Reward Shaping and Action Masking for Compositional Tasks using Behavior Trees and LLMs

Nicholas Potteiger NICHOLAS.POTTEIGER@VANDERBILT.EDU
Ankita Samaddar ANKITA.SAMADDAR@VANDERBILT.EDU
Taylor T. Johnson TAYLOR.JOHNSON@VANDERBILT.EDU
Xenofon Koutsoukos XENOFON.KOUTSOUKOS@VANDERBILT.EDU
Vanderbilt University, Nashville, TN

Abstract

Decomposing complex tasks into a sequence of simpler subtasks can improve learning efficiency for an autonomous agent. Reinforcement learning (RL) can be used to optimize agent policies to complete subtasks, but requires well-defined subtask rewards and benefits from action masking. Recent work uses large language models (LLMs) to automate reward shaping and action masking, however none of them fully address reactivity to subtask failure and modularity to varying objects for compositional tasks. To overcome these challenges, we develop masking reward behavior tree (MRBT), a symbolic structure used as a reactive and modular reward and action mask function. We design an MRBT template and derive logical specifications to construct and verify MRBTs for a sequence of object-interaction subtasks. Further, we develop an automated pipeline that uses an LLM to generate MRBTs robust to varying task objects, an SMT-solver to verify correctness of specifications, and a neurosymbolic RL loop to train agents on compositional tasks. Experiments demonstrate successful generation and refinement of five MRBTs, consistently improving training efficiency and task success rates over baselines and MRBTs without action masking. We further highlight three advantages of MRBTs: transferability, modularity, and verifiability.
Keywords: behavior tree, large language models, reward shaping, action masking, robotics

1. Introduction

Decomposing complex tasks allows autonomous agents to learn simpler behaviors. Many robotic tasks, such as assembly and drone delivery, involve sequential subtasks requiring navigation and manipulation of objects. Although reinforcement learning (RL) can train policies for these subtasks, it benefits from reward shaping and action masking to improve learning efficiency (Huang and Ontañón (2022)). Designing such rewards and action masks manually is challenging, as it requires identifying subtasks, ensuring reactivity to subtask failures, and maintaining modularity across varying task objects. These challenges motivate automated design of reactive, modular rewards and action masks for compositional tasks.

To automatically design rewards and action masks, previous work has leveraged large language models (LLMs) or vision language models (VLMs). They can be fine-tuned to evaluate task completion (Du et al. (2023); Fan et al. (2022)). Other approaches use pre-trained models to generate reward code (Ma et al. (2023); Venuto et al. (2024); Qu et al. (2025); Guo et al. (2025)) or construct reward machines (Alsadat et al. (2025); Alsadat and Xu (2025)). Similar to our work, VLM-CaR (Venuto et al. (2024)) decomposes tasks into a sequence of subtask reward code that is tested using expert demonstrations. Very few

works have examined how LLMs can be used for action masking, with one work using LLM outputs to generate dynamic action masks (Zhao et al. (2025)). While prior work provides a strong foundation for automatic reward shaping and action masking, it does not fully address reactivity or modularity in compositional tasks, nor does it consider joint reward and action mask functions. Related work section is in Appendix A.

To address reactivity and modularity, we develop a reward and action mask function using a behavior tree (BT). BTs are interpretable, reactive, and modular, making them well-suited for compositional tasks and LLM reasoning. The BT, referred as masking reward BT (MRBT), outputs rewards and action masks from leaf behaviors while the BT coordinates execution. We design an MRBT template for sequential object-interaction subtasks, with placeholders for subtasks, action masks, and first-order logic formulas conditioning rewards and action masks. The template enables reactive backtracking for subtask failure. We derive logic specifications to verify the correctness of the logic formulas with respect to task completion. The specifications are verified over finite-horizon trajectories generated by an SMT solver under a symbolic encoding of the environment's transition dynamics.

To automatically design rewards and action masks that are reactive to subtask failure and modular to varying task objects, we leverage the MRBT template, an LLM, and an SMT solver. We define a task space that encapsulates possible combinations of the same task structure with varying task objects. The LLM inputs the MRBT template and task-specific information to generate MRBTs that are robust to any task in a task space. The SMT-solver checks satisfiability of the specifications, re-prompting the LLM if unsatisfiable. The MRBT is integrated into a neurosymbolic RL loop to train an agent over the task space.

The main technical contributions of this work are:

1) Reward and action mask functions for compositional tasks using *masking reward behavior tree (MRBT)*. An MRBT outputs rewards and action masks from executed leaf behaviors.

2) An MRBT template to construct MRBTs for sequential object-interaction subtasks. In addition, logical specifications are derived to check subtask completion, proximity to subtask objects prior to subtask completion, and that subtasks can be composed to yield maximal reward without regression due to subtask failure. They are verified with an SMT-solver constrained by a symbolic model of the environment dynamics. Alternatively, expert demonstrations can be used for testing specifications.

3) An automated pipeline[1] for generating and verifying MRBTs from the MRBT template that are robust to any task in a task space. The MRBTs are integrated into a neurosymbolic RL training loop for optimizing an agent to complete any task in a task space.

4) A comprehensive evaluation of MRBTs is conducted, including pipeline analysis and an ablation study against two baselines across five task spaces in two environments. ChatGPT-5 (OpenAI (2025)) and Z3 (De Moura and Bjørner (2008)) are used to generate, verify, and refine MRBTs. Higher training efficiency and task success are consistently achieved using an MRBT compared to baselines and an MRBT without action masking. In the most complex task space, $\geq 80\%$ average success rate is achieved, compared to $\leq 70\%$ for baselines.

5) An evaluation of advantages of MRBTs. Agent policies trained using MRBTs transfer navigation skills to a realistic quadcopter simulator (AirSim (Shah et al. (2018))), exhibit

1. Code and LLM prompts available at https://github.com/npotteig/bt_as_reward

greater modularity than reward machines as task complexity grows, and offer theoretical guarantees via Z3.

2. Autonomous Agents for Compositional Tasks

We study compositional tasks expressed in natural language as sequences of subtasks with object-interaction, where a template defines a task space of structurally equivalent instances that vary only in the task objects. We represent a task space as $\mathcal{M} = \langle \ell, \mathcal{V}, \mathcal{G} \rangle$, where ℓ is a natural language template, $\mathcal{V}$ is a set of discrete symbolic variables that fill the template, and $\mathcal{G}$ is a set of first-order logic formulas that represent task completion.

The agent must complete tasks in an environment $\mathcal{E}$ represented as a partially observable Markov decision process with logic formulas, $\mathcal{E} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \omega, p, \mathcal{L}, l, r, \gamma \rangle$. $\mathcal{E}$ consists of a state space $\mathcal{S}$, an observation space $\mathcal{O}$, an action space $\mathcal{A}$, an observation function $\omega : \mathcal{S} \times \mathcal{A} \to \mathcal{O}$, a transition function $p(s' \mid s, a)$, a finite set of first-order logic formulas $\mathcal{L}$ constructed from predicates derived from $\mathcal{S}$, a labeling function $l : \mathcal{M} \times \mathcal{S} \to 2^{\mathcal{L}}$ that maps tasks and states to subsets of logic formulas we refer to as labels, a reward function $r : (2^{\mathcal{L}})^+ \to \mathbb{R}$, and discount factor γ. The reward function inputs a label history $\lambda_{t+1} = \langle l(s_0, m), ..., l(s_t, m), l(s_{t+1}, m) \rangle \in (2^{\mathcal{L}})^+$ to compute a scalar reward. Further, an action mask function $\mu : (2^{\mathcal{L}})^+ \to 2^{\mathcal{A}}$ can dynamically restrict available actions. The task input to the labeling function are values in $\mathcal{V}$ and $\mathcal{G}$.

