PBRMS: Priority-Based Reward-Machine Switching for Context-Aware Reinforcement Learning in Non-Stationary POMDPs

Andy Edmondson[1, 2], **Ronald P. A. Petrick**[1],

[1]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK
[2]School of Informatics, The University of Edinburgh, Edinburgh, UK
Andy.Edmondson@ed.ac.uk, R.Petrick@hw.ac.uk

Abstract

Reinforcement learning (RL) methods often struggle where tasks have scarce rewards which depend on temporally extended sequences of sub-goals. This is especially true where environments constitute a Partially Observable Markov Decision Process (POMDP) where observations do not uniquely identify the underlying state. Reward Machines (RMs), Hierarchical RMs (HRMs) and automata generated from Linear Temporal Logic (LTL) can render these rewards Markovian, but current techniques cannot adequately represent environments that require tracking a large and initially unknown number of concurrently active task instances, each with its own progress state. This work presents Priority-Based Reward Machine Switching (PBRMS), a modular approach in which the agent contains a Contextual Controller and concise RM definitions constrained to describe task sequence structure. Using this controller, the agent selects, suspends, and resumes tasks based on internal state built from observed environmental context, enabling contingent task switching in uncertain environments containing a large and unknown number of parallel task sequences. Guiding task completion, the agent includes a per-step contextual value distribution over geodesic distance in the RL decision-maker's state space. Results show that PBRMS supports efficient learning in a procedurally generated, non-stationary POMDP with variable parallel tasks, offering a flexible alternative to current methods for integrating learning and structured knowledge.

Introduction

Search and Rescue (SAR) tasks, such as finding survivors of natural disasters or locating people in a burning building and leading them to safety, inherently contain significant uncertainties. In many cases, it is not clear how many people are impacted, where they are located, and how serious any injuries may be. This is true of SAR operations in buildings during a fire and in larger urban or mountain rescue scenarios. In these cases, agents must be able to cope with *temporally-extended* tasks and keep track of a large and initially unknown number of parallel tasks. For instance, when searching a mountain, an agent may spot person A 1 km away and move towards them. On the path to A they may spot person B nearby, switch attention to triage B and, finding them mobile, direct them to safety before communicating their status to coordinating personnel and moving on. At person C, who is immobilised, the agent may administer aid and leave supplies, communicating with coordinators before pausing actions with C to reach person A, later returning to C to check status and perhaps administer further aid.

While *Reinforcement Learning (RL)* can effectively learn complex sequences of actions in *Markov Decision Process (MDP)* settings, it struggles to learn efficiently when goals require the completion of temporally-extended sequences whose rewards depend on history, breaking Markov assumptions (Sutton and Barto 2018). *Reward shaping* (Ng, Harada, and Russell 1999) is a common method of providing task-specific guidance to learning agents, but writing a single reward function which adequately describes these non-Markovian tasks is problematic. *Reward Machines (RMs)* formalise reward function transitions during operation, dynamically reflecting subtask context (Toro Icarte et al. 2022). Similarly, *Linear Temporal Logic (LTL)* statements are used to describe task sequences and compose subtask-specific RL agents to achieve larger goals guided by generated automata (Bergeron, Serlin, and Leahy 2024).

In *Partially Observable MDPs (POMDPs)* where task structure is revealed only through local observations, newly discovered subtasks can change the optimal ordering of actions in ways that cannot be predicted in advance. In such cases, where subtask relevance depends on discoveries made through exploration, encoding all possible contingent orderings, interruptions, and resumptions of subtasks leads to an exponential blow-up in the size of the automaton. *Hierarchical Reward Machine (HRM)* approaches mitigate exponential blow-up through a hierarchical set of RMs which call each other as required (Furelos-Blanco et al. 2023). Where partial observability creates uncertainty about propositions, *Temporal Dependency Modelling* approaches can provide belief consistency across RM states (Li et al. 2024).

However, many real-world partially-observable domains, such as the example SAR operation described above, must track an initially unknown number of task sequences operating in parallel, whereby the next sequence starts at some unpredictable point in the progression of sequences already in progress. Current methods are unable to manage such cases efficiently since in operation they rely on a combination of fixed propositional vocabulary and associated set of RMs. However, a POMDP containing a large and unknown num-

ber of people, only discoverable through exploration, where for efficiency (and safety) it is not appropriate to complete all tasks for each person sequentially while ignoring others, cannot be adequately expressed by pure RM approaches.

This work introduces *Priority-Based Reward Machine Switching (PBRMS)*, shown in Figure 1, a mechanism designed to overcome limitations of existing RM and LTL-based methods in POMDPs where a large and unknown number of parallel subtasks must be tracked while avoiding exponential growth of the reward machine representation. The key contributions of this work are:

1. PBRMS, which avoids exponential growth by maintaining a set of concise RMs that describe the sequential structure and priority order of each subtask. The Contextual Controller then switches between RM states depending on priority and environmental context, using an internal belief representation to track parallel subtasks.
2. The addition of a context-specific reward distribution over geodesic distance. Calculated each timestep, the distribution is used in reward calculations during training, and forms part of the RL observation, guiding action decisions by providing task context. This helps the agent generalise to previously unseen configurations.

These contributions allow the agent to track progress of a large number of parallel subtasks, meaning PBRMS can express emergent, history-dependent and varying task sequence structures that can't be encoded using other RM and LTL methods. The rest of the paper discusses related work, the PBRMS approach and its evaluation, and experimental results, conclusions, and future work.

Background and Related Work

Reinforcement Learning algorithms can be defined as those which learn a mapping from situations to actions through experience. This mapping can be achieved in various ways, but they all seek to learn actions that will lead to achieving a defined goal through *reward maximisation* (Sutton and Barto 2018). This is often defined as a *Markov Decision Process (MDP)* with the tuple $\mathcal{M} = (S, A, P, R, \gamma)$, where S is the set of states, A the set of available actions, P the transition function $P(s'|s, a)$, R the reward function $R(s, a, s')$ and γ a discount factor (Russell et al. 2022). This indicates that while in state $s \in S$, undertaking action $a \in A$ has the probability $p \in [0, 1]$ of transitioning to state $s' \in S$ and receiving reward $r \in R$ with discount factor $\gamma \in [0, 1]$.

