Robots that Learn to Solve Symbolic Novelties with Self-Generated RL Simulations

Christopher Thierauf †
Deep Submergence Laboratory
Woods Hole Oceanographic Institution
christopher.thierauf@whoi.edu

Matthias Scheutz
Human-Robot Interaction Lab
Tufts University
matthias.scheutz@tufts.edu

† *Work done at Human-Robot Interaction Lab, Tufts University*

***Abstract*—A well-known challenge with symbolic systems is that they are constrained by their domain: that is, they require a predetermined set of facts to compute over and struggle to handle novelty. Machine learning techniques offer a potential resolution to this problem by allowing exploration that expands the symbolic domain. To this end, we describe the "real–sim–real" problem, where an agent must construct its own simulation environment for training and then deploy the learned policy (correcting the simulation as necessary). We then present an approach which addresses this problem, and demonstrate that an agent can use sensor data to construct its own simulation environment, train in it, and then use this newly-obtained knowledge to solve a task planning problem. Further, we show that when there is a mismatch between the generated simulation environment and the real environment, this mismatch can be resolved by the agent through exploration in the real world, and then be again applied to solve the better-understood task planning problem. We integrate this work with prior work on object-centric policies, where the agent trains directly on the environment to understand how the environment should be manipulated, which improves sim-to-real transfer and allows training to occur in real time.**

I. Introduction

Symbolic domains are known for their ability to perform long-horizon task planning, but require a fully specified domain in order to operate. This is in contrast to approaches like Reinforcement Learning (RL), which struggle with long-horizon planning but excel in providing creative solutions to problems without requiring an explicit symbolic model. The development of new behaviors and an understanding of how they may contribute to a task planning problem is essential to addressing novelties, which is an area that reinforcement learning excels in. Of particular interest to us is novelties which impede progress towards a task because the environment was not well understood prior to deployment. These "environmental task novelties" occur frequently in human-robot interaction domains in particular: for example, a robot should not fail to complete a task simply because it enters a room it has not seen before, or a cabinet that opens differently than anticipated.

To work towards addressing this challenge, we propose a framework where agents autonomously construct simulation environments based on real-world observations, and then train in this environment before deployment. We describe how the use of a Visual Language Model (VLM) can be employed to iteratively assemble a simulation environment. We then show that we can train on the environment directly to produce behaviors that are grounded in symbolic knowledge and which can be mapped to symbolic behaviors. By taking this neurosymbolic approach, the reinforcement learning agent provides its creativity to grounded long-term reasoning of a symbolic agent, combining the strengths of both. We use reinforcement learning as a source of creativity, allowing an agent to explore how the environment can be interacted with symbolically. This provides an autonomous agent the ability to "imagine" the solution "in its head", similar to how we might first visualize the solution to a challenging problem before attempting it.

Fig. 1: The Kinova arm completes its task in a toy kitchen environment, which it has no prior knowledge of. This is accomplished by training in the simulation environment which is autonomously constructed and refined through exploration in the real world. Though the simulation environment is improper, we will provide a mapping function which allows it to provide enough information to the symbolic agent to be useful in creative problem solving. In this way, the simulation is used as a symbolic exploration tool, which is refined through real-world interactions until the task can be solved symbolically.

Further, we show that even though the environment may be incorrectly constructed, this is not prohibitive to operation. First, because of the symbolic grounding and use of the environment as an symbolic exploration tool, the environment construction only need to be largely similar. We then show that interaction with the environment can provide additional information to update environmental knowledge, working towards solving the current task. Much in the same way that our mental simulation of the environment is improved through failed attempts that work towards the final successful one, we show the agent is able to learn valuable information which refines the simulation while providing symbolic knowledge.

Thus, we show a neurosymbolic framework that can handle a portion of the novelties problem by contributing:

- A definition of the "real–sim–real" problem, where an agent constructs a simulation for training, and then trains for

deployment in its environment;
- A method for autonomously constructing a highly imperfect yet suitable simulation for acquiring symbolic knowledge;
- Training agents on this imperfect environment to provide symbolic knowledge about the task's solution (extending [27]);
- A planning mechanism which uses knowledge in a planning problem to solve the task or learn more about it, facilitating long-horizon reasoning in novel situations; and
- Demonstrating that the combination of these features allows an agent to solve a task containing novelties that would otherwise break purely symbolic planners.

We begin by formalizing the underlying assumptions and mechanisms of our approach before demonstrating it in a real-world robotic domain. Additional evaluations illustrate the strengths and limitations of the method, and suggest future directions for enabling more general and robust symbolic reasoning in the presence of incomplete knowledge.

II. Background

Creative Problem Solving. In creative problem solving, an agent is tasked with finding novel solutions to task planning problems. For example, in the "MacGyver Problem" [24], an agent is tasked with solving a new problem with limited resources (requiring that the existing resources be used in a creative manner should the task be solved). Some approaches to address this class of problem have focused on the development of motor behaviors that solve new tasks, e.g., [11, 10]. In another approach, hierarchical reinforcement learning is used to imitate human creative problem solving strategies [6]. Other approaches do not explicitly consider themselves to be robot creativity but could fall under this description: for example, [2] demonstrates that a symbolic system can perform problem solving in a novel domain by providing an abstraction of motor control behavior. In each of these, creativity provides an agent with improved ability to task solve by provided an improved ability to problem solve: rather than failing when a novel scenario is encountered, this class of agents will take steps explore and solve it.

