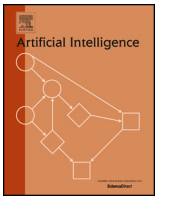
Artificial Intelligence 331 (2024) 104111

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



A neurosymbolic cognitive architecture framework for handling novelties in open worlds ☆

Shivam Goel [a,*,1], Panagiotis Lymperopoulos [a,*,1], Ravenna Thielstrom [a], Evan Krause [a], Patrick Feeney [a], Pierrick Lorang [a,c], Sarah Schneider [a,c], Yichen Wei [a], Eric Kildebeck [b], Stephen Goss [b], Michael C. Hughes [a], Liping Liu [a], Jivko Sinapov [a], Matthias Scheutz [a,*]

[a] Tufts University, 419 Boston Ave, Medford, 02155, MA, USA
[b] The University of Texas at Dallas, 800 W Campbell Rd, Richardson, 75080, TX, USA
[c] AIT Austrian Institute of Technology GmbH, Center for Vision, Automation & Control, Giefinggasse 4, Vienna, 1210, Austria

ARTICLE INFO

Keywords:
Open-world novelty
Cognitive architecture
Learning and perception
Creative problem solving

ABSTRACT

"Open world" environments are those in which novel objects, agents, events, and more can appear and contradict previous understandings of the environment. This runs counter to the "closed world" assumption used in most AI research, where the environment is assumed to be fully understood and unchanging. The types of environments AI agents can be deployed in are limited by the inability to handle the novelties that occur in open world environments. This paper presents a novel cognitive architecture framework to handle open-world novelties. This framework combines symbolic planning, counterfactual reasoning, reinforcement learning, and deep computer vision to detect and accommodate novelties. We introduce general algorithms for exploring open worlds using inference and machine learning methodologies to facilitate novelty accommodation. The ability to detect and accommodate novelties allows agents built on this framework to successfully complete tasks despite a variety of novel changes to the world. Both the framework components and the entire system are evaluated in Minecraft-like simulated environments. Our results indicate that agents are able to efficiently complete tasks while accommodating "concealed novelties" not shared with the architecture development team.

1. Introduction

Traditionally, AI research has focused on agents operating in "closed worlds" where all task-relevant concepts are assumed to be known in advance and designers can utilize this knowledge to construct specific algorithms based on this information [6,21]. Agents might still have to learn about instances of these concepts and their distributions, and maybe even cope with out-of-distribution cases, but at least they can assume that no *conceptual changes* will occur to task-relevant aspects that they would have to accommodate in

☆ This article belongs to Special Issue: Open-World AI.

* Corresponding authors.
E-mail addresses: Shivam.Goel@tufts.edu (S. Goel), plympe01@tufts.edu (P. Lymperopoulos), Matthias.Scheutz@tufts.edu (M. Scheutz).

[1] These authors contributed equally.

https://doi.org/10.1016/j.artint.2024.104111
Received 2 February 2023; Received in revised form 30 December 2023; Accepted 4 March 2024

Available online 15 March 2024

Fig. 1. An example of the gridworld environment. The novel actor is highlighted in blue, while the rest of the environment is known to the agent. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

order to complete their task (e.g., a self-driving car having to discover the concept of a car ferry which it would have to take to cross the river in order to get to its goal location as the bridge was just closed for traffic).

The transition from closed to open worlds thus necessitates that artificial agents be able to handle task-relevant novelties as they present themselves during task performance. While some of the unknown aspects of the environment might not have any impact on the agent's actions, others might be preventative in that without "accommodating" them, the agent can no longer accomplish its task. While accommodation does not necessarily require that the agent be able to detect the unknown aspect (e.g., a detour on the road leading directly onto the ferry), in general detection, and in some cases explicit characterization, of the novelty will often be a critical component (e.g., no detour, only a road sign suggesting to take the ferry).

The goal for the development of novelty-aware agents thus is to integrate methods into the agent architecture that allow the agent to detect novelties and accommodate them. We propose a hybrid symbolic-subsymbolic inference-based approach that at its core uses symbolic and subsymbolic statistical inferences to make predictions about possible and likely world states. When these predictions fail, the agent assumes that its knowledge about the world is incomplete and might have to be augmented. Depending on the impact of the prediction failure on its task performance, the agent might decide to continue with its task and explore the novelty later, or it might immediately engage in a comprehensive exploration process during which it attempts to acquire additional knowledge to be able to handle the novelty. Depending on the novelty, the agent might be able to generate an explicit symbolic characterization of the novelty that will allow it to reason and plan with it, or it might only have an implicit classification that is sufficient for triggering actions that utilize or avoid it.

Our contributions in this paper are: (1) An agent-centric categorization of novelties for goal-oriented planning agents, (2) a hybrid architectural approach that combines symbolic planning and counterfactual reasoning with reinforcement learning and deep computer-vision techniques to perform novelty detection and accommodation; (3) domain-general algorithms for novelty handling and explorations using inference and reasoning methodologies that include machine learning techniques; and (4) comprehensive evaluations of the agent architecture that range from evaluations of individual components to system evaluations performed by a dedicated evaluation team on "concealed novelties" that were not shared with the algorithm and architecture development team.

2. What is a novelty?

We depart from the assumption that novelty is a property of objects as it is sometimes treated in the recent literature on open-world novelty (e.g., [6]), and instead view novelty as an *intrinsically agent-relative* concept, i.e., a relation between aspects of an environment and an agent's cognitive system. This is easy to see because what is a novel object or concept for one agent (e.g., like a mass spectrometer for a diabetes patient, say, because the agent has not encountered it before or cannot derive any knowledge about it) might not be novel for another agent (e.g., the doctor using it to determine the distribution of metabolites). Hence, when we talk about novelty in this paper, we always have an epistemic agent in mind for whom something is novel because the agent has not experienced it and cannot derive representations of it from its knowledge base (cp. [41]). To better illustrate what kinds of novelties an agent might encounter and when it should care about them—when it ought be able to detect and handle them, and maybe characterize them to be able to utilize them for their purposes in the future—we will use a simple gridworld environment, based on the popular Minecraft game (see Fig. 1) where the agent's task is to craft a *pogostick*. We will later use the very same environment for comprehensive evaluations of the proposed novelty-aware agent architecture.

The environment contains any number of objects such as *trees* on which the task-performing agent (just called "the agent") can perform actions such as breaking them into *logs*, which, in turn, can be crafted into *planks* and *sticks*. There is also a *crafting-table* that the agent can use to craft complex items such as a *tree-tap* and a *pogostick* using pre-defined *crafting recipes*. The agent can also *mine diamond* from *diamond ore*. And the environment has a *safe* which can be used to store items. In addition to objects, the environment can also have various *actors*, i.e., agent types such as *traders* which can *interact* and *trade* items, or the *pogoist*, an adversary which competes with the agent in collecting resources from the environment to craft a *pogostick*.

Now consider an agent that has knowledge of all aspects of this environment, either through past experience or through having been endowed with the knowledge *a priori*. In other words, nothing in the environment is *novel for the agent*. Suppose a change is introduced while the agent is performing its task, e.g., in the form of new objects, new environment dynamics, new relations, etc. In the environment in Fig. 1, a new *actor*, referred to as *supplier* (highlighted in blue) was introduced and may have unknown effects on the agent's ability to craft a *pogostick*.

Depending on the characteristics of the new actor and its relation to the agent, some aspects of it may be novel for the agent. For instance, the supplier may relate to objects in the world in previously unknown ways, it may be visually distinct from any actor known to the agent, or may behave differently than any other actor previously encountered by the agent. Regardless of the aspect of the supplier that is novel to the agent, detecting the novelty can be useful or even necessary if the supplier can affect the agent's ability to achieve its task goals. For example, it may be that

- without interacting with this new actor, it is no longer possible to obtain the required ingredients to craft the pogostick,
- crafting a pogostick is still possible without interacting with the supplier, but the supplier can still assist the agent in crafting the pogostick more efficiently, or
- the supplier has no effect on the agent's task and interaction with it offers no utility to the agent.

To determine which is the case, the agent needs to explore different interactions with the supplier.

In general, novelty-aware agents need to be able to identify situations that are unknown or inconsistent with their knowledge of the environment, and have strategies to explore and incorporate new knowledge from those explorations into their knowledge representations.

Novelty detection. Novelty detection is the process of identifying representations inconsistent with the agent's prior knowledge. Depending on the knowledge representations that the agent maintains, the process of detecting novelty can vary. For instance, the agent encountering the supplier may recognize it as a novel agent by its appearance, which differs from other agents it has encountered in the past.

Novelty accommodation. Novelty accommodation, then, is the process of gaining and incorporating new knowledge related to the novel aspect into an agent's knowledge representation. Accommodation might be imperative when the novelty negatively affects the agent's ability to achieve its task. For instance, after visually identifying the supplier, the agent might need to explore interactions with it to characterize the supplier's potential impact on its goals. Different accommodation strategies may be necessary depending on whether the supplier is beneficial or detrimental to the agent's goals and no accommodation is necessary if the supplier has no impact on them.

Designers of novelty-capable agents need, to some extent, anticipate novelties to bootstrap the agent to start up with some ontology that allows it to carve up the world into different categories. However, the central motivation for developing agent architectures and methods that detect, characterize, and accommodate novelty is that in open worlds, designers are not able to anticipate all possible states that might constitute novelties for the agent.

Finally, since novelty is an agent-relative concept, an agent might find many novelties that are trivial or not task-related. Furthermore, any given unknown element of the environment can represent a large number of novelties, as not only is the element itself unknown, but so may its properties, its relations to other elements, etc. In this work, our agent is task-oriented and so exhaustively searches for novelties for the sake of discovering them but only handles them when necessary. This also blocks it from discovering many useless novelties (e.g., different arrangements of trees constituting different spatial relationships). However, our architecture can, in principle, be used as an information-gathering agent if the goal is to discover as many novelties as possible.

3. Theoretical framework

We start with general definitions of a *task environment* with *agents* and *objects* in order to be able to formally define what we mean by "novelty": *a relation between the agent and aspects of the environment* (cp. to the definition in [41]). We then define different types of novelties based on how they impact an agent's task performance.

3.1. The formal agent-environment framework

We consider a task environment $\mathcal{E} = \langle E, \mathcal{U} \rangle$ to consist of a (not necessarily finite) set of environmental states E and a set $\mathcal{U}$ of time-indexed maps $\mathcal{U}_t : E \mapsto E$ for $t \in \mathbb{N}$ that defines the evolution of states, i.e., given a state $e \in E$, $\mathcal{U}_t(e)$ is the state of the environment $\mathcal{E}$ at time t. We can define a reachability relation $\mathcal{R}(x, y)$ *iff* $\exists t \mathcal{U}_t(x) = y$ indicating whether a state is reachable from another state, given the environmental dynamics. In other words, the environment may contain any number of *objects* and *agents* as part of its state (e.g., placed in locations for spatially extended environments) with agents differing from objects in that they have internal states and can cause state changes through internal (computational) processes while objects are inert without internal states or internal processes.

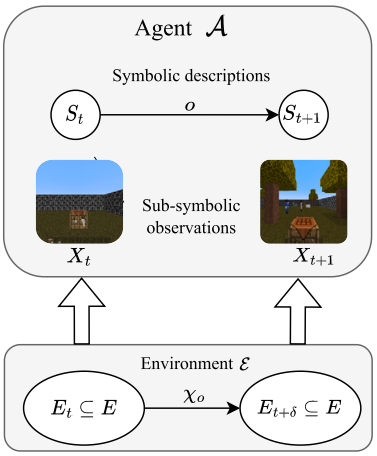


Fig. 2. Overview of the Agent-Environment Interface. The agent represents the environment state symbolically in the language $\mathcal{L}$. The agent does not have complete knowledge, so a state description can correspond to many true environment states. It also observes the environment subsymbolically. The agent uses an operator o of its planning domain to transition in the environment to a new state via the executor χ_o.

We define an agent $\mathcal{A} = \langle \zeta_{\mathcal{A}}, \Xi_{\mathcal{A}}, \mathcal{K}_{\mathcal{A}}, \vdash_{\mathcal{A}} \rangle$ to be equipped with a set of sensors $\zeta_{\mathcal{A}}$[2] that allow it to perceive the environment, a set of effectors $\Xi_{\mathcal{A}}$[3] that allow it to act upon the environment, a knowledge repository $\mathcal{K}_{\mathcal{A}} = (KB_{\mathcal{A}}, \Theta_{\mathcal{A}})$ (potentially empty) that can hold different representations, e.g., symbolic descriptions of E in some formal language or "subsymbolic representations" like sensory or effector states. The knowledge base $KB_{\mathcal{A}}$ is symbolic and stores knowledge explicitly in terms of facts and rules in a formal language. On the other hand, $\Theta_{\mathcal{A}}$ stores knowledge about subsymbolic and neural representations in a distributed manner, such as in the weights of a neural network. The agent also has an internal inference operator $\vdash_{\mathcal{A}}$ that enables it to perform manipulations of its knowledge representations, e.g., to perform different types of inference for different knowledge modalities in $\mathcal{K}_{\mathcal{A}}$. For $KB_{\mathcal{A}}$, we define $\vdash_{\mathcal{A}}$ to represent a first-order prover that can derive new facts from the $KB_{\mathcal{A}}$, while for $\Theta_{\mathcal{A}}$ we define it to represent a neural inference algorithm. In general, at a given time t, the agent's sensors will generate agent-internal representations $X_{\mathcal{A}}$ of partially observed environment states in E and the agent's effectors $\Xi_{\mathcal{A}}$ at time t, given the environmental state, will cause an update to the environmental state at time $t+\delta$, thus implicitly defining a set of "actions" $A_{\mathcal{A}}$ that specify how the agent can affect the environment. Actions can be triggered by the agent through its internal inference algorithms (the "agent function"). A note on notation: We will drop the agent subscript going forward if it is clear from the context which agent we are referring to.

While the above definitions allow for a diverse array of agent models, we will consider *cognitive agents* for the rest of the paper that can reason and plan using a first-order language (FOL) $\mathcal{L}$ while also representing aspects of the environment subsymbolically (e.g., visually in images) and reasoning with those representations non-symbolically. The agent has a set of $n-ary$ predicates P (denoting relations or atoms) over a set of variables V and constants C (with constants, for example, denoting objects or locations in the environment). An atom over the language $\mathcal{L}$ is represented by $p(u_1, \ldots, u_n), u_i \in C \cup V$ and $\neg p(u_1, \ldots, u_n)$ its negation. If $u_i \ldots u_n \in C$, then $p(u_1, \ldots, u_n)$ is a *grounded atom*. $\mathcal{L}$ can then be used to express knowledge about $\mathcal{E}$ such as facts, rules, and relations between states in E.

Agents can perform tasks in the environment using explicit task descriptions defined in a "planning domain" represented as $\Sigma = \langle S, \mathcal{O} \rangle$, where S is a (partial) description in $\mathcal{L}$ of states in E and $\mathcal{O}$ is a set of operators expressed in $\mathcal{L}$, with preconditions ψ_o and effects ω_o, corresponding to (sequences of) actions the agent can perform. Each planning domain operator in $\mathcal{O}$ is associated with a "lower-level action executor" $\chi \in \mathcal{X}$ defined as a triplet of functions $\langle \gamma, \pi, \beta \rangle$ over S. The functions γ and β indicate which state descriptions in S are the acceptable start and end state descriptions for the executor χ and the function $\pi : S \mapsto A$ is a policy indicating which actions are taken in each state. The state description of a state $s \in S$ includes information about all the objects O and actors Φ in the domain.

Fig. 2 illustrates the distinction between state descriptions, environmental states, and subsymbolic state observations. Symbolic state descriptions do not enjoy a one-to-one relation to environment states, as the agent may not be omniscient. As a result, a particular state description may correspond to a set of possible environment states. Similarly, subsymbolic state observations X (e.g., images) also do not fully describe the environment state. The agent uses planning operators $o \in \mathcal{O}$ to generate plans. The executor $\chi_o \in \mathcal{X}$ associated with o executes actions that, in accordance with the state transition relation $\mathcal{R}$, modify the environment state.

A planning task $T = \langle \Sigma, s_0, s_g \rangle$ defines a set of initial state descriptions s_0 and a set of goal state descriptions s_g using $\mathcal{L}$ in the planning domain Σ. The agent $\mathcal{A}$, using state descriptions $s \in S$, along with operators $o \in \mathcal{O}$, can use inference $\vdash$ to produce a plan $\mathcal{P} = [o_1, \ldots, o_{|\mathcal{P}|}]$ that solves the planning task T, indicated by $\mathcal{P} \rhd T$, if one exists and the agent's algorithm can find it. We consider

[2] We consider sensors to be maps from the set of environment states E and internal (sub)symbolic state representations to internal (sub)symbolic state representations. This distinction allows for internal states that influence perception.