Many approaches to RL have been tried, and one of the best understood is the *Deep Q-Network (DQN)* (Mnih et al. 2015). Used throughout this work, DQNs use a *replay buffer* to store experience tuples $e = (s, a, r, s')$ each timestep and *Deep Neural Networks (DNNs)* to approximate action-values. Updates are completed according to the loss:

$$L_i(\theta_i) = \mathbb{E}_{e\sim U(D)}\left[\left(r+\gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right],$$

where s is the current state, s' the next state, r the received reward, a the action taken, and a' an action available in s'.

When learning temporally-extended tasks, requiring several steps to achieve each sub-goal, the well known *credit assignment problem* slows learning since it becomes difficult to determine which actions contribute to delayed rewards (Sutton and Barto 2018). Reward shaping (Ng, Harada, and Russell 1999) addresses this by providing *potential-based* intermediate rewards of the form $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$ that guide the agent toward task completion.

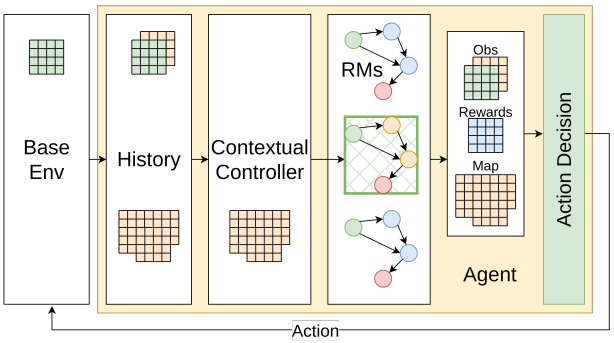

Figure 1: The PBRMS agent architecture. See Figure 3 for an example environment configuration.

Reward Machines (RMs) (Toro Icarte et al. 2022), are defined by the tuple $\mathcal{R} = (U, u_1, F, P, \delta_u, \delta_r)$, where U is a finite set of states, $u_1 \in U$ the initial state, F a finite set of terminal states, P a finite set of propositions, δ_u the state-transition function, and δ_r the state-reward function (Li et al. 2024). These structures extend reward functions using discrete states to decompose problems into structured subtasks able to provide context-specific rewards. This decomposition enables more efficient learning, particularly when combined with algorithms that learn separate policies for each subtask (Toro Icarte et al. 2022). Similarly, approaches based on LTL are used to specify task sequences and generate automata that guide the composition of subtask-specific RL agents (Bergeron, Serlin, and Leahy 2024).

POMDPs extend MDPs to settings in which the agent cannot directly observe the full state. Formally, a POMDP is a tuple $\mathcal{P} = (S, A, P, R, \mathcal{O}, O, \gamma)$, where S, A, P, R, and γ retain MDP definitions, O is the set of observations o, and $\mathcal{O}(o \mid s', a)$ is the probability of observing o after taking action a and transitioning to state s'. In this case, s_t is not observed directly; instead the agent receives the partial observation o_t, which in general does not uniquely identify the underlying state. Decision-making therefore relies on a belief state $b_t \in \Delta(S)$, a probability distribution over latent states inferred from the action-observation history obtained through P and $\mathcal{O}$ (Cassandra, Kaelbling, and Littman 1994).

In partially observable environments, RM and LTL automata can provide not only subtask decomposition but also form a memory of reward transition history which cannot be inferred from current observations (Toro Icarte et al. 2022). LTL-based composition approaches similarly use automaton states to guide task execution (Bergeron, Serlin, and Leahy 2024). This memory renders otherwise non-Markovian problems Markovian over the joint observation-automaton state space, enabling RL to learn effectively.

Complex tasks may consist of several subtasks where or-

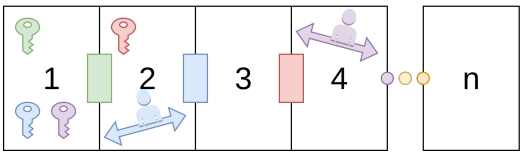

Figure 2: A simple environment with an initially unknown number of rooms and non-stationary people.

dering depends on discovered environmental context and may therefore change dynamically. A "flat" RM must explicitly encode every possible contingent transition, resumption point, and nested subtask sequence, producing an exponential number of states (Furelos-Blanco et al. 2023). HRMs, formally defined as $H = (\mathcal{M}, M_r, P)$, where $\mathcal{M}$ is a finite set of RMs, M_r the root RM, P a finite set of propositions across all $M \in \mathcal{M}$, avoid this blow-up by factoring behaviour into reusable sub-RMs and maintaining a call stack that preserves return transitions. This enables more concise representations, more efficient learning, and reuse of learned subtask policies (Furelos-Blanco et al. 2023).

A key function of RM and LTL automata is to decompose tasks into subtasks so as to define temporal structure and aid credit assignment. To achieve this, structures must provide informative rewards either from the MDP itself or through reward-shaping. A sophisticated approach to reward-shaping is demonstrated by Ewers, Anderson, and Thomson (2025), whereby a pre-calculated probability density function with peaks at key locations is used both to calculate rewards (based on probability mass accumulated) and to augment the agent's observation space, providing spatial beliefs about objective locations.