The objective of the agent is to find a policy $\pi^*(a \mid m, o)$, where for every task in a task space $\forall m^i = \langle \ell, v^i, g^i \rangle \in \mathcal{M}$, π^* generates a trajectory of state-action pairs $((s_0, a_0), ..., (s_T, a_T))$ where the terminal state s_T satisfies g^i. The task input is a sequence of integers, where each integer represents a word in the natural language task.

The problem is to use an LLM to design reward and action mask functions that guide the agent toward learning π^*. Given a task space $\mathcal{M}$ and environment $\mathcal{E}$, the LLM takes as input task-specific information such as, $\mathcal{M}$, predicates from $\mathcal{S}$, and $\mathcal{A}$. The output is a set of logic formulas $\mathcal{L}$, labeling function $l(s, m)$, reward function $r(\lambda)$, and action mask function $\mu(\lambda)$, given label history $\lambda \in (2^{\mathcal{L}})^+$, state $s \in \mathcal{S}$, $\forall m \in \mathcal{M}$.

We use MiniGrid LockedRoom (Chevalier-Boisvert et al. (2023)) as a running example, where the agent must find a key in one of six rooms, unlock a door, and reach the goal. The task space is defined by $\ell =$ *"get the* {*key_color*} *key from the* {*room_color*} *room, unlock the* {*door_color*} *door and go to the goal"*, $\mathcal{V} = \{key\_color, room\_color, door\_color\}$, and $\mathcal{G}$ contains a single logic formula g^0 that checks when the agent reaches the goal. Color values are drawn from {*red, green, blue, purple, yellow, grey*}. The state space $\mathcal{S}$ is an $N \times N$ grid with object information, including the agent, keys, doors, and the goal. Predicates derived from $\mathcal{S}$ include *agent_pos*, *key_pos*, *door_pos*, *door_state*, and *goal_pos*, where positions are 2D tuples and $door\_state \in \{open, closed, locked\}$. Object attributes may be indexed by color (e.g. key_pos_{red}), and missing or occluded objects are assigned a value of -1 for position and state. The action space is $\mathcal{A} = \{left, right, forward, pickup, drop, toggle, done\}$.

3. Masking Reward Behavior Trees

To learn agent policies for compositional tasks, we define a masking reward behavior tree (MRBT) used as a reactive and modular reward and action mask function. We design a

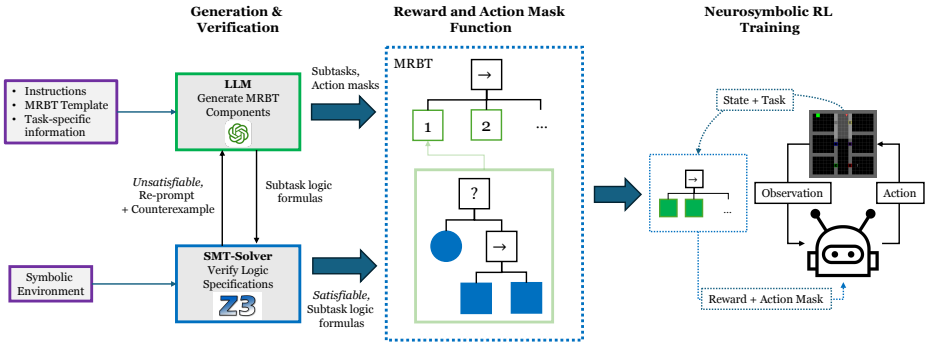

Figure 1: Automated pipeline to *generate* and *verify* MRBTs and their use in *training*.

template to construct MRBTs for sequences of object-interaction subtasks. From these MRBTs, we derive logical specifications to verify correctness of MRBT logic. Finally, we present an automated pipeline (Figure 1) that uses the template, specifications, an LLM, and an SMT solver to generate and verify MRBTs, which are then integrated into a neurosymbolic RL loop.

3.1. Preliminaries

Behavior Trees: BTs are symbolic policies known for reactivity, modularity, and interpretability. A BT executes by propagating a tick starting from its root through a finite set of behaviors in depth-first order; each behavior returns *Success*, *Running*, or *Failure*. Internal nodes are control nodes, such as Sequence and Fallback that control propagation of the tick to its children. Leafs nodes are execution nodes, such as Condition and Action, that execute behavior in the environment when ticked.

Reward Machines: A reward machine (RM) is a finite state machine that takes abstracted descriptions of the environment as input and outputs reward functions (Icarte et al. (2018)). Given a set of propositions, or more generally first-order logic formulas, $\mathcal{L}$, state space $\mathcal{S}$, and action space $\mathcal{A}$, an RM is a tuple $\langle U, u_0, F, \delta_u, \delta_r \rangle$ where U is a finite set of machine states, $u_0 \in U$ is the initial state, F is a finite set of terminal machine states such that $U \cap F = \emptyset$, $\delta_u : U \times 2^{\mathcal{L}} \to U$ is the state-transition function, $\delta_r : U \times 2^{\mathcal{L}} \to \left[\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}\right]$ is a state-reward function that outputs a reward function based on the RM state. A *simple reward machine* is only dependent on $\mathcal{L}$ where $\delta_r : U \times 2^{\mathcal{L}} \to \mathbb{R}$ outputs a scalar reward.

Neurosymbolic Reinforcement Learning: Neurosymbolic RL is an emerging class of methods that aims to enhance reasoning and learning by combining symbolic components (e.g., BTs) with RL components (e.g., neural networks) (Acharya et al. (2024), Potteiger et al. (2024)). The methods fall under one of three categories: *learning for reasoning*, *reasoning for learning*, and *learning-reasoning*. RMs fall under reasoning for learning, as they serve as symbolic components to guide the output of a neural policy by reward shaping. Our work falls under reasoning for learning. We use a BT to provide feedback to the agent.

3.2. Masking Reward BT Formalization

A masking reward BT (MRBT) is a type of BT that inputs abstracted descriptions of the environment and outputs a real-valued reward and discrete action mask. The execution nodes (leaves) of an MRBT are simple RMs with action masks defined as follows:

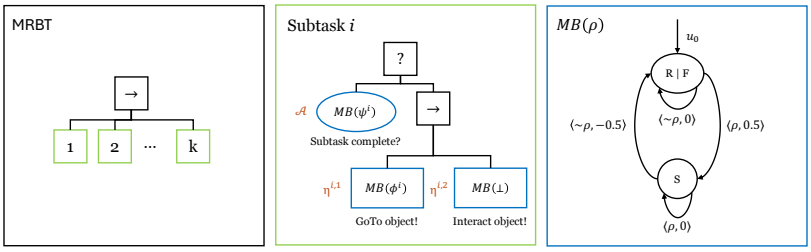

Figure 2: Masking reward behavior tree (MRBT) template.