[3] We consider effectors to be maps from the internal (sub)symbolic states and environmental states to the internal (sub)symbolic states to environmental states.

that $P \rhd T$ if executing the actions (through executors) corresponding to the operators in $\mathcal{P}$ in states consistent with descriptions in s_0 will take the agent to states consistent with descriptions in s_g.

3.2. Novelties

Given an environment $\mathcal{E}$, an agent $\mathcal{A}$, and a planning task T, the agent needs to complete the task T. However, it is often the case that $\mathcal{A}$ does not have complete knowledge of $\mathcal{E}$ as there may be aspects of environment states in E that cannot be derived from the agent's knowledge repository $\mathcal{K}$. As defined earlier, these elements constitute novelty for $\mathcal{A}$. Specifically, a representation ν of an aspect of an environment state E is a *novelty for the agent* $\mathcal{A}$ if $\mathcal{K} \nvdash \nu$ where the representation ν could be in $\mathcal{L}$ or a subsymbolic representation (e.g., an image). A set of novelty representations[4] is represented as $\mathcal{N}$. The process of incorporating novelty representations $\mathcal{N}$ in the agent's knowledge repository $\mathcal{K}$ is denoted as $KB \cup \mathcal{N}$ for symbolic representations and $\Theta \cup \mathcal{N}$ for subsymbolic representations of novelties.

The existence of a set $\mathcal{N}$ for the agent $\mathcal{A}$ can have varying effects on its ability to solve a task T and we will define some novelty types accordingly:

Definition 1. *[Prohibitive novelty] A novelty represented by ν is* prohibitive *(for agent $\mathcal{A}$ and task T) if for all plans $\mathcal{P}, KB \vdash \mathcal{P}, \mathcal{P} \not\rhd T$, but $\exists \mathcal{P}_\nu \rhd T$ and $KB \cup \{\nu\} \vdash \mathcal{P}_\nu$.*

In other words, prohibitive novelties for the agent $\mathcal{A}$ are aspects of the environment that the agent needs to represent and reason with to generate a successful plan.

Definition 2. *[Obstructive novelty] A novelty represented by ν is* obstructive *(for agent $\mathcal{A}$ and task T) if it causes the execution of an executor $\chi \in \mathcal{X}$ to fail.*

Obstructive novelties for the agent $\mathcal{A}$ thus may or may not impact the agent's task performance depending on whether the agent included the operator associated with the failing executor in its plan. Especially in cases where no other plan can be found, knowledge of the novelty might help the agent to either modify the failed executor or replace it with a new working executor.

Definition 3. *[Beneficial novelty] A novelty represented by ν is* beneficial *(for agent $\mathcal{A}$ and task T) if $\exists \mathcal{P}_\nu$ $KB \cup \{\nu\} \vdash \mathcal{P}_{\mathcal{N}}$ such that $\mathcal{P}_\nu \rhd T$ and $\forall \mathcal{P}, KB \vdash \mathcal{P}, \mathcal{P} \rhd T$, $|\mathcal{P}_\nu| < |\mathcal{P}|$.*

In other words, beneficial novelties for the agent $\mathcal{A}$ are novelties whose representations can help the agent solve the planning task T in fewer steps than its original plan $\mathcal{P}$ (alternatively, when action costs are defined for plans, such novelties would result in lower-cost plans).

Definition 4. *[Nuisance novelty] A novelty represented by ν is a* nuisance *(for agent $\mathcal{A}$ and task T) if*
$\forall \mathcal{P}_\nu$ $KB \cup \{\nu\} \vdash \mathcal{P}_\nu$ such that $\mathcal{P}_\nu \rhd T$, $\exists \mathcal{P} \rhd T, KB \vdash \mathcal{P}$ and $|\mathcal{P}_\nu| \geq |\mathcal{P}|$.

In other words, nuisance novelties for an agent $\mathcal{A}$ do not contribute to task performance nor do they obstruct it; they may only cause higher performance costs (in terms of plan length, or plan costs if a notion of action cost is defined).

It should be clear from the above definitions that all the novelty types are agent-relative and thus depend on a particular agent's makeup, including sensors, effectors, knowledge representations, etc. Hence, what is a prohibitive novelty for one agent, might be a nuisance novelty for another agent or no novelty for yet another one.

The goal for our agent design then is to handle novelties, i.e., when an agent $\mathcal{A}$ encounters a novelty represented by ν, it needs to assess how the novelty affects its task-solving ability (i.e., prohibitive, obstructive, beneficial, nuisance) and experiment with the novelty to expand or correct its knowledge repository $\mathcal{K}$. In cases where the agent's goal is knowledge discovery, exploration strategies informed by the detected novelty can be employed to further expand $\mathcal{K}$. In goal-oriented settings where task completion takes precedence, the potentially intractable number of novelties motivates limiting exploration to cases where task completion is compromised, or when there are very strong environmental cues (i.e. passive detection of novelty). In those cases, any additional knowledge incorporated into $\mathcal{K}$ can then be used for inference to solve its task T.

4. An architectural framework for novelty handling

We introduce the architectural framework (depicted in Fig. 3) for developing novelty-aware agents composed of several components that, through their function and interactions, define the agent's knowledge repository $\mathcal{K}$, its inference algorithms $\vdash$, and operate its sensors ζ and effectors Ξ.

[4] Note that the agent's symbolic or subsymbolic representational repertoire is a hard limit for the agent's ability to capture novelties. We are not considering "growing agents" here that might be able to extend their cognitive system's representational formalism and capacity.

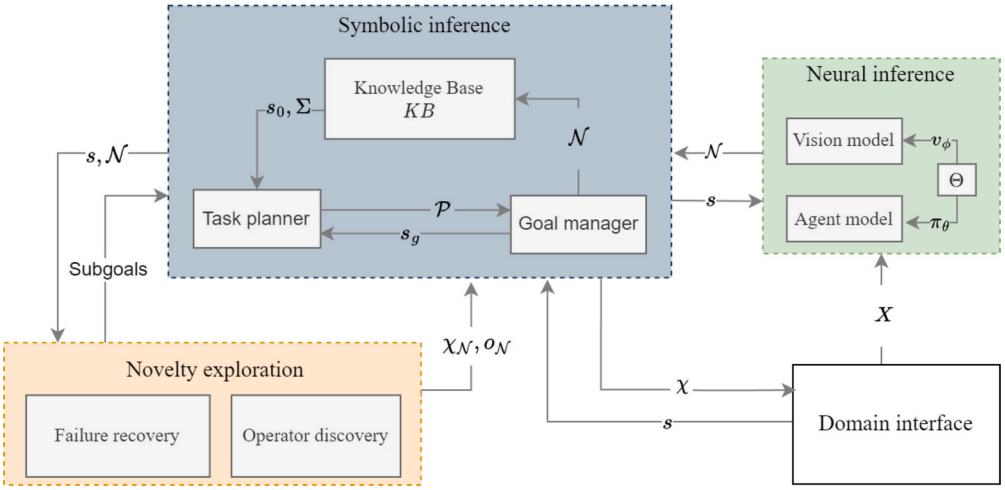


Fig. 3. High-level architecture diagram of the relevant components for novelty handling (highlighted in colored boxes) and the information flow among them (see text for details).

The agent has a DOMAIN INTERFACE component that mediates its interactions with the environment through the sensors ζ and effectors Ξ (this will typically be itself a set of components for different sensors and effectors but we are not concerned with those architectural details in this work). Sensory information is organized into symbolic state descriptions $s \in \mathcal{S}$ and passed on to other components for processing, including recognized actions of other actors in the environment. Additionally, subsymbolic perceptions X (e.g. images) acquired by the sensors are also made available by the DOMAIN INTERFACE.[5]

The knowledge base $KB \in \mathcal{K}$ stores facts using $\mathcal{L}$ about the environment as extracted from state description obtained from the domain interface, as well as the operators $\mathcal{O}$ available to the agent. The GOAL MANAGER and TASK PLANNER are the symbolic inference mechanisms of the architecture. The GOAL MANAGER detects novelty by comparing inferred expected state descriptions from rules and facts in KB to state descriptions obtained through the DOMAIN INTERFACE. The TASK PLANNER generates plans for the agent to execute in the environment. The GOAL MANAGER also detects novelty due to plan failure. Whenever novelty is detected, it is submitted to the KB for further inference and exploration.

The subsymbolic knowledge repository $\Theta \in \mathcal{K}$ is composed of the VISION MODEL and AGENT MODEL components, which make up the neural inference algorithms of the agent. The VISION MODEL receives subsymbolic state descriptions (e.g., images) and detects out-of-distribution samples using statistical methods. The AGENT MODEL receives symbolic state descriptions and, with statistical methods, monitors the behavior of other actors in the environment to detect when their behavior is inconsistent with the agent's expectations. When either component detects novelty, it sends a description of the novelty to the KB.

The NOVELTY EXPLORATION component is comprised of symbolic novelty exploration algorithms, as well as a reinforcement learner that can learn policies to form new executors if necessary. It accepts a description of the novelty and symbolic state from the KB. It returns sub-goals to the GOAL MANAGER that aim to explore the environment or specifications of operators that can be used to aid in solving task T. The operation of these components is described in detail in subsequent sections.

4.1. Symbolic inference

Symbolic inference is performed by the GOAL MANAGER and TASK PLANNER on facts and rules from the KB to solve task T and to detect novelties $\mathcal{N}$.

4.1.1. Knowledge base

The knowledge base $KB \in \mathcal{K}$ stores facts about symbolic state descriptions $\mathcal{S}$ using the language $\mathcal{L}$. It is populated with facts and rules defined *a priori* and through interaction with the environment. This may include objects, actors, functions, symbolic state representations, and a semantic type hierarchy used to describe planning tasks.

During execution, other components in the architecture may assert, retract or query facts in KB to detect novelty, create plans, or store new knowledge. The KB is part of the knowledge repository $\mathcal{K}$ against which novelties are detected. As a result, when information about a detected and explored novelty is asserted into KB, subsequent encounters with it will no longer be novel for the agent. This is not limited to novelties detected using symbolic inference, as novelties detected with neural inference are also eventually expressed in $\mathcal{L}$ and stored in KB.

[5] Ideally, all symbolic information such as state descriptions and recognized actions by other agents would be obtained from a vision or perceptual processing module, but because our focus is not on scene descriptions and action recognition from images, we allow the DOMAIN INTERFACE to already provide these pre-processed percepts.

4.1.2. Task planner

The TASK PLANNER is one of the two symbolic inference mechanisms utilizing the KB. Once a desired goal state description s_g is received from the GOAL MANAGER, the TASK PLANNER retrieves a planning domain definition Σ (operators, predicates, etc.) and an initial state description s_0 from the KB to form a planning task T and searches for a plan $\mathcal{P} \triangleright T$. Whether a plan is found or not, along with the plan itself, is then submitted to the GOAL MANAGER for further inference and execution.

4.1.3. Goal manager

The GOAL MANAGER is responsible for managing the agent's top-level goal(s), which can consist of multiple concurrent goals, and is responsible for detecting novelties in symbolic state descriptions. The component submits goals to the TASK PLANNER and performs inference on state descriptions, a process during which novelties of different types may arise due to planning failure, execution failure, or unexpected aspects of state descriptions. The general principle for detecting novelty with symbolic inference is predicting next environment states and comparing predictions with observed states. Once a novelty is detected, the GOAL MANAGER produces a description of it and asserts it to the KB. Depending on the novelty, the produced description contains information about the environment state, the aspects of it that are novel as well as any operators that may exhibit unexpected behavior. Depending on the novelty type, the NOVELTY EXPLORATION component (see Section 4.3) may be invoked. The GOAL MANAGER is also the primary component communicating with the DOMAIN INTERFACE.

Prohibitive novelty. Prohibitive novelties are most often encountered due to the failure of the TASK PLANNER. For a given goal, the GOAL MANAGER consults the TASK PLANNER for a plan $\mathcal{P}$ to execute. If the TASK PLANNER fails to find a plan for a particular goal, assuming that the goal is achievable based on the agent's knowledge, then the failure may be due to an prohibitive novelty.

Obstructive novelty. Obstructive novelty is most likely encountered due to plan execution failure. Once a plan sequence has been generated, the GOAL MANAGER enters an execution phase where it verifies that each operator's preconditions are met. In this inference task, the DOMAIN INTERFACE is consulted to verify that the preconditions ψ_o of an operator $o \in \mathcal{O}$ hold in the current symbolic description of the environment state. If inference on prior states indicates that all of the operator's preconditions ψ_o should be met, but they are not in the current state, this constitutes a novelty. If all preconditions are met, the executor χ_o of the operator is sent to and executed by the DOMAIN INTERFACE component. The agent may receive feedback through the DOMAIN INTERFACE component indicating the success or failure of the executor. In case of failure when success was anticipated, this again constitutes novelty. Additionally, obstructive novelty may occur even when an executor succeeds, if its effects are not consistent with the agent's expectations.

Beneficial and nuisance novelty. Success in the task does not eliminate the possibility of novelty. The GOAL MANAGER may still encounter beneficial or nuisance novelties during its inference of the expected state of the world. If the inferred state does not match the observed state of the world solicited from the DOMAIN INTERFACE, then that still constitutes a novelty, regardless of its effect on the task. Otherwise, plan execution continues until all operators in the plan have been executed.

In all cases where novelty is detected, a symbolic description of it is asserted to KB and sent to the NOVELTY EXPLORATION component, which utilizes different exploration policies depending on the way novelty is encountered. Novelty descriptions involve information about the aspects of the environment state that are novel such as novel objects and unexpected or missing operator effects. If the agent's goal is knowledge discovery, GOAL MANAGER can also enumerate sub-goals for the agent to achieve to test for novelty in different environment states or to further explore previously detected novelty. However, in goal oriented environments such extensive exploration is limited to novelties that prevent task completion.

4.2. Neural inference

The architecture's neural inference components are used to detect novelty in the subsymbolic state representations X as well as in statistical relationships between elements of the state descriptions S.

The knowledge repository $\Theta \in \mathcal{K}$ stores the agent's knowledge of these modalities in the form of machine learning model parameters. The first component for neural inference is VISION MODEL, which implements a visual novelty detector based on images from the DOMAIN INTERFACE. The second component of neural inference is an AGENT MODEL that models the behavior of other actors in the world to detect deviations from their known behaviors. If either module detects model deviations with sufficient confidence, a symbolic representation of the hypothesized novelty is submitted to the knowledge base for further inference and exploration. A general design principle for neural novelty detectors is an emphasis on caution: Given the distributional nature of neural models, we set conservative thresholds for novelty detections to minimize false positives. This is especially important when the agent's priority is task completion, as false positive detections can result in superfluous exploration that may hinder task performance.

4.2.1. Vision model

The VISION MODEL is responsible for detecting visual novelty. At every step, the DOMAIN INTERFACE provides a 2-dimensional color image of the agent's current view of the world. The VISION MODEL's task is to determine whether each new input image represents a plausible view of the known standard environment, or a different "novel" distribution. Some authors refer to similar problems as anomaly detection or out-of-distribution detection [47]. Visual novelties could include new object types, new agent appearances, or new backgrounds. Scene composition properties could also change (sizes, frequencies, relative locations, etc.).

To solve the detection problem, our VISION MODEL takes a deep autoencoder approach, following Abati et al. [1]. Deep autoencoders are often applied to visual novelty detection for their ability to learn effective representations of data through self-supervision [43]. The model consists of two component neural networks: an encoder e_ϕ with weights ϕ that maps image X to a code vector $z \in \mathbb{R}^C$, and a decoder $\mathrm{d}_{\phi'}$ with weights ϕ' that maps code vector z back to an image X. After a model is suitably trained (see below), given a new image $\tilde{X}$, the model assesses possible novelty via reconstruction error: $r(\tilde{X}; \phi, \phi') = ||\tilde{X} - d_{\phi'}(e_\phi(\tilde{X}))||_2$, which measures Euclidean distance between the input and its reconstruction. Intuitively, a well-trained model should have low reconstruction error for images that represent the normal environment used for training, while images containing novelty will yield higher error. The effectiveness of this novelty detection method is determined by the separability of the normal and novel reconstruction error distributions [45]. The VISION MODEL can pass the reconstruction error signal directly to the Symbolic Inference. Naturally, we can also apply a threshold τ to produce binary detection decisions, where "novelty" is called if $r(\tilde{X}) > \tau$ and "normal" otherwise. The value of τ can be selected on a validation set containing labeled examples of normal and novel data to maximize a performance metric of interest.

The VISION MODEL can be trained in advance on a dataset of images depicting the "normal" environment. Given N training images, we seek encoder and decoder weights ϕ, ϕ' that minimize the total reconstruction error: $\sum_n r(X_n; \phi, \phi')$. This can be solved via stochastic gradient descent [1], and the optimal weights ϕ, ϕ' stored within Θ for later processing.