Having the rewards available is not enough, however, since complex environments often contain action sequences which are only occasionally experienced, meaning the agent fails to learn optimal policies for these situations (Fuchs et al. 2021). *Curriculum Learning* (Bengio et al. 2009) can help mitigate this by ensuring important action sequences are experienced sufficiently often for learning to occur. Even with a curriculum, DNN-based agents that are required to learn several sequential tasks can suffer from *Catastrophic Interference* (McCloskey and Cohen 1989) as updates for new tasks cause weights to diverge from those learned for previous tasks. An elegant solution is to decompose the system into several DNNs, each responsible for a single aspect of the system or single task, the review by Khetarpal et al. (2022) identifying several examples. Bergeron, Serlin, and Leahy (2024) use this approach for zero-shot task composition of agents which fulfil LTL specifications.

Multi-Skill Learning in POMDPs

To examine the broader concepts required to learn a multi-task POMDP across different domains we consider the simplified environment shown in Figure 2, where a number of people are stuck in locked rooms. Here, an agent must navigate between an unknown number of rooms numbered 1 to n, collect the colour-coded keys required to unlock matching doors and lead any people encountered to the exit and safety. The agent receives a limited line-of-sight observation window, so propositions related to keys, doors, or people become true from the agent's perspective only when observed in the local field of view. Over time, people move unpredictably in space, causing some observed proposition, such as `person_Adam_found`, to become true and then false again as they enter and leave the known area, independently of agent actions. Keys may also be encountered at any point, not only in the room containing the matching locked door, so efficient operation requires opportunistic collection.

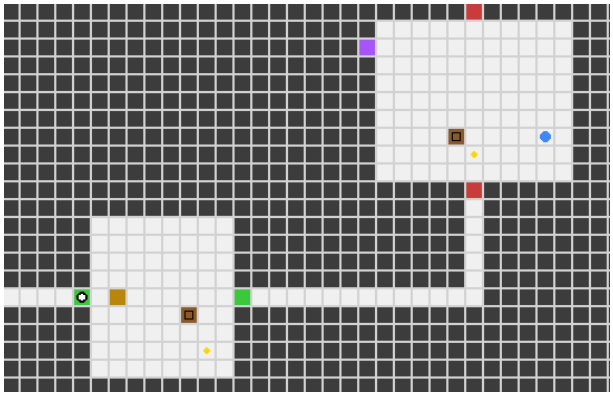

Figure 3: Example environment configuration section. The agent is a white circle, cabinet key a yellow dot, unlocked cupboard containing a keycard a yellow square with black item, person a blue dot, locked cupboard containing a keycard a brown square, locked door a red square, unlocked door a green square, exit a purple square.

This combination of partial observability and non-stationarity produces subtask relationships that are: **inherently contingent**, since subtasks become available only when their preconditions are observed, and non-stationarities mean these may become true or false independently of the agent; **interruptible**, such as when a person is observed as the agent approaches a door; **emergent**, since the available subtask structure at a given point depends on changing environmental context. Additionally, environments are **unbounded**, since there may be any number of rooms and keys, or people to be rescued, which must be tracked independently and in parallel.

However, both flat RMs and HRMs assume a fixed, finite set of propositions P and a fixed, finite automaton structure during operation where (in the flat case) the sole RM, or (in the HRM case) $\mathcal{M}$, must be defined prior to operation (Furelos-Blanco et al. 2023). While these approaches can track multiple predefined task instances, in an uncertain environment the agent may discover person $k > n$ when only $\mathcal{M} = [M_1, M_2, \ldots, M_n]$ RMs were specified, and no mechanism to dynamically add instances at runtime is provided. Therefore, tasks which require dynamic instantiation to track an unbounded and unknown number of parallel subtask sequences cannot be expressed with the RM or HRM formalisms without violating finiteness assumptions.

These limitations motivate the need for a method that sup-

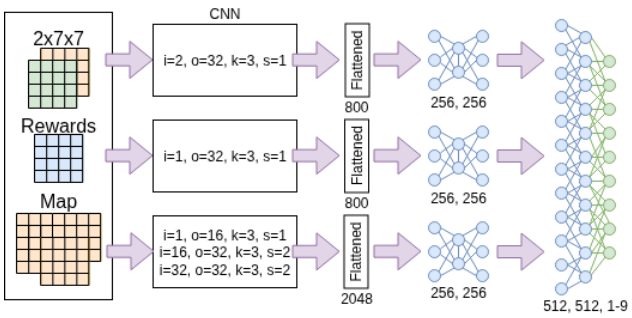

Figure 4: Architecture of the 6 identical DQNs. Training runs for $750k$ steps with $\gamma = 0.99$, replay size $= 100k$, $\alpha = 1e^{-4}$, ϵ decays from $1.0 - 0.01$ and the target network is updated every 5 steps.

ports dynamic, priority-based switching between subtasks, allowing behaviours to be interrupted, resumed, or skipped based on propositions revealed through partial and time-varying observations without constructing an exponentially large automaton. Most importantly, a suitable method must enable the tracking of a large and unknown number of parallel task progressions. The PBRMS method is introduced to address these challenges by using a set of concise RMs to describe task structure, and Contextual Controller to pause and dynamically switch between tasks according to priority and context before resuming the paused task. The next sections describe an environment for evaluating an agent's ability to learn this class of domain, followed by a detailed description of PBRMS and a discussion of evaluation results.

Non-Stationary POMDP Environment

To evaluate the proposed PBRMS method, Rescue Gridworld[1] (Edmondson and Petrick 2026) is used. This environment is designed such that the agent must complete sequential tasks while adapting to partial observations which reveal higher priority tasks. Procedurally generated, the entire layout including room placement, numbers of task-relevant items and their locations is randomised to present a different number and sequence of subtasks each episode. The agent must explore to find the exit, rescuing the moving people. To progress, the agent must find keys (yellow dots), open cupboards containing keycards (brown squares with black markings), and unlock doors (red squares). Each key unlocks one specific cupboard, each keycard one specific door. Ideally, the agent will explore sufficiently to find all the people in the maze, despite not knowing ahead of time how many there will be. Figure 3 shows an example configuration.

