ProtoCRL: Prototype-based Network for Continual Reinforcement Learning

Michela Proietti, Peter R. Wurman, Peter Stone, Roberto Capobianco

Keywords: Continual reinforcement learning, experience replay, event tables, prototype-based architecture, Gaussian mixture model, variational inference

Summary

The purpose of continual reinforcement learning is to train an agent on a sequence of tasks such that it learns the ones that appear later in the sequence while retaining the ability to perform the tasks that appeared earlier. Experience replay is a popular method used to make the agent remember previous tasks, but its effectiveness strongly relies on the selection of experiences to store. Kompella et al. (2023) proposed organizing the experience replay buffer into partitions, each storing transitions leading to a rare but crucial event, such that these key experiences get revisited more often during training. However, the method is sensitive to the manual selection of event states. To address this issue, we introduce ProtoCRL, a prototype-based architecture leveraging a variational Gaussian mixture model to automatically discover effective event states and build the associated partitions in the experience replay buffer. The proposed approach is tested on a sequence of MiniGrid environments, demonstrating the agent's ability to adapt and learn new skills incrementally.

Contribution(s)

1. This paper introduces ProtoCRL, a prototype-based architecture for continual reinforcement learning. ProtoCRL features a variational Gaussian mixture model to automatically identify effective event states and build the associated event tables, i.e., partitions within the experience replay buffer (ERB) storing transitions that lead to a particular event state.
   Context: Experience replay is a common strategy used in continual reinforcement learning (Liotet et al., 2022; Luo et al., 2023). Kompella et al. (2023) showed that partitioning the ERB into event tables increases sample efficiency and improves the agent's generalization performance. However, the method is sensitive to the manual selection of event states. ProtoCRL automatizes the construction of the ERB, making event tables suitable to applications in which the identification of event states is nontrivial.
2. The learned Gaussian mixture components practically serve as prototypical representations of an event state. By inspecting the assignments of the input experiences to the Gaussian mixture components, we show that ProtoCRL identifies meaningful event states that the agent needs to visit more often to remember previously learned tasks.
   Context: In the literature, prototypes have been used to either explain pre-trained black-box agents (Borzillo et al., 2023) or to improve the agents generalization performance of agents trained on single tasks (Liu et al., 2023). In this work, we leverage the learned prototypical representations to both guide experience replay and gain insights into what information is useful for the agents to maintain the ability to perform multiple tasks learned in sequence.
3. We show that ProtoCRL achieves comparable performance to manually defined event tables and even higher performance when reducing the ERB capacity.
   Context: We test ProtoCRL on a sequence of three MiniGrid environments (Chevalier-Boisvert et al., 2018), comparing its performance in terms of average return and forgetting to manually defined event tables and to ContinualDreamer (Kessler et al., 2023).

ProtoCRL: Prototype-based Network for Continual Reinforcement Learning

Michela Proietti[1], **Peter R. Wurman**[2], **Peter Stone**[2,3], **Roberto Capobianco** [2]
mproietti@diag.uniroma1.it,
{peter.wurman,peter.stone,roberto.capobianco}@sony.com

[1]**Sapienza University of Rome, Italy**
[2]**Sony AI**
[3]**The University of Texas at Austin**

Abstract

The purpose of continual reinforcement learning is to train an agent on a sequence of tasks such that it learns the ones that appear later in the sequence while retaining the ability to perform the tasks that appeared earlier. Experience replay is a popular method used to make the agent remember previous tasks, but its effectiveness strongly relies on the selection of experiences to store. Kompella et al. (2023) proposed organizing the experience replay buffer into partitions, each storing transitions leading to a rare but crucial event, such that these key experiences get revisited more often during training. However, the method is sensitive to the manual selection of event states. To address this issue, we introduce ProtoCRL, a prototype-based architecture leveraging a variational Gaussian mixture model to automatically discover effective event states and build the associated partitions in the experience replay buffer. The proposed approach is tested on a sequence of MiniGrid environments, demonstrating the agent's ability to adapt and learn new skills incrementally.

1 Introduction

Recent advances in deep Reinforcement Learning (RL) have achieved super-human performance in several applications (Silver et al., 2016; Wurman et al., 2022). However, unlike humans, RL agents lack the ability to continuously and incrementally learn new skills, which is particularly crucial in open-world games featuring diverse challenges. In these settings, the major obstacle to continual learning is catastrophic forgetting (McCloskey & Cohen, 1989; French, 1999), a phenomenon consisting of new knowledge overwriting previously learned information. Continual RL (CRL) aims to address this issue by proposing methods to mitigate forgetting and improve the agent's performance in sequential tasks (Khetarpal et al., 2022).