Definition 1 *(masking behavior reward machine) Given a set of logic formulas $\mathcal{L}$ and discrete action space $\mathcal{A}$, a masking behavior reward machine is $MB = \langle U, u_0, \delta_u, \delta_r, \eta \rangle$, where U is a set of states $\{Success, Running, Failure\}$, u_0, δ_u, and δ_r are defined as in a simple reward machine (see Section 3.1), and $\eta \in 2^{\mathcal{A}}$ is an action mask.*

Masking behavior reward machines (MBRMs) are structured into an MRBT according to the following definition:

Definition 2 *(masking reward behavior tree) An MRBT is a BT $\mathcal{T}$ whose leaves are a set of MBRMs $\mathcal{B}_{leaf}$ defined over a common set of logic formulas $\mathcal{L}$ and action space $\mathcal{A}$. An MRBT maintains a state assignment $u : \mathcal{B}_{leaf} \to U$ initialized by u_0.*

An MRBT starts from the initial state assignment u_0, and at each timestep t, maintains a state assignment $u_t : \mathcal{B}_{leaf} \to U$. Given a label assignment $\sigma_{t+1} \subseteq \mathcal{L}$, a tick is propagated through $\mathcal{T}$ starting from the root, producing an ordered sequence of ticked leaves $\varepsilon_{t+1} = (b_1, \ldots, b_j)$, where $b_i \in \mathcal{B}_{leaf}$. The state assignment is updated by applying the transition function only to the ticked leaves:

$$u_{t+1}(b) = \begin{cases} \delta_u^b\big(u_t(b), \sigma_{t+1}\big) & \text{if } b \in \varepsilon_{t+1} \\ u_t(b) & \text{otherwise} \end{cases} \tag{1}$$

The reward is the sum of rewards of the ticked leaves $r_{t+1} = \sum_{b \in \varepsilon_{t+1}} \delta_r^b\big(u_t(b), \sigma_{t+1}\big)$ and the action mask is the mask of the last ticked leaf $\eta_{t+1} = \eta^{b_j}$.

3.3. Masking Reward BT Template

We introduce an MRBT template (Figure 2) to systematically construct an MRBT and a labeling function for a given environment $\mathcal{E}$ and task space $\mathcal{M}$. The template assumes tasks are decomposed into sequential subtasks involving navigation and object interactions, making it suitable for domains like robotic assembly and drone delivery.

The template inputs k subtasks and $2k$ action masks and outputs an MRBT with labeling function $l(s, m)$. Each subtask i has two masks, $\eta^{i,1}, \eta^{i,2} \in \mathcal{A}$. We first define logic formulas $\mathcal{L}$ and BT $\mathcal{T}$. $\mathcal{L}$ contains $2k$ formulas: ψ^i for subtask COMPLETION and ϕ^i for OBJECT PROXIMITY to the subtask object. The BT has a sequence root with k subtree children, each comprising a fallback, a sequence, and three MBRMs (one condition and two actions) represented as $MB(\rho)$ that share a transition structure conditioned on $\rho \in \mathcal{L}$. The transition structure enables a positive reward to be outputted when ρ is switched to

True and a reward penalty when switched to *False*. Penalties signal to the agent that a subtask must be re-completed, and state changes in the MBRM trigger MRBT backtracking when a state moves from *Success* to *Failure* or *Running*, ensuring the correct action mask is outputted.

The condition MBRM $MB(\psi^i)$ returns *Success* if the subtask is complete and *Failure* otherwise. The navigation MBRM $MB(\phi^i)$ returns *Success* when the agent is near the object and *Running* otherwise. The interaction MBRM $MB(\bot)$ always returns *Running* until preempted by subtask completion or navigation. $MB(\psi^i)$ outputs the full action space, while $MB(\phi^i)$ and $MB(\bot)$ output $\eta^{i,1}$ (navigation) and $\eta^{i,2}$ (interaction), respectively. Finally, $\mathcal{L}$, $\mathcal{T}$, and $\mathcal{A}$ are used to derive an MRBT by Definition 2.

Executing the MRBT requires evaluating the logic formulas in $\mathcal{L}$ to obtain a label assignment $\sigma_{t+1} = l(s_{t+1}, m)$ using a labeling function $l(s, m)$. The labeling function is expressed as the logic formulas in $\mathcal{L}$ satisfied from state $s \in \mathcal{S}$ and task $m \in \mathcal{M}$:

$$l(s,m) = \left\{\psi^i \mid 1 \le i \le k,\ (s,m) \models \psi^i\right\} \cup \left\{\phi^i \mid 1 \le i \le k,\ (s,m) \models \phi^i\right\} \tag{2}$$

Given a task $m \in \mathcal{M}$ and current state assigment u_t, executing one tick of the MRBT with input σ_{t+1} produces a sequence of ticked leaves ϵ_t. The reward and action mask are then given by

$$r(\lambda_{t+1}) = \sum_{b \in \varepsilon_t} \delta_r^b(u_t(b), \sigma_{t+1}) \tag{3}$$

$$\mu(\lambda_{t+1}) = \eta^{b_j} \tag{4}$$

where b_j is the last leaf in ϵ_t, and $\lambda_{t+1} = (\lambda_t, \sigma_{t+1})$.

Example 1 *An MRBT for MiniGrid LockedRoom contains $k = 4$ subtasks: Open key-room door, Pick up key, Unlock/open locked door, Reach goal. For subtask 1, ψ^1 checks if the door is open or occluded due to the agent being at the door position: $[door\_state_{room\_color} = open \vee door\_state_{room\_color} = -1]$ and ϕ^1 checks if the Manhattan distance between the agent and door is ≤ 1 or the door is occluded: $manhattan\_distance(agent\_pos, door\_pos_{room\_color}) \le 1 \vee door\_state = -1$. The action mask $\eta^{1,1}$ contains actions* {left, right, forward} *to navigate and $\eta^{1,2}$ contains the actions* {left, right, toggle} *to open the door.*

3.4. Verification of Masking Reward BTs

To verify the subtask logic formulas of an MRBT constructed from the template, we develop a symbolic model of the environment transition dynamics $\mathcal{E}^{sym}$ in an SMT-solver. We derive logical specifications derived from the formulas that are checked over task space $\mathcal{M}$ and trajectories $\tau = (s_0, \ldots, s_{H-1})$ of horizon H generated by the SMT-solver constrained by $\mathcal{E}^{sym}$. See Appendix C for a table of specifications.

Completion: We verify that if the task is complete at final timestep $H - 1$, then the subtask must have been completed. This ensures that $MB(\psi^i)$ will always transition to *Success* at least once when the task is complete. Further, to prevent trivial solutions (i.e. $\psi^i = \top$) where the subtask will be complete in most trajectories and tasks, we verify the existence of N distinct solutions where ψ^i is never *True*.

Object Proximity: We verify that the agent is in proximity to the subtask object at the timestep immediately preceding subtask completion. This ensures that $MB(\phi^i)$ is at the

Success state at time t, before the subtask completes at $t+1$, and implies that $MB(\perp)$ is in the *Running* state at t. Together, these conditions preserve the sequence of navigating to and interacting with the subtask object prior to completion in the MRBT template. Further, we verify the existence of N distinct solutions where ϕ^i is never *True*.

Non-regressive Maximal Reward: To achieve maximal reward under the MRBT template, all subtasks must be completed. Assuming deterministic dynamics and non-regressive subtasks (i.e., ψ^i can remain *True*), we verify the existence of N distinct solutions such that if the task is complete at final timestep $H-1$, then each ψ^i persists to be *True*.

3.5. Automating Generation, Verification, and Training

We present an automated pipeline to generate and verify MRBTs and use them to train an agent policy over a task space. Given instructions, an MRBT template, and task-specific information, the pipeline uses an LLM to generate subtasks, action masks, and subtask logic formulas. The logic formulas are verified for satisfiability using an SMT-solver and symbolic environment model $\mathcal{E}^{sym}$ using the specifications (Section 3.4).

The LLM inputs a system prompt, including instructions, the MRBT template, and task-specific information such as, $\mathcal{M}$, predicates from $\mathcal{S}$, and $\mathcal{A}$. The instructions prompt the LLM to generate subtasks and logic formulas for a BT. First, the LLM is prompted to generate a list of subtasks. For each subtask i, the LLM generates a completion logic formula ψ^i, an object proximity logic formula ϕ^i, and action masks: $\eta^{i,1}$, $\eta^{i,2}$. The prompts enforce dependence on both state and task, ensuring robustness across the state and task spaces. The resulting ψ^i, ϕ^i, $\eta^{i,1}$, $\eta^{i,2}$, are used in the MRBT defined by the generated subtasks (illustration in Appendix B).

