NovelGym: A Flexible Ecosystem for Hybrid Planning and Learning Agents Designed for Open Worlds

Shivam Goel
Tufts University
Medford, USA
shivam.goel@tufts.edu

Yichen Wei
Brown University
Providence, USA
yichen.wei@brown.edu

Panagiotis Lymperopoulos
Tufts University
Medford, USA
panagiotis.lymperopoulos@tufts.edu

Klára Churá
Tufts University
Medford, USA
klara.chura@tufts.edu

Matthias Scheutz
Tufts University
Medford, USA
matthias.scheutz@tufts.edu

Jivko Sinapov
Tufts University
Medford, USA
jivko.sinapov@tufts.edu

ABSTRACT

As AI agents leave the lab and venture into the real world as autonomous vehicles, delivery robots, and cooking robots, it is increasingly necessary to design and comprehensively evaluate algorithms that tackle the "open-world". To this end, we introduce NovelGym[1], a flexible and adaptable ecosystem designed to simulate gridworld environments, serving as a robust platform for benchmarking reinforcement learning (RL) and hybrid planning and learning agents in open-world contexts. The modular architecture of NovelGym facilitates rapid creation and modification of task environments, including multi-agent scenarios, with multiple environment transformations, thus providing a dynamic testbed for researchers to develop open-world AI agents.

KEYWORDS

open world learning, neurosymbolic learning, benchmarking environments

ACM Reference Format:
Shivam Goel, Yichen Wei, Panagiotis Lymperopoulos, Klára Churá, Matthias Scheutz, and Jivko Sinapov. 2024. NovelGym: A Flexible Ecosystem for Hybrid Planning and Learning Agents Designed for Open Worlds . In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

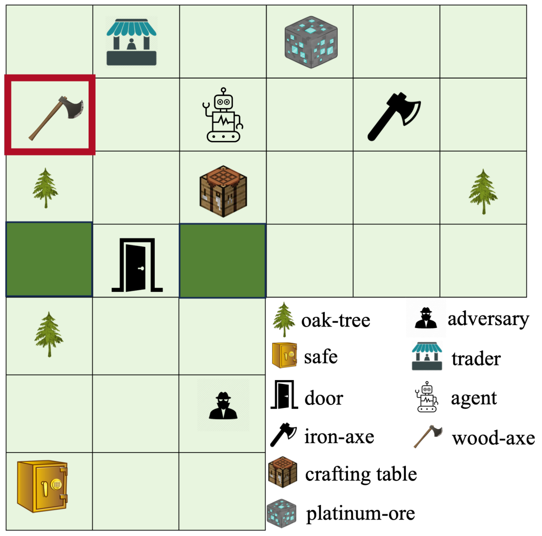

Figure 1: NovelGym environment representation. The figure shows a gridworld environment with various entities, as described in the legend. The red box highlights the novelty in the environment.

1 INTRODUCTION

As AI research ventures beyond "closed-worlds" where agents know all task-relevant concepts in advance, the ability to recognize, learn, and adapt to *conceptually* new situations becomes increasingly important. While significant research effort has been invested in creating "open-world" systems [4, 7, 18, 22], comprehensively evaluating them remains a challenge due to 1) the varying and conflicting interpretations of novelty as a concept [4, 6, 15], 2) the varying architectural choices made in designing novelty-aware agents [3, 8, 12, 24] and 3) the unbounded space of novelties that an agent may encounter.

In this work, we consider *novelty* an intrinsically agent-relative concept: An aspect of the world is *novel for an agent*, if that agent has not experienced it in the past or cannot derive it from its current knowledge. As such, depending on the particular cognitive and perceptual capabilities of a given agent, different aspects of the world may or may not constitute novelty. Similarly, a particular aspect of the world that is novel for an agent may also be irrelevant to it, making adaptation unnecessary. For instance, the height of a lamp is a novel concept for an automated vacuum cleaner but one that is irrelevant with respect to its cleaning task. We also emphasize that what may be novel for one agent may not be novel for another. Therefore, when comparing agents' capabilities in novelty adaptation, it is important to control for such differences. As a result, evaluation environments for novelty-aware agents need to be flexible enough to accommodate varying agent architectures, easy to

[1]Project website and codebase source: https://novelgym.github.io/

This work is licensed under a Creative Commons Attribution International 4.0 License.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

extend to enable rapid development of novelties and tasks, including for multi-agent scenarios, and offer agent-agnostic evaluation metrics that can measure the agent's ability to adapt to novelty compared to non-novelty aware agents of similar capabilities.

In our work, we propose a new benchmark for the evaluation of novelty-aware agents that is consistent with the aforementioned desiderata. Specifically, NovelGym offers:

(1) A flexible and modular environment featuring easy task and novelty design for developing and evaluating open-world agents, in single and multi-agent scenarios.
(2) An ecosystem that seamlessly works with agents of different architectures, including symbolic planning agents, reinforcement learners, and hybrid neurosymbolic architectures.
(3) Benchmarks of various state-of-the-art learning and hybrid methods for novelty handling.
(4) Agent-agnostic evaluation metrics for novelty adaptation.

The rest of our paper is structured as follows: First, we discuss related work on novelty-aware agents and environments to evaluate them, followed by establishing the theoretical framework upon which the design of NovelGym is based. We present the environment architecture and current environment transformations that may serve as novelties for tested agents. We also explain the modular implementation of the environment and how it facilitates further design of tasks, agents, and novelties. Finally, we present evaluation measures and benchmark evaluations of novelty handling agents.

2 RELATED WORK

Recently, there has been increasing interest in creating agents that can adapt to sudden and abrupt changes (i.e., novelties). Klenk et al. [13] present a trainable model for novelty adaptation called WorldCloner, where a symbolic representation of the pre-novelty world is learned and then used to detect novelties. WorldCloner uses a gridworld where an agent must complete a task that may or may not be obstructed by novelties, but the task is simple and does not involve a complex sequence of operations like crafting or breaking. Stern et al. [22] also presents a model-based framework named HYDRA that uses a domain-independent planner for the popular video game Angry Birds. Sarathy et al. [18] proposed SPOTTER, an approach that goes beyond pure reinforcement learning methods to learn new operators needed to solve a task when symbolic planning cannot due to novelties. Pardo [16] introduced Tonic, a benchmarking library for deep reinforcement learning that is configurable but limited to the scope of compatible environments and agents.