Experience Replay (ER) is a common strategy used in both RL and CRL, albeit with slightly different objectives. In both contexts, ER involves storing the agent's experiences in an Experience Replay Buffer (ERB) and randomly sampling from it to update the agent's policy. In standard single-task RL settings, ER thus leads to better generalization and higher stability, as it ensures that learning relies on a more diverse set of experiences and breaks the correlation between consecutive experiences (Lin, 1992). In CRL, instead, the ERB also stores data from previous tasks, which is periodically replayed to prevent catastrophic forgetting (Rolnick et al., 2019b).

In environments where crucial events occur rarely, Kompella et al. (2023) propose to partition the ERB into event tables, each storing transitions leading to a particular event, i.e. a rarely visited state that is essential for task completion. Their approach, called Stratified Sampling from Event

Tables (SSET), demonstrates increased robustness to perturbations compared to other ER strategies, showcasing its effectiveness in mitigating catastrophic forgetting. However, manual selection of meaningful event states might not always be straightforward, especially in complex games in which the environment has many diverse features.

To overcome this limitation and to ensure a more flexible approach for CRL, we introduce ProtoCRL, the first prototype-based architecture for CRL. By relying on a Variational Gaussian Mixture Model (VGMM), ProtoCRL automatically identifies event states, assigning each experience to one of the VGMM components, thus eliminating the need for manual event definitions. Moreover, given that the VGMM can learn to use fewer clusters than its maximum capacity, we just need to define the maximum number of components and it will autonomously learn to use additional clusters, and thus additional event tables, for new tasks as they arise. When evaluated on three sequential MiniGrid tasks, ProtoCRL not only reaches similar performance to baselines that rely on predefined event tables, but also surpasses them under memory constraints. Moreover, inspecting the automatically built event tables offers better insight into the agent's decision-making. Finally, although we use ProtoCRL as online and target network in DDQN (Hasselt et al., 2016), its modular design makes it easily adaptable to other algorithms.

Our main contributions are:

- Proposing the first prototype-based architecture featuring a VGMM for automatic event discovery. ProtoCRL additionally makes event tables more flexible and thus suited to CRL settings, as the VGMM can dynamically leverage fewer clusters than the maximum number given, leaving extra capacity to accommodate additional event tables as new tasks appear.
- Showing that ProtoCRL identifies meaningful event states by inspecting the assignments of observations to the VGMM's components, thus gaining insights into the agent's decision process.
- Providing evidence of ProtoCRL comparable performance to baselines with manually defined event tables and superior performance in the presence of memory constraints on shuffled sequences of three MiniGrid tasks.

2 Preliminaries

2.1 Reinforcement Learning

Markov Decision Process In this work, we consider an agent acting in a *Markov Decision Process* (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, \mathcal{P}, \gamma, \mathcal{I}, \beta \rangle$ (Sutton et al., 1998), with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to Pr[\mathcal{S}]$, discount factor $\gamma \in [0, 1)$, initial state distribution $\mathcal{I} : Pr[\mathcal{S}]$, and episode termination function $\beta : \mathcal{S} \to \{0, 1\}$. At time step t, the agent observes state s, uses its current policy $\pi : \mathcal{S} \to Pr[\mathcal{A}]$ to choose action a, and receives reward r and next state s'. The episode ends if a termination condition is verified ($\beta(s') = 1$) or if a horizon of H is reached. The action-value function $Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(s,a)}[V^\pi(s')]$ represents the expected discounted return when taking action a in state s and following policy π thereafter. $V^\pi(s) = \mathbb{E}_{a \sim \pi(s)}[Q^\pi(s, \pi(s))]$ is the state value function representing the expected return when starting in state s and following policy π.

Model-free Off-policy Methods The aim of the agent is to find an optimal policy π^* and the corresponding $Q^*(s, a)$ that maximizes the expected discounted return. *Model-free* methods learn the value function or policy directly from interactions, without explicitly modeling the environment's dynamics (transition and reward functions). Additionally, *off-policy* methods can learn from experiences generated by a different policy (behavior policy) than the one being improved (target policy), enabling the reuse of past experiences. Consequently, model-free off-policy methods learn $Q^*(s, a)$ directly from data by incrementally updating the action-value function based on the temporal difference (TD)-error $\delta = r + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a)$, where $\max_{a'} Q_k(s', a')$ represents the maximum action-value in the next state (approximating the optimal state value). DDQN (Hasselt et al., 2016) is one such approach, in which the action-value function is represented as a deep neural network with parameters ϕ, that are updated along the gradient of the TD-error. With respect to DQN (Mnih et al., 2015), DDQN uses two different value networks: an online network for current Q-values and a target network for computing the target values that is updated with lower frequency.