If the VISION MODEL detects abnormal images, it generates a description $\mathcal{N}$ that is sent to KB. The description includes an identifier that allows the KB to associate a visually-detected novelty with the state description of the environment state in which it was observed. Since no additional information is extracted, it is difficult to accommodate novelties that are detected only visually.

4.2.2. Agent model

The AGENT MODEL is a crucial part of the architecture when operating in multi-agent environments. It uses neural inference over symbolic state descriptions and is responsible for maintaining knowledge about other actors. The aim of the AGENT MODEL is to evaluate whether facts about other agents contained in symbolic state descriptions are consistent with its knowledge of those agents' behavior. One major factor we consider here is the *type* (e.g. *supplier* or *pogoist* from the example in Section 2) of other agents.

Behavioral modeling with behavioral cloning. The main approach of the AGENT MODEL is to model other agents' behaviors via behavioral cloning (BC).[6]

Suppose other actors Φ identified in the state descriptions S have actor-types in a set Q, also provided in the state description. For each type $q \in Q$, the AGENT MODEL learns the policy $\pi(a|s, q)$ of an agent of type q. It is implemented with a neural network $\mathrm{nn}_\theta(s, q)$, which outputs the probability of actions in the action space A_q of the agent of type q. Here the input (s, q) is represented as a feature vector. If necessary, other representations such as graphs and text can also be consumed by neural networks. If complete knowledge of another actor's state is not known, then a pseudo-state s' can be inferred using knowledge in $\mathcal{K}$. Then the model becomes $\mathrm{nn}_\theta(s', q)$ and the action likelihood can be estimated. The model is trained via supervised learning using the true actions each actor took at that state (described in symbolic representations). The resulting model parameters θ are stored in Θ and used to evaluate action likelihoods during execution. The exact architecture, hyper-parameters, training procedure and feature representation is also application dependent and is discussed in the experiment section.

Unlikely action detection. During operation, the AGENT MODEL uses the learned neural network to monitor an actor's behaviors and detect unlikely actions. Our agent may encounter actors that deviate from their actor-type's policy. Such an event is inconsistent with the agent's $\mathcal{K}$ in the sense that the actors exhibit behavior that is unlikely under the actor policies encoded in Θ. Such novelty can be detected by evaluating the likelihood of observed actions using nn_θ. Focusing on a single actor of type $q \in Q$ and its action $a \in A_q$ from a symbolic state description $s \in S$, the likelihood under the learned model is computed by $\hat{l} = \mathrm{nn}_\theta(s, q)$. Then, $\hat{l}$ is compared to previously encountered values to decide if a deviates from the known distribution. For instance, if nn_θ is trained offline with a dataset of trajectories from the environment, then a portion of the dataset can be used for validation to set a threshold below which actions are considered inconsistent with the model for that actor type.

Actor type classification. Using the model nn_θ, the AGENT MODEL can also classify actors into the known actor types Q. During operation, a voting scheme over the Q known actor types is used to classify actors into types using their action likelihoods. The voting-based actor type classifier is described in Algorithm 1.

The predicted types can be compared with the types extracted from the symbolic description to determine if some actors deviate from their actor types.

If the AGENT MODEL detects novelties due to unexpected actor behavior, it can provide feedback to the rest of the architecture for accommodation. It produces a symbolic description ν that includes the state description in which novelty was detected, any unlikely actions detected for each actor in the environment and an indicator of the inferred agent-types using the vote-based classifier, if they deviate from those expected. The novelty description is submitted to the KB and used for accommodation if necessary.

[6] Behavioral cloning [4,11,3] is an offline reinforcement learning technique that learns policies from a dataset of actor trajectories. That dataset can be collected from repeated interactions with the environment and the policies are learned using supervised learning.

Algorithm 1 Voting-based actor classification.

Inputs:
$D = \{(s_i, a_i) | i = 1..M\}$ ▷ state and actions observations
nn_θ ▷ Learned actor policies
Q ▷ Set of known actor types
procedure:
$\mathbf{v} \leftarrow \mathbf{0}$, $\mathbf{v} \in \mathbb{N}_0^{|Q|}$ ▷ Vote vector
while not empty(D) **do**
  $(s, a) \leftarrow D.pop()$
  $b \leftarrow argmax_{q \in Q} [\text{nn}_\theta(s, q)]_a$ ▷ Vote for actor-type b
  $v_b \leftarrow v_b + 1$
end while
return $q \leftarrow argmax(\mathbf{v})$ ▷ Plurality Vote

4.3. Novelty exploration component

The NOVELTY EXPLORATION component receives symbolic descriptions of novelties from the GOAL MANAGER and is responsible for generating exploration strategies depending on the type of novelty encountered. These include heuristic search strategies on the symbolic descriptions of states (and on failed operators) as well as knowledge-guided reinforcement learning-based exploration to learn new executors for existing failed operators.

Knowledge discovery. If the NOVELTY EXPLORATION component receives a novelty description before the agent attempts to generate a plan, then the type of the novelty (i.e. how it may impact its ability to solve the task) has not yet been determined. In that case, a cursory knowledge discovery routine is utilized to gather information about the encountered novelty. The exploration strategy is dependent on the symbolic description of the novelty and can involve exploratory subgoals and operators related to the novelty.[7] To avoid taking too much time away from the agent's task, this routine is only given a limited time to run before the agent moves on and begins to plan for its main goal. If additional information about the novelty is gathered from knowledge discovery, it is appended to the symbolic novelty descriptions $\mathcal{N}$ and, if the task is interrupted later on, this information can be used during failure recovery (as described in the next section) or knowledge discovery can resume with any exploration strategies of the novelty that it previously did not have time for.

For instance, if a novel actor is encountered (see example in Section 2), the NOVELTY EXPLORATION component would submit subgoals to the GOAL MANAGER involving interactions with the novel actor. These subgoals are generated using the agent's type hierarchy: known operators applicable to known actors are used on the novel actor. Such interactions may reveal additional information about the novelty and aid in failure recovery depending on the novelty type.

4.3.1. Failure recovery

The FAILURE RECOVERY component is responsible for deploying recovery strategies for novelty accommodation and is invoked when novelty causes a planning or execution failure.[8] Depending on the novelty description it receives, it employs various recovery strategies to address the particular failure.

Fig. 4 (Left) illustrates the agent's recovery policies for different failure cases attributed to novelty. Special attention is paid when novelty is attributed to the effect failure of an operator. Fig. 4 (Right) shows in detail the strategies employed when an operator's known effects are inconsistent with the state description after its execution. Depending on the severity of the failure, the agent employs different recovery strategies. If the state description is consistent with only a subset of the operator's effects (partial effect failure), then the KB is updated with the new effects, and the agent attempts to replan. If instead, none of the operator's effects are observed (total effect failure), the agent attempts to discover new operators that may assist it in solving the task. This process involves hypothesizing new preconditions, a heuristic search over known operators to uncover unknown effects, and a reinforcement-learning-based exploration methodology that learns new executors for failed operators. These strategies may be executed in the order presented, or may be applied to novelties at any order depending on the domain implementation and novelty descriptions. Overall the failure recovery policies are general and flexible, as each may be involved in recovery from multiple novelty types and they can be composed to recover from complex failures.

The rest of this section discusses the use of these recovery policies in the context of the novelty types described in Section 3.

Recovery from prohibitive novelty. If the TASK PLANNER fails to produce a plan, then the NOVELTY EXPLORATION component follows its plan failure policy that employs knowledge discovery. It generates and executes sub-goals to acquire additional information about plan failure. The generated exploration subgoals attempt to prioritize operators that explore currently unobserved environment states to update (if necessary) the agent's prior knowledge about those states. If knowledge discovery fails to produce a plan, the NOVELTY

[7] For example, the operators can be selected using the agent's type hierarchy: If the novelty is a previously unknown property of a known object, then operators applicable to that object may be attempted. Similarly, if the novelty involves novel actors, appropriate operators may be invoked.

[8] It should be noted that the word failure is used to describe a failure in the agent's knowledge and inference to describe the environment, not necessarily a failure in the task.

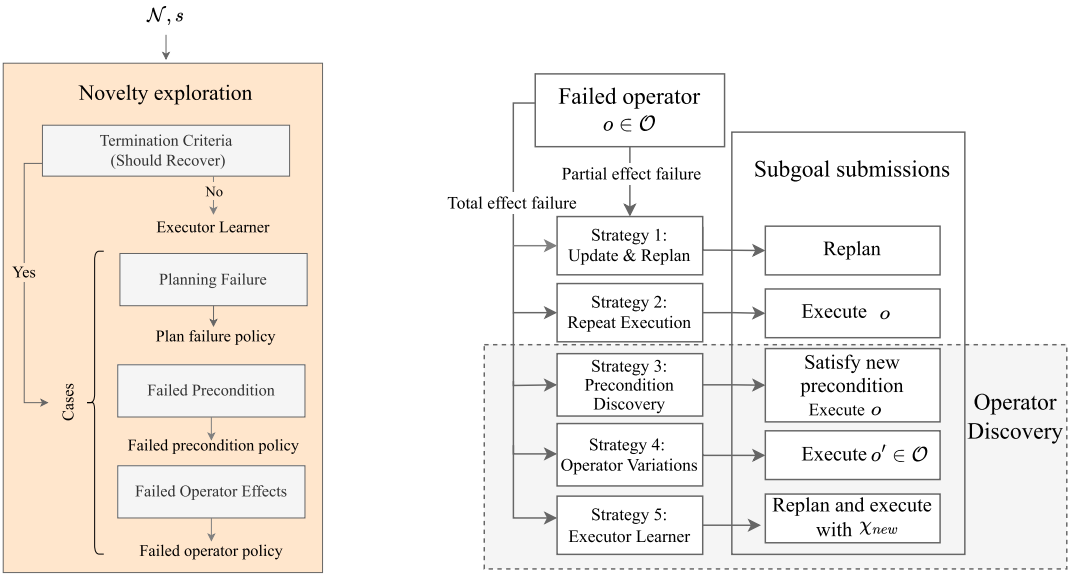


Fig. 4. Diagrammatic representation of the agent's recovery policies. (Left) The recovery policies initiated under different failure conditions. (Right) Outline of specific strategies of the failed operator policy (concerning the failed effects of the operator). Different recovery policies are initiated depending on how novelty is encountered, and individual strategies may apply to multiple novelty types.

EXPLORATION component utilizes the EXECUTOR LEARNER to discover states from which the agent can produce successful plans to solve the task.

Following the gridworld example from Section 2, consider a scenario in which the crafting table, an object essential to task completion, is not present in the symbolic state description. In that case, the agent fails to plan and needs to explore the environment to find alternate solutions for task completion. For instance, the agent may investigate ways to acquire a crafting table, such as, opening the safe, visiting other rooms or by interacting with other actors. The agent can then attempt to replan if a way to acquire a crafting table is discovered.

Recovery from obstructive novelty. The two conditions associated with obstructive novelty are handled using different exploration strategies.

Precondition failure. As discussed previously, novelty due to unmet preconditions indicates that the agent's knowledge of the environment state is inaccurate. Therefore, the recovery strategy updates the agent's KB with the accurate state description obtained through the DOMAIN INTERFACE. It then prompts the task planner to re-plan.

For instance, a novel actor in the environment may interfere with the agent's resource gathering by stealing resources before the agent can collect them but after forming a plan to do so. As a result, the preconditions for that operator no longer hold true (e.g., resources are no longer available). In this scenario, the agent should update its KB with the accurate state description and attempt to re-plan.

Partial Effect failure. If all the preconditions of an operator are met, the operator is executed, and the operator's effects are compared against the symbolic state description. If a subset of the effects are inconsistent, the GOAL MANAGER infers that the effect specification of the operator may be inaccurate. The FAILURE RECOVERY component then proceeds with recovery strategies as shown in Fig. 4 (right).

The failed effects recovery procedure attempts to repair an operator $o \in \mathcal{O}$ with unexpected effects ω. The agent applies strategy 1 in Fig. 4 (right), which first updates the operator description in the KB to be consistent with its observed effects and then attempts to re-plan.

Total effect failure. If instead there are no observed effects after executing the executor of an operator, the operator itself can be said to have failed. The NOVELTY EXPLORATION component has a series of strategies to repair the failed operator. The first strategy is the same as used for the partial effect failure: the agent updates the operator to reflect that no effects occur and replans. Another strategy (labeled as *Strategy 2* in Fig. 4 (right)) involves repeated attempts to execute the operator to account for instances of circumstantial failure. More substantial novelty exploration occurs when the agent utilizes the OPERATOR DISCOVERY component, which employs knowledge-guided search strategies to discover new operators to solve the task. The OPERATOR DISCOVERY component is discussed in detail in Section 4.3.2.

An example of total effect failure occurs in a more complex variant of the supplier novelty scenario discussed in Section 2, where the supplier offers to trade the agent a pogostick in return for all the usual ingredients that the agent would use to craft the pogostick. When the agent initially uses Knowledge Discovery to investigate the new actor in the world, it will interact with the *supplier*, receive the trade offer, and incorporate the trade operator into its knowledge base. Since this method of obtaining a pogostick is as efficient as simply crafting one (the agent needs to gather all the ingredients either way), the agent may ignore the supplier and continue with

its usual plan of crafting the pogostick. If the crafting operator fails when the agent attempts to craft, the novelty is determined as obstructive. When the agent attempts to re-plan as part of the Failed Effects recovery policy, it will use the knowledge gained from interacting with the *supplier* earlier to create a new plan to obtain the pogostick via the *supplier*.

Recovery from beneficial and nuisance novelties. Novelty due to unexpected operator effects is not necessarily obstructive. If there are different effects than expected, rather than no effects at all, it may be that the novel effects are irrelevant or even beneficial for task completion.

Partial effect failure. As discussed in the recovery for obstructive novelties, when an operator's effects are inconsistent with the world state, one of the agent's accommodation strategies is to modify the operator's description in KB to be accurate with respect to the environment state descriptions. As a result, when the agent replans, the planner will automatically take advantage of beneficial novelties if they allow a shorter plan to be created. Similarly, the planner will automatically not involve nuisance novelties in plans.

The agent may encounter a beneficial novelty in a scenario in which the *supplier* (referring to the example discussed in Section 2), upon interaction, directly provides the agent with the pogostick at no cost. During Knowledge Discovery, the agent may interact with the new actor and immediately encounter an Effect Failure when it observes the unexpected effect of receiving a pogostick. This results in a new operator being created, which the agent can in the future use to acquire a pogostick very quickly.

4.3.2. Operator discovery

In cases when prohibitive or obstructive novelties are attributed to operator failure, the agent needs to discover new operators that may allow it to solve the task.

Precondition discovery. The first strategy to discover new operators involves precondition exploration. For a given failed operator $o \in \mathcal{O}$ with preconditions ψ_o, a new operator o' is constructed with the same effects and executor as o but with a new set of preconditions. The new preconditions $\psi_{o'}$ are constructed using a priority-based order[9] of all possible preconditions encountered in $\mathcal{O}$. The preconditions are added to $\psi_{o'}$ one by one, and goals are submitted to the GOAL MANAGER to satisfy them and attempt the new operator o'. If the operator succeeds, the broken operator o is replaced with this new operator o_i and added to KB.

In the earlier gridworld example, consider a scenario where trees cannot be broken into logs without the agent holding a tool, even though the agent's prior experience suggests it is possible. In this scenario, precondition discovery will consider a new precondition of holding a tool before breaking a tree, as the agent knows that holding a tool is a precondition for breaking other resources in the environment.

Operator variations If the precondition discovery fails to produce a working operator, the agent actively searches for known operators with unknown effects. This process is guided by the agent's type hierarchy and has two phases. In the first phase, the agent attempts (compatible) operators with the same parameters as the failed one. For example, if the operator for breaking a tree is no longer working, the agent will attempt other operators that accept trees as a parameter to test if any of these operators also yield unexpected effects. In the second phase, the agent prioritizes operators that act on different parameters of the same type as the failed operator. In a situation where trees come in different types, if the break operator failed on a birch tree, the agent may attempt to interact with an oak tree, as it belongs to the same type as the birch, to see if the operator failure is consistent across tree types. If any operator produces unexpected effects, a new operator o' is added to KB with those effects noted, and the agent re-plans.

If both precondition discovery and operator variations fail to produce an operator that enables a successful plan, then the agent attempts to learn a new executor for the failed operator using knowledge-guided reinforcement learning.

4.3.3. Knowledge-guided executor learner

Due to the exponentially large search space of precondition discovery and operator variations, the agent cannot exhaustively search that space. After expending some effort in those directions to no avail, a more intelligent search strategy is employed to create a new executor for the failed operator. That way, the agent can discover ways to achieve a desired effect that would assist it in solving the task.