Neurosymbolic Approaches in Robotics. By "neurosymbolic" systems, we refer to approaches where the strengths of symbolic logics (long horizon explicit reasoning) are combined with the strengths of reinforcement learning or similar approaches (discovering new information or behaviors from high volumes of data). The result of combining these two different approaches to robot behavior is that the respective strengths cover the respective weaknesses: a symbolic agent can benefit from the creativity of reinforcement learning while a reinforcement learning agent can benefit from the high-level direction provided by explicit reasoning.

This approach of employing a symbolic system for high level behavior and reinforcement learning as subsymbolic has been previously explored. For example, the DIARC architecture [25] has integrated reinforcement learning strategies for problem-solving (e.g., [18], see [12] for detail). Other examples provide towards complex action planning by, for example, integrating a symbolic sense-plan-act robot architecture with an LLM as a neurosymbolic planner [4], or improving the understandability of learned policies by integrating the learning process with symbolic logics [13]. Other approaches have shown that this integration provides an improved situational awareness in an agent [21], and both [9] and [16] show that this allows agents to reason in the abstract: as we will discuss, this is a highly valuable skill for an agent to possess. Our simulations can be considered a form of reasoning in the abstract because the simulations cannot be used directly for control policy training, but can still provide valuable information as a source of exploration.

Object-Centric Reinforcement Learning. In object-centric reinforcement learning [27], the action and observation space are the objects in the environment. This approach allows an agent to learn the positions and velocities over time (e.g., forces) required to accomplish a task. By mapping the steps taken in the environment to executable robot behaviors, the task can be accomplished. These robot behaviors can be provided by a symbolic architecture and solved using classical kinematic solvers, allowing the reinforcement learning agent to provide creative problem solving skills while remaining within a symbolic architecture for planning, reasoning, and understanding.

This approach presents several advantages for our work (while also introducing some assumptions that can be limiting, see discussion in Section V). One-shot policy transfer is enabled from a simulation environment to a real-world deployment. Similarly, one-shot policy transfer is enabled across homogenous and, in some cases, heterogenous agents. Because the policy remains symbolically grounded, it is symbolically interpretable. Actions are dramatically faster to learn. However, this does require a mapping function that translates intended behavior to real-world action. This is often a safe assumption to make – classical kinematic planners are widely available – but must be mentioned. However, it is this mapping which permits imprecise construction of the simulation environment (as we will discuss). In this sense, the agent is performing a more abstract and hypothetical reasoning.

Vision Language Models (VLMs). VLMs have recently provided roboticists a powerful new tool for autonomously understanding a robot environment. Consider, for example, the "Flamingo" model [1], which successfully demonstrates few-shot learning on image and video datasets alongside textual input, or the various proprietary ChatGPT models [3] These advances have been rapidly adopted by roboticists. Consider, as a non-exhaustive sampling from the last year, usage of VLMs to enable better interpretation of human instruction [7], better intuition of the properties of objects which they will grasp [8], better interpretation of human behavior to enable more socially-appropriate robot navigation [26], or teaching via demonstration [29]. In each of these, the VLM provides a form of common-sense knowledge that a broader architecture can benefit from. Our approach does the same.

Autonomously Constructing Simulations. In [17], the authors present an approach in which a physics simulation is generated based on sensor data for use in an object manipulation task. The successful transfer of real-world data to inform a simulation environment, then application of this data to learn a reinforcement learning policy, is shown to be effective. It is also, to our knowledge, the first (and currently, perhaps only) use of the term "real–sim–real" as a research goal, although we do contribute a more formal representation of the problem. Like us, their approach uses observations of existing candidate objects to assemble a simulation environment. However, we build off of this work by contributing the ability to learn new

information about the environment and re-integrate it into the simulation. Further, we contribute construction of larger objects by assembling smaller candidate geometries (as we will discuss), and make use of a VLM (which provides additional assumptions about the environment). Finally, we contribute the formulation of the real–sim–real problem in a manner that can be integrated within a symbolic architecture, and then show this allows for the integration of symbolic and reinforcement learning approaches.

III. Proposed Method

To present our method, we first present the problem formally and then describe a technical approach to solving it. We later demonstrate this approach.

Problem Representation. We present the "real–sim–real" problem, which closely mirrors the "sim to real" problem, but adds the additional steps of constructing and then improving the simulation environment from observations of the real-world environment.

Building from [27], we first consider an environment as being composed of a set of objects. We define an object $o \in O$ as an element of the robot's scene which can be impacted by forces in the environment. An object will have some degrees of freedom in linear and rotational dimensions, and may or may not be constrained. For example, some objects can be lifted and rotated freely, while others, like doors, are limited to a specific rotation. We must therefore define an *object mobility function* $\zeta(o)$ which, given some $o \in O$, returns the n-tuple describing the degrees of freedom of that particular object. From this n-tuple we can produce an action a describing which degrees of freedom should be impacted and how.

