning scenarios. Extensive simulations conducted in a robotics domain corroborate the superiority of our approach, demonstrating faster convergence to higher performance compared to traditional methods. The success of our framework in navigating diverse novelty scenarios within a continuous domain underscores its potential for critical real-world applications.**



Hybrid Planning and Goal-oriented Learning

Fig. 1. Our method leverages symbolically conditioned learning.

I. Introduction

Recent advancements in Task and Motion Planning (TAMP) have significantly improved the integration of high-level task reasoning with low-level motion planning in robotics. These methods enable solutions to complex, long-horizon tasks by assuming access to task structures and environmental configurations, typically represented through planning domains like PDDL [1], coupled with low-level controllers like motion planners. The process results in an ordered list of operators from a task plan, which the robot executes using its knowledge base.

Despite these advancements, challenges remain, particularly in scenarios where obtaining comprehensive and accurate high-level information demands substantial engineering efforts. This limitation often leads to suboptimal task performance, especially if the available information is flawed or incomplete. To address this, recent studies have combined TAMP with Reinforcement Learning (RL), allowing the system to learn policies for low-level controllers, thereby reducing reliance on hard-to-engineer information [2].

However, a notable limitation of these combined approaches is their assumption of complete access to task distributions. They struggle in 'open-world' scenarios characterized by sudden, unpredictable changes in environmental dynamics, task goals, or the agent's capabilities. In these 'open-world' situations, agents encounter 'novelties' that challenge their existing world knowledge and decision-making strategies. This inadequacy of symbolic knowledge in novel scenarios renders traditional purely symbolic approaches [3] ineffective. To overcome these challenges, agents must be capable of adapting to sudden changes, necessitating exploration and learning to acquire the skills necessary for addressing new tasks. Recent work [4] addresses novelty accommodation by employing a hybrid TAMP and RL approach. However, this method is highly sample-inefficient, requiring the agent to learn a new RL policy from scratch for each encountered novelty, even for similar situations. As an example, consider a robot that does not have operators to perform the novel *Pick(Apple)* and *Pick(Orange)* actions. Such an approach will learn two separate RL policies for these novelties even though one policy could intuitively be reused to learn the second. Our approach introduces a multi-goal continual learning framework to overcome this inefficiency, see Fig.1. By leveraging a symbolic goal-oriented RL protocol with transfer learning we abstract and reuse previously learned knowledge to efficiently handle multiple novelties in continual learning scenarios, ensuring robust recovery in dynamic environments. Our contributions are:

(1) Our methodology augments a hybrid planning and learning architecture with a novel symbolic goal-oriented learning mechanism. In the face of novelties, our approach activates an exploration protocol that adeptly navigates the continuous environment using symbolic representations of goals, establishing neural connections between planning and learning. (2) To improve sample efficiency, we utilize the planner inside the learning protocol to discover symbolic shortcuts as new goals. We also employ transfer learning to efficiently reuse previously acquired policy knowledge.

We conducted evaluations within a continuous robotics

[1]Tufts University, 419 Boston Ave, Medford, 02155, 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

domain, using the Mujoco simulator. Our analysis included comparisons with both traditional RL algorithms and advanced hybrid agents, especially in scenarios requiring ongoing adaptation to novel situations. The results provide strong evidence of the superiority of our approach, demonstrating notable advancements over existing methods in handling dynamic and unpredictable environments.

II. Related Work

Many hybrid approaches for TAMP problems have demonstrated leveraging high-level planner representations alongside low-level policy-based learning approaches like RL [5], [6], [7], [8], [9], [10]. These approaches often rely on planning systems built under closed-world assumptions, rendering them ill-equipped for unforeseen situations in open-world settings.

Developing integrated architectures for open-world novelty accommodation is fairly recent [11], [12], [13]. Prior research in "life-long learning" with deep RL variants faced catastrophic forgetting and adaptability challenges in abrupt changes [14]. "Meta Experience Replay" [15] and "powerplay" [16] required extensive retraining. In contrast, RL-based learning of planning operators as higher-level abstractions simplifies reuse and storage, addressing catastrophic forgetting as RL policies are retained post-training [17]. This approach, however, has limitations in discrete environments and single novelty adaptations. While hybrid methods help in recovery, they encounter issues with data inefficiency and extended learning times, especially in continuous spaces.

Recovering from planning failures using Reinforcement Learning (RL) remains challenging due to expensive data requirements, while novelty handling demands responsiveness with limited data. Recent research on accommodating dynamic environments has focused on scenarios where agents encounter informed, gradual, or plan-level changes [2], [18], [19], [20], [21]. Some works prioritize solving tasks within a single trial, sacrificing robustness [22], while others explore leveraging human guidance [4]. Previous works have delved into goal-oriented RL within hybrid frameworks [23], [24] to help the data efficiency, yet none have fully exploited the integration of symbolic representation and reasoning as a goal selector to expedite exploration. *Despite advancements, enhancing learning efficiency in uninformed and abrupt open-world settings remains an ongoing challenge*.

III. Preliminaries

A. Symbolic Planning

Symbolic planning typically builds upon a domain rendered in a formal language such as PDDL [1]. Let $\sigma = \langle \mathcal{E}, \mathcal{F}, \mathcal{S}, \mathcal{O} \rangle$ be a domain description, where $\mathcal{E} = \{\varepsilon_1, \ldots, \varepsilon_{|\mathcal{E}|}\}$ is a set of entities within the environment. $\mathcal{F} = \{f_1(\odot), \ldots, f_{|\mathcal{F}|}(\odot)\}, \odot \subset \mathcal{E}$ is a set of boolean or numerical predicates. $\mathcal{S} = \{s_1 \ldots, s_{|\mathcal{S}|}\}$ is the set of symbolic states in the environment. $\mathcal{O}$ represents the set of known action operators, defined as $\mathcal{O} = \{o_1, \ldots, o_{|\mathcal{O}|}\}$. Each operator o_i is characterized by a set of preconditions and effects denoted $\psi_i, \omega_i \in \mathcal{F}$. The preconditions ψ_i and effects ω_i of o_i represents the predicates that must hold (or must not hold) before and after executing o_i, respectively. A planning task is typically described as a STRIPS task which we denote as $T = (\mathcal{E}, \mathcal{F}, \mathcal{O}, s_0, s_g)$, where $s_0 \subset \mathcal{S}$ represents the set of initial states and $s_g \subset \mathcal{S}$ represents the set of desired goal states. The solution to this planning task T is presented as an ordered sequence of operators, denoted as plan $\mathcal{P} = [o_1, \ldots o_{|\mathcal{P}|}]$.