Experience Replay with Event Tables In off-policy RL, experiences $\langle s, a, r, s' \rangle$ are typically stored in an ERB and then sampled in batches to perform gradient updates to the value networks parameters. Kompella et al. (2023) propose partitioning the ERB into n event tables $\mathcal{B}^{\nu_i}$ and a default table $\mathcal{B}^0$. The default table stores all transitions, while each event table is associated with an event specification $\nu = \langle \omega, \tau \rangle$, where $\omega : \mathcal{S} \rightarrow 0, 1$ is a boolean event condition and τ is a history length. Typical good candidates for event states are important states that occur rarely, such as goal states or bottleneck states, and their associated event conditions need to be specified by domain experts. All tables are handled in a FIFO manner and insertion and sampling operations are controlled by parameters η, κ, d_i, with $i \in [0, n]$, representing sampling probabilities, capacity sizes, and minimum data requirements for each table (including the default table, corresponding to $i = 0$), respectively.

Continual Reinforcement Learning CRL extends the standard RL paradigm to scenarios where an agent is faced with a sequence of tasks or changing environments. Each new task is represented as a new MDP $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{A}_i, R_i, \mathcal{P}_i, \gamma_i, \mathcal{I}_i, \beta_i \rangle$, with $i = 1, ..., T$. During the i-th task training, the agent collects only experiences from that task.

2.2 Variational Gaussian Mixture Model

Gaussian Mixture Model A *Gaussian Mixture Model* (GMM) is a widely-used probabilistic model for clustering and density estimation. The GMM assumes that the observed data is generated from a mixture of several Gaussian distributions, each representing a different cluster or component in the data. In this work, we deal with multi-dimensional data, so we use a *multivariate GMM* which models the distribution of a set of N observed data points $\mathcal{X} = x_1, ..., x_N$, with $x_n \in \mathbb{R}_d$, as a weighted sum of K multivariate Gaussian components. The probability density function for the multivariate GMM is given by:

$$P(x_n|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_{|}\mu_k, \Sigma_k)$$

where π_k are the mixing weights, $\mathcal{N}(x_n|\mu_k, \Sigma_k)$ is a multivariate Gaussian distribution with mean μ_k and covariance matrix Σ_k, and $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$ represents the set of all parameters of the model. The mixing weights are such that $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$, as they represent the probability that a data point belongs to each mixture component.

Variational Inference Given a GMM, the end goal is to perform inference, i.e., to estimate the parameters of the mixture components given some observations:

$$p(\theta|\mathcal{X}) = \frac{p(\mathcal{X}|\theta)p(\theta)}{p(\mathcal{X})}$$

where $p(\mathcal{X}|\theta)$ is the likelihood of the data given the parameters, and $p(\theta)$ is the prior distribution. However, the computation of the denominator, called evidence, is generally intractable. The main idea of *variational inference* is to find a simple distribution $q(\theta, \phi)$ from a family of distributions $\mathcal{Q}$, called variational distributions, that is a good approximation of the true posterior. Intuitively, instead of computing the exact (but intractable) posterior distribution over model parameters, we search for the best approximation within a tractable family of distributions. To this aim, we look for the distribution $q^*(\theta, \phi)$ that minimizes the Kullback-Leibler (KL) divergence between the true and approximate posteriors, which is equivalent to maximizing the *evidence lower bound* (ELBO):

$$q^*(\theta, \phi) = \arg \min_{q(\theta,\phi) \in \mathcal{Q}} D_{KL} \left[q(\theta, \phi) || p(\theta, \phi|\mathcal{X}) \right]$$

To this aim, it is necessary to place prior distributions over the model parameters θ. A model that applies this variational treatment to a Gaussian mixture, placing conjugate priors on the weights, means, and covariances, and approximating their joint posterior is called a *Variational Gaussian Mixture Model* (VGMM). The key advantage of VGMMs over standard GMMs is their ability to automatically determine the appropriate number of mixture components, leaving some components inactive during learning. This feature makes VGMMs particularly suitable for continual learning scenarios where the number of meaningful clusters may grow over time.