In the same way, we must then define the *object state transition function* $\sigma(o, a)$ which, given an object, returns the 18-tuple describing the full pose of the object after applying the action[1]. Although simulation environments often provide this information directly, real-world environments will require some observation strategy, such as vision processing[2].

Continuing to build upon [27], we can use this construction to produce an MDP $M = \langle S,A,\sigma,r \rangle$ where S is the set of states produced by applying the object state transition function ζ to all objects; A is the set of actions constrained by the state transition function σ; σ is used as the state transition function; and r is the reward function where $r(s)$ for some $s \in S$ provides a reward for task completion.

Crucial for this work is to finally define some mapping between object manipulations and actions which can be employed by the robot. We state that for every every *object mobility function* $\zeta(o)$, there exists a corresponding *object manipulation function* $\eta(o)$ which the robot can employ to actually produce the motion described by ζ. If it should be the case that η cannot provide some mobility described by the appropriate ζ, then ζ must be reduced to correctly reflect the capabilities of the platform. We modify the construction of [27] by stating that it may also be the case that there will be multiple manipulation functions for a single object. This allows the representation of uncertainty about the agent's environment: it may not know how an object should best be grasped or otherwise interacted with, and so we allow multiple simultaneous options to be considered. It will be necessary to discover which option is reality at deployment time.

The real–sim–real task, then, is to first perform the "real to sim" task. In this task, an agent must autonomously determine the full set of objects (which is O) and the way those objects can be manipulated (the mobility function, $\zeta(o)$). Finding O can generally be performed using object recognition and understanding techniques. Finding $\zeta(o)$ is more challenging, and initially requires common-sense knowledge before further real-world exploration can occur. Then, the "sim to real" task can be performed. However, in the real–sim–real task, the agent must continue to iterate between simulation and real environment to refine the simulation as necessary, until the task can be completed.

Object-Centric RL Framework. As previously discussed, we leverage the object-centric RL approach introduced in [27]. In this approach, the observation space and action space are the environment itself. This provides the ability to treat reinforcement learning as a source of creative symbolic knowledge discovery, rather than a strategy for precise motor control policies. This distinction is highly valuable, because the simulation environments are often imprecise. Therefore, rather than attempting to transfer a full motor control policy, we transfer the knowledge of what events must occur to solve the task and then resolve these events using classical kinematics solvers. For example, to open a door our approach does not learn a motor policy that opens it: instead, it learns that the door must be opened and that a kinematic solver can be employed to grasp the door and pull it open. This leverages the strengths of reinforcement learning as a creative problem solver while allowing symbolic knowledge and solving to provide long-horizon reasoning.

This assumes a perfect controller, which is actually a realistic assumption to make: all real-world deployments of kinematics systems assume that the system can operate in its environment, and that if it cannot it will fail. We do not claim to add kinematic ability beyond what the mechanical system can already perform, only that we can extend the set of symbolic operators to extend the task planning problem.

Constructing Simulation Environments. To construct a simulation environment *Sim*, we start with common sense knowledge about different "primitive objects" $p \in P$. These primitive objects are the most basic representation of individual components which make up larger, more complex objects. For example, a kitchen scene is composed of a set of drawers underneath a cabinet. Should an agent encounter a scene it has never observed, we do not assume that it is in its set of primitive objects. However, we do assume here enough common sense knowledge to identify the primitive objects within it, and that different representations of each object are known (although which final scene is "correct" cannot be known at this time). This does introduce a limitation: a set of basic prior knowledge must be assumed about what *kinds* of objects the agent may encounter and that a 3D model exists for it; however, we reduce this requirement by allowing basic objects (hinges, drawers, etc) to construct more complex ones (as described next). These primitive objects will provided to the simulation

[1]Composed of x, y, z, roll, pitch, and yaw positions; plus the x, y, z, roll, pitch, and yaw velocities, plus the accelerations.

[2]While vision processing is the most common and approachable method of learning about the environment, note that we do not require the observations come from on-robot vision. A language parsing system, for example, could allow a human partner to describe the scene with spoken language.

environment using the standard .SDF format, and so any 3D format supported by .SDF files is feasible (ranging from full .STL meshes to the hand-assembled planes and prisms we show here).

Critically, however, we do not assume that a scene is produced by only one combination of primitive objects. When a drawer is encountered, it may open to the left, the right, or pull out; handles may require a twist, latch, or pull; the range of motion may be many degrees or just a few inches. Each of these possibilities introduces additional possible combinations. We explore reducing these combinations in the next section.