B. Goal-oriented Reinforcement Learning (RL)

We formalize the environment in which the agent acts as a Markov Decision Process (MDP) $M = \langle \widetilde{\mathcal{S}}, \mathcal{A}, R, \tau, \gamma \rangle$, where $\widetilde{\mathcal{S}}$ represents sub-symbolic states, $\mathcal{A}$ actions, $R : \widetilde{\mathcal{S}} \times \mathcal{A} \to \mathbb{R}$ the reward function, τ the probability transition function $\tau(\tilde{s}_{t+1}|\tilde{s}_t, a_t)$, and $\gamma \in (0, 1]$ the discount factor. A policy for M is defined as a probability distribution $\pi_M(a|\tilde{s})$ that an agent chooses action a in state $\tilde{s}$. We abbreviate π_M as π in this paper. An RL algorithm aims to find a policy that maximizes the expected sum of discounted rewards, $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, over time t. The value of each state-action pair under a policy π is defined as the expected return when starting in $\tilde{s}$ and following π afterwards, $Q^\pi(\tilde{s}, a) = E_\pi[G_t | \widetilde{S}_t = \tilde{s}; A_t = a]$. An optimal policy is defined as any policy π^* so that $Q^{\pi^*}(\tilde{s}, a) \geq Q^*(\tilde{s}, a)$, the optimal value function, for every possible state, action, and policy. The value of each state-action pair under policy π is denoted by $Q^\pi(\tilde{s}, a)$, representing the expected return starting in state $\tilde{s}$ and following policy π. $Q^*(\tilde{s}, a)$ denotes the maximum expected return achievable for any policy from state $\tilde{s}$ and action a. An optimal policy π^* ensures $Q^{\pi^*}(\tilde{s}, a) \geq Q^*(\tilde{s}, a)$ for all states and actions.

Schaul et al. [25] extend the RL framework with Unified Value Function Approximation (UVFA), that generalizes over both states $\widetilde{\mathcal{S}}$ and goals $\mathcal{G}$. UVFA enables representing a set of RL value functions with one unified function approximation. In UVFA, the agent acts upon the current state and the goal $g \in G$, defining the policy as $\pi(a|\tilde{s}, g)$. The resulting reward is also goal-dependent $r_t = r(\tilde{s}_t, a_t, g)$ and the corresponding Q-function involves a goal in addition to the state-action pair $Q^\pi(\tilde{s}_t, a_t, g) = E[R_t | \tilde{s}_t, a_t, g]$. Experiments in the LavaWorld environment [26] demonstrated UVFA's capability to generalize across various state-goal pairs, given adequate training data. UVFA's flexibility allows for its integration with off-policy algorithms such as Deep Q-Networks (DQNs) [27] and Soft Actor-Critic (SAC) [28].

In addition, UVFA has been combined with Hindsight Experience Replay (HER) [29], for sample-efficient learning from sparse and binary rewards. HER alters the RL sampling process by replaying experiences with various arbitrary goals, enabling the agent to derive learning values from sub-optimal outcomes. Specifically, for each transition $(\tilde{s}_t \to \tilde{s}_{t+1})$ within an experienced episode $(\tilde{s}_0, \tilde{s}_1, ..., \tilde{s}_T)$, HER stores the original goal from the episode and a selection of goals from prior episodes. This method of "experience replay" with an arbitrary goal, conducted via an off-policy RL algorithm, allows HER to imagine trajectories with

different goals, thereby enabling the learning of multiple task variations from a single experience.

C. Hybrid Plan & Learn for Novelty Accommodation

In open-world scenarios, agents may face novelties that create impasses in task completion T. Sarathy et al. [17] addressed this by proposing an Integrated Planning Task (IPT) framework, which integrates planning and learning. An IPT $\mathcal{T} = \langle T, M, d, e \rangle$ is comprised of a STRIPS task T, an MDP M, a detection function $d : \widetilde{\mathcal{S}} \rightarrow \mathcal{S}$ (mapping sub-symbolic states $\widetilde{\mathcal{S}}$ in M to high-level planning states $\mathcal{S}$ in σ, where d is surjective), and an execution function $e : \mathcal{O} \rightarrow X_M$ (mapping each planning operator $o_i \in \mathcal{O}$ to an executor in MDP). An executor $x = \langle I, \pi, \beta \rangle$ consists of an initiation indicator $I(\tilde{s})$, a policy $\pi(\tilde{s}) \rightarrow \mathcal{A}$, and a termination indicator $\beta(\tilde{s})$. This executor framework, an extension of the options framework by Sutton et al. [30], allows inferring initiation and termination conditions from the agent's observations or reasoning. A solution to an IPT $\mathcal{T}$ is an executor, and a planning solution is a plan $\mathcal{P} = \left[o_1, \ldots o_{|\mathcal{P}|}\right]$. If all operators are accurate, executing the corresponding executors $e(o_1), ..., e(o_{|\mathcal{P}|})$ in MDP M in sequence results in a final state $\tilde{s} \subseteq s_g$. An IPT $\mathcal{T}$ is *solvable* if a solution exists and *plannable* if a planning solution exists. A *plannable state* is a state in which following the plan $\mathcal{P}$ can achieve the task goal s_g.

IV. Method

A. Problem Statement

In our work, a novelty is a new encounter that impacts the known dynamics τ, or the state representation $\widetilde{\mathcal{S}}$ of the environment M, or does not belong to the agent's domain knowledge σ. A novelty is detected via an operator in the plan that, at execution time, cannot be executed or does not produce the expected effects and fails; we call this a failed operator. We focus on novelties that obstruct action execution and where no symbolic solution can be inferred.

Such a challenge is called the executor discovery problem. When an agent is deemed unable to successfully infer a solution to a task T because of a novelty, it needs to discover a recovery executor x capable of supplying the missing information. To solve this problem, the agent must generate on-the-fly a "Stretch-IPT" denoted as $\tilde{\mathcal{T}} = \langle T, M', d, e' \rangle$, where $M' = \langle \widetilde{\mathcal{S}}', \mathcal{A}, R, \tau', \gamma \rangle$, $\widetilde{\mathcal{S}}'$ being the new environment's state space and τ' the new transition function, both possibly impacted by the novelty occurrence.