The executor learning component employs reinforcement learning[10] [53], with a reward function encoding the failed operator's desired effects. A new operator is then created with preconditions derived from the state description where the failure occurred and effects identical to the failed operator. The learned policy is used as the new operator's executor, and the agent can use it to reach a state from which it can plan and complete the task. The algorithm used, called RAPid-learn [16], uses knowledge-guided exploration informed by the novelty. The component receives symbolic state descriptions $s \in \mathcal{S}$ and novelty descriptions $\mathcal{N}$ from the goal manager and temporarily guides the agent's behavior to explore the environment.

[9] The Preconditions can be ordered in many ways. For instance, they can be ordered by their frequency in known operators or using domain-specific heuristics. The specific implementation details for our agent instantiation in the evaluation environment can be found in the appendix.

[10] In RL, an agent interacts with its environment to achieve a goal specified by a reward function R. These problems are characterized using an episodic Markov decision process (MDP) $\mathcal{M} = \langle \tilde{S}, \tilde{A}, R, \iota \rangle$, where $\tilde{S}$ is a set of subsymbolic states, $\tilde{A}$ is a set of actions, a reward function R translates state-action pairings to scalar rewards. The agent aims to learn a policy $\pi_\mu(\tilde{a}|\tilde{s})$ (parameterized by μ), maximizing its discounted return $G_t = \sum_{k=0}^{K} \gamma^k R_{t+k+1}$ until the end of the episode at timestep K.

Algorithm 2 Executor learner $(T, \mathcal{P}, \omega_o, s_f) \rightarrow x_{new}$.

1: **Inputs:**
2: $T = \langle \Sigma, s_0, s_g \rangle$ ▷ Symbolic Planning Task
3: $\mathcal{P}$ ▷ Plan $\mathcal{P} = \{o_1, o_2, ..., o_{|\mathcal{P}|}\}$
4: N_{eps} ▷ Number of episodes
5: $\eta, \omega_o, \tilde{s}_f$
6: **Procedure:**
7: $\gamma(s)$: initiation indicator ▷ computed from s_f
8: $\beta(s)$: termination indicator ▷ computed from ω_o
9: Construct MDP $\mathcal{M} = \langle \tilde{S}, \tilde{A}, R, \gamma \rangle$ using T
10: **for** N_{eps} *episodes* **do**
11: $\pi_o^{new} \leftarrow$ ***Train(*** $\mathcal{M}$, T, β***)*** ▷ Train in $\mathcal{E}$
12: **if** $success(\pi_\mu^o, T) > \eta$ **then**
13: $\chi_{new} \leftarrow \langle \gamma, \pi_\mu^o, \beta \rangle$ **return** χ_{new}
14: **end if**
15: **end for**
16: **return** *failure*

Policy learning for failed executors. The procedure for learning a new executor is described in Algorithm 2. Consider a plan $\mathcal{P}$ which fails during the execution of an executor $\chi_o \in \mathcal{X}$ of an operator o. Assuming prior accommodation strategies failed, the learner is invoked to learn a new policy π_μ^o parameterized by μ to replace the policy of the executor χ_o. First, two indicator functions γ, β are defined over symbolic descriptions S to indicate whether a given state s is an acceptable initial or final state for the policy (Algorithm 2 lines 7 and 8). Then, an MDP $\mathcal{M}$ can be defined with a reward function R (Line 9) that provides a positive reward when the agent reaches states that satisfy the effects ω_o of the failed operator o, and from which a successful executable plan to the goal state exists.[11] The policy is represented with a neural network that accepts $\tilde{s} \in \tilde{S}$, which is a domain-dependent representation of the symbolic state description s.[12] The policy π_μ^o is trained by repeated interaction with the environment through reinforcement learning until it achieves a predefined success rate (line 12).

Knowledge-guided exploration of novelties. An important feature of the executor learner is its knowledge-guided exploration strategy. Using a description of novelties $\mathcal{N}$, the exploration strategy of the RL learner is biased towards states and actions that are related to the novelty.[13] For instance, if the presence of a novel object is detected in the environment, the RL learner may be encouraged to explore actions on that object. The knowledge-guided exploration improves the efficiency of exploration and makes the plan recovery process easier. Once a new policy π_μ^o for the executor χ_o is learned, the resulting operator is stored in the KB. The parameters of the policy μ are stored in $\mathcal{K}$ and retrieved when χ_o is executed.

5. Comprehensive evaluation

We conducted comprehensive evaluation experiments of the proposed novelty-aware architecture framework discussed in Section 3, evaluating core components of the framework as well as integrated agents. The details of the agent implementation, the simulation setup, and the hardware used for the evaluations can be found in the Appendix.

We start with a description of the two evaluation simulations followed by an overview of the experimental methodology and the evaluation results. Overall, evaluations were performed in multiple trials where the agent had to perform the crafting task in different random environmental settings (with different numbers of objects, etc.) divided into two phases: an initial "pre-novelty" phase (i.e., a varying initial number of trials with only known entities and dynamics) and a "post-novelty" phase (i.e., a set of trials where something novel to the agent was introduced in all trials). The agent's knowledge repository $\mathcal{K}$ was initialized with the information required to not only solve the task, but anticipate the outcome of every interaction in "pre-novelty" environments. The agent needed to modify and augment its knowledge for most "post-novelty" environments to be able to solve the task.

5.1. Evaluation setting

We used the *Polycraft World* domain [19] as a partially observable, multi-agent environment for the agent evaluations. Polycraft World is a "Minecraft mod" consisting of a multi-agent (cooperative and adversarial) turn-taking grid-world game where an agent competes with actors for resources to perform various crafting tasks. To accomplish a task like crafting a pogostick, an agent must explore the environment to perform a series of sub-tasks that involve collecting and crafting the needed materials for constructing a pogostick. Sub-tasks include:

[11] Each time the agent satisfies the effects of the failed operator, the task planner is called to produce and execute a plan to reach the goal state. If the plan is executed successfully, a positive reward is given.

[12] $\tilde{s}$ can take any form depending on the domain in question, such as a vector, graph, text, etc.

[13] Further details on knowledge-guided exploration are available in the original publication of RAPid-Learn [16].

Fig. 5. Snapshot of the evaluation environments. Left: Novelgridworlds, Right: Polycraft.

- mining trees, diamond, and platinum,
- trading with *trader* actors,
- crafting intermediate objects, including a tree tap,
- placing tree tap on a tree and collecting rubber from it,
- collecting a key from a chest,
- opening a door and navigating to another room to find a safe,
- unlocking a safe and collecting items from it,
- crafting a pogostick.

The agent can execute movement commands (turn, walk, and teleport), interaction commands (select-item, use, break-block, craft, collect, place, delete, trade, and interact), and sensing commands to make observations of the world state, including the direction the agent is facing, inventory items, and locations of every object and actor in the room occupied by the agent.

For the particular Polycraft World domain used here (shown in Fig. 5 right), each trial instantiates the environment with a random configuration that includes two to three rooms of varying sizes, one rival pogostick building actor ("*pogoist*"), two trader actors, and resources in the environment (e.g., trees, platinum blocks, diamond ore, a safe containing diamonds, a crafting table, and a chest containing a key). The rival *pogoist* actor is competing for resources, whereas the *trader* actors, when interacted with, offer recipes for possible trades (e.g., 18 diamonds for one platinum block) and will trade with the agent if it has the requisite materials.

The agent starts each trial with an iron-axe in its inventory, which must be equipped to mine diamond ore and platinum blocks, and only perceives information within the room it is currently in.

We used ten broad categories of novelties as shown in Table 1.[14] In total, evaluations include 288 novelties from the 10 categories, of which 216 were known during the development of the agent. Each novelty scenario is associated with three novelty tournaments, and each tournament is composed of 50 task instances (i.e., episodes) where the first 5-20 episodes (a random number n is selected between 5 to 20) do not contain novelty (i.e., pre-novelty phase) and the subsequent 30-45 episodes (i.e., post-novelty phase) contain the novelty being evaluated. In addition to the novelty tournaments, three no-novelty tournaments are included in the evaluation, each consisting of 50 unique no-novelty task episodes. In each tournament, the agent receives a positive reward for completing the task (e.g., obtaining a pogostick), and every action the agent takes has an associated negative cost. A cumulative score for the agent's performance is calculated by subtracting the negative cost assigned per action from the positive reward assigned for completing the task. The agent is awarded a score of zero in the episodes where it fails to achieve the task. The episode in which the agent reported novelty was also recorded. For comparison, the evaluation team used a non-novelty aware fast-forward planner agent otherwise unrelated to the evaluated agent to generate standard performance scores for the pogostick task. The non-novelty aware control agent achieved scores within 15% of the evaluated agent scores on pre-novelty episodes and had significantly decreased performance in post-novelty episodes.

Metrics for tournament-based evaluations. To evaluate our agent in tournament-based evaluations in the Polycraft domain, we define metrics to measure its novelty detection and novelty accommodation performance. The metrics are defined as follows:

- *False Negatives* ($\overline{FN}_{CDT}$): How many episodes on average after novelty has been introduced, does the agent fail to report novelty? A perfect score of 0 is obtained if novelty is reported in the first post-novelty episode.
- *Correctly Detected Trials (CDT%)*: Boolean for each trial. True for every trial where the agent i) reports novelty after novelty has been introduced and ii) does not report novelty before novelty is introduced.
- *False Positives (FP%)*: Boolean for each trial. True for every trial where the agent reports novelty before novelty is introduced.

[14] This categorization should not be confused with the novelty types described in Section 3. Moreover, this categorization is not necessarily exclusive as the novelties may generally belong to one or more of these categories. Nevertheless, to better motivate the discussion for novelty handling capabilities of an agent, we categorize them.

Table 1
Descriptions of the novelties categories with examples used for the comprehensive evaluation in Polycraft.

Category	Description	Example
Object	A new entity in the environment that does not have goal-oriented behavior.	A new block type 'fence' is added that obstructs access to trees
Attribute	Changes to the properties of previously-known entities in the environment.	A new tree variant 'birch' is added that can not produce rubber with a tree tap
Representation	Changes to how previously-known entities are specified in sensory percepts.	Item names are partially scrambled
Actor	A new entity in the environment that does have goal-oriented behavior.	A new entity 'thief' is added that steals items from your inventory
Action	A new goal-oriented behavior of a previously-known environmental agent.	The *Pogoist* agent now trades items instead of the *Traders*)
Relation	A new static property of the relationships between multiple entities.	*Traders* now spawn in different areas of the arena)
Interaction	A new dynamic property of behaviors or actions that impacts multiple entities.	*Traders* are now 'busy' sometimes when the player interacts with them
Environment	A new element of an open-world space that may impact the entire task space and is independent of a specific entity.	The new element 'wind' is present in various regions of the arena and alters player movement
Goal	A new objective of goal-oriented behavior for an environmental actor.	The *Pogoist* changes the resources it is seeking
Event	A new state change or series of state changes that are not each the result of volitional action by an agent/actor.	Trees rot and regrow over time

- *Novelty Reaction Performance (NRP%)*: Average post-novelty task score for the agent divided by the average pre-novelty task score of the non-novelty aware control agent.
- *Goal Achieved (GA%)*: Percentage of post-novelty tasks in each tournament where the agent achieved the goal.

We also use a previously developed "openAI gym" environment called *Novelgridworlds* [17] for component evaluations (shown in Fig. 5 left). Specifically, we implemented the same pogostick task in Novelgridworlds and developed several novelties for algorithm prototyping, internal evaluations, and showcasing important characteristics of our cognitive architecture. The need for a separate evaluation platform was due to the need for a tight development loop between designing novelties the agent could not handle and, subsequently, improving the agent's algorithms to master them (which was not possible in Polycraft). More importantly, many novelties designed by the evaluation team in Polycraft were purposely concealed and not accessible to the architecture development team in an effort to avoid any subconscious biases being introduced into the algorithm developments. Additional standard metrics are used for component-wise evaluations.

5.2. Component-wise evaluation

A component-wise evaluation is performed for various components of the architecture to comprehensively evaluate important aspects of the architecture. The VISION MODEL is evaluated on its ability to detect novelties in images of the Polycraft environment in a two-stage evaluation. The AGENT MODEL is evaluated in Polycraft shared novelties that involve actors with changed behavior. The EXECUTOR LEARNER is also evaluated separately in the NovelGridworlds environment to evaluate its efficiency in recovering from execution failures.

5.2.1. Vision model evaluation

Vision model training. We train our vision models on a custom open-access dataset called NovelCraft [13], which contains over 10,000 256×256 pixel RGB images from our agent's perspective as it solves the pogostick building challenge within the Polycraft world. Rather than apply encoder and decoder to the entire image, we instead process *patches* of size 32×32 pixels. Patch-based autoencoders are faster and easier to train and maintain, while being at least as accurate as whole-image models in our tests [13]. For encoder and decoder, we use the specific network architectures in Abati et al. [1], with latent code size of 100. Hyperparameter search for architectural choices, learning rates, etc. was informed by NovelCraft's separately available validation set of images. Further details on training and implementation are available in the appendix and in Feeney et al. [13].

Two-round evaluation. Given our trained model, two distinct rounds of evaluation were then performed. First, an initial *known-novelty* evaluation was performed on the test set of the NovelCraft dataset [13]. While this is a predefined split whose images and novelties are distinct from the training set, the possible novelties in this test set were known to our team during model design, so we call this a "known-novelty" evaluation. There are 21 normal episodes and 440 novel episodes, where new novel *object* types, such as JukeBox or TNT, are inserted into the Polycraft environment. In total, 51 different *object* type novelties are used (for a full listing, see Appendix A of [13]).

Next, an additional *unknown-novelty* evaluation was conducted using a set of Polycraft novelties designed by an external team. We had no prior knowledge of these novelties during model development, which makes for a more robust assessment. This evaluation grouped 50 episodes into tournaments, with a set of normal episodes followed by a set of novel episodes depicting the same novelty. The transition from normal episodes to novel episodes occurs randomly, but there are 15 normal episodes on average. Tournaments for 9 different visual novelties were conducted with an additional tournament composed of 50 normal episodes. Three tournaments

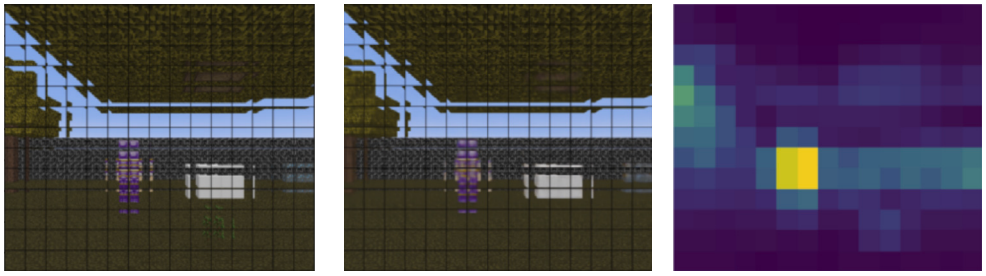


Fig. 6. Input and output example from the patch-based autoencoder of the agent's VISION MODEL. The image X obtained through the DOMAIN INTERFACE (left) is segmented into patches and reconstructed by the autoencoder (middle). The reconstruction error (right) is used as a signal to detect novelties, with higher values (lighter) indicating regions more likely to contain unknown visual elements.

Table 2
Visual novelty detection results on the known-novelty test set (from NovelCraft) and the unknown-novelty test set (content unknown to us during model development). We report performance at different thresholding choices selected on the known set. Mismatch between the novelties depicted in the evaluations results in the model yielding a lower than expected precision (PPV) on the unknown set while maintaining reasonable true positive rate (TPR).

Evaluation	TPR	TNR	PPV
Known Set (TPR 95%)	95.0	31.3	88.7
Known Set (PPV 95%)	66.5	80.6	95.0
Unknown Set (PPV 95%)	84.9	70.5	85.1

were randomly generated for each setting, resulting in 30 tournaments and 1,500 episodes total. In both evaluations, the model observes all of the episode's image patches then outputs a single real-valued novelty score for the whole episode. This novelty score is the model's prediction of the likelihood of a novelty appearing anywhere in the episode. For simplicity, the model computes a per-patch novelty score for each image patch in the episode using the reconstruction error signal from the deep autoencoder, then reports the maximum score over all frames. Fig. 6 showcases an example of its operation. Future work could pursue more sophisticated aggregation strategies across frames.