The base environment's observation is a configurable $n \times n$, $360°$ line-of-sight window of the map centred on the agent (see Figure 5), where default n is 7. Perfect object detection of observed items is assumed, meaning the agent can (for instance) identify the door number for each observed keycard and see the number of each observed door, this information provided along with the $n \times n$ window. The window itself is a 2 dimensional array of integer values signifying the content of the square. The tile codes are as follows: 0 = empty, 5 = wall, 10 = locked door, 20 = unlocked door, 30 = locked cupboard, 40 = unlocked cupboard, 50 = unlocked cupboard with keycard present, 60 = key present, 70 = person present, 80 = exit, 100 = occluded and 255 = unknown. Values are spread through the 0-255 range to provide contrast to CNN feature extractors. The agent interacts through a set of 9 action codes: 0 = up, 1 = down, 2 = left, 3 = right, 4 = collect key, 5 = unlock cupboard, 6 = collect keycard, 7 = unlock door and 8 = talk to person.

[1] https://pypi.org/project/rescue-gridworld

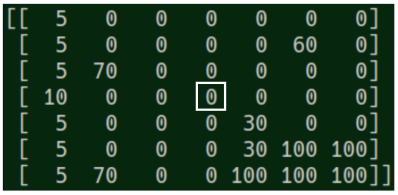

Figure 5: Example partial observation. The agent is at the centre, code 100 shows occlusions by code 30 (cupboards).

Learning Agent Architecture

The agent architecture, shown in Figure 1, consists of five main components: Observations perceived by the agent are first passed to a *History* block implemented as a wrapper, subclassing the Gymnasium `ObservationWrapper`. This block takes each $n \times n$ line-of-sight observation and item information, and updates a local map and state representation, assuming perfect localisation and object detection. Additionally, and in line with Mnih et al. (2015), a stack of j $n \times n$ observations is retained to give the agent a sense of its recent trajectory. Through experimentation, a history of 2 was found to be most effective. At this point, the observation consists of a map built from partial $n \times n$ line-of-sight base observations, a stack of the j most recent $n \times n$ observations and an item state representation. This updated observation is passed to the *Contextual Controller* block which interrogates the constructed map and state representation to identify the current task context. The *Priority-Based Contextual Controller*, described in Algorithm 1, then dynamically switches between RMs depending on identified context. Since context is evaluated each step, the Contextual Controller is a wrapper around the History block.

Discrete RMs are defined as graphs using the Python NetworkX library (Hagberg, Schult, and Swart 2008) and encode the structure of subtasks in the environment. There may be any number of RMs of arbitrary size; an example graph definition is shown in Figure 6, structures with their propositions are shown in Figure 7. This approach, defining task structure in the RM and moving instance tracking into the Contextual Controller, allows significant simplification in definitions. For instance, in this environment the base task is to explore and find the exit. While exploring, an unknown number of people could be encountered at any stage. With the RM only guiding structure of the active task, and progress of each RM instance tracked by the Contextual

```
graph = {  'nodes': [
  {'id': 0, 'name': 'UNKNOWN',
    'peak': 5.0, 'sigma': 10},
  {'id': 1, 'name': 'EXIT',
    'peak': 5.0, 'sigma': 10},
  {'id': 2, 'name': 'GOAL'}],
  'edges': [{'source': 0, 'target': 1},
          {'source': 1, 'target': 2}]}
```

Figure 6: Reward-machine definition of the *Explore* task.

Controller, interruptions by any number of people can be managed in parallel. Since the RM doesn't track specific histories, information about (in this case) which key or keycard has been collected and the observed id of each door and cupboard is stored separately within the Contextual Controller. The controller only moves the agent to higher-priority task states, returning to lower-priority subtasks once those are completed or are no longer entailed by the belief state.

Once the appropriate reward machine has been chosen, its definition is used to calculate the rewards for training and for final operation. The example reward machine definition shows a *peak* value which is the reward given for achieving a subtask or reaching a specific tile location. Additionally, the reward machine contains a *spread* which indicates how far to propagate the reward. Using this information, at each timestep the agent internally updates a *pointwise-max Gaussian radial-basis potential over geodesic distance* relative to current target locations. This provides a dynamic and rich reward surface across the map which changes according to subtask context. The full calculation of the reward surface is detailed below. In addition to providing a rich reward signal for DQN updates, a local $n \times n$ window of the calculated reward-surface is added to the DQN input, aligning in space with the most recent observation window from the environment. This provides a guide to the agent as is navigates during a task, especially during exploration, and is similar to ideas presented by Ewers, Anderson, and Thomson (2025) who use a generated Gaussian PDM to guide an RL agent's priorities for search and rescue tasks.

Actions in the environment are taken by a set of *Deep Q-Networks (DQNs)*, each of which is trained to perform one of the task primitives such as *collect key* or *unlock door*. During training and operation the appropriate DQN is chosen according to the identified context of the environment and the active reward machine's state. As well as the benefits of task-primitive composition, discussed earlier, this approach means each DQN learns a subset of actions, reducing the combinatorial complexity of training. For example, the environment presented here has 9 actions, while each of the 6 DQNs used have a maximum of 5. Each DQN is created using the Stable-Baselines3 (Raffin et al. 2021) library with modified feature extractors. Specifically, `BaseFeatureExtractor` is subclassed since the map, history and reward surface are passed as images too small for standard Stable-Baselines3 CNN configurations. Following feature extraction, the representation is flattened and passed to the final two hidden layers and the output layer. An architecture overview is shown in Figure 4.