An executor $x_{recovery}$ solution to a particular Stretch-IPT either captures a new path in the subjacent MDP to achieve a known operator transition (i.e., for an operator o_f that failed, $e'(o_f) = x_{recovery}$), or captures a completely new operator ($o_{new} \notin \mathcal{O}, e'(o_{new}) = x_{recovery}$) which provides a functioning symbolic path in σ to the task goal. $\tilde{\mathcal{T}}$ is then said to be *Solvable* if a path (symbolic and continuous) exists from s_0 to s_g. Our aim in this paper is to propose an approach capable of benefiting from previous recovery executors to increase the agent's adaptability over time on continual and uninformed learning scenarios.

B. Methodology Overview

Our proposed methodology builds upon a hybrid planning and learning agent architecture that utilizes domain knowledge to generate plans for tasks. When an impasse occurs due to environmental novelty, the agent creates a Stretch-IPT on-the-fly for executor discovery. In our work, executor discovery integrates goal-conditioned learning, where executors are conditioned on symbolic goals, allowing the agent to generalize strategies across various task instances. The agent utilizes the planner to continually discover relevant goals to condition the policies, see Fig 2 *right*, and search for shortcuts see Fig 3, facilitating adaptation to novelty.

C. Hybrid Architecture Extension

A hybrid architecture typically consists of a dual-layer design: a symbolic planning layer and a continuous execution layer. In the symbolic layer, a planner is utilized to generate a plan starting from s_0 to the task goal state s_g using the domain knowledge (described in PDDL). In the execution layer, the executors mapped to each operator in the plan get executed in the MDP environment one operator at a time.

We extend the hybrid planning and learning architecture to accommodate continual learning scenarios by associating multiple executors with the same operator. Formally, we define this mapping as $e(o) = \mathcal{X}_o = \{x_1, x_2, ...\}$, see Fig 2 *left*. For each novelty that leads to an operator failure (o_f), we add a recovery executor to the set $\mathcal{X}_f$ mapped to o_f through the executor function $e : \mathcal{O} \rightarrow \mathcal{X}$. In particular, the agent executes each operator o_i in the plan, gathering $\mathcal{X}_i = e(o_i)$ with chronological ordering delineated. Executors in $\mathcal{X}_i$ are prioritized based on the indicator function, which outputs 1 for executors deemed suitable for the current state. The success of an executor is verified by confirming whether the agent reaches a state satisfying the expected effects of the corresponding operator o_i.

The execution protocol is then detailed in Alg. 1. The agent iterates over the plan until a novelty in the environment, modifying M to M', causes an execution impasse[1], and in turn causes an operator o_f to fail. An execution impasse means that either all executors in $\mathcal{X}_f$ failed. i.e., the operator failed, or that $|\mathcal{X}_f| = \emptyset$, thus no symbolic solution can be inferred from the current agent knowledge. To solve such an impasse, we instantiate a *Stretch-IPT* $\tilde{\mathcal{T}}$ based on M' to discover a recovery executor ($x_{recovery}$). We set the reset condition for the executor discovery based on the I indicator derived from s_f, the state where o_f failed (Eq. 1), and initialize the termination condition using the β indicator computed from ω_{o_f} (Eq. 2 with $\mathcal{G}_p$ initialized to $\{\omega_{o_f}\}$). Upon successfully learning, the agent either maps $x_{recovery}$ to the failed operator (adding it to $\mathcal{X}_f$) or abstracts a new operator and executes it to overcome the impasse caused by the novelty. In the event of failure to learn a recovery executor, the framework returns *false*.

[1]It is beyond the scope of this work to discern whether a failure stems from the introduction of novel elements or due to the agent's shortcomings.

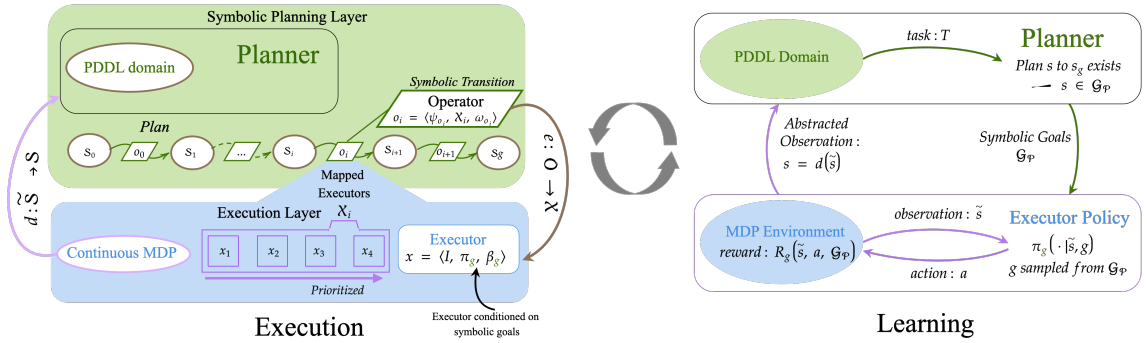


Fig. 2. Overview diagram of the framework. Dual layers (planning & execution) with different space representations. Typical Hybrid approaches achieve the dual layer execution and restrict the search space for learning using the symbolic domain. In addition, our framework enables multiple executors mapping per operator and establishes learning connections between symbolic and continuous layers.

D. Executor Discovery using Goal-conditioned Learning

In goal-oriented RL, a goal space $\mathcal{G} \subseteq \widetilde{\mathcal{S}}$ is commonly considered. In our approach, we adopt symbolic representations for goal spaces in RL, i.e., $\mathcal{G} \subseteq \mathcal{S}$, enabling broader task perspectives beyond sensor-level data. By conditioning executors on symbolic propositions, the agent gains insight on the higher symbolic structure of the continuous environment, which facilitates faster executor discovery. Because $\mathcal{S}$ is an abstraction of $\widetilde{\mathcal{S}}$, we still have $\mathcal{G} \subseteq \widetilde{\mathcal{S}}$. Any goal $g \in \mathcal{G}$ can thus be expressed as a set of grounded predicates ($g = \{f_1(\odot), f_2(\odot), \ldots\}$) and the transcription from $\widetilde{\mathcal{S}}$ to $\mathcal{G}$ is also facilitated by the detection function d.