Several environments and frameworks for novelties or open-world scenarios have been developed to aid the research and development of those frameworks. NovGrid [2] is a novelty generator built on MiniGrid that allows the injection of novelties into any existing minigrid environment. Silver and Chitnis [21] present PDDLGym, which is a gym environment for RL research that can be generated from symbolic, PDDL domain files commonly used in planning. Goel et al. [9] present NovelGridWorlds, a Minecraft-inspired grid world environment to study the detection and adaptation of novelty that works with Planning and Learning. However, none of the environments provides an easy injection of novelties and the ability to integrate Planning and Learning seamlessly. Our environment provides a modular and highly configurable interface and the flexibility to integrate with both planning and RL agents in multiple ways, which, to the best of our knowledge, doesn't exist. Hence, the contribution of such an ecosystem can enable research in the direction of open-world problem-solving.

3 THEORETICAL FRAMEWORK

3.1 Running example

Let us consider a gridworld as shown in Figure 1; the environment is laid out as the two rooms environment separated by a door and two cells (shown in dark green). As shown in Figure 1, an agent is facing a *crafting table* (used to craft items such as tree tap, axe, etc.). The agent can collect resources from the environment by moving around with navigation actions (move forward, turn left, turn right). The agent can collect resources by breaking them and then craft them into specific items using certain recipes. For example, an agent can break the oak tree to get two logs, and one log can be crafted into four planks. The agent's goal is to craft a *pogostick*. The recipe for crafting a pogostick involves collecting logs, diamonds, and platinum and crafting various intermediary items such as planks, sticks, tree-tap, etc. There are other entities, such as traders, with whom the agent can interact and *trade* items. The environment also has an adversarial agent that competes with the agent in getting resources to craft the *pogostick*.

3.2 Environment

We formalize the environment E as $E = \langle G, \mathcal{E}, \mathcal{R}, \mathcal{S}, A, \tau, C \rangle$, where each component is defined as follows:

Grid. $G \subseteq \mathbb{Z}^2$ represents the set of all grid cells in a 2D gridworld. For an $(m \times n)$ grid, any cell can be uniquely identified by its position (i, j) where $(1 \leq i \leq m)$ *and* $(1 \leq j \leq n)$. For example, the gridworld shown in Figure 1 consists of 7 rows and 6 columns.

Entities. $\mathcal{E} = \{< t_1, P_{e_1} >, < t_2, P_{e_2} >, \ldots, < t_n, P_{e_n} >\}$ is the set of all the entities in the environment where,

- t_i is the type of the i^{th} entity from the set T of all possible entity types: $T = \{t_1, t_2, \ldots, t_t\}$. For instance, an entity "tree" might have a type "oak-tree" within T.
- P_{e_i} is the set of properties of the i^{th} entity. Each property set is a subset of the global property set ($P_{e_i} \subseteq P$), which encompasses all possible properties an entity can exhibit: $P = \{p_1, p_2, \ldots, p_p\}$
- An entity e is thus represented as a tuple $e =< t, P_e >$ where $t \in T$ and $P_e \subseteq P$.
- Some entities are dynamic, such as other agents or adversaries (refer to Figure 1), which can actively take actions in the environment, influencing state transitions.
- Entities can be located at individual grid cells, in an agent's inventory, or nested within other entities (e.g., stored inside a chest or safe).

For example, as depicted in Figure 1, the gridworld environment includes entities like oak-*tree*, representing the tree entity with *oak* type. This tree might have properties such as *breakable*. Similarly, an *axe* entity in the environment might have types like *wood* or *iron* and can possess properties like *graspable*.

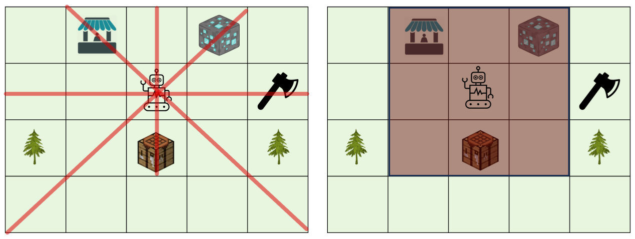
Figure 2: Illustration of the sensor representation of the agent in the environment. (Left) shows a LiDAR representation. (Right) shows an image based local view representation.

Recipes. $\mathcal{R} = \{r_1, r_2, \ldots, r_r\}$ is a set of transformation rules, where a rule $r \in \mathcal{R}$ defines how one or more entities can be transformed into another set of entities. Each rule is defined as $r : \mathcal{E}^* \times L_r \rightarrow \mathcal{E}^*$, where $L_r \subseteq G$ represents the locations in the grid world, i.e., a recipe can be applied at different places in the environment. For example, some recipes work only in front of the *crafting table*, or in front of other dynamic entities like *traders* (traders and crafting table shown in Figure 1). Consider the rule r : $(\{1 \text{ platinum } 1 \text{ stick } 2 \text{ plank}\}, L_{\text{crafting\_table}}) \rightarrow \{1 \text{ platinum axe}\}$. It indicates that, 1 unit of *platinum* and *stick*, and two units of *plank* can be crafted into platinum axe at the crafting table.

States. $\mathcal{S}$ is the set of all possible states in the environment. For example, as shown in Figure 1, a possible state of the environment can be the locations of all the entities in the world.

Primitive Actions. $A = \{a_1, a_2, \ldots, a_a\}$ is the set of all available primitive actions. While all actions are available to the agent in every state, some actions only result in changes to the state in specific contexts. For example, *break* action can be executed anywhere in the gridworld, but would only have an effect if performed in front of an entity that has the *breakable* property (example, a tree).

Transition Dynamics. $\tau = \mathcal{S} \times A \rightarrow \mathcal{S}$ denotes the transition dynamics and is responsible for determining the progression from one state to another based on a given action. Formally, the transition dynamics can be represented as: $\tau : \mathcal{S} \times A \times (A_{e_1}, A_{e_2}, \ldots, A_{e_m}) \rightarrow \mathcal{S}$, where $\mathcal{S}$ represents the state space, A denotes the action space of the primary agent, and the tuple $(A_{e_1}, A_{e_2}, \ldots, A_{e_m})$ captures the sequence of actions taken by each dynamic entity in the environment. These dynamic entities can act as adversaries or other environmental actors, influencing the state transitions.