The SMT-solver inputs the generated logic formulas, checks for satisfiability of the logic specifications subject to $\mathcal{E}^{sym}$ for trajectories of size H, and either terminates with the specifications being satisfied or returns a debug prompt that is used to refine the generated logic formulas. The debug prompt includes the subtask logic formula that violated the specification and a failure description. For Non-Regressive Maximal Reward, it reports the first subtask violating persistence. For Completion and Object-Proximity correctness, an unsatisfiable specification yields a counterexample trajectory–task pair satisfying the negated specification, which is appended to the prompt.

The MRBT is used as a reward function and action mask function in a neurosymbolic RL loop to learn π^* using Equations 3 and 4 (see example in Appendix B).

4. Evaluation

We conduct a comprehensive evaluation that includes an analysis of the pipeline outputs and an ablation study that compares agent task success using four ablations of MRBT across five task spaces in two environments. We conduct ten experiments across all task spaces under both deterministic and stochastic dynamics. The deterministic setting evaluates whether MRBT reactivity improves training when subtask failures arise from the agent's own actions. The stochastic setting evaluates whether MRBT reactivity benefits the agent under unpredictable changes (e.g., randomly dropping a key). We use ChatGPT 5 as the LLM for generation of MRBTs and verify generated MRBTs using Z3 SMT-solver.

4.1. Environments

The environments are *MiniGrid*, with discrete states and actions, and *MuJoCo Fetch* (de Lazcano et al. (2024)), with discrete actions and continuous states. *MuJoCo Fetch* models simple assembly tasks in which a gripper agent must pick up a block and move it to a target position. For *MuJoCo Fetch*, continuous position offsets were discretized and gripper actions reduced to open/close to allow discrete action masking. *Stochastic* dynamics are modeled with a 0.05 transition probability in both environments: in *MiniGrid*, a picked-up key may drop to an adjacent cell; in *MuJoCo Fetch*, the gripper may open unexpectedly, potentially dropping the block. Episodes may end upon task completion or at a timestep limit.

Task spaces for *MiniGrid* include *DoorKey*, *LockedRoom*, and *DroneSupplier*, while *MuJoCo Fetch* includes *PickAndPlace* and *PickAndPlace2*. In *DoorKey*, the agent picks up a key, opens a locked door, and reaches a goal; in *LockedRoom*, it retrieves a key from one of six rooms to open a locked door to a goal; in *DroneSupplier*, it collects a key from one of six boxes to unlock one of six doors. The grid in *DroneSupplier* is extracted from a neighborhood map in AirSim. We relate keys to supplies and the locked door as a person in need. In *PickAndPlace*, a gripper agent moves a block to a target, and in *PickAndPlace2*, to one of two targets. More details on the environments and tasks are in Appendix D.

4.2. Automated Pipeline Results

We use ChatGPT-5 for its strong reasoning and accessibility. Details on pipeline setup are in Appendix E.1. We report generated subtasks for each task space. Task space *DoorKey* has subtasks *Acquire Key, Open Door, Reach Goal*, task space *DroneSupplier* has subtasks *Open the box, Pick up the key, Open the door*, task space *PickAndPlace* has subtasks *Pick Block, Move To Target*, task space *PickAndPlace2* has subtasks *Grasp Block, Move Block to Target*. The action masks for each task space follow a similar pattern. The first action mask $\eta^{i,1}$ for each subtask i contains navigation actions to get near the subtask object. The second action mask $\eta^{i,2}$ enables specific manipulation actions and can disable navigation. *LockedRoom* subtasks and action masks are listed in Example 1.

Z3 can be used to refine generated logic formulas. For ψ^1 in Example 1, the LLM initially generated the logic formula $[door\_state_{room\_color} = open]$. Z3 found the Non-regressive Maximal Reward specification to be unsatisfiable due to ψ^1 not persisting. After re-prompting the LLM with this failure, it correctly revised ψ^1 to include $door\_state_{room\_color} = -1$. In *PickAndPlace2*, Z3 detected an object proximity counterexample where the block reached the target before the agent was evaluated in proximity using the object proximity logic formula; the LLM resolved this by increasing the distance thresholds. All correct specifications were verified in under ten minutes, with most taking less than one (see Appendix F).

4.3. Ablation Study

We train an agent policy $\pi(s, m)$ using neurosymbolic RL with four MRBT ablations, two of which are baselines, within each task space and evaluate performance based on the task *success rate* of the agent. The *success rate* is the proportion of tasks completed from step t to $t + I$, where $I = 2048$ timesteps.

The ablation methods of MRBT are as follows:

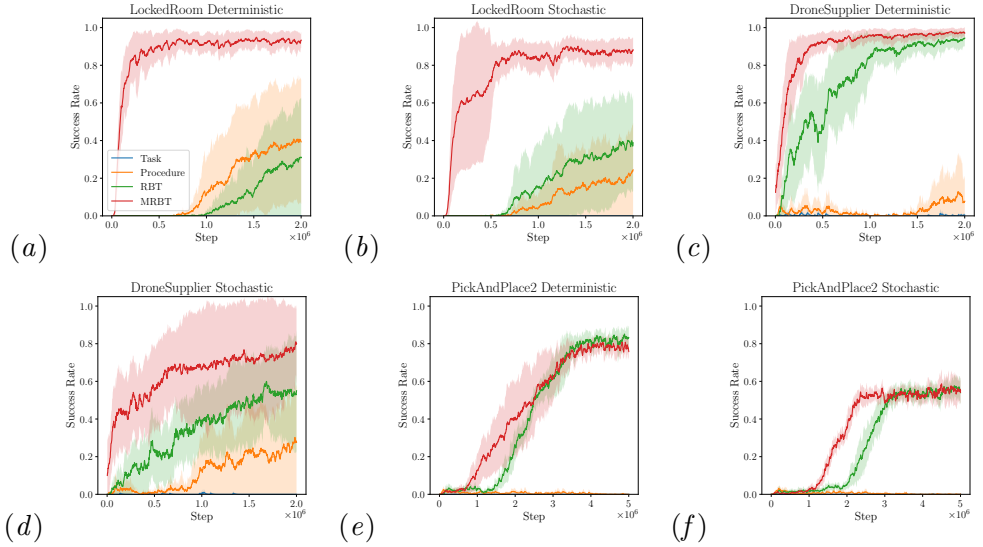

Figure 4: Success rate of agents during training for 4 random seeds with *deterministic* and *stochastic* dynamics; Solid lines and shaded region represent mean and standard deviation.

1) *Task* - (Baseline) This baseline provides a binary reward if a task is complete.

2) *Procedure* - (Baseline) This baseline is based on the reward function in VLM-CaR, that is a procedure of reward code for evaluating completion of each subtask. Our implementation is a modified MRBT where each subtask reward is issued only once (i.e. no reactive backtracking) and there is no action masking.

3) *RBT* - (Ours) The reward function uses an MRBT, but no action masking.

4) *MRBT* - (Ours) Reward function and an action mask function use MRBT.

Procedure, *RBT*, and *MRBT* use a shared MRBT structure and include the *Task* reward as additional feedback. Training algorithm and hyperparameter details are in Appendix E.2.

Ablation Study Results: *MRBT* results in consistently high average task success across all experiments, whereas other ablations exhibit lower or more variable performance (Figure 4). Using *Task* results in near-zero task success across experiments, demonstrating the need for task decomposition. Specifically in *LockedRoom*, the most complex task space with the largest number of subtasks ($k = 4$) and task variants (36), the action masks significantly improve average task success ($\geq 80\%$) over methods without action masking ($\leq 70\%$) such as *RBT*. In *PickAndPlace2*, using *Procedure* results in near-zero task success, whereas *MRBT* and *RBT* lead to $> 60\%$ task success highlighting the need for reactive rewards to guide agent learning. Using action masks in *MRBT* leads to improved learning efficiency and either comparable or higher average task success than using *RBT*. A limitation of the action masks is that their bias can restrict the agent from learning a more optimal solution, as seen in *PickAndPlace2* where there are instances the average task success rate of *RBT* is higher than *MRBT*. However, we find this degradation to be minimal, as the standard deviations of task success across four training runs overlap, and in all experiments *MRBT* achieves consistently high task success. Results for *DoorKey* and *PickAndPlace* are in Appendix G.