We expand the definition of the executor $x \in \mathcal{X}$ to incorporate the goal space in the policy definition: $\pi_g : \widetilde{\mathcal{S}} \times \mathcal{G} \rightarrow \mathcal{A}$, as well as in the termination indicator $\beta_g(\tilde{s})$. Notably, such executor is symbolically conditioned, i.e., $\pi_g : \widetilde{\mathcal{S}} \times \mathcal{S} \rightarrow \mathcal{A}$, because $\mathcal{G} \subseteq \mathcal{S}$. We refer to such executors as "symbolic goal-conditioned" executors, denoted as $x = \langle I, \pi_g, \beta_g \rangle$. Specifically, for a new MDP M', we initialize an executor x' with an initiation indicator I' computed from the current operator's failed state, a symbolic goal-conditioned policy $\pi_{g'}$ and a termination indicator $\beta_{g'}$ initially computed from the grounded predicates of the failed operator's expected effects.

When executing an executor mapped to the operator o_i, we set the goal $g = \omega_{o_i}$. We then sample actions from the symbolic goal-conditioned policy: $a \sim \pi_g(\cdot|\tilde{s}, \omega_{o_i})$. The executor is terminated when β_g outputs 1, indicating that the agent reached g. From this state, the agent can re-plan a functioning path to the task goal s_g.

By integrating HER, the agent learns numerous versions of a task through experience replay, utilizing arbitrary goals derived from a symbolic space. As a consequence of the inherent generalization capabilities across state-goal pairs, our agent has the ability to generalize policy strategies for operators across various groundings. For instance, our agent can generalize knowledge from executing `Pick(``Mug'')` to `Pick(``Dish'')`, employing the same symbolic goal-conditioned policy for `Pick(·)`. It can also generalize the information that the predicate `open(``Door'')` evaluated as `True` is relevant for any navigation task, for example.

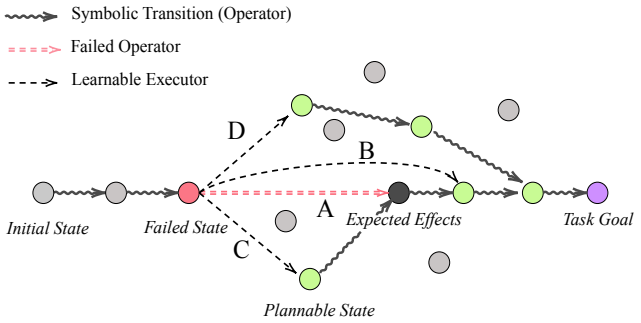


Fig. 3. We discover goals and shortcuts by utilizing the planner during learning. Green dots represent plannable states. Classical hybrid approaches typically pursue paths A or occasionally B. We aim to find shortcuts like paths C and D, which are more cost-effective to learn.

To adapt to new situations quickly, an agent leverages its existing knowledge by integrating transfer learning within our framework. When facing a novelty that requires a new executor, the agent transfers the weights from the most recently used policy of the failed operator instead of starting anew. For a failed operator o_f, we select the latest executor $x = \langle I, \pi_g, \beta_g \rangle$ from $e(o_f)$, trained in environment M. The policy's parameterizing function, π_g, serves as the foundation for π'_g, the policy for $x_{recovery}$ in a new setting, M'. However, to prevent the agent from learning from outdated information, we do not reuse old replay buffers.

Over time, the agent amasses experiences across diverse symbolic states while operating at a continuous level, establishing links between the abstract domain and the MDP.

E. Goal-conditioning & Search for Shortcuts

Finding relevant goals to condition a policy that swiftly accommodates a novelty is essential. In our architecture, we integrate the planner into the learning process to expedite the search for pertinent goals. Notably, we leverage the planner to explore "Shortcuts" (as depicted in Fig. 3)—symbolic states for which a plan to s_g already exists. Upon identifying a Shortcut state, we validate whether the symbolic plan execution from this state to s_g is viable in M'. If the execution proves successful and the agent reaches s_g, the

Algorithm 1 ***Execution*** *($\mathcal{T}$, adapt=true)*

Require: $\mathcal{T} = \langle T, \mathcal{M}, d, e \rangle$ ▷ Integrated Planning Task
Require: $T = \langle \mathcal{E}, \mathcal{F}, \mathcal{O}, s_o, s_g \rangle$
$\mathcal{P} \leftarrow$ **Plan**(T) ▷ $\mathcal{P} = \langle o_1, o_2, ..., o_{|\mathcal{P}|} \rangle$
if $\mathcal{P}$ **is** $\emptyset$ **then return** *false* ▷ Abort
end if
for $o_i \in \mathcal{P}$ **do**
  success $\leftarrow$ *false* ▷ Operator success
  $\mathcal{X}_i \leftarrow e(o_i)$
  for x in ***Prioritized***$(\mathcal{X}_i)$ **do** ▷ Executors Prioritization
    $\tilde{s} \leftarrow$ **Execute**(x)
    $s \leftarrow d(\tilde{s})$
    success $\leftarrow$ *check*$(\omega_{o_i} \subseteq s)$ ▷ Check if ω_{o_i} hold in s
    if *success* **then**
      break for ▷ Operator success. Continue with $\mathcal{P}$
    end if
  end for
  if $\neg$*success* $\wedge \neg$*adapt* **then**
    return *false* ▷ If all $x \in \mathcal{X}_i$ failed $\wedge \neg$*adapt*, abort
  end if
  if $\neg$*success* $\wedge$*adapt* **then** ▷ Else if *adapt*, proceed to *Recovery*
    $\tilde{s_f}, s_f, o_f, \mathcal{X}_f \leftarrow \tilde{s}, s, o_i, \mathcal{X}_i$ ▷ Failure information
    $\widetilde{\mathcal{T}} = \langle T, \mathcal{M}', d, e' \rangle$ ▷ Stretch IPT
    $x_{recovery} \leftarrow$ ***Adapt***$(\widetilde{\mathcal{T}},\ o_f,\ \tilde{s_f})$
    $\tilde{s} \leftarrow$ **Execute**$(x_{recovery})$ ▷ Execute Learned Executor
    $s \leftarrow d(\tilde{s})$
    $\mathcal{P} \leftarrow$ **Plan**(T,s) ▷ Check if T has a plan from s
    success $\leftarrow \neg(\mathcal{P}$ **is** $\emptyset)$
    if *success* **then**
      if $\omega_{o_f} \subseteq s$ **then** ▷ New executor mapping
        $e'(o_f) \leftarrow \mathcal{X}_f \cup \{x_{recovery}\}$
      else ▷ New operator abstraction
        $o_{new} \leftarrow \langle \psi_{new} = s_f, \omega_{new} = s - s_f \rangle$
        $\mathcal{O}'$, $e'(o_{new}) \leftarrow \mathcal{O} \cup \{o_{new}\}$, $\{x_{recovery}\}$
      end if
      break for ▷ Operator success. Continue with $\mathcal{P}$
    else
      return *false* ▷ Abort if still $\neg$*success*
    end if
  end if
end for
return *true* ▷ $\mathcal{T}$ solved successfully