Evaluation criteria. The model is evaluated on the true positive rate (TPR, aka recall), the true negative rate (TNR), and the positive predictive value (PPV, aka precision) of the per-episode visual novelty detector. Given the count of true and false positives (TP and FP) as well as true and false negatives (TN and FN) the metrics are defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \ \text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}, \ \text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Evaluating these rates at chosen operating thresholds is more informative than area-under-the-curve metrics when making binary judgments of normal or novel [13]. The known-novelty evaluation examines two possible ways to select a threshold. First, the novelty score threshold is chosen to achieve 95% TPR on the NovelCraft test set to examine performance when avoiding false negatives matters most. Second, the novelty score threshold is chosen to achieve 95% PPV to examine when avoiding false positives matters most. The unknown-novelty evaluation uses the latter threshold to prioritize preventing false positives on the unseen set of novelties.

Results. Table 2 quantifies the visual novelty detection performance. The known-novelty evaluations show that enforcing a TPR of 95% maintains a good PPV but results in a low TNR. A much better performance in terms of TNR is achieved by enforcing 95% PPV while the TPR decreases in this regime. Reusing the same 95% PPV threshold in the unknown-novelty evaluation performs well despite having a 85.1% PPV. The TPR is increased by 14.4 percentage points with only a 10.1 percentage point decrease in TNR.

The results show that focusing on high TPR values comes at the cost of a low TNR - although many novel episodes are detected as novel, comparably many normal episodes are considered to be novel too. By emphasizing high PPV, the performance in terms of TNR is significantly improved while a similar TPR is maintained. Compared to the high TPR regime, less normal episodes are reported as novel but some of the novel episodes are missed.

The threshold for the high PPV regime in the unknown-novelty evaluations is the same as in the known-novelty evaluation. However, PPV and TNR are decreased while TPR is increased. This suggests that novelties in the unknown-novelty evaluation are easier to detect than object novelties and that the unknown-novelty evaluation includes more normal episodes depicting the normal environment in a way that was not present to a great extent in the validation set.

Table 3
Tournament-wide (left) and episodic (right) AGENT MODEL evaluations using *only* detections from the AGENT MODEL on actor-related novelties.

Tournament-wide		Episodic	
Metric	**Value**	**Metric**	**Value**
CDT%	64.71%	*F1*	0.728
$\overline{FN}_{CDT}$	0.36	*Precision*	0.997
FP%	2.94	*Recall*	0.573

Discussion. The results demonstrate some of the challenges when applying visual novelty detection methods for automated *binary* novelty decisions - a threshold must be chosen based on available validation data. There is a tradeoff between prioritizing avoiding false negatives and avoiding false positives and a threshold might need to be chosen specifically for the given task [13]. Furthermore one must keep in mind that results achieved on the validation set might differ from the expected performance on a test set if the sets deviate "too much" from another. Detecting novelties in Polycraft is a challenging problem - novel episodes hardly ever show the novelty in a focused, large and central manner. A novel input image shows a complex scene in which the novelty might be depicted on only a very small fraction of the scene's content while large image regions are occupied by normal objects, agents and textures. The novelty can be far in the back of a scene or partially occluded. However, for open-world applications, the detection of novelties in these scene-based scenarios is crucial.

In future work, novelty descriptions produced by the VISION MODEL could further inform novelty exploration strategies to verify suspected novelty. For instance, potential novelties, identified by increased pixel-level image reconstruction errors, could be examined in more detail by including *localization* information in the novelty descriptions which the agent could use to better focus exploration efforts.

5.2.2. Agent model evaluation

Using the novelties shared in the Polycraft environment, the AGENT MODEL's detection capabilities of actor-related novelties are evaluated. From the total of 36 novelties shared, 7 are selected that involve known actors that exhibit altered behavior. These novelties originate from the Actions and Goals[15] categories. Each environment is associated with 9 tournaments and the novelties are spread over 3 sub-variants, yielding a total of 63 test tournaments. Each tournament is run for 30 games, yielding a roughly equal split between pre- and post-novelty episodes in each tournament.

Evaluation criteria. The novelty detection performance of the AGENT MODEL *alone* is evaluated using both the tournament-wide detection metrics on the episode-level standard anomaly detection metrics. Further analysis reveals strengths and weaknesses of the agent model which are discussed using two examples of novelty from the Polycraft environment.

Model training. The AGENT MODEL is trained and validated on trajectories generated by randomly repositioning objects in the initial states of the 100 no-novelty environment configurations provided in the Polycraft repository. In total, 1000 trajectories are generated and 80% of the trajectory steps are used for training and the remaining 20% for validation and threshold calculation. AGENT MODEL is evaluated by running the architecture on the novel environment configurations and recording only the novelties detected by the AGENT MODEL. Additional implementation details are available in the appendix.

Tournament-wide results. Table 3 (left) quantifies the novelty detection performance of the AGENT MODEL in tournament-wide metrics. The AGENT MODEL is able to correctly identify novel trials in the majority of the novelties. It is consistent in its detections, only producing a small number of false negatives in correctly detected trials. Additionally, its tournament false positive rate is low, which makes it a viable addition to the symbolic novelty detection system which tends to produce very few false positives.

Episode-level results. Table 3 (right) shows the episodic evaluation of the AGENT MODEL detector. As suggested by the tournament-wide evaluation, the agent has strong performance and is generally cautious, producing a very small number of false positives as evident from its high precision score. However, its weaker recall indicates that there are many episodes where the AGENT MODEL fails to detect novelty. As indicated by the prior analysis those cases seem to be concentrated in environments where interaction is required for the novelty to be made apparent. This suggests that the architecture would benefit from tighter integration of its AGENT MODEL and its planner and knowledge base.

Discussion. A more thorough investigation of the results reveals that the AGENT MODEL exhibits high variability across different novelties, with nearly perfect detection performance in some novel environments and worse performance in others. One way to distinguish which novelties the AGENT MODEL can reliably detect is whether the behavior change of actors is dependent on interaction

[15] The actors category typically involves actors of novel types, detection of which is handled with symbolic inference as they are included in the symbolic state descriptions.

Table 4
Component-wise evaluation results comparing the GA% of the *Base agent* and the *Base*+ EXECUTOR LEARNER *agent* on the four novelty scenarios evaluated in Novelgridworlds.

Novelty\Agent	Base GA%	Base+ EXECUTOR LEARNER GA%
This is just random...	2.0	21.0
Convince me.	0.00	59.0
Sapling can't grow here!	5.0	10.0
Show me your card first.	0.00	21.0

with the agent. Novelties that arise from a change in actor behavior independent from the agent's actions are consistently detected by the AGENT MODEL, whereas novelties that require interactions to manifest are more difficult to detect.

One example of a novelty type in Polycraft that the AGENT MODEL consistently detects, involves a *pogoist*-type actor altering its resource-gathering strategy. This change is detected by the AGENT MODEL as a set of actions that are very unlikely under the learned policy for that agent type. As a result, the AGENT MODEL detects the novelty and produces a description indicating which performed actions are highly unlikely. In this example, the AGENT MODEL reports that the unlikely action taken by the actor involves collecting the "diamond" resource, which is essential to task completion. As a result, the GOAL MANAGER generates subgoals to gather that resource with high priority, to ensure the agent can continue to solve the task.

A failure case for the AGENT MODEL in Polycraft is a variant of a novelty that gives the *pogoist*-type actor the ability to trade resources with our agent much like the no-novelty behavior of the "Trader"-type agents. One variant of the novelty involves the actor changing its policy and acting exactly like the no-novelty "Trader" actors. That change is quickly detected by the AGENT MODEL as a set of unlikely actions. Additionally, using the actor type classifier, the AGENT MODEL infers that the *pogoist* actor is acting like a *trader*. That novelty can be submitted to the KB and accommodation strategies can be applied. However, a different variant of this novelty involves the actor acting as expected most of the time, unless the agent interacts with it. In that case, it is available for trading as before. This novelty is not detectable without attempting to interact with the actor, and therefore, the AGENT MODEL never detects this variant of the novelty.

5.2.3. Executor learner evaluations

To showcase the effectiveness of the EXECUTOR LEARNER component, we evaluate it on four novelty scenarios. The evaluations were performed using a similar experimental setup described in Section 5.3.1 with a pre and post-novelty scenario. We inject novelties into the environment and measure the task performance capability of the agents. Specifically, we compare the performance of the *Base*+ EXECUTOR LEARNER with the *Base agent*, in which the Base agent consists of the symbolic inference components, and the *Base*+ EXECUTOR LEARNER consists of the symbolic inference and the EXECUTOR LEARNER. The rationale is to showcase the usefulness of an rl-based learner to accommodate novelties, especially when extensive symbolic inference strategies (described in Section 4.3.1) fail to accommodate it. We further describe the experimental setup and evaluation criteria and discuss the results.

We design the novelty scenario for the *internal evaluation* in the Novelgridworlds environment as it gives us more flexibility to implement specific scenarios we want to analyze. These novelties are designed to showcase the benefits of using a reinforcement-learning-based learner[16] - in particular stochasticity, dynamic behavior, spatial and temporal relations. A description of all novelty scenarios used for this evaluation is available in the appendix.

Evaluation metrics. Both agent configurations (Base and Base+ EXECUTOR LEARNER) were evaluated on the above four novelty categories. Each novelty tournament comprised 50 episodes. Each episode was limited to a maximum of 600 time steps or 15 minutes. We record the percentage of times each agent successfully achieved the goal, i.e., crafting a pogostick (The GA% metric described earlier in the text). Each novelty tournament was run for 10 independent trials, and the GA% was averaged across these 10 independent trials. It should be noted that the EXECUTOR LEARNER performed online exploration and learning during the evaluations and was randomly initialized for each run. For the scope of this work, we do not evaluate the robustness of the policy in this evaluation.[17] We showcase the results in Table 4.

Discussion. The results of the component-wise analysis (Table 4) show that the *Base*+ EXECUTOR LEARNER *agent* performs better overall than the *Base agent* in accommodating to novelties. Adding an EXECUTOR LEARNER component to the base architecture helps the agent to be more adaptive and learn policies that the symbolic explorer cannot find. The *Base*+ EXECUTOR LEARNER is better at capturing complex dynamics where the *Base agent* alone completely fails in the novelty scenarios described above. We detail each novelty scenario to understand how the EXECUTOR LEARNER aids learning to accommodate the novelties.

[16] The implemented novelties cannot be solved by another module in the architecture alone. We ensure that the EXECUTOR LEARNER is needed to accommodate these novelty scenarios. It is unknown whether any of these or similar novelties were included in the Systemic evaluation.

[17] For thorough evaluation details and results on the robustness of policies learned with this algorithm please refer to our original publication of the algorithm used RAPid-Learn [16].

Table 5
Overall results comparing the five configurations of the agents in the systemic evaluation.

Agent	$\overline{FN}_{CDT}$	CDT%	FP%	NRP%	GA%
Base	0.69	83.3	3.2	73.5	64.4
Base+ AGENT MODEL	**0.58**	**86.1**	6.1	72.4	64.0
Base+ EXECUTOR LEARNER	0.76	84.7	**2.3**	**74.2**	**66.6**
Base + AGENT MODEL + EXECUTOR LEARNER	**0.39**	**88.7**	1.6	68.1	61.9
VISION MODEL	3.44	56.6	34.1	–	–

5.3. Systemic evaluation

Systemic evaluations were performed across the entire Polycraft novelty set to showcase the integrated capabilities of the agent. We evaluate three novelty-aware agent configurations in all 864 novelty tournaments[18] and three pre-novelty tournaments. These evaluations utilize a novelty firewall, where novelties outside the shared set of novelty tournaments were concealed from the algorithm and architecture team and evaluated by a separate evaluation team.

5.3.1. Evaluation methodology

The systemic evaluation focuses on the measurement of novelty detection and accommodation performance across all novelty categories using different configurations of the agent architecture. Three evaluations are presented on different configurations of the agent architecture.

Base agent. The first agent configuration we evaluated includes the first-order inference components and the novelty exploration component except the EXECUTOR LEARNER. This configuration aims to evaluate the agent's performance using only symbolic reasoning, without any learning-based components.

Base agent+ AGENT MODEL. The second configuration we evaluate augments the base agent with the AGENT MODEL, which aims to improve the agent's novelty detection performance, especially in actor-related novelties.

Base agent+ EXECUTOR LEARNER. The third configuration we evaluate augments the base agent with the EXECUTOR LEARNER, which aims to improve the agent's ability to recover from execution failures unresolved by OPERATOR DISCOVERY. It should be noted that the EXECUTOR LEARNER in this agent is used only after all other exploration strategies fail to accommodate novelty within a preset time-limit.

Base agent + AGENT MODEL + EXECUTOR LEARNER We also evaluate the combined version of all the models with the base agent to demonstrate the capability of our architecture in terms of novelty detection and novelty accommodation.

Vision agent We include a vision-only agent to demonstrate the challenge of performing novelty detection through only visual perception. In this configuration, the default version of the Base agent is used, but no novelties discovered through symbolic inference are reported. However, since all accommodation strategies rely on detailed symbolic descriptions of novelty that the vision model cannot produce, we only evaluate the vision agent on novelty detection using the $\overline{FN}_{CDT}$, CDT%, and FP% metrics.

Evaluation criteria. The evaluation of the integrated agent is performed using the tournament-wide metrics defined earlier. Specifically, we evaluate the novelty detection performance of the agent using CDT%, FP%, and $\overline{FN}_{CDT}$, and the novelty accommodations performance using NRP% and GA%. Overall results are computed by averaging all tournaments. Additionally, per-category results are obtained by averaging tournaments within the same category. Due to the high computational cost of the systemic evaluations, we only run a single set of experiments per agent configuration.

5.3.2. Results & discussion

Table 5 quantifies the novelty handling performance of each of the five agent variants in the Polycraft environment. The base +AGENT MODEL and base +AGENT MODEL + EXECUTOR LEARNER agents outperform the other variants in $\overline{FN}_{CDT}$ and CDT%, which indicates increased novelty detection capability but with higher variation in performance, which is attributed to the AGENT MODEL being the only learning based novelty-detection component in those systems. The base+ EXECUTOR LEARNER agent performs best in NRP% and GA%, which indicates the agent with an rl-based learner has better novelty accommodation capability. While the VISION MODEL demonstrates potential to improve the novelty detection capabilities of the agent, its evaluations suggest that further effort is required to bring the false positive rate to a level that would not be detrimental to the agent's performance in the pogostick task in Polycraft.

[18] 288 novelties with 3 tournaments each.

Table 6
Detailed systemic evaluation of Base agent on all the novelty categories.

Metric	$\overline{FN}_{CDT}$	CDT%	FP%	NRP%	GA%
Objects	0.00	95.9	1.6	95.8	82.9
Attributes	0.51	86.7	3.5	63.2	58.5
Representations	0.18	85.0	2.8	74.8	66.7
Actors	0.00	92.3	7.7	63.3	52.6
Actions	3.76	61.1	0.0	103.0	94.4
Relations	0.00	96.3	3.7	70.7	48.4
Interactions	0.06	64.2	1.9	71.7	65.4
Environments	0.10	92.2	4.4	44.2	41.1
Goals	3.21	70.0	4.4	99.1	81.2
Events	1.12	77.3	2.3	51.0	48.8

Table 7
Detailed systemic evaluation of Base+ AGENT MODEL agent on all the novelty categories.

Metric	$\overline{FN}_{CDT}$	CDT%	FP%	NRP%	GA%
Objects	0.00	88.9	8.7	91.5	82.3
Attributes	0.65	85.4	6.9	66.3	59.0
Representations	0.65	84.3	4.6	73.3	66.1
Actors	0	96.3	3.7	56.9	49.6
Actions	1.23	74.1	7.4	99.5	94.6
Relations	0.00	90.7	9.3	66.0	47.0
Interactions	0.24	68.5	3.7	68.2	62.6
Environments	0.48	95.5	4.5	42.7	40.9
Goals	0.13	95.6	4.4	104	82.7
Events	2.44	77.3	5.7	49.0	47.5

It should be noted the NRP metric may disadvantage reinforcement-learning agents as it punishes extended exploration in the environment.[19] The EXECUTOR LEARNER achieves a measurable improvement over the other two configurations in NRP% despite this, indicating a positive influence on novelty accommodation. A detailed analysis of the evaluation (Table 8) reveals that the EXECUTOR LEARNER improves the exploration capability of the agent in various novelty categories. Table 10 showcases examples of the accommodation strategies learned by EXECUTOR LEARNER in some novelty categories. We discuss these examples in detail in Section 5.3.3.

Table 6 details the systemic evaluation results of the base agent configuration by novelty category. The base agent performs strongly in novelty handling, though it is weaker than the other configurations.

Regarding novelty detection performance, the base agent seems to be weakest on the action, interaction, and goal novelties, receiving CDT% scores of 61.1, 64.2, and 70, respectively. Action and goal novelties also show high $\overline{FN}_{CDT}$, which indicates that the base agent is not very consistent in detecting those novelties.