Cost function. $C : \mathcal{S} \times A \rightarrow \mathbb{R}^+$, denotes the function that assigns a fixed, non-negative cost to each state and action pair. Specifically, for each action $a \in A$, in a state $s \in \mathcal{S}$, $C(s, a)$ provides the associated cost of performing the action in a specific state. For example, the cost associated with moving one step in the grid can be 1, or breaking a tree can have a cost of 5, and depending on the context, it may cost less, for example, breaking when holding a tool.

3.3 Agent

We define an agent $\mathcal{A} = \langle Sen, Act, \mathcal{K}, \Pi \rangle$ as an entity in the environment equipped with sensors Sen to generate observations O of the world, actuators Act that can affect the world, a knowledge repository $\mathcal{K}$ that contains knowledge (learned or provided), and a function $\Pi : O \rightarrow A$ that maps observations to actions. Agent observations can be high-dimensional sub-symbolic vector representations or can be symbolic representations. For example, if the agent has a LiDAR-like sensor (as shown in Figure 2 (left)), then the sensors will produce distances to the objects in the world; similarly, an image-based local view would represent the grid with one hot vector encoding on the entity types (shown in Figure 2 (right)); similarly the mapping can be from a state to a high level symbolic state described using PDDL [1](as shown in Table 1).

types	`air,crafting_table,diamond,platinum - physobj`
	`tree,diamond,platinum - breakable`

predicates	`(holding ?v0 - physobj)`
	`(floating ?v0 - physobj)`
	`(facing ?v0 - physobj)`

fluents	`world(?physobj)`
	`inventory(?object)`

action	`approach`
params	`(?physobj01 ?physobj02)`

preconditions	`(and`
	`(>= ( world ?physobj02) 1)`
	`(facing ?physobj01))`

effects	`(and`
	`(facing ?physobj02)`
	`(not (facing ?physobj01))`

Table 1: PDDL representation of the symbolic domain.

The agent, using actuators Act can act in the environment. The actions can be primitive actions or parameterized actions that implement the primitive-level actions as a sequence. For example, an agent can have an action as `approach <entity>`, which is implemented by a planner and uses primitive actions (move forward, etc.) to execute the action. The agent's knowledge repository, $\mathcal{K}$, can be initially empty and accumulate knowledge over time. Knowledge can be in the form of parameterized policies or a description of the world symbolically through first-order logic. An agent's behavior function Π can be implemented based on two popular paradigms for decision-making agents, namely, Planning and Reinforcement Learning, as well as hybrid approaches combining the two. We further formalize the symbolic planning and reinforcement learning frameworks to further describe different agents.

3.3.1 Symbolic Planning. In specifying a planning problem, we define $\mathcal{L}$ as a first-order language containing atoms $p(t_1, ..., t_n)$ and their negations $\neg p(t_1, ..., t_n)$, where each atom t_i may be constant or variable. We define a planning domain in $\mathcal{L}$ as $\mathcal{D} = \langle \tilde{\mathcal{S}}, O, \tau_\alpha \rangle$, where $\tilde{\mathcal{S}}$ represents the set of symbolic states, O the set of finite action operators, and τ_α as the transition function that describes how the state changes as a result of an action operator being executed in the environment [19]. We then define a planning problem as $\mathcal{P} = (\mathcal{D}, \tilde{s}_0, \tilde{S}_g)$, where $\tilde{s}_0$ is the initial state and $\tilde{S}_g$ is the set of goal states. The agent begins in a start state $\tilde{s}_0$ and establishes a plan π for reaching one of the goal state contained in $\tilde{S}_g$. Hence, the plan $\pi = [o_1, o_2, ..., o_{|\pi|}]$ is a solution to the planning problem $\mathcal{P}$, where each $o_i \in O$ is an action operator that has a set of preconditions and effects. The preconditions describe the states before executing the operator in the environment, and the effects describe the state of the environment after the agent has executed the operator.

3.3.2 Reinforcement Learning. We formalize an RL problem as a Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}_\beta, A_\beta, \tau_\beta, R, \gamma \rangle$. At time-step t, the agent is given a state representation $S_t \in \mathcal{S}_\beta$. Exploring the environment by taking action $A_t \in A_\beta$, the agent is assigned a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ based on the state S_{t+1} it lands in by choosing an action A_t in a state S_t. The goal of the agent is to learn a policy that maximizes the expected return value $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ for every state at time t. The importance of immediate and future rewards is determined by the discount factor $\gamma \in [0, 1)$.

3.4 Novelty

We simulate the open-world using a base environment and a set of environment transformations that act on one (or more) of its constituent elements. The transformations may introduce *novelties* for some agents, depending on their knowledge, perception, and representations. More formally, we can define the transformation of the environment as a function $\nu : E \longrightarrow E'$. The function ν may transform the environment as:

- **Layout changes:** The transformation can affect the layout or the grid size G such that $G \neq G'$in E'. In other words, in the transformed environment E', the layout of the grid G' is not the same as the original layout G.
- **Entity alterations:** New or existing entities may be introduced or modified. This can be represented as a transformation in the set of entities $\mathcal{E}$ such that $\mathcal{E} \neq \mathcal{E}'$in E'.
- **Recipe modifications:** The set of transformation rules $\mathcal{R}$ can undergo changes, leading to $\mathcal{R} \neq \mathcal{R}'$ in E'.
- **Action alterations:** The set of available actions A can experience modifications, resulting in $A \neq A'$ in E'.
- **Transition dynamics change:** If for some state $s \in \mathcal{S}$ and $a \in A$, we have $\tau(s, a) \neq \tau'(s, a)$ where τ and τ' are the transition dynamics of E and E' respectively, then the transition dynamics have changed.
- **Cost function alterations:** The cost function C can be modified, leading to $C \neq C'$ in E'. This implies differences in costs for certain actions in specific states between the original and transformed environments.