Algorithm 2 ***Adapt*** *($\tilde{\mathcal{T}}$, o_f, $\tilde{s_f}$)* $\rightarrow x_{recovery}$

Require: an off-policy RL algorithm **A**
Require: hyper-parameters H
Require: N_{eps} ▷ Maximum number of episodes
Require: n_{steps} ▷ Maximum number of steps per episodes
Require: η ▷ A success rate threshold
Require: *transfer* ▷ Boolean parameter for transfer learning
$\mathcal{G} \leftarrow$**Compute_Space**$(\mathcal{E}, \mathcal{F})$ ▷ *Goal Space*
$\mathcal{G}_\mathcal{P} \leftarrow \{g \in \mathcal{G} | \omega_{o_f} \subseteq g\}$ ▷ Goal set
$\mathcal{S}_{\mathcal{P}\emptyset} \leftarrow \{s_f\}$ ▷ States without symbolic solution to $T.s_g$
$I \leftarrow$ *test*$(d(\tilde{s})$ **is** $s_f)$ ▷ Initiation indicator
$\beta_g \leftarrow$ *test*$(d(\tilde{s}) \in \mathcal{G}_\mathcal{P})$ ▷ Termination indicator
$R_g \leftarrow$**Compute_Reward**$(\mathcal{G}_\mathcal{P})$ ▷ Symbolic Goal-conditioned Reward
if *transfer* **then**
  $x_{source} \leftarrow$ ***Prioritized***$(\mathcal{X}_f)$
  $\pi_g \leftarrow x_{source}.\pi_g$ ▷ Symbolic Goal-conditioned Policy
else
  Initialize π_g
end if
for N_{eps} *episodes* **do**
  $\tilde{s} \leftarrow$ **Sample_state**(M', I) ▷ Reset the MDP s.t. $I(\tilde{s})$ **is** 1
  $g \leftarrow$ **Sample_goal**$(\mathcal{G}_\mathcal{P})$
  done $\leftarrow$ *false*
  while $\neg$done **do**
    $s \leftarrow d(\tilde{s})$
    $a \sim \pi_g(\cdot|\tilde{s}, g)$
    $\tilde{s}' \sim \tau(\cdot|\tilde{s}, a)$ ▷ Environment step
    $s' \leftarrow d(\tilde{s}')$
    if $(\neg(s'$ **is** $s) \wedge s' \notin \mathcal{G}_\mathcal{P} \wedge s' \notin \mathcal{S}_{\mathcal{P}\emptyset})$ **then**
      $T' \leftarrow \langle \mathcal{E}, \mathcal{F}, \mathcal{O}, s', s_g \rangle$
      $\tilde{\mathcal{T}}' \leftarrow \langle T', M', d, e' \rangle$
      if ***Execution***$(\tilde{\mathcal{T}}'$, *false*$)$ **is** *true* **then**
        $\mathcal{G}_\mathcal{P} \leftarrow \mathcal{G}_\mathcal{P} \cup \{s'\}$ ▷ s' is a functioning plannable state
      else
        $\mathcal{S}_{\mathcal{P}\emptyset} \leftarrow \mathcal{S}_{\mathcal{P}\emptyset} \cup \{s'\}$ ▷ s' has no solution to reach s_g
      end if
    end if
    $\pi_g \leftarrow$***Train_with_HER***$(\tilde{s},\ a,\ \tilde{s}',\ R_g)$
    if $(\beta_g(\tilde{s}')$ **is** $1) \vee$ *reached*(n_{steps}) **then**
      done $\leftarrow$ *true*
    end if
    $\tilde{s} \leftarrow \tilde{s}'$
  end while
  if *success*$(\pi_g, \tilde{\mathcal{T}}) > \eta$ **then**
    $x_{recovery} \leftarrow \langle I, \pi_g, \beta_g \rangle$ **return** $x_{recovery}$
  end if
end for
$x_{recovery} \leftarrow \langle I, \pi_g, \beta_g \rangle$
return $x_{recovery}$

Shortcut becomes a viable goal for $x_{recovery}$ to pursue.

The goal-conditioning and discovery of recovery executors is explained through Alg.2. The policy π_g is goal conditioned and can be trained on any goal in $\mathcal{S}$. In our framework, a desired goal g is a functioning plannable state, i.e., whose plan execution from g to that task goal s_g is successful in the modified MDP M'. We proceed to such verification by invoking Alg.1 *Execution(·)* on the modified task T' with initial state $s'_0 = g$, and *adapt* deactivated. Multiple such goals can exist and we call the set of desired goals $\mathcal{G}_\mathcal{P} = \{g_1, g_2, g_3, ...\}$. More formally, if we consider $s'_0 = s$ as initial state in the modified task $T' = \langle \mathcal{E}, \mathcal{F}, \mathcal{O}, s'_0, s_g \rangle$ embodied in $\tilde{\mathcal{T}}' = \langle T', M', d, e \rangle$, the set of desired goals is $\mathcal{G}_\mathcal{P} = \{s |\ \tilde{\mathcal{T}}'$ **is** *Solvable*$\}$. During the learning process, we initialize $\mathcal{G}_\mathcal{P}$ set using the expected effects of the failed operator o_f, i.e., any state in which ω_{o_f} is satisfied. $\mathcal{G}_\mathcal{P}$ then expands as the agent discovers functioning plannable states through random exploration.