Similarly, we define "primitive location relations" $r \in R$, which is a first-order symbolic representation of how the location of each each $p \in Sim$ can physically relate to other p in the environment. For example, we might state that `on(block,table)`, meaning the block is on the table, or `above(microwave,sink)`, meaning the microwave is above the sink. These relations are constrained to the implementation: for object A to be placed "on" object B in simulation, a function must be implemented which sets the location of object B to the location plus height of object A. However, these functions are easy to implement; in our demonstration we provide functions for "on", "above", "left of", "right of", "below", "in front", "behind", and "inside". When constructing a simulation by defining transforms within an .SDF file, the modifications made to the transforms is straightforward. However, it will not be accurate (e.g., for above – how high above?). This inaccuracy will be addressed by not treating the environment as a perfect digital twin, but as a tool for mental simulations that informs future real-world exploration.

The value of such a construction is that it can be produced by a VLM and can be interpreted programmatically. As outlined in Algorithm 1, a simulation is constructed by first asking a VLM for possible objects in the scene (Alg 1:3) before then using this knowledge to query for position relative to some reference object (Alg 1:6) This strategy does introduce ambiguity in assembly: for example, when we state that two shelves are above a desk, does this indicate that the shelves are stacked on top of each other or that they are next to each other? However, as we will later discuss, we only use this simulation for exploration that will inform a symbolic system and so this ambiguity is not actually problematic for problem solving.

An additional source of ambiguity introduced here comes from multiple candidate implementations of an object: For example, the VLM may correctly identify a hinged door, but may not be able to identify if it opens out, left, right, etc. To address this, the task of constructing a simulation environment is actually to construct a set of viable simulation environments, from which the "real" environment can later be selected. Therefore, when ambiguity is observed, the set of simulation environments is expanded to include each possible simulation (Alg 1:7-10). This concept also includes how different object manipulation functions may be mapped onto one conceptual object: it may be necessary to turn, push, or pull a door, and each of these cases should construct a new environment.

Algorithm 1 Construction of a simulation environment.

Input: A camera feed.
Output: A set of simulation environments suitable for training via object-centric RL.

1: Let primitive objects $P = \emptyset$, relations $R = \emptyset$.
2: Let S be a set of simulation environments.
3: Query VLM to obtain P. see footnote 3
4: Randomly select reference object o from R
5: **for** n where $1 \leq n \leq |P|$ **do**
6: Get p^n relative to o via VLM, add to R see footnote 4
7: **if** only one 3D model for this o **then**
8: Place 3D model of o in sim as specified by R
9: **else**
10: Expand S such that each 3D model for o maps to an $s \in S$
11: **end if**
12: **end for**
13: **return** the full set of simulation environments

Algorithm 2 Refining the set of simulation environments.

Input: A set of real-time sensor inputs and set of simulation environments.
Output: Reduced sim set.

1: $E = \{e^0, e^1, ...e^{|E|}\}$ is a set of sim environments.
2: a is an action applicable to the current state s.
3: Perform a and observe the current state s'.
4: **for** each $e \in E$ **do**
5: s'_e is s' in e after a.
6: **if** $s'_e \neq s'$ **then**
7: Remove e from E.
8: **end if**
9: **end for**
10: **return** E.

Refining Simulation Environments through Exploration.

To refine the simulation environment (via Algorithm 2), we make use of the fact that multiple simulation environments have been constructed and it is therefore possible to produce multiple potential solutions to a single problem. In some cases these minutiae will not be relevant to the task at hand, and so they can safely be ignored. However, in many cases, they will be highly impactful: if an object is found to be difficult to grasp, or if a handle must be twisted before being pulled, etc.

In the case that it is impactful, it is observable. We employ a simple observability heuristic of task success versus failure, with our method of detecting failure being that effort to complete the task should not exceed some predetermined reasonable value. However, more complex task observability strategies are available, all of which are beyond the scope of this work. For broader works on identifying robot task failure, consider as a survey [28] or as a taxonomy [14]. But, more complex strategies than failure detection are additionally worth considering. For example, tactile sensing

[3]The VLM is provided a list of possible objects where 3D models have been obtained, and asked to list which are visible. E.g., "Provide a list of objects in the scene. Possible objects are the 'desk', 'shelf', 'microwave', 'cabinet'.". This allows the set of objects the simulation will produce remain within the set of items the symbolic agent is capable of addressing. Regular expressions are then used to ensure that only known words are passed to the rest of the system (responses like "OK, here's a list" are pruned).

[4]Similar to querying for possible objects, the object relations are provided. E.g., "Where is the desk relative to the shelf? Possible positions include 'on', 'above', 'left', or 'right'.". Regular expressions are again used to refine the output.

Algorithm 3 Action selection with real–sim–real.

Input: A set of simulation environments E and a symbolic goal s^g
Output: Action selected.

while A solution has not yet been returned: **do**
  Randomly select some $e \in E$.
  Construct reward where 1 *iff* s^g, 0 otherwise
  Attempt to solve s^g in e.
  if s^g can be solved in in e **then**
    return the policy
  else
    Remove e from E
  end if
  if $|E| == 0$ **then**
    return failure.
  end if
end while

Algorithm 4 Task handling with real–sim–real.