Importantly, these transformations are composable, enabling the creation of arbitrarily complex environments. By taking into account an agent's knowledge, perceptions and representations, we can apply transformations and their compositions to create novelties for that agent. For instance, an agent that assumes a specific environment layout for navigation would encounter novelty under a transformation that flips the environment layout, whereas an agent that operates with LiDAR sensors may not.

4 NOVELGYM: ARCHITECTURE & IMPLEMENTATION

The NovelGym ecosystem serves as a platform for developing and evaluating AI agents, with a focus on open-world novelty-aware agents. Comprising a game engine and several aiding modules, the system diagram of the ecosystem can be visualized as shown in Figure 3. As seen in the figure, NovelGym has two main components. The first component, environment (highlighted in blue), represents the modules that implement the environment and the game engine. The second component, agent (highlighted in purple), showcases the modules that can be used to implement agent architectures. A novelty injector (shown in red) transforms the environment based on the given specifications. We now describe each component of the system while laying out the important features that enable open-world novelty-aware agent training and evaluations.

4.1 Environment

Our environment implementation is based on PettingZoo [23] and OpenAI Gym [5]. The environment component of NovelGym houses a core engine that is responsible for the implementation of entities (entity module shown in Figure 3) and actions (action module shown in Figure 3). The design of separate modules for all the features of the environment ensures easy task creation and novelty implementation. The core engine maintains the state of the world through the state module. The configuration of the world can be specified using the specification module. Our environment implementation supports multi-agent systems. Thus, each movable entity (agent) takes turns in executing actions. The world map keeps track of the location of entities in the grid world and the coordinates of the rooms. We now describe each module.

4.1.1 Specification module. The specification module reads the YAML configuration file and initializes the world. It also loads the action and entity modules and initializes them with the parameters and properties as specified in the configuration file.

4.1.2 Entity module. The entity module is responsible for the entities in the environment. Based on the specification, the entities are initialized and may be modified by writing custom entity modules[2]. Entity properties can also be specified directly in a configuration YAML file. For example, objects may be configured to be breakable by hand, by tools, or non-breakable.

4.1.3 Action module. The action module is responsible for the action implementations in the environment. Our environment supports primitive actions as well as higher-level action operators (as described in Section 3.3.1). The action module executes the actions in the environment and ensures that the state of the world is updated by updating the environment.

4.1.4 World map/state module. The world map/state module generates the representation of gridworld given the current state. It has a state tracker implemented inside it that keeps track of the entities (including agents). The state tracker also runs scheduled tasks at the end of each timestep. These scheduled tasks may include updating the state for durative actions. For example, the action of break-tree introduces one sapling in the world after three timesteps. In this case, the state tracker will ensure that the world is updated with a sapling after three timesteps of action execution.

4.1.5 Novelty module. The novelties are implemented as extensions or modifications of the environment. The novelty module helps in modifying the entity, action, and state modules depending on the novelty configuration.

[2]A detailed tutorial on how to use and customize the environment can be found here.

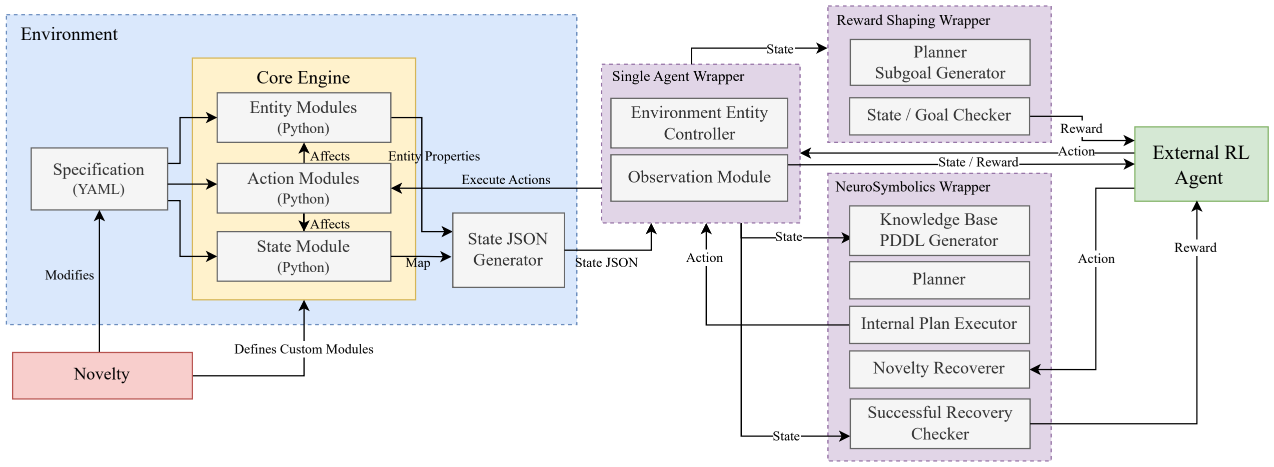

Figure 3: System design of the NovelGym ecosystem. Blue highlights the environment modules, and purple highlights the agent modules.

4.2 Agent training architecture

The agent training architecture comprises three main components to aid the development of hybrid learning agent architectures.

4.2.1 Single agent wrapper. While our environment supports multi-agent systems, we focus on a single primary agent that solves the task. Other agents in the environment are movable entities that can take actions in the environment. The single-agent wrapper converts the PettingZoo [23] multi-agent environment into a single-agent one by taking over the action execution of all other agents while exposing the control to the primary agent. The single agent wrapper includes an observation conversion module that converts the state of the world into a desired representation, e.g., a local view of the map, or a lidar representation of the world (see Figure 2). The customization allows the testing of different state representations. For object-centric observation spaces, the conversion module can be configured to automatically expand the observation and action spaces in cases of additional entities and actions in novelties.

4.2.2 Neurosymbolic wrapper. The neurosymbolic wrapper is responsible for combining the symbolic planning agents and reinforcement learning agents. The wrapper maintains a knowledge base that can be in the form of PDDL. With a pre-defined PDDL template, the wrapper automatically generates PDDL files by referring to the information from the pre-novelty configuration file and individual object and action modules. The generated PDDL can be sent to a planner (implemented in the planner component through MetricFF [10]) to generate plans. The plan executor ensures that each operator in the plan is executed in the environment. The wrapper also has a novelty recovery component that can be used to implement routines for novelty recovery.