5. Advantages of MRBTs

Transferability: We evaluate the transferability of the *DroneSupplier* agent policy in *MiniGrid* to a realistic quadcopter simulator: AirSim. Since AirSim does not support manipulator actions, we focus on evaluating navigation and represent interaction actions as transitions in *MiniGrid*. Using AirSim's API, we implement forward motion via `moveToPosition` and directional changes via `moveByVelocity`. We compare the task success rates of the ablation methods, in Section 4.3, over 100 episodes. Results in Table 1 show that the task success rates under MRBT outperforms all ablation methods.

Method	Success Rate (Deterministic)	Success Rate (Stochastic)
Task	0.00	0.00
Procedure	0.05	0.57
RBT	0.94	0.45
MRBT	0.97	0.94

Table 1: Task success rate for agent policies transferred to AirSim.

Modularity: The MRBT integrates action masking and is more modular than a hierarchical reward machine (HRM) (Furelos-Blanco et al. (2023)) with comparable reward logic (Appendix H). For k=3 subtasks, the MRBT comprises 16 behaviors, 18 RM states, and 36 RM edges, while the HRM has 13 states and 24 edges. Although the HRM is structurally more compact, the MRBT is more modular: adding a subtask to the MRBT requires $\mathcal{O}(1)$ additional storage by attaching a new subtree to the root, whereas adding a subtask to the HRM requires $\mathcal{O}(k)$ storage for back edges to prior states. Prior work similarly shows that BTs scale more modularly than finite-state machines as task complexity increases (Iovino et al. (2023, 2025); Olsson (2016)).

Verifiability: We use Z3 to develop $\mathcal{E}^{sym}$ and verify logic specifications to provide theoretical guarantees for the MRBT. For *MiniGrid*, we develop $\mathcal{E}^{sym}$ to match the true dynamics in $\mathcal{E}$, allowing us to conclude that the verified specifications hold during training for trajectories of size H. For *MuJoCo Fetch*, $\mathcal{E}^{sym}$ abstracts $\mathcal{E}$. Although $\mathcal{E}$ is not fully modeled in $\mathcal{E}^{sym}$, we see from the results that the generated MRBTs improve training efficiency. In cases where $\mathcal{E}^{sym}$ is not available, expert demonstrations can be used to test logic formulas and can be useful priors for action masks (see Appendix I).

6. Conclusion

We introduce masking reward behavior trees (MRBTs), used as reward and action mask functions for compositional tasks. We design an MRBT template and logical specifications to construct and verify MRBTs for sequential object-interaction subtasks. We design a pipeline to automatically generate and verify MRBTs using an LLM and SMT-solver and use the MRBTs in a neurosymbolic RL training loop. We generate five MRBTs using the pipeline and demonstrate how the SMT-solver was used to refine the generated output. An ablation study shows that MRBTs improve learning efficiency and task success over ablations, with the largest gain in task success in the most complex task space. We further show that MRBTs offer transferability, modularity, and verifiability. Future work will explore generalizing the MRBT template. Generative AI was used to improve writing quality (ChatGPT-5) and to assist with code development (Claude Sonnet 4.5 (Anthropic (2025))).
This material is based upon work sponsored by the Air Force Research Lab (AFRL) and DARPA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL or DARPA.

References

Kamal Acharya, Waleed Raza, Carlos Dourado, Alvaro Velasquez, and Houbing Herbert Song. Neurosymbolic reinforcement learning and planning: A survey. *IEEE Transactions on Artificial Intelligence*, 5(5):1939–1953, 2024. doi: 10.1109/TAI.2023.3311428.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millicah, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Shayan Meshkat Alsadat and Zhe Xu. Large language model-based task learning for swarm systems in reinforcement learning with reward machines. In *2025 5th International Conference on Computer, Control and Robotics (ICCCR)*, pages 469–477, Red Hook, NY, USA, 2025. Curran Associates Inc. doi: 10.1109/ICCCR65461.2025.11072658.

Shayan Meshkat Alsadat, Jean-Raphaël Gaglione, Daniel Neider, Ufuk Topcu, and Zhe Xu. Using large language models to automate and expedite reinforcement learning with reward machine. In *2025 American Control Conference (ACC)*, pages 206–211, Red Hook, NY, USA, 2025. Curran Associates Inc. doi: 10.23919/ACC63710.2025.11107701.

Anthropic. Claude sonnet 4.5 system card. Technical report, Anthropic, September 2025. URL https://www.anthropic.com/system-cards/.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6065–6073, Red Hook, NY, USA, 7 2019. Curran Associates Inc. doi: 10.24963/ijcai.2019/840. URL https://doi.org/10.24963/ijcai.2019/840.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: modular & customizable reinforcement learning environments for goal-oriented tasks. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2024. URL http://github.com/Farama-Foundation/Gymnasium-Robotics.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 120–136, Brookline, MA, USA, 22–25 Aug 2023. PMLR. URL https://proceedings.mlr.press/v232/du23b.html.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: building open-ended embodied agents with internet-scale knowledge. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. Hierarchies of reward machines. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, Norfolk, MA, USA, 2023. JMLR.org.

gaiin-platform. gaiin-platform. https://github.com/gaiin-platform, 2025. GitHub organization.

Qi Guo, Xing Liu, Jianjiang Hui, Zhengxiong Liu, and Panfeng Huang. Utilizing large language models for robot skill reward shaping in reinforcement learning. In Xuguang Lan, Xuesong Mei, Caigui Jiang, Fei Zhao, and Zhiqiang Tian, editors, *Intelligent Robotics and Applications*, pages 3–17, Singapore, 2025. Springer Nature Singapore. ISBN 978-981-96-0783-9.

Ashutosh Gupta, John Komp, Abhay Singh Rajput, Krishna Shankaranarayanan, Ashutosh Trivedi, and Namrita Varshney. Integrating explanations in learning ltl specifications from demonstrations, 2024. URL https://arxiv.org/abs/2404.02872.

Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. In Roman Barták, Fazel Keshtkar, and Michael Franklin, editors, *Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Hutchinson Island, Jensen Beach, Florida, USA, May 15-18, 2022*, FL, USA, 2022. Florida Online Journals. doi: 10.32473/flairs.v35i.130584. URL https://doi.org/10.32473/flairs.v35i.130584.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2107–2116, Brookline, MA, USA, 10–15 Jul 2018. PMLR. URL https://proceedings.mlr.press/v80/icarte18a.html.

Matteo Iovino, Julian Förster, Pietro Falco, Jen Jen Chung, Roland Siegwart, and Christian Smith. On the programming effort required to generate behavior trees and finite state machines for robotic applications. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5807–5813, Red Hook, NY, USA, 2023. Curran Associates Inc. doi: 10.1109/ICRA48891.2023.10160972.

Matteo Iovino, Julian Förster, Pietro Falco, Jen Jen Chung, Roland Siegwart, and Christian Smith. Comparison between behavior trees and finite state machines. *IEEE Transactions on Automation Science and Engineering*, 22:21098–21117, 2025. doi: 10.1109/TASE.2025.3610090.