Table 7 details the systemic evaluation results of the *Base*+ AGENT MODEL configuration by novelty category. The *Base*+ AGENT MODEL performs best in CDT% and $\overline{FN}_{CDT}$ out of the three configurations but has the highest FP%. In the three pre-novelty trials ran, the agent model produced false positives in two out of three tournaments in a total of 3 out of 150 episodes. This per-episode false positive rate is consistent with our component-wise evaluation results and indicates that even a comparatively low false positive rate of about 2-3% can be magnified in tournament-wide evaluations. The additional false positives also explain the slight NRP% and GA% decrease, as they may force the agent to explore unnecessarily. It should be noted that the agent configurations that include AGENT MODEL exhibit higher variation in detection performance, particularly in false positive rate, than the agent configurations that only use symbolic inference. This is due to small random effects in the environment (e.g. where broken objects fall to the ground) that may trigger false positive detections from the neural models. Still, both configurations of the agent with AGENT MODEL overall outperform the other agents in novelty detection.

The *Base*+ AGENT MODEL configuration performs especially well in novelty detection of action and goal novelties, increasing CDT% by 13% and 25.6%, respectively. It also significantly decreases $\overline{FN}_{CDT}$, decreasing the average time-to-detection to 0-2 episodes for both categories. A smaller improvement is also seen in CDT% for the interactions and environments novelty categories. This indicates that the AGENT MODEL is able to supplement the base agent's novelty detection capability in actor-related novelties in which modeling of actor behavior is required.

A surprising result is that the combined base+ AGENT MODEL + EXECUTOR LEARNER agent performs worse than the other agents in the novelty accommodation metrics NRP% and GA%. Table 9 indicates that this is consistent across novelty categories and not concentrated on one. We attribute this to unforeseen interactions between the additional components in that configuration: When

[19] Extended exploration accumulates negative scores because the agent executes many actions in the environment. Therefore, reinforcement learners, which tend to require prolonged exploration to learn policies, are disadvantaged by this metric as possible gains in task completion performance can be masked.

Table 8
Detailed systemic evaluation of Base+ EXECUTOR LEARNER agent on all the novelty categories.

Metric	$\overline{FN}_{CDT}$	CDT%	FP%	NRP%	GA%
Objects	0.00	95.2	2.4	92.5	82.0
Attributes	0.48	88.1	2.8	62.9	61.2
Representations	0.64	86.0	3.7	75.9	68.6
Actors	0.00	98.1	1.9	64.4	54.5
Actions	4.86	64.8	1.9	101.8	96.5
Relations	0.00	98.1	1.9	70.2	49.3
Interactions	0.06	64.8	1.9	71.3	64.6
Environments	0.09	94.4	2.2	42.2	40.5
Goals	2.82	67.8	1.1	106.5	86.5
Events	1.23	80.7	2.3	55.8	56.7

Table 9
Detailed systemic evaluation of Base + AGENT MODEL + EXECUTOR LEARNER agent on all the novelty categories.

Metric	$\overline{FN}_{CDT}$	CDT%	FP%	NRP%	GA%
Objects	0.00	95.2	1.6	92.5	80.1
Attributes	0.95	91	1.4	63.0	57.6
Representations	0.13	85.2	0.9	74.7	66.7
Actors	0	98.1	1.9	53.5	47.8
Actions	1.93	77.8	0.0	91.6	86.8
Relations	0.00	96.3	3.7	46.8	39.8
Interactions	0.06	66.0	1.9	72.9	64.7
Environments	0.10	93.3	3.3	32.2	39.9
Goals	0.21	98.9	1.1	101.2	82.1
Events	0.78	77.5	0.0	40.6	41.5

Table 10
Description of some novelties in which the Base+ EXECUTOR LEARNER agent performs better than the base agent on the systemic evaluation conducted on Polycraft.

Category	Description	Adaptation
Attribute	Chest objects contain valuable task items accessible with *break* or *collect*.	EXECUTOR LEARNER learns to break and collect from multiple chests on the ground to get the necessary items.
Attribute	Birch tree types fail to produce rubber.	EXECUTOR LEARNER learns to remove objects that result in operator failure.
Action	Traders plant new trees around themselves.	EXECUTOR LEARNER learns to handle actions from Actors that previously didn't perform certain actions.
Concealed	Concealed	EXECUTOR LEARNER performed better on some concealed novelties.

the EXECUTOR LEARNER explores the environment, this may induce changes to the environment that in turn may cause changes in the behavior of other agents. As a result, the AGENT MODEL may detect novelty, which in turn may lead the agent to further explore the detected novelties, hurting its task performance.

5.3.3. Examples

Several cases are identified from the systemic evaluations conducted on the Polycraft environment to demonstrate the architecture's strengths and weaknesses in novelty handling. Below is a detailed walk-through of how the architecture reacted to each novelty scenario. Additional examples are available in the appendix.

Must break tree tap to get rubber. Within this novelty scenario, the operator to collect rubber from a tree tap does not work. The agent must now break the tree tap to obtain rubber. This novelty belongs to the Attribute category. The base agent sometimes succeeds at solving this novelty and sometimes fails. These two cases are outlined below.

Explanation. The agent attempts to collect rubber from a tree tap, but the operator fails with total effect failure, prompting the NOVELTY EXPLORATION component to enter a failed operator effects recovery policy. The failed operator is represented in the agent's knowledge base as the operator *collect(tree tap,tree)* where *(*tree) is the tree that the tree tap is placed onto. Our current implementation of the failed operator effects policy only utilizes the Operator Variation strategy on operators with a single parameter, which means that the strategy is not executed in this case. As discussed in the appendix, this is due to the large search space resulting from considering operators with multiple parameters. As a result, the operator variation *break(tree tap, tree)* is never attempted, and ultimately the agent is unable to solve the task via any other strategies, opting to give up.

The agent is successful in the rare occasion in which the rival *pogoist* actor breaks the tree holding the tree tap after the agent has placed the tree tap on the tree, but before the agent has executed the "collect" operator. When the GOAL MANAGER attempts to

execute the "collect" operator, a precondition fails: *nextTo(tree tap, tree)*, which represents the spatial relation of the tree tap to the tree. This condition fails because the tree has been chopped down. As a result, the agent utilizes the precondition failure strategy, in which the agent updates its knowledge base with the accurate state and replans. In the plan generated by the agent, it first breaks the tree tap to obtain the tree tap item and takes it to another tree. Upon breaking the tree tap, it enters the effects failure recovery policy when that action unexpectedly places rubber in the agent's inventory. The agent updates its representation of the operator to include acquiring rubber as an effect. Thus, the agent succeeds in accommodating the novelty due to the actions of the rival *pogoist*.[20]

Pogoist prioritizes diamond. The rival *pogoist* actor changes strategy to prioritize mining diamond ore, which means the agent will be unable to mine any if it does not also prioritize diamond. This is categorized as a Goals novelty.

Explanation. Without the agent model added to the base architecture, our agent is able to solve this novelty relatively well because the Task Planner prioritizes mining diamond ore. This is coincidental and not due to any intentional value placed on mining diamond ore over any other action in the plan, so our base agent's success in this novelty is an interesting example of successful performance without explicit accommodation. Moreover, the symbolic inference does not even detect this as a novelty.

However, *with* the AGENT MODEL added on to the base architecture, our agent can successfully detect and intentionally accommodate the novelty. Each type of actor in the world (including *pogoist*) is linked to a model of behavior that is trained on non-novel trials, as described in Section A.1. The *pogoist* breaking diamond ore so early in the episode is differentiated from the model of expected behavior, and therefore, the AGENT MODEL reports it as an unlikely action from the *pogoist*. The agent considers the 'unlikely' action to be a novelty and records this in its knowledge base. At the beginning of the next episode, the agent spends a little time investigating any recorded novelty descriptions in Knowledge Discovery, including this action, which it does by copying the action and mining the diamond. Thus novelty is detected, and the agent performs well by obtaining the ore early.[21]

Cannot collect rubber from Birch trees. A new species of birch tree, belonging to a new subtype of the tree object, is introduced to the environment alongside the known oak trees. Rubber cannot be collected from the birch trees, only the oak trees. This is categorized as an Attribute novelty.

Explanation. The base agent performs fairly inconsistently in this novelty, entirely dependent on whether the planner decides to collect the rubber from an oak or a birch on any given trial, since the inability to collect rubber from birch trees is unknown to the agent. When it tries to collect from a birch tree and fails, the agent attempts to repair the operator using the total effects failure policy. The failed operator is *collectFrom(tree tap,birch tree)*, and the operator that would provide success is *collectFrom(tree tap,oak tree)*. But as mentioned earlier, our current implementation of the Total Effects Failure policy only utilizes the operator variation strategy on operators with a single parameter to avoid search space explosion, leading the agent to give up after trying other unsuccessful policies.

However, the agent utilizing the EXECUTOR LEARNER manages to overcome this implementation compromise in our architecture. Rather than giving up, when the agent utilizes the EXECUTOR LEARNER, to learn a surprising new policy to accommodate this novelty. It prevents the failure from happening again by breaking all the birch trees in the environment and replanning. This forces the symbolic task planner to collect rubber from oak trees for the rest of the trial because it doesn't have the option to collect it from anywhere else. This unexpected accommodation strategy showcases the flexibility of the EXECUTOR LEARNER in integrating with the implementation of the TASK PLANNER. The learner's reward function encourages reaching states from which the agent can plan, which may yield different recovery policies based on the characteristics of a particular implementation of the architecture.

6. Related works

Open-world novelty detection and accommodation is an emerging direction in AI. Some researchers have already attempted to address unprecedented experiences in the world, a process that often begins with the detection of novelty. One way to detect anomalies is to use statistical techniques to detect visual novelties or anomalies in time-series data [37]. In particular, convolutional neural networks (CNNs) trained as classifiers on normal classes can have their probabilistic outputs repurposed to compute a novelty score for new images [35,30,32]. For example, a method named *NDCC* [8] first trains a classifier, then computes a novelty score for a new image by computing the distance between that image and the closest normal class's centroid in the learned feature space. Other variants of neural network techniques for visual novelty detection include autoencoder-based approaches [1,27], Generative Adversarial Networks (*GAN*s, 18), and adversarial autoencoders based on GANs [49]. However, these techniques alone only allow the detection of anomalies.

To make better inferences about new visual events, some researchers have proposed cognitive architectures. These architectures are able to identify and locate unknown activities in video data in an open world, such as *OW-TAL* [61]. Others are also able to analyze and generate explanations and theories, as well as revise and update beliefs [56]. Alternatively, attempts are made not only to recognize new experiences but also to acquire additional knowledge about them. The *CoTTA* cognitive architecture [55] explores relationships between topics and structural elements to capture similarities and provide a better semantic representation for retrieval. Li et al. [33] present a cognitive architecture to understand visual scenes from a very sparse "tabula rasa" knowledge of the world, and learn to automatically extract relevant information from the world using only unsupervised and RL techniques. These

[20] A narrated video of our agent encountering this novelty can be viewed at the following link: https://www.youtube.com/watch?v=ILgFudhi6i8.

[21] A narrated video of our agent encountering this novelty can be viewed at the following link: https://www.youtube.com/watch?v=EZOgBy9cDc4.

cognitive architectures are able to extract new visual representations from the world, but they do not give the agent any capabilities to understand how to plan and act when it encounters changes in the world.

Various aspects of planning and acting in non-stationary environments have been studied before (e.g. planning in the open world, see 54), sometimes under the umbrella term *"lifelong learning"* and often using variants of *deep reinforcement learning* [26]. However, re-planning approaches can not recover from *obstructive novelties*. Proposed solutions for continuous [2,31] and non-stationary [9] learning are often plagued by catastrophic forgetting and difficulties in dealing with abrupt changes in the task environment, which can completely block the learner's ability to adapt and lead to permanent task failure.

Some work in the field of continual learning has shown performance stability despite repeated exposure to novelties [28]. In reality, however, continual learning is neither designed to recognize novelties nor to adapt to a growing domain and would fail in the open world. Lifelong learning methods with adaptive capabilities such as *Meta Experience Replay* [46] or *powerplay* [50], require task traces from previous experiences for retraining to prevent forgetting. This is often unrealistic (e.g., with the limited computational and memory capacity available to autonomous embodied systems such as robots) and sometimes not feasible (because of the time it takes to relearn the system).

Some classical approaches to open-world inference are also relevant to novelty-capable agents. Methods based on Adaptive Resonance Theory [60,51], non-monotonic logic [40,12] and growing neural gas networks [38,39] may form learning and reasoning components of larger novelty-capable agents.

Several recent hybrid planning and learning approaches have shown how to take advantage of high-level planning representations and reasoning methods while utilizing low-level policy-based learning approaches like RL [52,25,29,24]. Guan et al. [20] learn a metacontroller over the learned skills by using the provided operators to acquire skills to access a variety of terminal states (also called landmark states). However, these approaches do not consider open-world settings. Recent work on dynamic environment accommodation is typically limited in several ways. Agents may receive dense rewards [5], or have extra symbolic knowledge about a change when it appears [7,15,23].

Other efforts assume agents may face gradual changes [42,44] or plan-level novelties only [58]. On the other hand, Sarathy et al. [48] used RL to accommodate changes caused by the introduction of novelties. By adapting existing and/or learned executors for plan operators, they enabled the agent to find a successful path to the goal after novelty injection [16,36].

Some cognitive architectures explore learning to deal with changes in the world. *FANS-RL* [14] improves reinforcement learning efficiency and stability in dynamic environments. However, FANS-RL is designed to respond to changes in environment dynamics such as motor malfunctions or changing goals in the form changing reward functions, rather than sudden and novel changes (such as *prohibitive novelties*) that may include new objects, properties, and relations. Muhammad et al. [41] propose an architecture capable of handling abrupt novelties where detection, symbolic characterization, and reasoning are required. Nevertheless, their work may encounter impasses when the available understanding of the environment is insufficient for adapting to changes (such as *obstructive novelties*). The architecture we propose detects changes in the world either through new visual information or through comparing expected models of the world and adapts accordingly to keep performing despite these changes being novel, abrupt, uninformed, and of diverse forms. Our architecture includes a complete strategy for exploring the environment when a change is detected, abstracting information about it, and learning to adapt to it.

Recent advancements in the development of fault-tolerant systems have significantly contributed to domains such as network systems [34], sensor networks [59,57], unmanned aerial vehicles (UAVs) [22], and spacecraft [10]. These methods, however, are largely domain-specific, which highlights the need for more versatile architectures. Our approach introduces a suite of methods and algorithms designed with a focus on universality, enabling their adaptation for open-world novelty handling in diverse systems.

7. Limitations and future work

The comprehensive multi-stage evaluations demonstrated that the proposed components and their embedding in a fully functional cognitive agent architecture are able to detect and accommodate a large variety novelties even without prior foreknowledge of what novelties to expect. Of course, this is only a start and the proposed methods and components can and must be extended to cover a broader set of novelties in more open-world environments. We will briefly address some of the limitations and directions for future work.

Subsymbolic modalities and the extraction of symbolic information. While the proposed architecture included visual novelty detection methods based on images, we did not use visual processing for localizing novelties or generating environmental state descriptions. Visual processing could also be used to model the actions of other agents over time. A more faithful evaluation, closer to the real world, could require agents to use only perception-based processing and ignore any pre-processed symbolic descriptions. Moreover, since other modalities like sound might be important for real-world tasks, our present architecture would have to be augmented to handle those perceptual components, utilizing similar techniques to those employed for visual novelty detection (e.g., to extract symbolic descriptions from audio such as new words or speech elements). Finally, given the relatively high false positive rates observed in our visual detector evaluations, developing improved methods that better control false positive rates could make real-world applications possible.

Novelty handling limitations. The agent architecture cannot handle conceptual novelties and novel ontological categories. In addition, there are some novelties that our agent cannot detect due to implementation limitations, even though they could, in principle, be detected and handled by our architecture. Such novelties include higher-order novelties (i.e., novelties in properties of properties

or properties of relations, etc.) as well as elements of the world that the agent cannot sense (i.e., sound). Finally, in non-episodic environments with irreversible actions, the executor learner may, during exploration, eliminate every possible solution path.

Other types of environments. While the environments we used for evaluations can get fairly complex, the employed tasks were reasonably simple and did not involve drastic changes to the environment. Moreover, discrete abstractions in terms of space and time were also simplifications that allowed us to focus on the core problems of novelty handling. Ultimately real-world agents will have to deal with the continuity of real-time and real-space tasks. However, note that our theoretical framework is not limited to discrete states and thus still applies to a wide range of real-world problems that can be addressed by a system that at the core operates on discrete symbolic representations that abstract over continuous dimensions. To cope with continuous states, we can use various components to discretize space and time. For simple cases, we can directly discretize continuous variables produced by the Domain Interface. For complex cases, the Domain Interface can pass in continuous information and have it discretized by a learning model subcomponent within the Neural Inference module.