Algorithm 1: Priority-Based Contextual RM Controller

1: Initialise $\mathcal{M} = (M_1, M_2, \ldots, M_n)$, the set of RMs.
2: Enumerate $\mathcal{T}$, the set of task primitives, from $\mathcal{M}$.
3: Initialise the belief B to unknown.
4: $p \leftarrow 1$, indexing the lowest priority RM subtask.
5: $c \leftarrow l_p$, indexing M_p's lowest priority state.
6: $\phi_{t=0} \leftarrow 0$, initialise reward surface ϕ to 0.
7: **function** STEP(action a from Algorithm 2:14)
8: Act with a and gather partial observation o_t.
9: Augment B with o_t.
10: Calculate ϕ_t and r_{t-1} as per Equation 5.
11: Step $M_{p,c}$ and observe the result.
12: **if** at M_p goal state OR $B \not\models M_{p,c}$ **then**
13: Reset M_p to initial state.
14: **end if**
15: $(p, c) \leftarrow \underset{(j,k):B_t \models M_{j,k}}{\arg\max} \pi(M_{j,k})$.
16: $SetState(M_{p,c})$.
17: Create O_t with $[o_t, o_{t-1}, \ldots, o_{t-j}]$, B, and ϕ_t.
18: **return** reward r_t and full observation O_t.
19: **end function**

To keep everything separate, each DQN is held in a `TaskAgent` class along with its replay buffer and training update methods, allowing each DQN to have different hyperparameters. Additionally, utility methods translate between DQN action indices and environment action codes, and perform actions such as saving and loading the model. In this work, although each DQN could be initialised differently, they are identical except in the number of outputs which is automatically configured from the associated RM.

Priority-Based Contextual RM Controller

Unlike classical RM or HRM formulations in which the automaton state fully encodes task progress, PBRMS separates task structure (finite RM templates) from task instance state (maintained by the controller). Let $M = (U, V, F, P, \delta_u, \delta_r)$ be a finite RM describing a subtask, where $(U, F, P, \delta_u, \delta_r)$ retain standard RM definitions (Toro Icarte et al. 2022). V is used in place of u_1 since the controller can dynamically enter M at $u \in V \subseteq U$, contingent on observed context.

As shown in Figure 7, each RM state maps injectively to a priority level based on its position in the set of RM definitions. Algorithm 1 specifies the controller's *step* method, called each timestep during operation and from the learning algorithm described in Algorithm 2. Before training starts, the set of RMs $\mathcal{R}$ is interrogated to determine the set of task primitives $\mathcal{T}$. The pointer p references the lowest priority RM and c the lowest priority RM state l_p. Next, the reward surface ϕ is set to 0 and belief B initialised to unknown.

Each step (line 7), the function receives the DQN's action decision which is passed to the base environment. Having acted, the $n \times n$ line-of-sight window observation o_t is returned (line 8) and added to X, filtering for occlusions so as not to lose information (line 9). The reward-surface ϕ_t is then calculated followed by r_{t-1} (line 10).

The active reward machine instance can now be stepped,

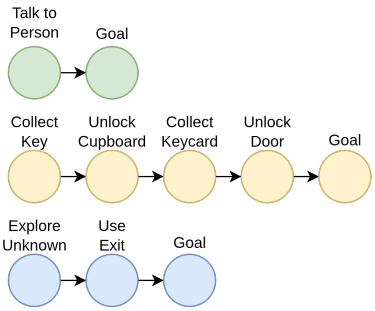

Figure 7: RM subtask proposition sequences. Highest priority RM on top, highest priority state on the left. The controller can move from any state to any higher priority state.

moving internal state if transition criteria have been met. If M_p is at an RM goal state, or the belief no longer entails the active RM state, the active RM is reset (lines 12-14). Next, the agent the obtains the highest priority (π) valid RM state (line 15). This requires Assumption 1 below to hold true since a valid $M_{j,k}$ (that is, an RM state entailed by the current belief) must be selectable. The active state is then set to the new $M_{p,c}$ (line 16). Finally, the observation is generated as described in the architecture section (line 17) and returned (line 18), along with the reward.

Note that PBRMS is intentionally agnostic to the specific data structures used for tracking subtask instance progress. In the same way that RM states capture task-relevant temporal progress while abstracting away underlying state history, the Contextual Controller abstracts over potentially many concurrently active instances of the same subtask template. Any mechanism capable of maintaining instance progress and priority (symbolic, spatial, or learned) can therefore be used. For instance, in this implementation an agent-generated map is updated with new observations, a separate pair of dictionary objects tracking which keys and keycards have been collected and the locations of associated doors and cupboards, using this information when selecting the active RM and generating the reward surface.

Assumption 1 (Epistemic Validity): To ensure correctness in POMDPs, we assume the underlying belief implementation satisfies the property that criteria for at least one $M_{j,k}$ are met within the belief state at every timestep. For example, while *Use Exit* is false we may expect *Explore Unknown* to remain true. Where belief is uncertain; if the current belief holds that tasks are complete but exit conditions unmet, this assumption can be satisfied by resetting some locations to unknown, triggering further exploration.

Theorem 1 (Well-Definedness): Given Assumption 1, the candidate set of valid states $\mathcal{V}$ is strictly non-empty at every timestep t. Therefore, Algorithm 1 always evaluates and transitions to a valid RM state.

Proof. Let $\mathcal{V}_t = \{(j,k) \mid B_t \vDash M_{j,k}\}$ be the candidate set of valid RM state index tuples at timestep t. Since Assumption 1 guarantees $\mathcal{V}_t \neq \varnothing$, the evaluation of $\arg\max$ across $\mathcal{V}_t$ is well defined. Therefore, the controller is guaranteed to select a valid state index tuple $(p,c) \in \mathcal{V}_t$. □

Theorem 2 (Priority Soundness): Given a user-enforced priority mapping π injective over RM states, the $\arg\max$ operation at Line 15 uniquely selects, and the controller guarantees transition to, the highest-priority state B_t entails.