$\mathcal{G}_\mathcal{P}$ is employed to compute the reward function, now denoted as R_g, as shown in Eq.3. $\mathcal{G}_\mathcal{P}$ is also utilized to compute the termination function, now referred as β_g in Eq.2. We treat the indicator functions as test functions:

$$I = \begin{cases} 1, & \text{if } d(\tilde{s}) = s_f, \\ 0, & \text{otherwise.} \end{cases} \quad (1) \qquad \beta_g = \begin{cases} 1, & \text{if } d(\tilde{s}) \in \mathcal{G}_\mathcal{P}, \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

The agent builds a sparse reward function R_g which, in case the agent reaches a goal state $s_t \in \mathcal{G}_\mathcal{P}$, provides a fixed positive reward r; otherwise, a -1 penalty per time step in the MDP. Formally:

$$R_g\left(\tilde{s_t}, a_t, \mathcal{G}_\mathcal{P}\right) = \begin{cases} r, & \text{if } d(\tilde{s_t}) \in \mathcal{G}_\mathcal{P} \\ -1, & \text{otherwise.} \end{cases} \quad (3)$$

This approach revises the conventional HER reward function, which typically accounts for a single goal per episode. Here, the reward function considers a set of goals ($\mathcal{G}_\mathcal{P}$) instead. In our method, each goal $g \in \mathcal{G}_\mathcal{P}$ is assigned an equivalent reward value, yet each step within the MDP M entails a cost.

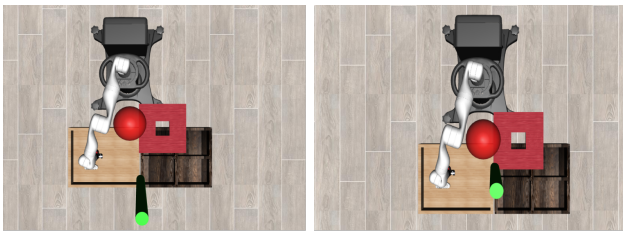


Fig. 4. ***Left:*** the original Pick&Place environment and the task of *picking and placing a can* (the red ball indicates the light-switch location). ***Right:*** the *Obstacle* novelty, a pole obstructs the path to the drop-off area (bin on the right).

Consequently, our agent is designed to identify and prioritize the goal g^* that enables reaching a functioning plannable state in the fewest possible steps. This strategy effectively creates an efficient route for adapting to novelties, thereby reducing the computational demands of learning. Notably, the planner is not engaged at every step of the MDP during the learning phase. Instead, its use is optimized and occurs only when the agent progresses from one symbolic state to another, and the current symbolic state has not been verified through Alg.1 before. If $x_{recovery}$ learns to achieve g^* when o_f fails such that $\omega_{o_f} \not\subseteq g^*$, then we abstract a new operator $o_{new} \notin \mathcal{O}$. In such case, $\psi_{new} = s_f$, $\omega_{new} = g^*$, and $e'(o_{new}) = \{x_{recovery}\}$.

V. Experiment

A. Environment

We evaluate our approach using the continuous RoboSuite [31] collection. Our primary focus was on the Pick and Place Can environment featuring a robotic arm, highly relevant to real-world robotics. The task involves picking and placing a can in a bin, a seemingly simple yet complex task for an RL agent, as indicated by benchmarks [32]. The observation space includes the robotic arm joints state and data on observable objects, i.e., position and orientation of objects. The action space encompasses a 3D displacement of the arm's end effector (using a position-based controller) and the gripper aperture (distance between right and left finger).

B. Experimental Scenario

We introduce novelties sequentially, one at a time, in the environment, causing them to obstruct at least one indispensable operator for task achievement. We did not treat simultaneous injection of multiple novelties. The novelties in the experiments are induced by changes in the dynamics of the environment, which results in plan execution failures. We categorize the novelties as being either a *Shift*, where objects are shifted in space and can still be sensed easily, or as a *Disruption*, which implies an abrupt, binary, and firm boundary mechanism. We focus on five novelties that affect the `Reach(·)` operator in a domain where three operators exist: `Pick(·)`, `Reach(·)`, `Place(·)`. We describe the five novelties in Table I. The environment always returns the same binary reward function: positive if success, negative otherwise independently of the novelties.

TABLE I

Description of the five novelty scenarios.

	Novelty scenarios
Hole *(Shift)*	A wooden plate, usually away from the task area, is now positioned above the *drop-off bin*, blocking it. The agent must navigate the *can* through a hole in the plate to complete the drop.
Elevated *(Shift)*	The *drop-off bin* has been elevated by 15 cm, requiring the agent to adapt by dropping the object from a greater height than previously necessary.
Obstacle *(Shift)*	A pole typically situated at the outer edge of the *pick-up bin* is moved between the two bins. This repositioning obstructs the agent's usual path for dropping off the *can*.
Locked Door *(Disruption)*	A *door*, initially unobstructive, now blocks the path between two bins. The agent needs to unlock it using a motion sensor active in a specific zone. After unlocking, the agent can push the door open to continue with the *can* placement task.
Light Off *(Disruption)*	The light turns off, affecting the robotic arm's LiDAR-like sensors with false readings, except for the accurate distance to the light switch. The agent must turn the light on to regain LiDAR accuracy for placing the *can*.

C. Detection function

The detection function d takes the sensor-level observation from the environment as input and generates predicates or their negations based on the available information. While most predicates are boolean (for instance *locked(door)*), some can also be numerical (as distance(gripper, area) which outputs an integer). The collection of these predicates defines the symbolic state in which the agent currently resides. In instances where the observation does not correspond to any symbolic state, the agent maintains its previous detected state as its belief state, meaning it does not update it. Although the detection function might not recognize novelties, meaning that the symbolic goal space may not include them, the lower-level state-space in the MDP still possesses the ability to sense through lower-level sensing capabilities. Such capabilities enable it to detect environmental changes or novelties, ensuring the preservation of the Markov property.

D. Evaluations and Metrics

Our approach is called "Hybrid Goal-Oriented Adaptive Learning"("HyGOAL Framework"). We employ HER atop SAC as the RL algorithm to train the goal-conditioned reinforcement learning policies. We use metricFF [33] as the symbolic planner (as it can handle numerical fluents).