Input: Real-time sensor inputs, symbolic goal, and symbolic planner.
Output: A completed task.

while goal state s^g remains unachieved **do**
  if there exists some $\tau = \{a^0, a^1, ...a^{|\tau|}\}$ to solve the goal **then**
    for each $a \in \tau$ **do**
      Take action a
    end for
    return task completed.
  else
    Produce set of simulation environments (alg 1)
    Select set of actions τ to solve the novelty (alg 3)
    for each $a \in \tau$ **do**
      Perform a
      Refine simulation environments E (alg 2)
    end for
  end if
end while
return task completed.

in robot manipulators has long been explored [23, 15], and would provide interesting future work.

Handling Novelties with Real–Sim–Real. Performing task completion using this method is similar to the standard task planning problem, where an agent must find what series of action operators produce the desired goal state from a start state. However, where the typical construction of this problem would assume that there exists an action the agent knows it can perform to complete the goal at hand, we drop that assumption. It is instead necessary to find a new action operator that can bridge from two plannable states.

To perform this, a new action selection strategy is presented as Algorithm 3. In it, a reinforcement learning problem is constructed to accomplish some state, allowing the reward function to be constructed as a sparse binary reward (one if the goal state is accomplished, zero otherwise). We do not train a robot-based agent on the simulation environment and attempt to transfer to the environment, because this would not be effective (the environment is not suitably accurate). Instead, we use the object-centric reinforcement learning approach, in which the environment is manipulated by the agent directly to learn, in an abstract sense, what needs to be done to the environment to accomplish the task. A representative environment is chosen at random: this environment will either produce an action sequence which solves the task, or it will not and the agent will address this problem when it arrives at it.

This then constructs the broader task handling strategy (Algorithm 4). This is an extension of the typical sense–plan–act cycle: Algorithm 4:1-6 is, in fact, standard task planning and execution. However, we append the ability to attempt creative problem-solving in the event that no solution is available. In this case, the agent must produce a set of simulation environments, and use the new action selection strategy to generate a reasonable next step. If the chosen next step happens to be successful, then the agent has solved (or made progress towards solving) the task. In the event that it is is unsuccessful, then it has learned about its environment: the task failure suggests that the assumptions that produced its current environment cannot be correct, and so these environments built with the failed assumption can be pruned. When action selection is reattempted, it is reattempted with a new set of assumptions that have not yet been disproven.

Fig. 2: Deployment environment. The Kinova 7-DoF arm (visible on the left), can reach all elements of the kitchen play set (center). An Intel RealSense camera (right) captures the full scene.

IV. Demonstration

We consider a hypothetical kitchen environment in which the robot is tasked with helping to prepare food. However, a critical step of moving food from one location to another is impeded by the agent's lack of knowledge: though it knows its task should be completed with food in a pot on the stovetop, it does not know the steps to take to retrieve that food. Further, it does not have prior knowledge of the environment. While a set of common-sense knowledge about the environment is provided, this set of knowledge is still not enough to provide certainty about the environment dynamics. Despite this lack of knowledge, and the novelty that the agent will therefore encounter, the agent is able to learn enough about its environment to solve the task.

Setup. We employ the Kinova ULe3, a 7 degree-of-freedom robot arm that interfaces to the ROS MoveIt! suite. Mounted to the end of the arm is the Robotiq 2f-85, a segmented two-finger gripper. We modify the gripper slightly (by simply adding rubber bands) to more easily accommodate the different types of grasping it will encounter, but it is otherwise standard. The robot is placed alongside a children's play kitchen produced by the furniture manufacturer IKEA (the "DUKTIG"). This provides a reasonably robust and

standard environment for replication. Toy plush foods provide easy-to-manipulate stand-ins for real food. This also simplifies the complex task of object manipulation that falls outside the scope of our research. An Intel RealSense allows the entire scene to be pictured. A minor modification to the gripper is made to better allow it to interact with the handle. Scene is pictured as Figure 2.

The agent is provided with some prior knowledge, but not enough to complete its task. It has basic knowledge of object manipulation as it relates to the objects it can produce: for example, that if a handle is grasped and the arm moves to the appropriate location, it will have moved the door on which the handle is mounted. This knowledge is all represented using symbolic planning logics, and implemented using the "unified-planning" python library [19]. To allow the task to be completed, the agent is provided with symbolic knowledge of the task to be completed: the food in the top cabinet must be placed in the pot. The location of the pot is known as a symbolic fact, as is the symbolic knowledge of the food being 'in the pot' (or not being in the pot, depending on the timestep). This also produces the sparse binary reward function for the reinforcement learning problem.

The reinforcement learning agent is constructed using PPO, as implemented by stable-baselines [22]. The VLM is provided by ChatGPT (the GPT-4o model, as of January 2024).

Crucially, however, the agent is not provided with knowledge of how to complete this task, and it is missing critical planning operators to allow it to do so. While it can identify when the cabinet has been opened, it does not know that this is necessary to complete its task. Similarly, the agent does not know what specific type of cabinet it is about to encounter: it knows that some cabinets have handles that must be turned while some can be pulled directly. This particular environment has a latch that can simply be pulled, but we bias the agent towards incorrectly assuming the handle should turn for purposes of the demonstration.