Proof. Let $\mathcal{V}_t = \{(j,k) \mid B_t \vDash M_{j,k}\} \neq \varnothing$ be the candidate set of valid states at t. To determine (p,c) the controller evaluates $\arg\max$ across $\mathcal{V}_t$ with respect to the RM priority mapping π. Since π is injective with respect to RM states it follows that for any two states $M_{j,k},\ M_{l,m} \in \mathcal{V}_t$ where $(j,k) \neq (l,m)$ that $\pi(M_{j,k}) \neq \pi(M_{l,m})$. Therefore, the set of priorities $\Pi = \{\pi(M_{j,k}) \mid (j,k) \in \mathcal{V}_t\}$ has a single strict maximum, and $(p,c) = \arg\max_{(j,k)\in\mathcal{V}_t} \pi(M_{j,k})$ is unique.

The controller therefore transitions to the identified unique maximum $M_{p,c}$, activating the highest priority RM state entailed by B_t. □

Reward Surface Calculation

Rewards for navigation are calculated using potential-based reward shaping (Ng, Harada, and Russell 1999) with a bonus for successfully achieving an action, and a step penalty to encourage efficiency. To determine this reward, the Contextual Controller generates a rich reward surface by calculating a *pointwise-max Gaussian radial-basis potential over geodesic distance*, defined as follows:

Let Ω be the discrete map grid, let $i \in \Omega$ denote an individual grid cell location, and $m : \Omega \to \mathbb{Z}$ the integer tile-coding of the map as built by the agent following observations. Passable locations are indicated by $P(i) = \mathbf{1}[\,m(i) \in \mathcal{P}\,]$, where $\mathcal{P}$ is the set of codes for passable tiles. At each step, the set of target tile codes C is updated according to the active reward-machine state. Each target code $c \in C$ has an associated reward magnitude r_c and seed set $S_c \subseteq \Omega$ of grid locations corresponding to $S_c = \{i \in \Omega : m(i) = c\}$ representing each i which the agent believes to be of type c. For the special code $c = 255$, denoting unknown tiles, S_{255} is restricted to those geodesically closest to the agent in the manner of simple frontier-based exploration although other methods such as information maximisation could be used.

First, $\forall c \in C$ calculate the geodesic distance d to $S_c\ \forall i \in \Omega$:

$$d_c(i) = \begin{cases} 0, & i \in S_c, \\ \text{shortest-path length to } S_c, & \text{if reachable,} \\ +\infty, & \text{otherwise.} \end{cases} \quad (1)$$

The Gaussian radial-basis potential for each c is then:

$$g_c(i) = r_c \exp\left(-\frac{d_c(i)^2}{2\sigma_c^2}\right), \qquad (0 \text{ if } d_c(i) = \infty), \quad (2)$$

where σ_c is an RM-specified value, 5.0 in this case. Starting from $\phi^{(0)} \equiv 0$, potentials combine using the pointwise maximum followed by a global power-law shaping:

$$\tilde{\phi}^{(k)}(i) = \max\big(\phi^{(k-1)}(i),\, g_c(i)\big), \quad (3)$$

$$\phi^{(k)}(i) = M^{(k)} \left(\frac{\tilde{\phi}^{(k)}(i)}{M^{(k)}}\right)^{\gamma}, \quad (4)$$

Hyperparameter	Value
Learning rate	3e-5
Optimiser	Adam
γ	0.9
ϵ max (initial)	1.0
ϵ max (fine-tuning)	0.2
ϵ decay	100k steps
ϵ min	0.01
Buffer size	500k
Target update frequency	100 steps

Table 1: Key DQN training hyperparameters.

where $M^{(k)} = \max_{j \in \Omega} \tilde{\phi}^{(k)}(j)$ and $\gamma > 1$. The non-linearity in this case serving to increase contrast between values of $\phi(i)$ near the peak. The potential is $\phi = \phi^{(K)}$ after all target codes are processed. The final reward expression each step is therefore:

$$\begin{aligned} \epsilon_t =& \{\text{task } c \text{ completed at step } t\}, \\ i_t =& \text{Agent location } i \in m \text{ at step } t, \\ r_t =& (\gamma\phi(i_t) - \phi(i_{t-1})) + r_{c_t}\mathbf{1}[\epsilon_t] - 0.03, \end{aligned} \tag{5}$$

where at most 1 task can be completed at any timestep.

DQN Training Process

Training of the DQNs is completed through curriculum learning, with simplified environments procedurally generated to train each task primitive. This avoids the well-known issue of rare circumstances not being learned effectively (Song et al. 2023) by ensuring the associated DQN gathers many experiences. In this work, 6 task primitives were identified and trained on 2, 3, and 4 room configurations in a 9-stage curriculum for 750,000 steps per stage:

1. *Explore* an unknown 2-room environment.
2. *Collect Keys* in 2 rooms used to unlock cupboards.
3. *Unlock Cupboards* in 2 rooms containing keycards.
4. *Collect Keycards* in 2 rooms from unlocked cupboards.
5. *Unlock Doors* in 2 rooms to navigate between them.
6. *Talk to People* encountered while navigating 3 rooms.
7. *First Group* of tasks; doors of 3 rooms start unlocked.
8. *Second Group* of tasks; doors of 3 rooms are locked, cupboards unlocked making keycard immediately available.
9. *Full Task* with doors and cupboards in 3 rooms locked with probability 0.8.

Note that every stage uses the *Explore* primitive since this task enables exploration of the environment to observe and identify the context of other task primitives. Key hyperparameters are shown in Table 1.

As shown in Algorithm 2, a modification of Mnih et al. (2015) where multiple task primitives are active, learning is completed by contextually switching during action selection (lines $11-12$) and then cycling through each active DQN to perform updates (lines $17-25$). Note that each τ has an associated Q_τ, $\hat{Q}_\tau$ and D_τ, reducing complexity. Important to efficient learning is retaining the experience replays to avoid low-quality updates at the start of the following stage.