4.2.3 Reward-shaping wrapper. In complex tasks with a large state space and action spaces, reward shaping is a commonly used technique. With the help of the integrated planner, a filtered list of actions in the PDDL plan gets selected, from which the wrapper generates sub-goals. The user may define the filter criteria and the plan-subgoal correspondences. Through comparison of the state before and after each transition, the wrapper checks whether the subgoal is met and rewards the agent for reaching the sub-goals. The reward-shaping wrapper can help implement sophisticated routines for novelty handling by combining planning and learning.

4.2.4 External RL agent. Our architecture also implements a modular reinforcement learning framework (Tianshou [25]). The availability of this module helps in implementing RL algorithms for training agents. The connectivity of the module with our agent architectures enables sophisticated agent designs and helps users in experimenting with various learning algorithms.

5 EVALUATIONS

Evaluating open-world agents necessitates novel protocols. We must also adjust evaluation metrics, considering agent performance both pre- and post-novelty injection. We define two scenarios, *pre-novelty* and similarly, *post-novelty* scenario. In the *pre-novelty scenario*, the environment conditions are known to the agent, and the agent's knowledge and/or a pre-trained policy is enough to solve the task *successfully*. However, when a novelty is injected, the agent's knowledge may become incomplete to solve the task either *successfully* (and/or) *optimally*. We call this scenario the *post-novelty scenario*. To illustrate, let us consider the introduction of a novel entity axe in the environment (shown in the red box in Figure 1). In the subsequent sub-sections, we will detail the proposed evaluation protocol, followed by the proposed evaluation metrics.

5.1 Evaluation Protocol

The evaluation protocol is divided into four phases:

Initial training phase. In this phase, we train the agent in a controlled environment that is free from novelties. In other words, the agent's knowledge base is complete to solve the task. For a reinforcement learning agent, knowledge can be a shaped reward function, a predefined hierarchical task decomposition [14], or a Linear Temporal Logic (LTL) guided automaton as a reward function [11], etc. For a planning agent, knowledge can be a description of the task through PDDL [1](illustrative example in Table 1).

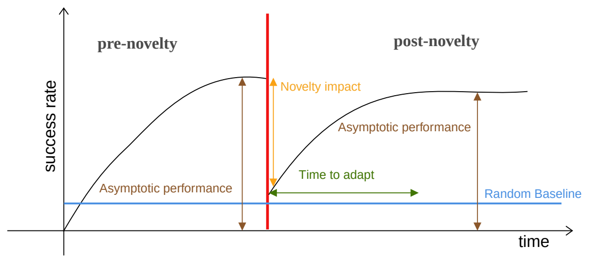

Figure 4: Illustration of performance metrics for open-world agents.

Novelty injection. In this phase, we introduce the novelty into the environment. The introduction of the novelty may hamper the performance of the agent. We can monitor the impact of the performance of the agent immediately upon the introduction of the novelty. Monitoring the impact can help in testing the robustness of the agent in the face of novelties.

Adaptation Phase. In this phase, we allow the agent to interact with the novel environment to solve the task efficiently. This phase is crucial in gauging how quickly and effectively the agent recalibrates its approach in response to the introduced novelty.

Post-Adaptation evaluation. After a period of adaptation (predefined time or convergence criteria), we can assess the performance of the agent in the novel environment.

5.2 Evaluation Metrics

We propose five evaluation metrics to evaluate agents in open-world environments, illustrated in Figure 4. The graph is plotted with respect to the success rates versus time. The *success rate* measures the agent's performance in successfully solving the task. The success rate can be measured at every epoch[3] *Success rate* (S): Given n episodes, where s of them are successful, the success rate is defined as $S = \frac{s}{n}$. We formally define the evaluation metrics:

(1) **Pre-novelty asymptotic performance ($S_{\textbf{pre-novelty}}$)**: This metric measures the final performance of the agent after the convergence criterion is met in the pre-novelty task (shown in brown in the pre-novelty part of Figure 4).

(2) **Novelty impact ($I_{\textbf{novelty}}$)**: This metric quantifies the immediate effect of introducing novelty on the agent's performance. It is calculated as the difference between the agent's performance before the novelty is introduced and its immediate performance after the novelty is encountered (illustrated in yellow in Figure 4):

$$I_{\text{novelty}} = S_{\text{pre-novelty}} - S_{\text{immediate post-novelty}}$$

If the agent cannot solve the task without further adaptation, its performance can theoretically drop to zero. Alternatively, in certain scenarios, even without immediate adaptation, the agent might still display non-zero performance.

(3) **Time to adapt ($T_{\textbf{adapt}}$)**: The time taken by the agent to reach the convergence criteria post novelty adaptation is the time to adapt (illustrated in green in Figure 4). The time taken can be measured in terms of time steps, number of actions taken, or CPU time.

(4) **Asymptotic adaptation performance ($S_{\textbf{post-novelty}}$)**: This metric measures the post-novelty adaptation performance by the agent. This is the success rate in the post-novelty scenario when the convergence criterion is met.

(5) **Post-Adaptation Efficiency (Δt)**: This metric quantifies the agent's policy efficiency after adjusting to novelty relative to its performance before the novelty. Specifically, it captures the potential for beneficial novelties that enable the agent to find task shortcuts. The metric is defined as:

$$\Delta t = t_{\text{pre-novelty}} - t_{\text{post-novelty}}$$

where $t_{\text{pre-novelty}}$ is the average time the agent takes to solve the task before encountering the novelty, and $t_{\text{post-novelty}}$ is the average time post-novelty adaptation.

[3]An epoch is the number of episodes/timesteps of training after which we evaluate the agent. In episodic tasks, the agent is provided a quota of timesteps to finish an episode.