Anssi Kanervisto, Stephanie Milani, Karolis Ramanauskas, Nicholay Topin, Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, Wei Yang, Weijun Hong, Zhongyue Huang, Haicheng Chen, Guangjun Zeng, Yue Lin, Vincent Micheli, Eloi Alonso, François Fleuret, Alexander Nikulin, Yury Belousov, Oleg Svidchenko, and Aleksei Shpilman. Minerl diamond 2021 competition: Overview, results, and lessons learned, 2022. URL https://arxiv.org/abs/2202.10583.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv: Arxiv-2310.12931*, 1(1), 2023.

M. Olsson. Behavior trees for decision-making in autonomous driving. Technical report, School of Computer Science and Communication, KTH Royal Institute of Technology, Stockholm, Sweden, 2016.

OpenAI. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025. URL https://arxiv.org/abs/2601.03267.

Nicholas Potteiger, Ankita Samaddar, Hunter Bergstrom, and Xenofon Koutsoukos. Designing robust cyber-defense agents with evolving behavior trees. In *2024 International Conference on Assured Autonomy (ICAA)*, pages 1–10, 2024. doi: 10.1109/ICAA64256.2024.00011.

Yun Qu, Yuhang Jiang, Boyuan Wang, Yixiu Mao, Cheems Wang, Chang Liu, and Xiangyang Ji. Latent reward: Llm-empowered credit assignment in episodic reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(19):20095–20103, Apr. 2025. doi: 10.1609/aaai.v39i19.34213. URL https://ojs.aaai.org/index.php/AAAI/article/view/34213.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Marco Hutter and Roland Siegwart, editors, *Field and Service Robotics*, pages 621–635, Cham, CH, 2018. Springer International Publishing. ISBN 978-3-319-67361-5.

Sumedh A Sontakke, Jesse Zhang, Sébastien M. R. Arnold, Karl Pertsch, Erdem Bıyık, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. Roboclip: one demonstration is enough to learn robot policies. In *Proceedings of the 37th International Conference on Neural*

Information Processing Systems, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Mathieu Tuli, Andrew C. Li, Pashootan Vaezipoor, Toryn Q. Klassen, Scott Sanner, and Sheila A. McIlraith. Learning to follow instructions in text-based games. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

David Venuto, Sami Nur Islam, Martin Klissarov, Doina Precup, Sherry Yang, and Ankit Anand. Code as reward: empowering reinforcement learning with vlms. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, Norfolk, MA, USA, 2024. JMLR.org.

Yanxiao Zhao, Yangge Qian, Jingyang Shan, and Xiaolin Qin. Camel: Continuous action masking enabled by large language models for reinforcement learning, 2025. URL https://arxiv.org/abs/2502.11896.

Appendix A. Related Work

Recent methods leverage LLMs and VLMs to automate reward and action mask design. VLMs such as OpenAI CLIP (Radford et al. (2021)) is used to compare embeddings of goal descriptions and current states, with high cosine similarity indicating goal completion. MineDojo (Fan et al. (2022)) fine-tunes CLIP for task success in *Minecraft* (Kanervisto et al. (2022)), while trajectory embeddings can also be used for evaluation (Sontakke et al. (2023)). Flamingo (Alayrac et al. (2022)) has been fine-tuned for visual question answering to detect mission completion (Du et al. (2023)), though fine-tuning and repeated inference during RL training incurs computational cost. Other approaches, including ours, generate code or symbolic rewards, requiring a pre-trained LLM or VLM during design or between training runs. Eureka (Ma et al. (2023)) uses an LLM to code and edit reward functions based on agent performance, while VLM-CaR (Venuto et al. (2024)) generates procedural subtask reward functions from initial and final image frames of a complex mission and validates them via a testing process. VLM-CaR effectively generates and tests reward code for complex missions but lacks support for varying missions in a mission space and reactivity to handle stochastic dynamics. LLMs can also produce multi-faceted rewards for richer feedback (Qu et al. (2025)). Further, policies learned from LLM generated rewards were shown to support sim-to-real transfer for robotic tasks (Guo et al. (2025)). Symbolic rewards such as reward machines and logic specifications have likewise been augmented by LLMs: reward machines can be generated or optimized via few-shot LLM prompts encoding domain knowledge (Alsadat et al. (2025); Alsadat and Xu (2025)), and LLM generated explanations can produce linear temporal logic formulas from expert demonstrations to construct reward machines or direct reward functions (Gupta et al. (2024); Camacho et al. (2019); Tuli et al. (2022)). Less work has focused on LLMs for action masking with one work using LLMs to generate suboptimal policies as guidance to dynamically restrict a continuous action space (Zhao et al. (2025)). Our proposed approach further expands the literature on methods for using LLMs to design action masks.

The prior work uses LLMs for single tasks to generate rewards and action masks, whereas our work focuses on designing joint reward and action mask functions that are reactive to subtask failure and generalize to variants of a task. Additionally, we verify specifications derived from the subtask logic formulas using an SMT solver, providing formal guarantees that are not attainable through testing with expert demonstrations.

Appendix B. Automated Pipeline More Details

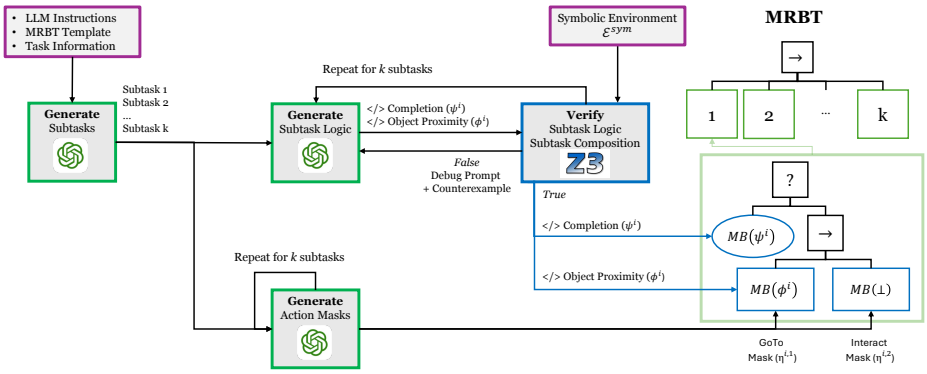

Figure 5: Automated pipeline to *generate* and *verify* MRBTs with an LLM and SMT solver.

An example of the neurosymbolic RL loop for a single mission of *MiniGrid LockedRoom* is in Figure 6. In this example, the color of the door ("red") in subtask 1, *Open key room door*, is determined by the episode mission, which specifies *"...the red room..."*.

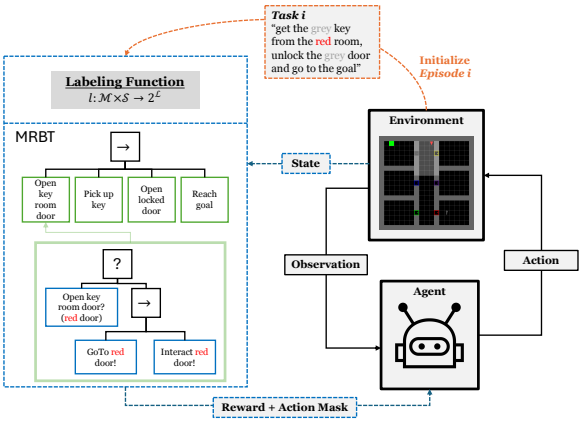

Figure 6: Integration of MRBTs into a neurosymbolic RL loop.

Appendix C. Logic Specifications for Verification

We compose specifications over the trajectory space $\tau \in \Gamma$ and task space $m \in \mathcal{M}$ that include logic formulas g^m, ψ^i, and ϕ^i. We denote, g^m_t, ψ^i_t, and ϕ^i_t as timestep evaluations, and $\exists^{=N}$ as a counting quantifier checking existence of N distinct solutions.