Algorithm 2: Multi-DQN with Contextual Switching

1: $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, the set of n task primitives.
2: $\mathcal{Q} = \{Q_\tau \mid \tau \in \mathcal{T}\}$, an associated set of Q functions.
3: **for** each $\tau \in \mathcal{T}$ **do**
4: Initialise replay memory D_τ to capacity N_τ
5: Initialise Q_τ with random weights θ_τ
6: Initialise target $\hat{Q}_\tau$ with weights $\hat{\theta}_\tau = \theta_\tau$
7: **end for**
8: **for** episode $= 1$ to M **do**
9: Reset environment and receive s_1
10: **for** $t = 1$ to T **do**
11: Identify appropriate τ and switch to Q_τ
12: With probability ε select a random action a_t
13: otherwise select $a_t \leftarrow \arg\max_a Q_\tau(s_t, a; \theta_\tau)$
14: r_t, $s_{t+1} =$ STEP(a_t) in Algorithm 1.
15: Store transition (s_t, a_t, r_t, s_{t+1}) in D_τ
16: **Every** U **steps:**
17: **for** each $\tau \in \mathcal{T}$ **do**
18: Minibatch of $(s_j, a_j, r_j, s_{j+1}) \sim D_\tau$
19: **for** each sampled experience **do**
20: $\hat{z} = \max_{a'} \hat{Q}_\tau(s_{j+1}, a'; \hat{\theta}_\tau)$
21: $y_j \leftarrow r_j + (1 - done)\,\gamma\hat{z}$
22: $\theta_\tau \leftarrow \theta_\tau - \alpha\nabla_{\theta_\tau}\big(y_j - Q_\tau(s_j, a_j; \theta_\tau)\big)^2$
23: **end for**
24: **Every** C **updates:** $\hat{Q}_\tau \leftarrow Q_\tau$
25: **end for**
26: **end for**
27: **end for**

Results and Discussion

Compute for experiments was a HP ZBook with 128 GB RAM, Intel Core i9-13950HX CPU and Nvidia RTX 5000 Mobile Ada GPU. Code is written in Python. Run duration for the full curriculum and tests was approximately 33 hours. Evaluation of the environment and PBRMS scaling shows it renders on the above hardware at 10 Hz to configurations of 500 rooms with 500 moving people, 1000 locked doors, cupboards, keys and keycards (Edmondson and Petrick 2026).

The DQN learning experiments were run in full for configurations with 2, 3 and 4 rooms. To evaluate stability, each configuration was trained 12 times with evaluation runs of 25 episodes every 10,000 steps. Following training, a final test of 200 configurations was run to evaluate model performance within the full environment. The agent reliably learned to perform tasks and reach the exit for each of 2-room ($M = 0.87, SD = 0.04$)), 3-room ($M = 0.86, SD = 0.034$) and 4-room ($M = 0.876, SD = 0.021$) configurations. Comparing distributions, the non-parametric Mann-Whitney-U test with respect to 2-rooms shows $p = 0.4$ for 3 rooms and $p = 0.79$ for 4 rooms, suggesting no reduction in performance. The mean and standard deviation of the 3-room curriculum's evaluation episodes are shown in Figure 8. For comparative evaluation, sparse rewards of 1 for success and 0 for failure are used.

As can be seen, individual tasks quickly achieve good rewards, with *Explore*, *Unlock Door* and *Talk to Person* reli-

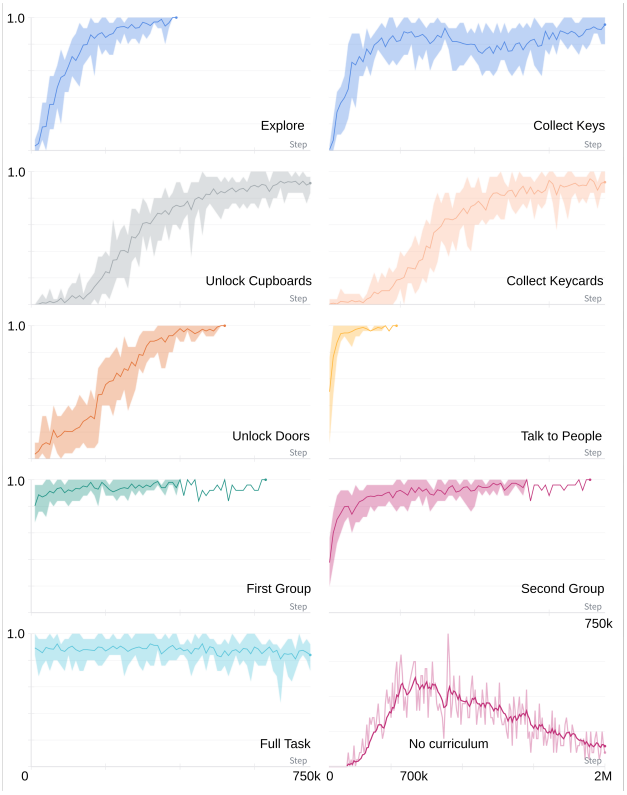

Figure 8: M and SD of reward (y-axis) during evaluation episodes across training steps (x-axis). Each shows a different stage of the curriculum (graphs 1-9) and without curriculum (bottom right).

ably achieving maximum reward for two 25-episode evaluations in a row, triggering early stopping. *First Group* and *Second Group* show that fine-tuning is necessary, with mean rewards eventually climbing to trigger early stopping. This is expected since agents will observe previously unseen tile codes as tasks are added. The full environment does not benefit from fine-tuning, reward declining after 400,000 steps. It is possible fine-tuning could be eliminated by adding random tile codes during single-task training, helping networks learn to ignore subtask-irrelevant codes earlier in training.

As shown bottom-right in Figure 8, removing the curriculum and training on full 2-room configurations fails. The agent completes tasks in approximately 50% of cases after 700,000 steps, but ultimately suffered from catastrophic forgetting. This is expected since, without a curriculum, the agent only infrequently observes later subtasks and may not collect sufficient experiences to cope with the variety of configurations encountered over time.

Keeping the curriculum and rich rewards but removing the reward surface leads to failure in 2-room configurations. Individual subtask completion rates range from 0.03 to 0.56, but subtask group stages are not completed.