6 EXPERIMENTS

6.1 Task

Pogostick task. The task is a Minecraft-inspired crafting task as illustrated in the running example (Section 3.1). The goal of the agent is to craft a pogostick while collecting resources and crafting various intermediary items. We simplified the task to make the task more tractable for a reinforcement learning agent. In the simplified task, the end goal of the agent remains to craft a pogostick while standing in front of the crafting table. The agent starts with a few items already in its inventory, such as a tree tap and iron pickaxe, and the steps required in order to achieve its goal are: (1) approach a block of platinum and (2) break it with the iron pickaxe; (3) approach a block of diamond and (4) break it with the iron pickaxe, (5) craft a plank, then (6) craft a stick, (7) select the tree tap and (8) collect rubber from an oak log while in front of the oak log, (9) approach the trader and (10) trade platinum for titanium, (11) approach the crafting table and (12) craft a block of diamond, and finally (13) craft a pogostick in front of the crafting table.

6.2 Noveltiesw

We implement a total of 12 transformations of the environment to test algorithms and frameworks for novelty handling in open-world learning These transformations comprise various aspects of task-solving and robust testing abilities.

6.2.1 Detrimental. A novelty is considered detrimental to an agent if it induces a change that hinders the agent from fulfilling the task for which it was designed. For example, if a tree in the environment can only be broken using a novel entity axe, and the agent has never utilized an axe to break trees. In that case, this transformation is considered detrimental to the agent's task in the environment.

6.2.2 Beneficial novelties. A novelty in the environment is considered beneficial for the agent if it induces a change that enhances the agent's ability to solve its task efficiently under a suitable performance metric (e.g. total reward, number of timesteps, etc.).

6.2.3 Nuisance novelties. A novelty in the environment is considered nuisance for the agent if it does not affect the agent's task or the agent's representation of the world. We selected 5 environmental

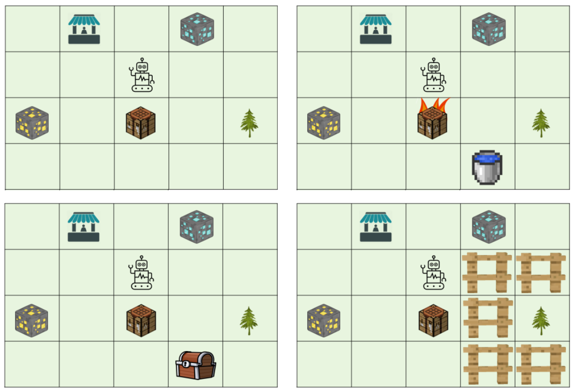

Figure 5: Illustration of the (clockwise) pre-novelty environment, fire novelty, fence novelty and chest novelty.

transformations for evaluations (details in Appendix E)

(1) Axe: In this transformation, the tree is unbreakable unless an axe is used to break it. (*Detrimental*)
(2) Chest: A chest is placed in the gridworld, and a new action `approach plastic chest` appears in the agent's action set. If the agent uses the `collect` action while standing in front of the *chest*, its inventory is filled with all the ingredients necessary to craft a *pogostick*. (*Beneficial*)
(3) Trader: One grid cell must be between the agent and a *trader* in order for the agent to be able to execute the `trade` action with the trader. (*Detrimental*)
(4) Fence: All oak logs in the gridworld are surrounded by a *fence* on the neighboring cells. The agent must break the *fence* first in order to access the *oak log*. (*Detrimental*)
(5) Fire: The crafting table is set on fire, and a water bucket is placed in the environment. The agent must first collect the water bucket and use it to put out the fire before using the crafting table for any crafting. (*Detrimental*)

6.3 Agent architectures

In order to develop agent architectures, we adapted a few existing architectures for novelty handling. The architectures ranged from learning approaches to neurosymbolic approaches and were adapted with sophisticated exploration methods. The modular nature of our proposed ecosystem helps in adapting and developing these hybrid architectures. Mainly, we had two approaches transfer learning and hybrid neurosymbolic approach.

Hybrid planning & learning approach: The hybrid planning and learning method was a direct implementation of [8]. The method assumes the pre-novelty task domain to be defined using PDDL. After novelty injection, if the agent cannot solve the task due to action execution failure, the method instantiates a learning (RL) problem. The goal of the RL problem is to find a plannable state. The plannable state is either a state that satisfies the failed operator's effects or can help the agent jump ahead in the plan. The instantiated learning problem can be solved by any off-the-shelf RL algorithm. We used PPO [20] as the RL agent. We also adapted the Intrinsic Curiosity Module (ICM) [17] for robust exploration. The modularity of NovelGym played a crucial role in the easy implementation of such a complicated agent architecture.

Transfer RL Due to the complex nature of the task, it was challenging to train an RL agent to solve it. We used a dense reward function to train the RL agent for the pre-novelty task (details in Appendix D) The RL agent reached comparable performance to a planning agent after 4M timesteps. With novelty injection, we expanded the agent's action and observation space, transferring pre-novelty task weights. Our architecture's flexibility facilitated automatic expansion on-the-fly. The benchmarked agent architectures include:

- RapidLearn^{+}(PPO+ICM): Hybrid planning and learning using PPO as the RL algorithm integrated with ICM.
- RapidLearn(PPO): Hybrid planning and learning with PPO.
- Transfer RL(PPO)+ICM: Transfer learning with ICM.
- Transfer RL (PPO): Transfer learning using PPO.

The agent's subsymbolic observation space is a LIDAR-like sensor that emits beams for every entity in the environment at incremental angles of $\frac{\pi}{4}$ to determine the closest entity in the angle of beam (similar illustration shown in Figure 2 (left)). The size of lidar sensor observation is $8 \times |\mathcal{E}|$, where $\mathcal{E}$ is the set of entities in the environment. The observation space is augmented by additional sensors that observe the agent's contents of inventory and entity selected by the agent. The agent's symbolic description was represented using PDDL, where entities were described with a type hierarchy, and predicates represented the relations between the types. The actions were described using preconditions and effects (details in Appendix G). We also implement an image-based local view representation. The local view representation is a one-hot vector encoding of all the entity types in the world in a grid of $n \times n$ around the agent (details in Appendix C) The agent has primitive actions such as move-forward, turn-left, etc., and parameterized actions such as approach<entity>, select <entity>, etc. (details in Appendix B) The action space size for the pre-novelty task is 28.