Evaluation of agent performance. There is an intrinsic tension in our system evaluations between the implicit goal to find novelties and the explicit goal to perform the pogostick task. Some of the failures to find novelties are thus due to the agent's pursuit of its explicit goal, which always takes precedence over finding novelties. Generally, the agent will only explore novelties that directly affect it while solving the task; its investigation of novel entities in the environment as discussed at in Section 4.3 is limited to in order to allow the agent to abandon novelty exploration in favor of its explicit goal of task-solving. This raises the question of how novelty detection should be evaluated in general: should it be part of another task (as in our case), or should there be a separate goal in which the agent might be rewarded specifically for novelty detection? In the latter case, the agent could then explicitly trade off the novelty detection task with the primary task. Of course, novelty detection and characterization could be also made the only task, but this approach seems quickly infeasible and pointless in environments with too many objects to investigate.

Interpretability There is an important final point to be highlighted about evaluations of novelty detection and accommodation: it is one thing to measure the extent to which an agent can handle *prohibitive* novelties or *obstructive*, because the agent cannot accomplish the task without "getting around" them, but whether the agent detected them (if it does not explicitly indicate detection), or whether it incorporated the novelty into its knowledge base in a way that matches our expectations, is another question. The latter case requires the agent to use *our* ontology and conceptual system, or at least one that we understand, but there is no guarantee that the way the agent recorded the novelty in its own internal representations will match with such a system. While symbolic representations are at least introspectible and new symbols can be traced back to how they originated in the agent, the subsymbolic components might not be amenable to those kinds of human introspection. It is possible that "explainability mechanisms" (developed for neural networks) might be able to address this problem, at least to some extent, but likely additional mechanisms and representations will be needed to understand what the agent has learned and how it characterized its new knowledge.

8. Conclusion

The aim of this paper was to present a novel cognitive architecture framework for handling novelties in open worlds. The proposed framework for embodied agents (with sensors and effectors) consists of both symbolic and subsymbolic components that together synergistically uses logical and statistical inferences for detecting novelties as well as offline and online machine-learning techniques for accommodating novelties. Extensive multi-stage evaluations of an architecture instance of the framework performing a crafting task in a Minecraft-inspired simulation environment demonstrated that the proposed methods can usually detect, to some extent characterize, and often accommodate several types of novelties, such as novel objects or novel agents.

These encouraging results point the way for future developments. Possible improvements could include the addition of other subsymbolic input modalities like sound or tactile sensors, extended symbolic inference and novelty exploration mechanisms that use planners to plan experiments for discovery instead of fixed strategies, and yet better ways for quickly experimenting with objects and actions in the environment to discover solutions to unforeseen and unexpected problems. The overall lesson is that more emphasis needs to be placed on problem-solving and solution discovery during task performance compared to pre-training and initial knowledge engineering. Only when we have solved the problem of how to effectively determine what to try in the interest of finding novelties that enable solutions to the task at hand will our artificial agents have made the full transition from closed to open worlds.

CRediT authorship contribution statement

Shivam Goel: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Panagiotis Lymperopoulos:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Ravenna Thielstrom:** Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Evan Krause:** Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Patrick Feeney:** Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Pierrick Lorang:** Data curation, Investigation, Methodology, Software, Validation, Writing – original draft. **Sarah Schneider:** Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Yichen Wei:** Data

curation, Methodology, Software, Validation, Visualization. **Eric Kildebeck:** Data curation, Investigation, Methodology, Resources, Software, Validation, Writing – original draft, Writing – review & editing. **Stephen Goss:** Data curation, Investigation, Methodology, Resources, Software, Validation. **Michael C. Hughes:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Writing – original draft, Writing – review & editing. **Liping Liu:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Validation, Writing – original draft, Writing – review & editing. **Jivko Sinapov:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Validation, Writing – original draft, Writing – review & editing. **Matthias Scheutz:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Panagiotis Lymperopoulos reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Shivam Goel reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Ravenna Thielstrom reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Evan Krause reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Patrick Feeney reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Pierrick Lorang reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Sarah Schneider reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Yichen Wei reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Eric Kildebeck reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Stephen Goss reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Michael C. Hughes reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Liping Liu reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Jivko Sinapov reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006. Matthias Scheutz reports financial support was provided by Defense Advanced Research Projects Agency, grant number W911NF-20-2-0006.

Data availability

Polycraft code is proprietary, Novelgridworlds code is open-source. Various components of the architecture may be made available upon request.

Appendix A

A.1. Agent implementation

The proposed novelty detection and accommodation capabilities and were evaluated in two different environments, described here. For all evaluations, we created a Polycraft Interface Component (however, it works identically with novelgridworlds). Upon any *Sense* action taken by the agent, this component parses Polycraft's sensory information from a JSON format into the first-order logic predicate form used by the rest of the architecture.

The KBis implemented as a set of facts and rules in Prolog. Prolog is queried every time the truth value of a fact is needed, such as when the preconditions and postconditions of an action are being checked. The set of Prolog facts is translated to PDDL to be inserted into a problem file. Actions are written in our Action Script Language, ASL, which is also translated into PDDL as a planning domain. Our Symbolic Planner can be configured to use any PDDL planner, and for this work we're using an off-the-shelf planner, the fast-forward planning system Metric-FF. During planning, the PDDL domain and problem files are dynamically generated by scraping information from KBI and the action database. The action database contains the set of all actions that can be executed by the agent, and is stored in the GOAL MANAGER component. When populating the PDDL domain, we filter for actions that have at least one known effect, so they can be utilized by the planner.

Time spent exploring. The amount of time that the agent may wish to spend on any one exploration strategy mentioned in Section 4.3 is domain-specific. In our implementation with Polycraft, because our agent is being evaluated on whether it can achieve success before a time-out, certain phases of exploration are restricted to a set time limit, such as the Knowledge Discovery phase. This phase occurs before the agent even generates a plan to achieve the main task goal, so it is important that the agent does not waste time exploring a novelty such that it does not have enough time left over to actually solve the task. The search space of Operator Discovery strategies (other than reinforcement learning, which we allow to keep exploring until Polycraft times out due to it being the final strategy employed when all other strategies have failed) is intentionally cut down in order to limit exploration time as well. Precondition Discovery, for example, only focuses on adding variations of the *holding(self,X)* predicate, for all X where X is a novel object in the environment, as this predicate represents a prominent state aspect in Polycraft. While this has the advantage of ensuring the agent does not spend too much time on any one exploration strategy, it also creates large blind spots for the agent's

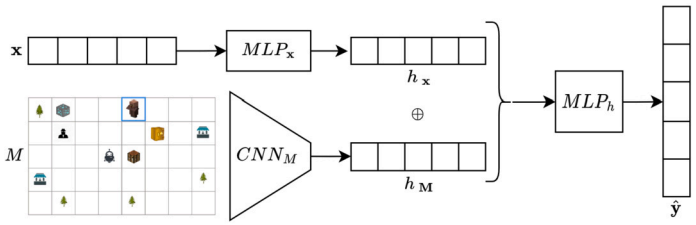


Fig. 7. Architecture diagram of the neural network underlying the agent model. The map of the environment is summarized into a vector and concatenated with state and action-history information. The network outputs probabilities for each action. ⊕ represents concatenation.

novelty solving abilities (again, reinforcement learning is utilized as a last resort to fill some of these gaps in the base agent's abilities, the success of which is discussed in Section 5.3.2).

Operator failure strategy selection. Choosing which strategies to follow in the Total Effect Failure policy, and how those strategies were executed, was implemented based on the specific operator that failed. For example, Strategy 1 (update operator and replan) is not attempted when a movement operator fails, as movement is an extremely essential part of task-solving in Polycraft and deleting its expected effects could entirely destroy the agent's ability to move anywhere. Instead, we consider the more likely explanation that the agent's knowledge base is inaccurate and the agent believes its path is clear when it is not, which is an error addressed by Strategy 2 (repeat the action). Similarly, Strategies 3 and 4 are excluded for failed operators that have more than one parameter (and thus a larger state space to explore) out of a reluctance to spend excessive time symbolically exploring a novelty instead of simply trying to find an alternative plan that avoids the novelty. These heuristics are not necessarily applicable to all other tasks or domains, but we find they are effective in Polycraft.

The initial ontology provided to the agent is accessible on this link: https://tufts.box.com/s/pab7xze409uhe5n9461m7x8dz0dn9xg9.

A.1.1. Agent Model *implementation details*

State representation. From the Polycraft symbolic state descriptions obtained through Domain Interface, a set of inputs $s'_c = (\mathbf{x}_c, M)$ are derived for the neural networks of the Agent Model for each actor ι in the state description. The vector $\mathbf{x}_\iota \in \mathbb{R}^d$ is a vectorized representation of the actor's internal states such as its inventory, inferred by tracking its actions throughout a trajectory. The matrix $M \in \{0,1\}^{40\times40\times d_2}$ is a representation of the map of the environment, as a 40x40 square is sufficient to hold every configuration of the pre-novelty polycraft map. The third dimension of M with size $d_2 \in \mathbb{N}$ stores a binary indicator indicating which known environment entity is present in each position.

As mentioned in the text, unlikely actions are detected using a threshold calculated over a validation set of state-action pairs from pre-novelty environments.

Model architecture. Each neural network in the implementation of Agent Model for the Polycraft domain is composed of 3 convolutional layers with filter size (3,3) and stride 1, and 4 multilayer perceptrons with 1 hidden layer. All layers have a hidden dimension of 64. All layers except the output layer use ReLU activations. The architecture is illustrated in Fig. 7 and summarized as follows:

$$H_M = CNN_M(mlp_M(M))$$
$$\mathbf{h}_M = flatten(H_M)$$
$$\mathbf{h}_x = mlp_x(\mathbf{x})$$
$$\hat{\mathbf{y}} = mlp([\mathbf{h}_M, \mathbf{h}_x]).$$

The output vector $\hat{\mathbf{y}}$ represents logits that compute probabilities for each action using a softmax layer.

Additional training details. The agent model neural networks are trained for a total of 100 epochs with early stopping based on the validation loss, with a patience value of 5. They are trained using the Adam optimizer with a learning rate of 0.001. Class weights are utilized during training to ensure the model does not overfit the most common actions taken by other actors (such as navigation commands). Weights are computed based on the frequency of classes (actions) in the dataset so that all classes (actions) are weighed equally.

A.1.2. Vision Model *implementation details*

Image preprocessing. Before splitting into patches, input images are cropped to remove the "Minecraft item bar" by removing the bottom 22 pixel rows of the images. For training, patches of size 32×32 pixels are sampled randomly from the cropped images and normalized between 0 and 1. Additionally, we add Gaussian noise with a standard deviation of $\frac{1}{40}$ is added to the input data.

Table 11
Hyperparameters of EXECUTOR LEARNER for reinforcement learning.

Hyperparameter	Value
ϵ_{max}	0.3
ϵ_{min}	0.05
ρ_{max}	0.35
ρ_{min}	0.05
λ decay	$-\frac{\log(0.01)}{50}$
decay rate	0.99
network weight update rate	1
positive reinforcement	1000
negative reinforcement	-100
step cost	-1
number of episodes checked for convergence	100
successful runs required to decide convergence	20

Training settings. Hyperparameter tuning of the latent representation dimension takes place over the values 50, 100, and 200 and batch sizes 32, 64, and 128. The selected value for the representation dimension is 100 and the batch size is set to 128. Training takes place over 8000 with a learning rate of 0.001 with the Adam optimizer and default Pytorch values for other parameters (0 weight decay and $(\beta_1, \beta_2) = (0.9, 0.999)$). Additional details on the architecture and training settings are available in [13].

A.2. EXECUTOR LEARNER *implementation details*

The EXECUTOR LEARNER uses one neural network for each operator. Each neural network has 1 hidden layer with 128 neurons. Table 11 summarizes all hyperparameters for the RAPid-learn [16] algorithm.

A.3. Non-novelty aware agent

The non-novelty aware agent is a simple PDDL-based planning agent. The domain and problem files as well as the code used to integrate it with the Polycraft domain are available here: https://tufts.box.com/s/qeypcyn6xyq60vvm6l0adqu647ltttgw.

A.4. Simulation setup and hardware details

All systemic evaluations are run on the LoneStar6 HPC in Texas Advanced Computing Center. Simulation details for the Polycraft domain are available in [19]. Training for neural models took place on a server with 4 NVIDIA RTX 2080Ti GPUs, and an Intel(R) Core(TM) i9-9940X processor with 130 GB of memory

A.5. Executor learner novelty handling discussion

In this section, we provide a detailed discussion of some novelty scenarios in the internal evaluation of the EXECUTOR LEARNER agent.

In the *This is just random* novelty, we showcase that the agent with an EXECUTOR LEARNER can learn to adapt to stochasticity in the environment. Table 4 shows that the EXECUTOR LEARNER learner configuration successfully solves the task in 20% of the cases, whereas the symbolic explorer solves it in a mere 2%.

In *Convince me* novelty, we observe that the EXECUTOR LEARNER aids the agent in learning to accommodate a beneficial novelty. Table 4 shows that in this novelty, the base agent's symbolic explorer fails to solve the task, while the EXECUTOR LEARNER helps solve the task almost 60% of the time.

In the *Sapling can't grow here!* novelty, we observe that the EXECUTOR LEARNER helps accommodate about 10% of the time. The goal of this novelty was to showcase the learning of spatial aspects of the learner. However, we could not observe the expected performance due to implementation limitations. The EXECUTOR LEARNER has access to the same higher-level operators as the base agent. However, to learn the spatial aspects of this novelty, the agent should have access to the lower-level navigation actions to learn a successful policy to solve the task.

In the *show me your card first* novelty, we demonstrate that the RL agent can learn a sequence of operators to accommodate the novelties. The search space of *sequences* of operators is intractable for the other exploration strategies, as they would have to explicitly enumerate all combinations of operators of different lengths. Instead, the RL-based learner has a more flexible knowledge representation in the form of neural network parameters, which allows it to generalize the knowledge it gains to unexplored states. That, combined with its knowledge-guided exploration strategy, makes it more efficient in searching over the sequences of operators. From the results in Table 4, the agent configuration that uses the EXECUTOR LEARNER successfully solves the task 20% of the time, while the symbolic explorer can never accommodate it.

Table 12
Results comparing configurations of the agents in the systemic evaluation of known novelties.

Agent	$\overline{FN}_{CDT}$	CDT%	FP%	NRP%	GA%
Base	0.41	83.8	3.0	76.4	67.7
Base+ AGENT MODEL	0.43	84.6	6.70	75.4	67.0
Base+ EXECUTOR LEARNER	0.57	85.9	2.5	76.5	69.0
Base + AGENT MODEL + EXECUTOR LEARNER	0.42	88.1	1.7	72.6	65.0

Table 13
Results comparing configurations of the agents in the systemic evaluation of unknown novelties.

Agent	$\overline{FN}_{CDT}$	CDT%	FP%	NRP%	GA%
Base	1.34	79.9	3.7	64.8	57.1
Base+ AGENT MODEL	0.91	89.5	4.9	65.5	57.2
Base+ EXECUTOR LEARNER	1.23	81.00	1.9	68.2	61.3
Base + AGENT MODEL + EXECUTOR LEARNER	0.33	89.6	1.5	58.1	54.7

A.6. Additional evaluation data

In this section, we include additional results from our systemic evaluations.

Tables 12 and 13 break down the performance of the 4 accommodation-capable agent configurations between known and unknown novelties. It is important to note that the agent configurations including the AGENT MODEL component perform better on the unknown set. This is because the novelties related to other agents were not balanced across the two sets, making the unknown set contain proportionally more such novelties. The agent model was able to detect those novelties often and as such the performance on that set is higher.

The dataset imbalance originates in the development cycle of the agent. The agent was developed in three phases, with each phase focusing on different categories of novelties. In each phase, a small number of novelties from each category were known, and the rest were unknown. At the end of each phase, the unknown novelties for that phase were revealed. In the paper, we present the final version of the system, in which unknown novelties from the first two phases are revealed and the third are hidden. Therefore, the set of unknown novelties consists of novelties from phase 3 categories, which are Environments, Goals and Events.

A.7. Additional examples

New item needed to craft pogostick. This novelty scenario belongs to the object category. In this scenario, a new unknown item (gold) is required in the crafting recipe to craft a pogostick. The agent must now learn to associate the recipe ingredient "gold" with a previously unknown resource block that now exists in the world. It must mine that block to acquire the gold and then use it to craft a pogostick.