Training a single DQN on the 2 room configuration with curriculum fails to learn as well as multiple models, but does better than the multi-DQN architecture without curriculum. Individual tasks achieve success rates between 0.7 and 0.95, and group task fine-tuning helps mean final-task completion reach approximately 0.75. During the curriculum, the DQN network must remember 6 tasks one after the other, by which time the replay buffer contains no experiences from early tasks. Fine-tuning then helps the agent recover, seeming to remind it of actions related to earlier tasks.

Since recurrent networks have often been used within POMDP environments (Bakker 2001; Meng, Gorbet, and Kulić 2021; Li et al. 2024), an LSTM-PPO agent from SB3's Contrib library (Raffin et al. 2021) was used as a baseline. Using the same perception layers as the DQN agent it was trained for 2M steps without curriculum or reward surface, instead receiving sparse rewards for individual subtask completion. This agent performed poorly, unable to complete 2-room configurations, succeeding in fewer than 10% of cases.

The objective of this work, and what the results show, is that the PBRMS approach with contextual switching, geodesic reward shaping, curriculum learning, and action primitive networks enables agents to complete variable chains of tasks within partially observable environments expressed as concise, discrete RMs.

Conclusions and Future Work

This work has shown that environments which require a large and unknown number of subtasks to be tracked in parallel cannot be adequately expressed by standard RM approaches. With its Contextual Controller, PBRMS tracks the state of a large number of parallel tasks and switches contextually, later returning to complete lower-priority subtasks. Using this method, an agent was able to successfully navigate 2, 3 and 4 room variants of the environment, demonstrating its applicability for temporally-extended tasks with varying task structures and numbers of parallel subtasks. Additionally, during scaling experiments the environment and PBRMS agent operated with configurations to 500 rooms and 500 people, 1000 locked doors, cupboards, keys and keycards rendering at 10 Hz. With 1000 rooms and 1500 people it renders at 3 Hz, and without rendering runs at 50 Hz (Edmondson and Petrick 2026). In future, we plan to implement this approach in a robotics simulator, with the long-term plan of showing that PBRMS can operate as a high-level controller for agents completing search and rescue tasks in continuous environments where observations may be uncertain or noisy.

During experiments, it was noted that injecting irrelevant task codes into the agent-constructed map may help it ignore such codes and remove the need for fine-tuning. The authors will also look at this in future work.

Finally, while the utility of Large Language Models in scenarios of this type is not fully established, we plan to identify the extent to which a semantic controller can provide more nuanced control than predefined priorities.

References

Bakker, B. 2001. Reinforcement Learning with Long Short-Term Memory. In Dietterich, T.; Becker, S.; and Ghahramani, Z., eds., *NeurIPS*, volume 14. MIT Press.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *ICML-26*, 41–48. Montreal Quebec Canada: ACM. ISBN 978-1-60558-516-1.

Bergeron, T.; Serlin, Z.; and Leahy, K. 2024. CompLTL: Temporal Logic Planning via Zero-Shot Policy Composition. In *AAAI 2025 Workshop on Generalization in Planning (GenPlan)*. Philadelphia.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the 12th AAAI National Conference on Artificial Intelligence (AAAI 1994)*, 1023–1028. Seattle, Washington: AAAI Press.

Edmondson, A.; and Petrick, R. P. A. 2026. Solving Scaleable POMDP Mazes with Parallel Temporally Extended Task Sequences. In *ICAPS System Demo*, volume 36.

Ewers, J.-H.; Anderson, D.; and Thomson, D. 2025. Deep reinforcement learning for time-critical wilderness search and rescue using drones. *Frontiers in Robotics and AI*, Volume 11 - 2024. Doi:10.3389/frobt.2024.1527095.

Fuchs, F.; Song, Y.; Kaufmann, E.; Scaramuzza, D.; and Durr, P. 2021. Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, 6(3): 4257–4264.

Furelos-Blanco, D.; Law, M.; Jonsson, A.; Broda, K.; and Russo, A. 2023. Hierarchies of Reward Machines. In *ICML*, volume 40. Honolulu, Hawaii: PMLR.

Hagberg, A. A.; Schult, D. A.; and Swart, P. J. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In Varoquaux, G.; Vaught, T.; and Millman, J., eds., *Proceedings of the 7th Python in Science Conference*, 11 – 15. Pasadena, CA USA.

Khetarpal, K.; Riemer, M.; Rish, I.; and Precup, D. 2022. Towards Continual Reinforcement Learning: A Review and Perspectives. *JAIR*, 75: 1401–1476.

Li, A. C.; Chen, Z.; Klassen, T. Q.; Vaezipoor, P.; Icarte, R. T.; and McIlraith, S. A. 2024. Reward Machines for Deep RL in Noisy and Uncertain Environments. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *NeurIPS*, volume 37, 110341–110368. Curran Associates, Inc.

McCloskey, M.; and Cohen, N. J. 1989. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. volume 24 of *Psychology of Learning and Motivation*, 109–165. Academic Press.

Meng, L.; Gorbet, R.; and Kulić, D. 2021. Memory-based Deep Reinforcement Learning for POMDPs. In *2021 IEEE/RSJ IROS*, 5619–5626.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. ICML 1999. San Francisco.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *JMLR*, 22(268): 1–8.

Russell, S. J.; Norvig, P.; Chang, M.-w.; Devlin, J.; Dragan, A.; Forsyth, D.; Goodfellow, I.; Malik, J.; Mansinghka, V.; Pearl, J.; and Wooldridge, M. J. 2022. *Artificial intelligence: a modern approach*. Pearson series in artificial intelligence. Harlow: Pearson, fourth edition, global edition edition. ISBN 978-1-292-40113-3.

Song, Y.; Romero, A.; Müller, M.; Koltun, V.; and Scaramuzza, D. 2023. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82).

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, second edition. ISBN 978-0-262-03924-6.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *JAIR*, 73: 173–208.