Specification	Logic Formula
Completion Correctness	$\forall(\tau \in \Gamma, m \in \mathcal{M})\ g^m_{H-1} \implies \bigvee_{t=0}^{H-1} \psi^i_t$
Completion Non-Triviality	$\exists^{=N}(\tau \in \Gamma, m \in \mathcal{M}) \bigwedge_{t=0}^{H-1} \sim \psi^i_t$
Object Proximity Correctness	$\forall(\tau \in \Gamma, m \in \mathcal{M}) \bigwedge_{t=0}^{H-2} (\sim \psi^i_t \wedge \psi^i_{t+1}) \implies \phi^i_t$
Object Proximity Non-Triviality	$\exists^{=N}(\tau \in \Gamma, m \in \mathcal{M}) \bigwedge_{t=0}^{H-1} \sim \phi^i_t$
Composition Persistence	$\exists^{=N}(\tau \in \Gamma, m \in \mathcal{M})\ g^m_{H-1} \implies \bigwedge_{t=0}^{H-2} \bigwedge_{i=1}^{k} (\psi^i_t \implies \psi^i_{t+1})$

Table 2: Logic specifications used to verify subtasks.

Appendix D. Environment and Task Descriptions

D.1. MiniGrid

MiniGrid (Chevalier-Boisvert et al. (2023)) is a discrete state, discrete action environment where an agent (red triangle) can pick up and interact with objects. The discrete state space $\mathcal{S}$ is a $N \times N$ grid where each grid cell is a tuple $(OBJECTID, COLOR, STATE)$. $STATE$ stores data for the direction of the agent or if an object is open, closed, or locked. The observation space $\mathcal{O}$ is a $J \times J, J \leq N$ field of view of the state. Predicates of $\mathcal{S}$ contain the positions of objects (e.g. keys, doors, agent, goal, ...) and agent direction. The action space is {*left, right, forward, pickup, drop, toggle, done*}. Stochastic dynamics are modeled as a 0.05 probability in the transitions probability function p that the key will drop to an adjacent cell after picked up. The color values are *"red", "green", "blue", "purple", "yellow", "grey"*.

DoorKey. The agent must pick up a key, open a locked door, and then travel to the goal on the other side of a wall. The grid we consider is size 16×16 that has one wall and door that separates two rooms, where the left room has a key, and the right room has a goal. The task space is $\mathcal{M}_{dk} = \langle \ell_{dk}, \mathcal{V}_{dk}, \mathcal{G}_{dk} \rangle$. $\ell_{dk} =$ textit"use the key to open the door and then get to the goal", $\mathcal{V}_{dk} = \emptyset$, and $\mathcal{G}_{lr}$ contains one function that checks if the agent is at the same location as the goal. The maximum number of steps is 500.

LockedRoom. The agent must pick up a key in one of six room, then open a locked door for another room to reach a goal. The grid is 19×19 with walls and doors to separate rooms and one key, one locked door, and one goal. The task space is $\mathcal{M}_{lr} = \langle \ell_{lr}, \mathcal{V}_{lr}, \mathcal{G}_{lr} \rangle$. $\ell_{lr} =$ *"get the {lockedroom colour} key from the {keyroom colour} room, unlock the {door colour} door and go to the goal"*, $\mathcal{V}_{lr} = \{lockedroom\ colour, keyroom\ colour, door\ colour\}$, and $\mathcal{G}_{lr}$ contains one logic formula that checks if the agent is at the same location as the goal. The maximum number of steps is 190.

DroneSupplier. The agent must pick up a key from one of six boxes and use the key to unlock and open one of six doors. The grid is a 25×25 slice of a neighborhood in

Microsoft AirSim (Shah et al. (2018)) where each object above an altitude of 5 meters in the neighborhood is represented as a wall. The key is equivalent to supplies and the locked door is equivalent to a car or person in need. The task space is $\mathcal{M}_{ds} = \langle \ell_{ds} \mathcal{V}_{ds}, \mathcal{G}_{ds} \rangle$. ℓ_{ds} = *"open the { box color } box, pick up the key, then open the { door color } door"*, $\mathcal{V}_{ds} = \{$*box color*, *door color*$\}$, and $\mathcal{G}_{ds}$ contains logic formulas that check if the *door color* door is open. The maximum number of steps is 500.

D.2. MuJoCo Fetch

MuJoCo Fetch is a *Gymnasium-Robotics* (de Lazcano et al. (2024)) environment we modified as a continuous state, discrete action environment where the agent (gripper) must pick up a block and move it to targets. The continuous state space, $\mathcal{S}$, contains a $21-$*dimensional* vector for gripper and block dynamics, a $3h-$*dimensional* desired target vector for h possible target positions. Predicates of $\mathcal{S}$ contain the gripper and block positions, linear velocity, target positions, and gripper finger displacement. The observation space is the state space, $\mathcal{O} = \mathcal{S}$. The discrete action space is {*left, right, forward, backward, up, down, open gripper, close gripper*}. Stochastic dynamics are modeled as a 0.05 probability in the transition probability function p that the gripper will open when closed, potentially leading to the block dropping. The objects are *"block", "target", "agent"*. The *color* values for targets are *"green", "yellow"*. For both task spaces, the maximum number of steps is 100.

PickAndPlace. The agent must pick up a block from a table and move it to a target position above or on the table. The task space is $\mathcal{M}_{pp} = \langle \ell_{pp}, \mathcal{V}_{pp}, \mathcal{G}_{pp} \rangle$. ℓ_{pp} = *"pick up and move the block to the target location"*, $\mathcal{V}_{pp} = \emptyset$, and $\mathcal{G}_{pp}$ contains one logic formula that checks if the block is within 5cm of the target.

PickAndPlace2. Same as *PickAndPlace*, but with two target positions to select from. The task space is $\mathcal{M}_{pp2} = \langle \ell_{pp2}, \mathcal{V}_{pp2}, \mathcal{G}_{pp2} \rangle$. ℓ_{pp2} = *"pick up and move the block to the { target color } target location"*, $\mathcal{V}_{pp2} = \{$*target color*$\}$, and $\mathcal{G}_{pp}$ contains two logic formulas that check if the block is within 5cm of the *green* or *yellow* target.

Appendix E. Experimental Setup

E.1. Automated Pipeline Details

The pipeline interfaces with ChatGPT-5 using the Vanderbilt Amplify Platform (gaiin-platform (2025)), an open-source web platform for hosting LLM services. However, any API could be adapted into the pipeline with a similar expected output. System prompts include manually curated LLM instructions and the MRBT template. For *MiniGrid*, the task space $\mathcal{M}$ and action space $\mathcal{A}$ were web-scraped from the official documentation, while symbolic variables used for predicates from state space S were Python Z3 variable definitions. In our custom *MiniGrid* task space, *DroneSupplier*, $\mathcal{M}$ was manually specified, with the task renamed to *Unlock* to reflect its intent. For *MuJoCo Fetch*, $\mathcal{M}$, predicates of $\mathcal{S}$, and $\mathcal{A}$ were manually specified similar to *MiniGrid*.

The prompts used for generating subtask logic formulas and action masks were similar for both environments. For generating subtask logic formulas, the LLM was instructed to generate a Python function compatible with inputting a vector of Z3 state variables and mission variables and outputting a Z3 expression. These functions could then be

integrated into Z3 constraints for verification with symbolic environment $\mathcal{E}^{sym}$. Other details were included in prompting the model to exclude Manhattan or Euclidean distances from generated formulas, which can significantly increase verification time in continuous-state environments like $MuJoCoFetch$. To generate action masks, the LLM was instructed to fill in vectors of size $|\mathcal{A}|$ with the available actions for the $2k$ action masks.