Explanation. Before executing the pogostick goal, the agent detects the presence of the gold in the environment as a novelty through the GOAL MANAGER's initial comparison of the agent's knowledge base to the current state of the world. Before the GOAL MANAGER attempts to achieve the pogostick goal, the NOVELTY EXPLORATION component enters Knowledge Discovery and generates subgoals that involve the novel object. The GOAL MANAGER executes these subgoals. One of these subgoals, *break(gold)*, results in the effect of gold being deposited into the agent's inventory, which the GOAL MANAGER observes through the DOMAIN INTERFACE. The Exploration component creates a new operator for break(gold) that reflects this effect and stores it in the agent's knowledge repository. The agent can then plan to accomplish the requirements of the novel crafting recipe (having gold in its inventory) by mining more gold and successfully completing the task. A narrated video of our agent encountering this novelty can be viewed at the following link: https://www.youtube.com/watch?v=7X6EUkYcHSc.

New supplier actor. A previously unknown actor (the *supplier*) appears in the world (note that this is similar to the example illustrated in Section 2). The *supplier* offers a pogostick if the agent performs the interact action on it. In this variant, the pogostick can no longer be crafted, leaving the supplier as the only means of acquiring it. This scenario is categorized as an Agent novelty and constitutes a beneficial novelty for our agent, as acquiring a pogostick can be achieved with a shorter plan than usual.

Explanation. Before executing the pogostick goal, the agent detects the presence of the *supplier* in the environment in the GOAL MANAGER's initial comparison of the agent's knowledge base to the actual state of the world. Before the GOAL MANAGER attempts to achieve the pogostick goal, the NOVELTY EXPLORATION component enters Knowledge Discovery and generates subgoals that involve the novel entity, including interacting with the *supplier*. When this happens, and the *supplier* deposits the pogostick into the agent's inventory, this is observed through the DOMAIN INTERFACE, the GOAL MANAGER notes the discrepancy between the expected effect and the observed effect, and prompts the Exploration component to enter failed effect recovery. The Exploration component creates a new operator for interactWith(supplier) that reflects this effect and stores the new operator in the agent's knowledge repository. In subsequent games, the agent can plan to obtain the pogostick immediately, using the knowledge of this beneficial novelty. A narrated video of our agent encountering this novelty can be viewed at the following link: https://www.youtube.com/watch?v=06B6pHMakFs.

Traders spawn in side rooms. The *traders* do not appear in the main room that the agent starts in as in pre-novelty. They appear in a side room, which means their existence is not initially known to the agent. This is categorized as a Relations novelty.

Explanation. The agent needs to interact with the *traders* to get the available trades and later use the trades to solve the task. Without the knowledge of the trades, the agent cannot solve the task. Therefore, upon discovering the absence of traders in the main room, the agent cannot find a successful plan and encounters this novelty as prohibitive novelty.

The NOVELTY EXPLORATION component executes the recovery policy for prohibitive novelties, which executes operators to explore the unobserved parts of the environment. One of these operators is exploreRooms(), in which the agent enters all the side rooms and observes what is inside them, including the *trader* actors. Upon knowing their location, the agent interacts with the *traders* to receive information about their trade offers, generating new trade operators to represent them. From here, the agent resumes normal operation after replanning with the new operators and successfully solves the task.

A narrated video of our agent encountering this novelty can be viewed here https://www.youtube.com/watch?v=gwxdPXZiYsE.

Different items in chests. In this novelty scenario, the ingredients to craft the pogostick are scattered throughout the environment inside various identical chests, and these ingredients are not obtainable otherwise. This is categorized as Object novelty.

Explanation. The base agent often fails to solve this problem because it does not distinguish between chests and assumes that opening each chest yields the same contents. Initially, this novelty prohibits the agent from being able to plan to craft a pogostick, which prompts the Planning Failure recovery policy. During this policy, the agent will open a chest as an exploratory action to discover what lies inside it. When opening a chest and finding unexpected contents, such as rubber, the agent enters the Partial Effects Failure recovery policy and creates a new operator to reflect that opening a chest yields rubber. From this point, however, the agent still fails to plan because it needs more of the ingredients hidden in the chests but assumes that all chests only contain rubber. It eventually will give up after failing to accommodate the novelty.

However, the base+EXECUTOR LEARNER agent is successful at accommodating this novelty. Instead of giving up, the agent begins exploring using reinforcement learning. The Executor Learner places a higher priority on executing novel operators during reinforcement learning exploration, and because the agent has already created a novel operator due to the Partial Effects Failure policy, this operator is attempted often. This eventually results in the agent collecting ingredients from all the chests and being able to replan to solve the task. This is an interesting example of the EXECUTOR LEARNER overcoming unforeseen limitations of the symbolic reasoner implementation: The agent assumes all chests in the environment are interchangeable, which leads to this failure. However, the EXECUTOR LEARNER has much more flexibility in its knowledge representations and is, therefore, able to discover an executor that accommodates the novelty.

References

[1] D. Abati, A. Porrello, S. Calderara, R. Cucchiara, Latent space autoregression for novelty detection, arXiv:1807.01653, 2019.
[2] D. Abel, Y. Jinnai, S.Y. Guo, G. Konidaris, M. Littman, Policy and value transfer in lifelong reinforcement learning, in: ICML, PMLR, 2018, pp. 20–29.
[3] B.D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robot. Auton. Syst. 57 (2009) 469–483.
[4] M. Bain, C. Sammut, A framework for behavioural cloning, in: Machine Intelligence, vol. 15, 1995, pp. 103–129.
[5] Z. Bing, D. Lerch, K. Huang, A. Knoll, Meta-reinforcement learning in non-stationary and dynamic environments, IEEE Trans. Pattern Anal. Mach. Intell. 14 (2022), https://doi.org/10.1109/TPAMI.2022.3185549.
[6] T.E. Boult, S. Cruz, A.R. Dhamija, M. Gunther, J. Henrydoss, W.J. Scheirer, Learning and the unknown: surveying steps toward open world recognition, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 9801–9807.
[7] D. Bryce, J. Benton, M.W. Boldt, Maintaining evolving domain models, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, AAAI Press, 2016, pp. 3053–3059.
[8] J. Cheng, N. Vasconcelos, Learning deep classifiers consistent with fine-grained novelty detection, in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Nashville, TN, USA, 2021, pp. 1664–1673.
[9] W.C. Cheung, D. Simchi-Levi, R. Zhu, Reinforcement learning for non-stationary Markov decision processes: the blessing of (more) optimism, in: ICML, PMLR, 2020, pp. 1843–1854.
[10] D. Codetta-Raiteri, L. Portinale, Dynamic Bayesian networks for fault detection, identification, and recovery in autonomous spacecraft, IEEE Trans. Syst. Man Cybern. Syst. 45 (2014) 13–24.
[11] S. Daftry, J.A. Bagnell, M. Hebert, Learning transferable policies for monocular reactive mav control, in: International Symposium on Experimental Robotics, Springer, 2016, pp. 3–11.
[12] T. Eiter, M. Šimkus, Linking open-world knowledge bases using nonmonotonic rules, in: Logic Programming and Nonmonotonic Reasoning: 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings 13, Springer, 2015, pp. 294–308.
[13] P. Feeney, S. Schneider, P. Lymperopoulos, L. Liu, M. Scheutz, M.C. Hughes, Novelcraft: a dataset for novelty detection and discovery in open worlds, https://doi.org/10.48550/ARXIV.2206.11736, 2022.
[14] F. Feng, B. Huang, K. Zhang, S. Magliacane, Factored Adaptation for Non-stationary Reinforcement Learning, 2022, pp. 1–32, arXiv:2203.16582v2.
[15] C. Gehring, M. Asai, R. Chitnis, T. Silver, L. Kaelbling, S. Sohrabi, M. Katz, Reinforcement learning for classical planning: viewing heuristics as dense reward generators, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 32, 2022, pp. 588–596.
[16] S. Goel, Y. Shukla, V. Sarathy, M. Scheutz, J. Sinapov, Rapid-learn: a framework for learning to recover for handling novelties in open-world environments, in: IEEE International Conference on Development and Learning (ICDL), London, UK, September 12-15, 2022, IEEE, 2022, pp. 1–8, https://arxiv.org/pdf/2206.12493.
[17] S. Goel, G. Tatiya, M. Scheutz, J. Sinapov, Novelgridworlds: a benchmark environment for detecting and adapting to novelties in open worlds, in: Proceedings of 13th Workshop on Adaptive and Learning Agents (ALA) at the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), 2021.
[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Adv. Neural Inf. Process. Syst. 3 (2014), https://doi.org/10.1145/3422622.
[19] S.A. Goss, R.J. Steininger, D. Narayanan, D.V. Olivença, Y. Sun, P. Qiu, J. Amato, E.O. Voit, W.E. Voit, E.J. Kildebeck, Polycraft world ai lab (pal): an extensible platform for evaluating artificial intelligence agents, arXiv preprint, arXiv:2301.11891, 2023.
[20] L. Guan, S. Sreedharan, S. Kambhampati, Leveraging approximate symbolic models for reinforcement learning via skill diversity, arXiv preprint, arXiv:2202.02886, 2022.

[21] C. Hewitt, P.d. Jong, Analyzing the Roles of Descriptions and Actions in Open Systems, Technical Report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1983.
[22] Y. Hsiao, Z. Wan, T. Jia, R. Ghosal, A. Mahmoud, A. Raychowdhury, D. Brooks, G. Wei, V. Reddi, Mavfi: an end-to-end fault analysis framework with anomaly detection and recovery for micro aerial vehicles, in: 2023 Design, Automation & Test in Europe Conference & Exhibition (Date), 2023.
[23] R.T. Icarte, T.Q. Klassen, R.A. Valenzano, S.A. McIlraith, Reward machines: exploiting reward function structure in reinforcement learning, J. Artif. Intell. Res. 73 (2020) 173–208.
[24] M. Jin, Z. Ma, K. Jin, H.H. Zhuo, C. Chen, C. Yu, Creativity of ai: automatic symbolic option discovery for facilitating deep reinforcement learning, arXiv preprint, arXiv:2112.09836, 2021.
[25] R. Karia, S. Srivastava, Relational abstractions for generalized reinforcement learning on symbolic problems, in: Lud De Raedt (Ed.), Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 3135–3142, https://doi.org/10.24963/ijcai.2022/435.
[26] K. Khetarpal, M. Riemer, I.R. Doina Precup, Towards continual reinforcement learning: a review and perspectives, arXiv:2012.13490v1, 2020.
[27] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, Computing Research Repository (CoRR), arXiv:1312.6114, 2014.
[28] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A.A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, R. Hadsell, Overcoming catastrophic forgetting in neural networks, Proc. Natl. Acad. Sci. 114 (2017) 3521–3526, https://doi.org/10.1073/pnas.1611835114.
[29] H. Kokel, A. Manoharan, S. Natarajan, B. Ravindran, P. Tadepalli, Reprel: integrating relational planning and reinforcement learning for effective abstraction, in: ICAPS, 2021, pp. 533–541.
[30] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, in: Advances in Neural Information Processing Systems, 2017.
[31] E. Lecarpentier, D. Abel, K. Asadi, Y. Jinnai, E. Rachelson, M.L. Littman, Lipschitz lifelong reinforcement learning, arXiv preprint, arXiv:2001.05411, 2020.
[32] Y. Lee, P. Kang, Anovit: unsupervised anomaly detection and localization with vision transformer-based encoder-decoder, https://doi.org/10.48550/ARXIV.2203.10808, 2022.
[33] H. Li, R. Dou, A. Keil, J.C. Principe, A self-learning cognitive architecture exploiting causality from rewards, Neural Netw. 150 (2022) 274–292, https://doi.org/10.1016/j.neunet.2022.02.029.
[34] Z. Li, L. Yuan, P. Mohapatra, C.N. Chuah, On the analysis of overlay failure detection and recovery, Comput. Netw. 51 (2007) 3828–3843.
[35] S. Liang, Y. Li, R. Srikant, Enhancing the reliability of out-of-distribution image detection in neural networks, in: International Conference on Learning Representations, 2018, https://openreview.net/forum?id=H1VGkIxRZ.
[36] P. Lorang, S. Goel, P. Zips, J. Sinapov, M. Scheutz, Speeding-up continual learning through information gains in novel experiences, in: 4th Planning and Reinforcement Learning (PRL) Workshop at IJCAI-2022, 2022.
[37] P. Lymperopoulos, Y. Li, L. Liu, Exploiting variable correlation with masked modeling for anomaly detection in time series, in: NeurIPS 2022 Workshop on Robustness in Sequence Modeling, 2022.
[38] S. Marsland, J. Shapiro, U. Nehmzow, A self-organising network that grows when required, Neural Netw. 15 (2002) 1041–1058.
[39] N. Masuyama, N. Amako, Y. Yamada, Y. Nojima, H. Ishibuchi, Adaptive resonance theory-based topological clustering with a divisive hierarchical structure capable of continual learning, IEEE Access 10 (2022) 68042–68056.
[40] D. McDermott, J. Doyle, Non-monotonic logic i, Artif. Intell. 13 (1980) 41–72.
[41] F. Muhammad, V. Sarathy, G. Tatiya, S. Goel, S. Gyawali, M. Guaman, J. Sinapov, M. Scheutz, A novelty-centric agent architecture for changing worlds, in: Proceedings of 20th International Conference on Autonomous Agents and Multiagent Systems, 2021.
[42] R.K. Nayyar, P. Verma, S. Srivastava, Differential assessment of black-box ai agents, Proc. AAAI Conf. Artif. Intell. 36 (2022) 9868–9876, https://doi.org/10.1609/aaai.v36i9.21223.
[43] G. Pang, C. Shen, L. Cao, A.V.D. Hengel, Deep learning for anomaly detection: a review, ACM Comput. Surv. 54 (2022) 1–38.
[44] W.M. Piotrowski, S. Mohan, Model-Based Novelty Adaptation for Open-World AI, 2020.
[45] C. Richter, N. Roy, Safe visual navigation via deep learning and novelty detection, in: Robotics: Science and Systems XIII, 2017.
[46] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, G. Tesauro, Learning to learn without forgetting by maximizing transfer and minimizing interference, in: Seventh International Conference on Learning Representations, 2019.
[47] L. Ruff, J.R. Kauffmann, R.A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T.G. Dietterich, K.R. Müller, A unifying review of deep and shallow anomaly detection, Proc. IEEE 109 (2021) 756–795.
[48] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, M. Scheutz, Spotter: extending symbolic planning operators through targeted reinforcement learning, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, 2021, pp. 1118–1126.
[49] T. Schlegl, P. Seeböck, S.M. Waldstein, G. Langs, U. Schmidt-Erfurth, f-anogan: fast unsupervised anomaly detection with generative adversarial networks, Med. Image Anal. 54 (2019) 30–44, https://doi.org/10.1016/j.media.2019.01.010.
[50] J. Schmidhuber, Powerplay: training an increasingly general problem solver by continually searching for the simplest still unsolvable problem, Front. Psychol. 4 (2013) 313, https://doi.org/10.3389/fpsyg.2013.00313.
[51] L.E.B. da Silva, I. Elnabarawy, D.C. Wunsch II, A survey of adaptive resonance theory neural network models for engineering applications, Neural Netw. 120 (2019) 167–203.
[52] L. Steccanella, A. Jonsson, State representation learning for goal-conditioned reinforcement learning, arXiv preprint, arXiv:2205.01965, 2022.
[53] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, 2018.
[54] K. Talamadupula, J. Benton, S. Kambhampati, P. Schermerhorn, M. Scheutz, Planning for human-robot teaming in open worlds, ACM Trans. Intell. Syst. Technol. 1 (2010) 14:1–14:24.
[55] Q. Wang, O. Fink, L. Van Gool, D. Dai, Continual test-time domain adaptation, in: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.
[56] T. Wanyana, An Agent Architecture for Knowledge, vol. 2, Springer International Publishing, 2021.
[57] J. Xin, C. Zhou, Y. Jiang, Q. Tang, X. Yang, J. Zhou, A signal recovery method for bridge monitoring system using tvfemd and encoder-decoder aided lstm, Measurement (2023) 112797.
[58] F. Yang, D. Lyu, B. Liu, S. Gustafson, Peorl: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making, 2018, pp. 4860–4866.
[59] Z. Yemeni, H. Wang, W.M. Ismael, A. Hawbani, Z. Chen, Cfddr: a centralized faulty data detection and recovery approach for wsn with faults identification, IEEE Syst. J. 16 (2021) 3001–3012.
[60] C. Zhang, C. Jiang, Q. Xu, Pretrained back propagation based adaptive resonance theory network for adaptive learning, J. Algorithms Comput. Technol. 17 (2023) 17483026231205009.
[61] Y. Zhang, X.Y. Zhang, H. Shi, OW-TAL: learning unknown human activities for open-world temporal action localization, Pattern Recognit. 133 (2023) 109027, https://doi.org/10.1016/J.PATCOG.2022.109027.