The system is evaluated based on its ability to complete the task. The agent will attempt to interact with the environment despite the fact that it has incomplete information and incomplete symbolic operators. We evaluate as a demonstration. Note that despite the usage of reinforcement learning, the learning process occurs within a fully symbolic and deterministic framework. For this reason, testing the agent over many seeds is not attempted, because this would not be appropriate: the deterministic output of the system means the results are identically successful or unsuccessful, depending on the implementation.

Learning and Adapting to Novel Situations. When the agent begins its task, it is unaware of how to complete it: the symbolic knowledge of what steps must be employed to result in this cabinet being open is not available. It is therefore forced to employ Algorithm 1. It constructs a set of simulation environments, one of which may accurately model the real-world environment. Through experimentation in simulation, it finds a possible solution to the problem (per Algorithm 3), visible as Figure 1. With this, an action to attempt is deployed.

In the case of this demonstration, the appropriate action is to grab the handle. However, the agent is unaware of this: it is equally feasible that the handle should be twisted. Not knowing which is appropriate, it incorrectly selects the 'twisting' behavior[5]. When the action is deployed, the effort measured in the wrist reveals that turning has failed. Therefore, per Algorithm 2, the environment that produced this incorrect behavior cannot be correct, and so a new environment must be selected. The agent resets and selects a new environment: in this case, the handle does not require turning, and so a behavior is realized that it can simply be pulled. This assumption happens to be correct, allowing the task to be resolved as planned.

Despite the initial task failure, the agent has not failed in its overall task. The task initially failed because insufficient knowledge is available about the correct way to solve the task. Without knowing which simulation is accurate, the agent made its best guess and applied it to the real world. Although the guess was incorrect, this failure is valuable: it provides a deeper understanding of the real environment, providing both symbolic knowledge and allowing the simulation environments to be pruned so that they are not explored further. In this manner, the agent has both learned valuable information about its environment while solving the task.

Full Demonstration. Thus, we arrive at the full demonstration outlined in Figure 3. In Figure 3a, the Kinova has been tasked with a symbolic task to retrieve a piece of food from the top cabinet and place it in the pot at the bottom. The agent has previously constructed a simulation environment of the scene, and has been manually informed that the desired food is in the top cabinet (necessary because this information is not directly observable, see Section V).

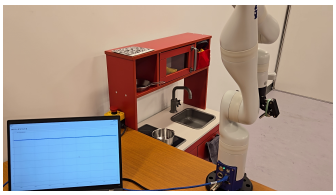
The agent has been tasked with a symbolic requirement of placing a food in the pot found on the counter. This concept is known to the agent symbolically, but the full task is not: knowledge of how to open the top cabinet is intentionally left missing. The agent has generated a simulation environment, and ambiguity in the mapping function has turned one simulation environment into several: it may be the case that the environment can be solved by deploying the 'turn handle' strategy, or it may be fine to simply pull. The agent first attempts the 'turn handle' strategy (Figure 3b), and fails (Figure 3c). It can therefore reject the assumption that the handle should be turned, and moves to a new one: that the handle can simply be pulled (Figure 3d) leading to the door being opened (Figure 3e). The agent now returns to symbolic task planning: the food can be grabbed (Figure 3f, g) and placed in the pot (Figure 3h).

V. Limitations

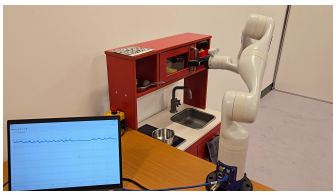
This approach introduces three key assumptions that can be considered limitations and opportunity for future work. We employ existing architectures, such as vision language models and inverse kinematics solvers, which do have their own well-documented challenges beyond our scope here – for example, VLMs may not comprehensively perceive their environment and IK solvers may encounter singularities. While these assumptions do also present issues for other types of agents, we observe that they are worth additional consideration in our approach. We now discuss these assumptions in more detail.

The Common Sense Assumption. We assume there exists a certain set of "common sense" knowledge in two key ways.

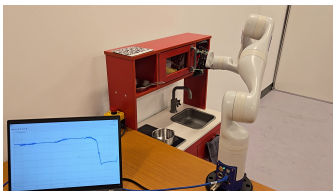
[5]While this selection would generally occur at random, we do force it to chose the wrong option for purposes of this demonstration



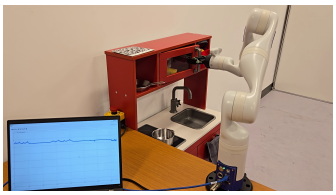
(a) Tasked with putting food in pot, but not aware of how to use cabinet.



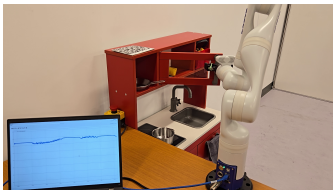
(b) Deploying the chosen strategy.



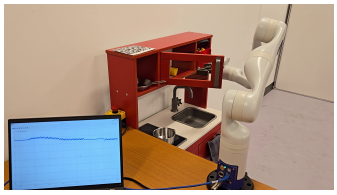
(c) Fail: turning handle is wrong action (known via monitoring force).