We compare our method to two Hybrid Symbolic and Learning baselines. The first is "RapidLearn" [4], a Hybrid planning and knowledge-guided learning approach, extended for continual learning. For RapidLearn, we used SAC as the RL algorithm and metricFF as the planner. The second baseline is a reward machine-based method utilizing low-level goal-oriented learning [23], extended for novelty accommodation and termed "LTL&GO." It employs Linear Temporal Logic (LTL) with planning domain for reward machine generation and utilizes goal-oriented learning (SAC+HER, with a sensor level goal space) for lower-level policies. Each agent was provided with a starting set of executors $\mathcal{X}$, a detection function d, and an execution function e ensuring equal performance on the original task/domain. We also

TABLE II
RESULTS AVERAGED ACROSS 10 SEEDS PER AGENT.
↓ SIGNIFIES LOWER IS BETTER, ↑ SIGNIFIES HIGHER IS BETTER.

Novelty	Agent	T_{adapt} ↓ ($\times 10^4$)	$SR_{post\text{-}training}$ ↑
Hole *(Shift)*	HyGOAL	39.8 ± 13.7	0.41 ± 0.50
	RapidLearn	47.8 ± 6.96	0.18 ± 0.35
	LTL& GoalOriented	41.6 ± 16.2	0.21 ± 0.34
	SAC	50.0 ± 0.00	0.00 ± 0.00
TL	HyGOAL *TL*	34.0 ± 17.9	0.61 ± 0.44
	RapidLearn *TL*	18.0 ± 8.43	0.86 ± 0.14
	LTL&GO *TL*	4.60 ± 1.75	1.00 ± 0.00
	SAC *TL*	50.0 ± 0.00	0.03 ± 0.06
Elevated *(Shift)*	HyGOAL	16.6 ± 13.0	0.91 ± 0.30
	RapidLearn	21.4 ± 12.4	0.84 ± 0.30
	LTL&GO	16.2 ± 18.0	0.76 ± 0.37
	SAC	50.0 ± 0.00	0.00 ± 0.00
TL	HyGOAL *TL*	23.4 ± 22.9	0.51 ± 0.45
	RapidLearn *TL*	21.2 ± 16.6	0.72 ± 0.33
	LTL&GO *TL*	9.00 ± 14.5	0.86 ± 0.32
	SAC *TL*	50.0 ± 0.00	0.02 ± 0.03
Obstacle *(Shift)*	HyGOAL	39.0 ± 16.0	0.38 ± 0.48
	RapidLearn	36.2 ± 13.6	0.66 ± 0.40
	LTL&GO	50.0 ± 0.00	0.04 ± 0.10
	SAC	50.0 ± 0.00	0.00 ± 0.00
TL	HyGOAL *TL*	36.2 ± 18.3	0.34 ± 0.47
	RapidLearn *TL*	45.8 ± 8.0	0.24 ± 0.35
	LTL&GO *TL*	37.2 ± 20.7	0.29 ± 0.47
	SAC *TL*	50.0 ± 0.00	0.00 ± 0.00
Locked Door *(Disruption)*	HyGOAL	13.8 ± 5.00	0.90 ± 0.28
	RapidLearn	29.4 ± 15.8	0.72 ± 0.38
	LTL&GO	39.6 ± 17.3	0.21 ± 0.30
	SAC	50.0 ± 0.00	0.00 ± 0.00
TL	HyGOAL *TL*	28.2 ± 23.1	0.51 ± 0.51
	RapidLearn *TL*	50.0 ± 0.00	0.11 ± 0.23
	LTL&GO *TL*	28.0 ± 23.2	0.50 ± 0.47
	SAC *TL*	50.0 ± 0.00	0.00 ± 0.00
Light Off *(Disruption)*	HyGOAL	26.4 ± 14.8	0.80 ± 0.37
	RapidLearn	31.0 ± 11.6	0.73 ± 0.32
	LTL&GO	41.6 ± 17.8	0.23 ± 0.38
	SAC	50.0 ± 0.00	0.00 ± 0.00
TL	HyGOAL *TL*	41.0 ± 14.5	0.30 ± 0.48
	RapidLearn *TL*	47.2 ± 8.90	0.09 ± 0.20
	LTL&GO *TL*	50.0 ± 0.00	0.08 ± 0.19
	SAC *TL*	50.0 ± 0.00	0.00 ± 0.00

include a pure RL baseline (SAC+HER), pre-trained to match Hybrid techniques' performance on the initial task/domain.

We also evaluated transfer learning in Hybrid agents. 'Agent *TL*' refers to agents with transferred executor policies between novelties, to assess if this transfer enhances adaptability to continual novelty injections.

After each novelty injection, agents were trained on the new scenario until meeting a convergence criterion—either a success rate above 80% or a maximum of $500,000$ training steps in the Pick&Place environment. In the RoboSuite environment, an episode entails a maximum of $1,000$ interactions. We averaged the results across 10 seeds per agent. Performance was evaluated by running 20 episodes every $20,000$ training steps and calculating the mean success rate at each evaluation point. The same RL hyperparameters were consistently applied across all novelties. Our evaluation focuses on two key metrics: T_{adapt}, indicating the number of MDP time steps required for the agent to achieve the convergence criteria, and $SR_{post\text{-}training}$, representing the agent's success rate upon reaching asymptotic convergence.

VI. RESULTS

Table II summarizes all agents' performance. Pure RL, along with re-planning (not listed in the table), yielded zero success across all novelties within $500,000$ steps, highlighting RL inefficiency in handling robotics novelties, as also supported by RoboSuite Benchmark. This underscores RL limitations even in basic tasks. Conversely, Hybrid approaches demonstrated adaptability to five sequential novelties, effectively managing significant task alterations by utilizing domain abstraction for targeted updates where novelties impact the agent's knowledge base.

Our method "HyGOAL" exhibited increased sample efficiency compared to other Hybrid approaches, outperforming all of them on four out of five scenarios both in terms of time to adapt and asymptotic success rate. In such uninformed scenarios, no human guidance is available, limiting the leverage other baselines tend to utilize to speed up their adaptation. "HyGOAL" was only outperformed once by "RapidLearn" on the *(Shift) Obstacle* novelty, a scenario in which our agent does not benefit from the framework extension. "HyGOAL" achieved convergence criteria with impressive speed in both *Disruption* novelties, just 138 episodes for *Locked Door*, thereby demonstrating a faster adaptation capability.