7 RESULTS & DISCUSSION

The results of our experiments are detailed in Table 2. Listed in the table are the mean and standard deviation of the metrics across all seeds. Notably, the metric $S_{\text{pre-novelty}}$ consistently scored perfectly. To achieve this, the RL agents underwent training of 4 million timesteps. Upon introducing novelty, there was a sharp performance decline across all agents for every novelty type, with the exception being the "chest" novelty. The chest novelty, being *beneficial* to the agent in the given environment, does not interfere with the task. Hence, the agent's success rate remains unaffected. The hybrid agent, given its design, does not adapt to novelties unless an execution failure is detected. Therefore, for the hybrid agent, this is not a novelty. The Δt metric (lower the better), which measures adaptation efficiency, highlights the superior performance of the transfer RL agents in adapting to "chest" novelty. This performance can be attributed to the exploratory nature of the RL algorithms. The I_{novelty} metric (lower the better) was computed by measuring the performance of each agent before training for adaptation for the novelty. Its values reveal that learning-based strategies are more sensitive to novelty introduction. However, hybrid models can

Novelty	Agent	$S_{\text{pre-novelty}} \uparrow$	$I_{\text{novelty}} \downarrow$	$T_{\text{adapt}} \downarrow$	$S_{\text{post-novelty}} \uparrow$	$\Delta t \downarrow$
Axe	RapidLearn^{+}(PPO+ICM)	1 ± 0	0.83 ± 0.205	71040 ± 17413	1.0 ± 0.0	122.7 ± 15.61
	RapidLearn (PPO)	1 ± 0	0.69 ± 0.207	68160 ± 17934	1.0 ± 0.0	116.9 ± 9.88
	Transfer RL (PPO)+ICM	1 ± 0	1.0 ± 0.0	169440 ± 97949	0.97 ± 0.021	61.0 ± 8.95
	Transfer RL (PPO)	1 ± 0	1.0 ± 0.0	114240 ± 19651	0.97 ± 0.018	63.1 ± 8.17
Chest	RapidLearn^{+}(PPO+ICM)	1 ± 0	–	–	–	–
	RapidLearn (PPO)	1 ± 0	–	–	–	–
	Transfer RL (PPO)+ICM	1 ± 0	0.0 ± 0.004	24000 ± 0	1.0 ± 0.0	−3.0 ± 0.78
	Transfer RL (PPO)	1 ± 0	0.01 ± 0.009	24000 ± 0	0.98 ± 0.011	−3.5 ± 3.13
Trader	RapidLearn^{+}(PPO+ICM)	1 ± 0	0.45 ± 0.069	102720 ± 20724	0.95 ± 0.017	126.9 ± 9.97
	RapidLearn (PPO)	1 ± 0	0.5 ± 0.07	96480 ± 21191	0.96 ± 0.02	122.8 ± 8.93
	Transfer RL (PPO)+ICM	1 ± 0	0.96 ± 0.108	227040 ± 112120	0.99 ± 0.011	93.6 ± 12.05
	Transfer RL (PPO)	1 ± 0	0.94 ± 0.128	350880 ± 237039	0.96 ± 0.025	89.3 ± 7.87
Fence	RapidLearn^{+}(PPO+ICM)	1 ± 0	0.38 ± 0.102	69120 ± 15657	0.96 ± 0.02	161.6 ± 7.6
	RapidLearn (PPO)	1 ± 0	0.41 ± 0.052	66400 ± 20999	0.95 ± 0.008	168.7 ± 6.15
	Transfer RL (PPO)+ICM	1 ± 0	1.0 ± 0.0	854400 ± 52974	0.93 ± 0.02	73.4 ± 4.7
	Transfer RL (PPO)	1 ± 0	1.0 ± 0.0	701760 ± 100589	0.92 ± 0.024	90.9 ± 9.49
Fire	RapidLearn^{+}(PPO+ICM)	1 ± 0	1.0 ± 0.0	263520 ± 81979	0.95 ± 0.02	133.2 ± 21.38
	RapidLearn (PPO)	1 ± 0	1.0 ± 0.0	252000 ± 49663	0.95 ± 0.016	142.5 ± 13.71
	Transfer RL (PPO)+ICM	1 ± 0	1.0 ± 0.0	372960 ± 198096	0.94 ± 0.016	105.3 ± 9.85
	Transfer RL (PPO)	1 ± 0	1.0 ± 0.0	127200 ± 37932	0.96 ± 0.029	103.3 ± 13.22

Table 2: Evaluations of the agents across five novelty scenarios. ↑ indicates higher value is better, ↓ indicated lower value is better and – indicates that the evaluated novelty is not a novelty for the corresponding agent.

sometimes exhibit marginally superior immediate adaptation upon the introduction of novelty. This might be due to their targeted novelty adaptation approach. We can observe from the results that fire novelty resulted in the sharpest decline in the performance of all the agents. This shows that the adaptation of this novelty based on the agents we evaluated is relatively harder than others. The correspondence of the I_{novelty} metric to the higher values of T_{adapt} can be observed in all the novelty and agents cases. For all the novelties, hybrid agents surpassed transfer learning methods in the T_{adapt} metric. This superiority may arise from hybrid architectures' focused learning and their ability to effectively reuse knowledge. Furthermore, we observed that the inclusion of ICM enhances adaptation in some cases. However, in some cases, we can see that the ICM approach, especially when adapted to the transfer learning agent, deteriorates the performance (fire novelty). The variance in chest novelty in the T_{adapt} is 0 because the agent did not achieve a low success rate to adapt. Therefore, it was not trained based on a convergence but rather a fixed set of episodes. $S_{\text{post-novelty}}$ metric results do not show perfect scores for some agents because the agents were trained to satisfy convergence criteria. They may reach perfect scores if trained longer.

Our results highlight essential elements for crafting AI agents capable of adeptly handling novelties. Specifically, the T_{adapt} metric reveals that even with advanced hybrid methods, a significant number of environmental interactions are needed for agents to recalibrate to their original performance levels after facing novelty. In real-world scenarios, especially in robotics, the opportunity for such extensive interactions is limited. This underscores the importance of our current domain and architectural approach, positioning it as a promising avenue for furthering open-world learning research.