The time horizon $H = 25$ timesteps for verification in both environments. For finding N distinct trajectories, we add constraints to prevent the initial state from being equal to previous initial states. For computational efficiency, we allow for setting a wallclock timeout $T_{timeout}$. The verifier attempts to find a counterexample, if none are found within $T_{timeout}$ the result is inconclusive. In such cases, the pipeline assumes correctness, interpreting a timeout without counterexamples as evidence that violations are difficult to find; thus, the logic formulas are expected to provide useful feedback for guiding training despite potential edge-case failures. $T_{timeout} = 900$ seconds (15 minutes) for our experiments. While no verification step reached this timeout (see Table 3), it serves as a practical safeguard when verifying logic formulas in more complex environments with potentially intractable verification times.

E.2. Training Setup

The RL algorithm for training is proximal policy optimization (PPO) with a learning rate of $3e-4$, $\gamma = 0.99$, and the number of environment steps per optimization step is 2048. For *MiniGrid*, the agent policy consists of a feature encoder and a linear control head. The feature encoder comprises three convolutional layers with 16, 32, and 64 filters of size 2×2, each followed by a ReLU activation. The output is flattened and passed through two additional fully connected layers of size 64, that output action logits. For *MuJoCo Fetch*, the agent policy consists of the linear control head with two fully connected layers of size 128.

Appendix F. Verification Runtimes

We report the wallclock time to verify each specification in Section 3.4 using Z3 for each correct subtask logic formula generated by ChatGPT-5. The results are in Table 3. The task spaces with the most time to verify are from *LockedRoom* and *DroneSupplier* because they have more task variants. All verification times are below the timeout $T_{timeout} = 900$.

Table 3: Verification time for logical specifications across five task spaces. Task spaces are *DoorKey* (DK), *LockedRoom* (LR), *DroneSupplier* (DS), *PickAndPlace* (PP), and *PickAndPlace2* (PP2).

Specification	Env	Subtask Verification Time (s)			
		1	2	3	4
Completion Correctness	DK	3.18	1.72	0.93	-
	LR	199.21	51.03	14.53	1.04
	DS	69.54	475.87	17.71	-
	PP	1.33	5.22	-	-
	PP2	0.36	60.13	-	-
Completion Non-Triviality	DK	17.74	25.21	24.11	-
	LR	61.74	61.17	71.42	43.21
	DS	56.18	68.98	75.82	-
	PP	4.41	2.61	-	-
	PP2	2.12	2.29	-	-
Object Proximity Correctness	DK	3.54	8.96	3.08	-
	LR	243.77	164.40	132.32	49.41
	DS	69.32	235.19	78.66	-
	PP	2.66	5.91	-	-
	PP2	5.17	8.09	-	-
Object Proximity Non-Triviality	DK	22.43	20.95	21.77	-
	LR	42.02	38.67	38.49	48.52
	DS	55.59	78.17	60.54	-
	PP	3.39	2.37	-	-
	PP2	2.40	1.94	-	-
Non-regressive Maximal Reward	DK	32.03			
	LR	437.73			
	DS	359.79			
	PP	11.06			
	PP2	8.23			

Appendix G. More Experiments Results

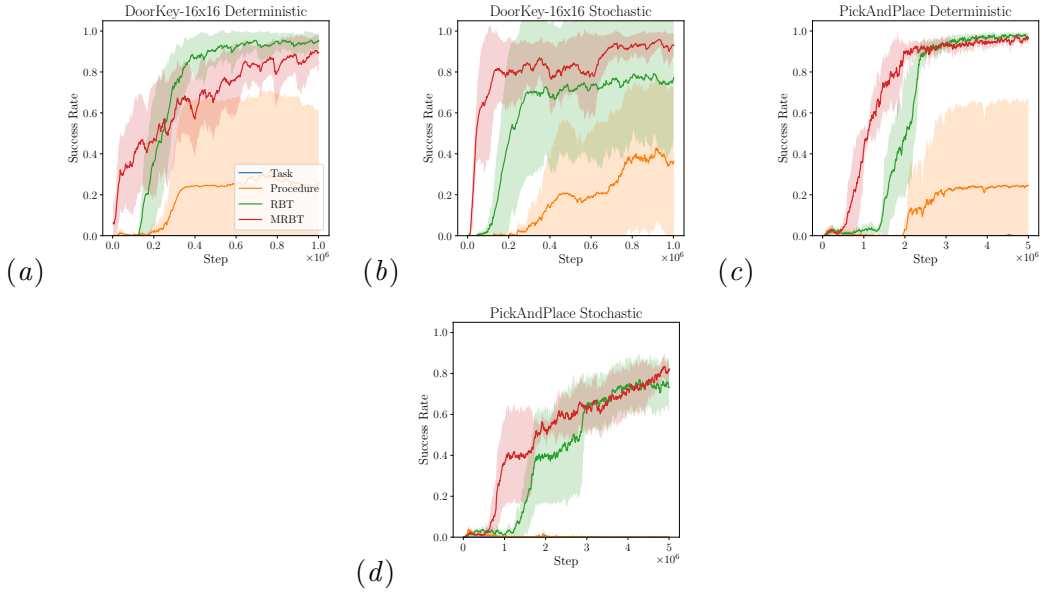

Figure 7: Task success rate of the agent during training for *MiniGrid DoorKey* and *MuJoCo Fetch PickAndPlace*.

Appendix H. Hierarchical Reward Machine

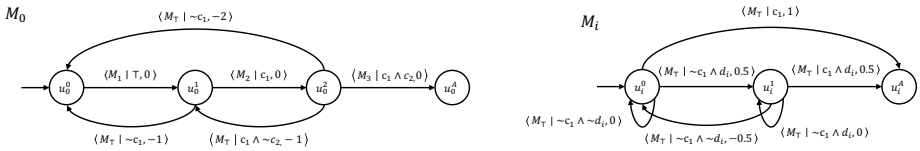

Figure 8: Hierarchical reward machine with three subtasks; the root RM is M_0, u^A is an accepting state, M_i are subtask RMs, and $M_\top$ is an RM that is u^A. Edges are labeled as $\langle M_i \mid e, r \rangle$ where M_i is a RM called when exit condition e is true, executing until it transitions to u^A, then outputting scalar reward r.

Appendix I. Testing MRBTs using Demonstrations

The specifications from Section 3.4 can be tested using expert and random demonstrations. The demonstrations are trajectories of states of arbitrary length constrained by $\mathcal{E}^{sym}$ and a task. The demonstrations are collected by an expert policy designed for $\mathcal{E}$ with deterministic dynamics. The Completion Correctness, Object Proximity Correctness, and Non-regressive Maximal Reward specifications are checked to see if they are satisfied for all trajectories or $N = 10$ demonstrations. For Completion Non-Triviality and Object Proximity Non-Triviality demonstrations using a random action policy are collected. We assume that the

random action policy performs poorly on the task. The specifications are checked to see if they are satisfied for $N = 10$ demonstrations. Assuming the expert demonstrations complete the subtasks sequentially, an initial set of action masks can be obtained by recording the actions taken while each MBRM is running. These action masks are then provided to the LLM for further refinement, enabling it to leverage the expert-derived action masks as a prior rather than generating them from scratch.

Using expert and random demonstrations enables testing on in-distribution data without relying on $\mathcal{E}^{\text{sym}}$. However, such demonstrations are biased by the strategies of the expert and random policies, may not cover all possible behaviors, and do not provide theoretical guarantees. For instance, in *LockedRoom*, an expert might always drop the key after opening a door before proceeding. As a result, all expert demonstrations would violate the Non-regressive Maximal Reward specification, resulting in the LLM being re-prompted. In reality, there exist demonstrations where the key is held until the agent reaches the goal, but the expert policy did not explore this strategy. Therefore, if one were to replace the SMT solver with demonstrations, multiple expert policies should be considered to improve coverage of potential behaviors.