3 Related Work

Continual Reinforcement Learning In recent years, interest in CRL has grown considerably (Khetarpal et al., 2022). However, the study of catastrophic forgetting and the more general stability-plasticity dilemma has a much longer history (McCloskey & Cohen, 1989; French, 1999; Mermillod et al., 2013). While some approaches aim at reducing forgetting by applying some form of weight or function regularization (Kirkpatrick et al., 2016; Kessler et al., 2022), architecture-based strategies feature task-specific sub-networks or dynamic networks to prevent task interference (Rusu et al., 2016; Mallya & Lazebnik, 2017). Another popular solution is experience replay (ER), first introduced by Lin (1992) as a method to improve sample efficiency in RL by storing and reusing past experiences. In the context of CRL, ER consists of storing and replaying experiences from old tasks (Rolnick et al., 2019a; Li et al., 2021; Liotet et al., 2022; Luo et al., 2023). For instance, CLEAR (Rolnick et al., 2019a) balances plasticity and stability by alternating off-policy learning from the ERB and on-policy learning from new experiences, using behavioural cloning between the current policy and its past version to increase stability. The way in which we populate and sample from the ERB significantly affects the final performance, as demonstrated by Kessler et al. (2023). In particular, ER is not particularly suited in environments featuring crucial event states that occur very rarely during each episode. These states might get sampled too infrequently from the ERB, leading to the agent being unable to efficiently and effectively learn the task at hand. Kompella et al. (2023) address this issue by dividing the ERB into event tables and ensuring they get sampled frequently enough. However, the effectiveness of event tables strongly relies on the definition of event conditions by domain experts, and a wrong choice of events might lead to suboptimal performance. The purpose of ProtoCRL is to avoid this problem by automatizing the construction of event tables.

Prototypes in Reinforcement Learning The term *prototype* has divergent meanings across different research sub-fields. In representation learning, a prototype is a latent vector encoding features relevant for achieving a certain outcome, i.e., a prototypical embedding in the latent space. In explainable artificial intelligence, a prototype is typically an input instance or a part of it (e.g., an image patch) that is representative of a specific class.

While early prototype-based networks focused on supervised image classification (Chen et al., 2019; Rymarczyk et al., 2022), recent works in RL adapt these principles to interpret black-box image-based RL agents. Ragodos et al. (2022) exploit the black-box model's demonstrations to train a self-interpretable agent by imitation learning, while Borzillo et al. (2023) and Kenny et al. (2023) use the pre-trained black-box agent as an encoder. In representation learning, instead, self-supervised learning and task-agnostic pre-training allow learning prototypical embeddings to enhance the generalization capabilities of RL agents (Yarats et al., 2021; Liu et al., 2023; Mazoure et al., 2022). For instance, DreamerPro (Deng et al., 2022) clusters observations into trainable prototypes and predicts the cluster assignment from both the world model's state and an augmented view of the observations, while ProtoCAD (Wang et al., 2024) utilizes prototypical representations to extract contextual information from varying dynamic environments.

To our knowledge, no existing work applies a prototype-based network to CRL. We address this gap via an architecture featuring a VGMM (Corduneanu & Bishop, 2001; Nasios & Bors, 2006) for clustering latent representations and identifying prototypical embeddings of event states. Unlike prior methods, our architecture does not require a pre-trained black-box agent or self-supervised pre-training.

4 ProtoCRL

CRL requires an agent to learn new tasks without forgetting earlier ones and using a fixed-size replay buffer. In this section, we introduce ProtoCRL, our prototype-based architecture that addresses these challenges by integrating a prototype-based latent representation with an event-based experience replay mechanism. By learning to cluster latent features via a VGMM, ProtoCRL automatically identifies prototypical event states that are critical for learning. These prototypes are then used to form event tables in the replay buffer, ensuring that diverse transitions are revisited during training to avoid forgetting within a constant memory budget.

4.1 Network Architecture

The network, illustrated in Figure 1, consists of three interdependent components: an encoder f_e, a VGMM module, and an output layer f_o.