Transfer Learning (TL) had mixed effects on agents, proving helpful for repurposing past learning for adaptation but harmful when unlearning previous information was necessary. An ablation study [2] on the sequence of novelty injections in a discrete domain revealed TL's limitations, particularly when discarding old information outweighed the benefits of leveraging past experiences. Our findings suggest that goal-conditioned policy transfer enhances transfer robustness compared to traditional policy transfer, with "LTL&GO TL" and "HyGOAL TL" performing best. "LTL&GO TL" achieved better and faster convergence on the first two Shift novelties. A Shift novelty causes certain objects in the environment to change position in 3D space. This can be effectively captured by using HER with a low-level distance sensor to represent the objects' goals, as performed in "LTL&GO TL". However, "HyGOAL TL" demonstrated superior performance across various novelty injections, particularly excelling in handling Disruption novelties. The symbolic goal space effectively captured new symbolic mechanisms, including abrupt transitions, anchoring domain information and enhancing resilience against forgetting. In summary, TL involves risks, demanding advanced assessment skills to weigh benefits and drawbacks prior to learning.

VII. DISCUSSION

Our hybrid approach streamlines the search space both horizontally and vertically. Horizontally, the state space is decomposed into operators, while vertically, providing the agent with pertinent goals derived from symbolic descriptions directs its exploration. The refinement of the goal space involves including only meaningful states, specifically the sub-states $\widetilde{s}$ that are imbued with symbolic significance.

[2]Code and appendix available via this link.

Consequently, owing to their symbolic nature, these substates focus the agent's attention on relevant states, thus increasing their likelihood as plannable states. This dual effect is in line with the hybrid approach, which integrates a planner into the learning process.

This work opens up promising research directions, particularly in hybrid approaches for handling open-world novelty. However, some limitations need consideration. Neurosymbolic approaches face scalability challenges in continual learning due to accumulating information. Agents should efficiently share information between executors, cluster symbolic states by domain, filter relevant data, and discard outdated information. The static detector function also needs to evolve with time and novelty injections. Future work should focus on knowledge abstraction to improve inference and mitigate scalability issues while enabling adaptive detection, which is crucial for enhancing adaptation and sample efficiency. Evaluating constrained state space exploration and refining policy transfer could lead to more focused explorations, making hybrid methods and RL-based techniques more practical for online learning in open worlds.

VIII. Conclusion

We introduced a hybrid framework that integrates planning and learning, using multi-goal executors conditioned on symbolic goals derived through planning. Our approach effectively adapts to novelties—sudden, uninformed changes that disrupt environment dynamics and task completion. Experiments show that combining planning with learning significantly outperforms purely learning-based methods. Symbolic goal-oriented learning enables the generalization of symbolic knowledge to low-level actions, strengthening neural connections and improving adaptability and transfer learning. The framework's success in various novelty scenarios highlights its potential in real-world applications, where efficiently handling unforeseen challenges is essential.

References

[1] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl," 1998.

[2] C. Gehring, M. Asai, R. Chitnis, T. Silver, L. Kaelbling, S. Sohrabi, and M. Katz, "Reinforcement learning for classical planning: Viewing heuristics as dense reward generators," *ICAPS*, vol. 32, no. 1, pp. 588–596, Jun. 2022.

[3] E. Gizzi, M. G. Castro, and J. Sinapov, "Creative problem solving by robots using action primitive discovery," in *2019 Joint IEEE 9th Intl. Conf. on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE, 2019, pp. 228–233.

[4] 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 ICDL*, 2022.

[5] L. Steccanella and A. Jonsson, "State representation learning for goal-conditioned reinforcement learning," *arXiv:2205.01965*, 2022.

[6] R. Karia and S. Srivastava, "Relational abstractions for generalized reinforcement learning on symbolic problems," *arXiv*, p. 2204, 2022.

[7] H. Kokel, A. Manoharan, S. Natarajan, B. Ravindran, and P. Tadepalli, "Reprel: Integrating relational planning and reinforcement learning for effective abstraction," in *ICAPS*, May 2021.

[8] L. Guan, S. Sreedharan, and S. Kambhampati, "Leveraging approximate symbolic models for reinforcement learning via skill diversity," *arXiv:2202.02886*, 2022.

[9] N. Kumar, W. McClinton, R. Chitnis, T. Silver, T. Lozano-Pérez, and L. P. Kaelbling, "Learning operators with ignore effects for bilevel planning in continuous domains," *arXiv:2208.07737*, 2022.

[10] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *ICRA*, 2020.

[11] 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," *arXiv*, 2023.

[12] B. Liu, S. Mazumder, E. Robertson, and S. Grigsby, "Ai autonomy: Self-initiated open-world continual learning and adaptation," *AI Magazine*, 2023.

[13] 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 *AAMAS*, 2021.

[14] K. Khetarpal, M. Riemer, and I. R. amd Doina Precup, "Towards continual reinforcement learning: A review and perspectives," *arXiv:2012.13490v1*, 2020.

[15] M. Riemer, I. Cases, R. Ajeman, M. Liu, I. Rsh, Y. Tu, and G. Tesaro, "Learning to learn without forgeting by maximizing transfer and minimizing interference," in *ICLR*, 2019.

[16] J. Schmidhuber, "Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem," *Front. in psychology*, 2013.

[17] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, and M. Scheutz, "Spotter: Extending symbolic planning operators through targeted reinforcement learning," in *AAMAS*, 2021.

[18] R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *JAIR*, vol. 73, pp. 173–208, 2020.

[19] R. K. Nayyar, P. Verma, and S. Srivastava, "Differential assessment of black-box ai agents," *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 36, no. 9, pp. 9868–9876, Jun. 2022.

[20] Y. Seo, K. Lee, I. Clavera, T. Kurutach, J. Shin, and P. Abbeel, "Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning," 2020.

[21] 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.

[22] A. S. Chen, A. Sharma, S. Levine, and C. Finn, "Single-life reinforcement learning," in *NeurIPS*, 2022.

[23] D. Xu and F. Fekri, "A framework for following temporal logic instructions with unknown causal dependencies," 2022.

[24] 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.

[25] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *ICML*, F. Bach and D. Blei, Eds., vol. 37. PMLR, 07–09 Jul 2015.

[26] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg, "Ai safety gridworlds," *arXiv preprint arXiv:1711.09883*, 2017.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, 2013.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, 2018.

[29] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *CoRR*, 2017.

[30] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, 1999.

[31] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv:2009.12293*, 2020.

[32] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, "Surreal: Open-source reinforcement learning framework and robot manipulation benchmark," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 767–782.

[33] 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.