8 CONCLUSION & FUTURE WORK

We introduced NovelGym, a flexible platform tailored for the implementation and injection of novelties and easy task creation in gridworld environments. We highlighted the modularity of our proposed ecosystem, substantiated by the integration of multiple novelties and the implementation of complex agent architectures. To further support the evaluation of novelty-aware open-world agents, we proposed an evaluation protocol complemented by evaluation metrics. Our empirical results offer insights into the performance dynamics of various agent architectures in the face of multiple novelties, evaluated against the backdrop of our introduced metrics. Our benchmarking data also presents the intricacies and challenges of open-world learning. Building on the strengths of NovelGym, the platform's adaptable and robust architecture presents a ground for many research directions. A possible direction can be the procedural and automatic generation of novelties, paired with an emphasis on continual agent learning. As a part of future work, we aim to expand NovelGym's capabilities to fully embrace multi-agent dynamics for deeper collaborative and competitive learning paradigms. We would also like to extend the environment to support human-in-the-loop learning, as novelty handling could benefit by demonstrations through humans and incorporating those into learning.

ACKNOWLEDGMENTS

This work was funded by DARPA grant W911NF- 20-2-0006. We thank the anonymous reviewers for their valuable comments that improved the paper. We also thank Arvind Pillai for his contribution in helping develop the code base in the earlier stages of the project.

REFERENCES

[1] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. 1998. Pddl| the planning domain definition language. *Technical Report, Tech. Rep.* (1998).

[2] Jonathan Balloch, Zhiyu Lin, Mustafa Hussain, Aarun Srinivas, Robert Wright, Xiangyu Peng, Julia Kim, and Mark Riedl. 2022. Novgrid: A flexible grid world for evaluating agent response to novelty. *arXiv preprint arXiv:2203.12117* (2022).

[3] Jonathan Balloch, Zhiyu Lin, Robert Wright, Xiangyu Peng, Mustafa Hussain, Aarun Srinivas, Julia Kim, and Mark O Riedl. 2023. Neuro-Symbolic World Models for Adapting to Open World Novelty. *arXiv preprint arXiv:2301.06294* (2023).

[4] Terrance E Boult, Nicolas M Windesheim, Steven Zhou, Christopher Pereyda, and Lawrence B Holder. 2022. Weibull-Open-World (WOW) Multi-Type Novelty Detection in CartPole3D. *Algorithms* 15, 10 (2022), 381.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[6] Theresa Chadwick, James Chao, Christianne Izumigawa, George Galdorisi, Hector Ortiz-Pena, Elias Loup, Nicholas Soultanian, Mitch Manzanares, Adrian Mai, Richmond Yen, et al. 2023. Characterizing Novelty in the Military Domain. *arXiv preprint arXiv:2302.12314* (2023).

[7] Yan Ding, Xiaohan Zhang, Saeid Amiri, Nieqing Cao, Hao Yang, Chad Esselink, and Shiqi Zhang. 2022. Robot task planning and situation handling in open worlds. *arXiv preprint arXiv:2210.01287* (2022).

[8] Shivam Goel, Yash Shukla, Vasanth Sarathy, Matthias Scheutz, and Jivko Sinapov. 2022. RAPid-Learn: A Framework for Learning to Recover for Handling Novelties in Open-World Environments. In *IEEE International Conference on Development and Learning, ICDL 2022, London, United Kingdom, September 12-15, 2022*. IEEE, 15–22. https://doi.org/10.1109/ICDL53763.2022.9962230

[9] Shivam Goel, Gyan Tatiya, Matthias Scheutz, and Jivko Sinapov. 2021. Novelgridworlds: A benchmark environment for detecting and adapting to novelties in open worlds. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.

[10] Jörg Hoffmann. 2003. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *Journal of artificial intelligence research* 20 (2003), 291–341.

[11] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, Margarita P Castro, Ethan Waldie, and Sheila A McIlraith. 2023. Learning reward machines: A study in partially observable reinforcement learning. *Artificial Intelligence* 323 (2023), 103989.

[12] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. 2022. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research* 75 (2022), 1401–1476.

[13] Matthew Klenk, Wiktor Piotrowski, Roni Stern, Shiwali Mohan, and Johan de Kleer. 2020. Model-Based Novelty Adaptation for Open-World AI. In *International Workshop on Principles of Diagnosis (DX)*.

[14] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems* 29 (2016).

[15] Faizan Muhammad, Vasanth Sarathy, Gyan Tatiya, Shivam Goel, Saurav Gyawali, Mateo Guaman, Jivko Sinapov, and Matthias Scheutz. 2021. A novelty-centric agent architecture for changing worlds. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 925–933.

[16] Fabio Pardo. 2020. Tonic: A deep reinforcement learning library for fast prototyping and benchmarking. *arXiv preprint arXiv:2011.07537* (2020).

[17] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*. PMLR, 2778–2787.

[18] Vasanth Sarathy, Daniel Kasenberg, Shivam Goel, Jivko Sinapov, and Matthias Scheutz. 2020. Spotter: Extending symbolic planning operators through targeted reinforcement learning. *arXiv preprint arXiv:2012.13037* (2020).

[19] Vasanth Sarathy and Matthias Scheutz. 2018. MacGyver Problems: AI Challenges for Testing Resourcefulness and Creativity. *Advances in Cognitive Systems* 6 (2018). https://hrilab.tufts.edu/publications/sarathy2018MacGyverACS.pdf

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

[21] Tom Silver and Rohan Chitnis. 2020. PDDLGym: Gym environments from PDDL problems. *arXiv preprint arXiv:2002.06432* (2020).

[22] Roni Stern, Wiktor Piotrowski, Matthew Klenk, Johan de Kleer, Alexandre Perez, Jacob Le, and Shiwali Mohan. 2022. Model-Based Adaptation to Novelty for Open-World AI. In *Proceedings of the ICAPS Workshop on Bridging the Gap Between AI Planning and Learning*.

[23] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. 2021. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 15032–15043.

[24] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560* (2023).

[25] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. 2022. Tianshou: A Highly Modularized Deep Reinforcement Learning Library. *Journal of Machine Learning Research* 23, 267 (2022), 1–6. http://jmlr.org/papers/v23/21-1127.html