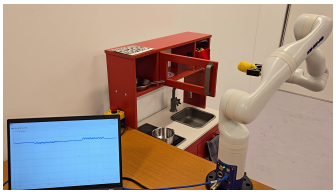
(d) Pulling instead is successful.



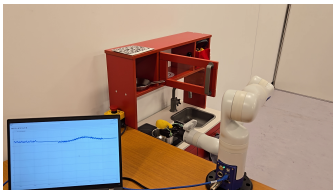
(e) Door is opened, and symbolic task resumes.



(f) Symbolic planner runs kinematic solver to grasp the handle.



(g) Food is grasped.



(h) Task complete.

Fig. 3: Demonstration of proposed method.

First, the VLM is trained on a high volume of data and therefore carries implicit assumptions. These assumptions are, in a sense, common sense knowledge: for example, what we consider as being a 'table' versus a 'desk' can be identified by a VLM even though these are both surfaces with table legs; similarly, how we define 'above' versus 'on top off'. Another form of common sense is in the primitive 3D models employed by the agent: the programmer is forced to impart some of these assumptions when they construct each primitive model, perhaps assuming that a shelf will be flat or that a cabinet can only open in one of a handful of ways.

In some ways, these assumptions are clearly limiting: while this approach does substantially widen the ability of an agent to handle novelty, it does still require that the novelty can be represented as a composition of known primitives. However, in other ways these assumptions can be valuable: common sense knowledge provided by a VLM can inform the physics dynamics of a simulation (e.g., by providing a reasonable assumption about the mass or friction of an object). Regardless, common sense knowledge remains a key element of this approach.

The Observability Assumption. We assume that everything that is necessary to complete the task is observable to the agent, or otherwise available. In the case of the food in the top cabinet, for example, the agent is unable to see and perceive that it is there, and so we are forced to provide it this knowledge directly. While this is a limititation, it is not as prohibitive as it may sound.

First, recall that this approach is integrated into a symbolic framework. As a result, the ability to inject symbolic knowledge is trivial. This knowledge can be provided by more than just vision processing: language can be used as a tool to provide symbolic knowledge (consider [20, 5]), and so this approach could become used as a part of a larger human interaction task where the goal is spoken alongside other task-critical information.

Additionally, this knowledge could be obtained during deployment-time and trigger a new generation of simulation environments. While our approach to exploring the environment focused on identifying task failure, approaches focusing on agent surprisal would be a valuable method of identifying that an agent has encountered new information that may be task-relevant and worth reconstructing a set of simulations over.

Effort of domain construction. This work does introduce a duplication of effort for the programmer: it is necessary to maintain both a physics domain for the reinforcement learning agent and a symbolic domain for the symbolic agent. For a purely reinforcement learning based approach, the symbolic domain would not be necessary; conversely, for a purely symbolic approach a set of components for a reinforcement learning domain would not be necessary. Additionally, a truly general agent will possess object primitives that it may never encounter (but must be prepared for anyway).

This is certainly a limitation, but not one that eliminates the viability of this approach. The ability to address novelty in reinforcement learning and symbolic domains has broad impact, and the ability to perform this outside of a simulation environment (which further differentiates our work from prior art) us also worth considering. However, in cases where handling novelty is not a priority, this approach admittedly adds development effort that would not otherwise be necessary. The decision to implement it, then, becomes a tradeoff that should be carefully considered before attempting it.

VI. Conclusion

This work advances research towards an open challenge in robots that use reinforcement learning. Novelty handling in RL agents remains a valuable and unsolved problem, but by working towards the real–sim–real problem we have shown that these agents can more reasonably handle never-before-seen environments. With the object-centric RL approach, where the agent learns directly on the objects in the environment, our system simplifies the mapping between simulation and reality as a classical kinematics problem, which enables deployment while also benefiting from other features of that approach (in particular, symbolically grounded behavior). Future work could extend this approach by strengthening the common-sense knowledge assumption and further explore its application in dynamic, real-world tasks. Working towards this problem enables robots bring the strengths of reinforcement learning to real-world environments with novelties that could not otherwise be addressed.

References

[1] J. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, M. Arthur, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Bińkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan. Flamingo: a visual language model for few-shot learning. 2022. doi: 10.48550/arxiv.2204.14198.

[2] Barrett Ames, Allison Thackston, and George Konidaris. Learning symbolic representations for planning with parameterized skills. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 526–533. IEEE, 2018.

[3] OpenAI Authors. ChatGPT. Accessed January 2024, https://chatgpt.com/.

[4] Alessio Capitanelli and Fulvio Mastrogiovanni. A framework for neurosymbolic robot action planning using large language models. *Frontiers in Neurorobotics*, 18:1342786, 2024.

[5] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*, 2022.

[6] Thomas R Colin, Tony Belpaeme, Angelo Cangelosi, and Nikolas Hemion. Hierarchical reinforcement learning as creative problem solving. *Robotics and Autonomous Systems*, 86:196–206, 2016.

[7] Zichao Dong, Weikun Zhang, Xufeng Huang, Hang Ji, Xin Zhan, and Junbo Chen. Hubo-vlm: Unified vision-language model designed for human robot interaction tasks. *arXiv preprint arXiv:2308.12537*, 2023.

[8] Jensen Gao, Bidipta Sarkar, Fei Xia, Ted Xiao, Jiajun Wu, Brian Ichter, Anirudha Majumdar, and Dorsa Sadigh. Physically grounded vision-language models for robotic manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12462–12469. IEEE, 2024.

[9] Y. Ge, S. Zhang, Y. Cai, T. Lu, H. Wang, X. Hui, and S. Wang. Ontology based autonomous robot task processing framework. *Frontiers in Neurorobotics*, 18, 2024. doi: 10.3389/fnbot.2024.1401075.

[10] Evana Gizzi, Mateo Guaman Castro, and Jivko Sinapov. Creative problem solving by robots using action primitive discovery. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 228–233. IEEE, 2019.

[11] Evana Gizzi, Amel Hassan, Wo Wei Lin, Keenan Rhea, and Jivko Sinapov. Toward creative problem solving agents: Action discovery through behavior babbling. In *2021 IEEE International Conference on Development and Learning (ICDL)*, pages 1–7. IEEE, 2021.

[12] Shivam Goel, Panagiotis Lymperopoulos, Ravenna Thielstrom, Evan Krause, Patrick Feeney, Pierrick Lorang, Sarah Schneider, Yichen Wei, Eric Kildebeck, Stephen Goss, et al. A neurosymbolic cognitive architecture framework for handling novelties in open worlds. *Artificial Intelligence*, 331:104111, 2024.

[13] Amr Gomaa, Bilal Mahdy, Niko Kleer, Michael Feld, Frank Kirchner, and Antonio Krüger. Teach me how to learn: A perspective review towards user-centered neuro-symbolic learning for robotic surgical systems. *arXiv preprint arXiv:2307.03853*, 2023.

[14] Arda Inceoglu, Eren Erdal Aksoy, and Sanem Sariel. Multimodal detection and classification of robot manipulation failures. *IEEE Robotics and Automation Letters*, 2023.

[15] Zhanat Kappassov, Juan-Antonio Corrales, and Véronique Perdereau. Tactile sensing in dexterous robot hands. *Robotics and Autonomous Systems*, 74:195–220, 2015.

[16] N. Krishnaswamy and J. Pustejovsky. Affordance embeddings for situated language understanding. *Frontiers in Artificial Intelligence*, 5, 2022. doi: 10.3389/frai.2022.774752.

[17] Naijun Liu, Yinghao Cai, Tao Lu, Rui Wang, and Shuo Wang. Real–sim–real transfer for real-world robot control policy learning with deep reinforcement learning. *Applied Sciences*, 10(5):1555, 2020.

[18] Pierrick Lorang, Shivam Goel, Yash Shukla, Patrik Zips, and Matthias Scheutz. A framework for neurosymbolic goal-conditioned continual learning in open world environments. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12070–12077. IEEE, 2024.

[19] Andrea Micheli, Arthur Bit-Monnot, Gabriele Röger, Enrico Scala, Alessandro Valentini, Luca Framba, Alberto Rovetta, Alessandro Trapasso, Luigi Bonassi, Alfonso Emilio Gerevini, et al. Unified planning: Modeling, manipulating and solving ai planning problems in python. *SoftwareX*, 29:102012, 2025.

[20] Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*, 2023.

[21] M. Pomarlan, S. De Giorgis, R. Ringe, M. M. Hedblom, and N. Tsiogkas. Hanging around: cognitive inspired reasoning for reactive robotics. *Frontiers in Artificial Intelligence and Applications*, 2024. doi: 10.3233/faia241288.

[22] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[23] R Andrew Russell. *Robot tactile sensing*. Prentice-Hall, Inc., 1990.

[24] Vasanth Sarathy and Matthias Scheutz. Macgyver problems: Ai challenges for testing resourcefulness and creativity. *Advances in Cognitive Systems*, 6:31–44, 2018.

[25] Matthias Scheutz, Thomas Williams, Evan Krause, Bradley Oosterveld, Vasanth Sarathy, and Tyler Frasca. An overview of the distributed integrated cognition affect and reflection diarc architecture. *Cognitive architectures*, pages 165–193, 2019.

[26] Daeun Song, Jing Liang, Amirreza Payandeh, Amir Hossain Raj, Xuesu Xiao, and Dinesh Manocha. Vlm-social-nav: Socially aware robot navigation through scoring using vision-language models. *IEEE Robotics and Automation Letters*, 2024.

[27] Christopher Thierauf and Matthias Scheutz. Fixing symbolic plans with reinforcement learning in object-based action spaces. 2024.

[28] Monica L Visinsky, Joseph R Cavallaro, and Ian D Walker. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering & System Safety*, 46(2):139–158, 1994.

[29] Beichen Wang, Juexiao Zhang, Shuwen Dong, Irving Fang, and Chen Feng. Vlm see, robot do: Human demo video to robot action plan via vision language model. *arXiv preprint arXiv:2410.08792*, 2024.