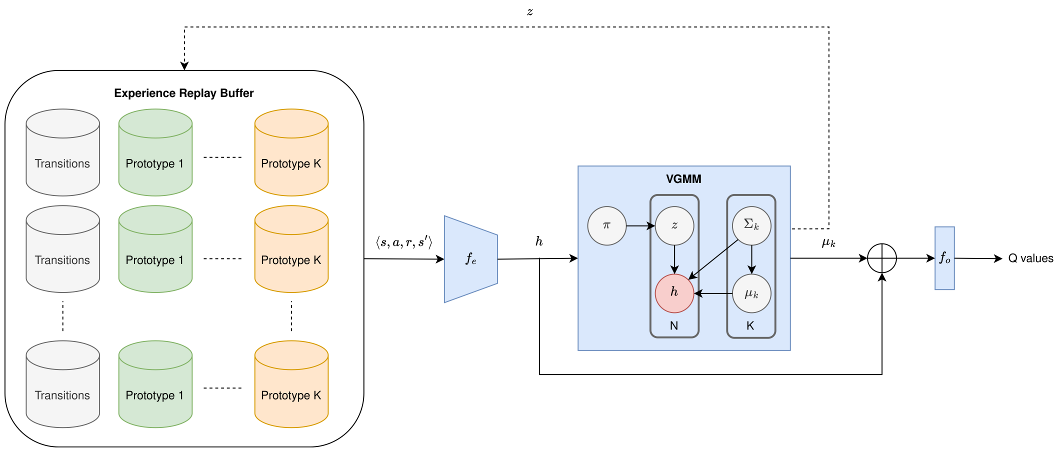


Figure 1: **ProtoCRL**. A transition $\langle s, a, r, s' \rangle$ is sampled from the ERB and state s is passed to the encoder f_e, which produces latent representation h. h is the input to a VGMM with K components. The VGMM assigns the input observation to one of its components through $z \in \mathcal{Z}^K$, which is a one-hot assignment vector. The VGMM is parametrized by mixing coefficients $\pi = (\pi_1, ..., \pi_K)$, means $\mu = (\mu_1, ..., \mu_K)$, and covariance matrices $\Sigma = (\Sigma_1, ..., \Sigma_K)$. The latent state representation h and the mean of the component it is assigned to μ_k are concatenated and passed to an output dense layer f_o, which predicts Q values. When collecting the experiences to store in the ERB, each experience is stored in the prototypical event table associated with the VGMM component the observation is assigned to.

Encoder The encoder f_e transforms high-dimensional raw observations into a compact latent representation. This component can be implemented differently, depending on the type of input observations. The learned latent space is crucial for capturing the relevant features needed for both control and for subsequent clustering. In our experiments, we implement the encoder as a simple dense neural network with three hidden layers and LeakyReLU nonlinearities.

Variational Gaussian Mixture Model The VGMM processes the latent representations produced by the encoder and clusters them into a set of prototypes.

As explained in Section 2.2, in order to maximize the ELBO objective, we need to place prior distributions over the model parameters θ. In this work, we use a Dirichlet prior for the mixing coefficients π_k, a Gaussian prior for the means μ_k, and a Gamma prior for the magnitude of the precision matrix, as reported in the literature (Lu, 2021). Additionally, to make this optimization scalable, especially for large datasets, we use Stochastic Variational Inference (SVI) (Hoffman et al., 2013). At each iteration, the gradient of the ELBO with respect to the variational parameters ϕ is estimated using the batch of latent representations produced by the encoder. The gradient is then used to update ϕ using a learning rate η_t, following the update rule:

$$\phi_{t+1} = \phi_t + \eta_t \nabla_\phi ELBO(\phi_t, \mathcal{X}_{mini-batch})$$

By iterating over mini-batches, the variational parameters ϕ converge to values that approximate the true posterior distribution of the model parameters. The main advantage of using variational inference is that after defining the maximum number of components in the VGMM, a smaller number of components might be used based on the input data. This is particularly useful in our CRL setting, in which an increasing number of components are needed as the number of seen tasks grows.

Output Layer The output layer maps the prototype-augmented features, obtained by concatenating the encoder's output with the mean of the GMM component it is assigned to, to action-value estimates using a shared network head. In our continual learning setting, the use of a shared head across tasks promotes knowledge transfer while reducing the overall model complexity. However, this choice may also introduce interference if tasks are highly divergent, which is mitigated by our selective replay strategy.

4.2 Prototypical Event Tables

To enhance learning and avoid catastrophic forgetting, ProtoCRL employs event tables (Kompella et al., 2023). Specifically, for each task $i \in [1, T]$, we define a default table $\mathcal{B}_i^0$ and prototypical event tables $\mathcal{B}_i^{\nu_k}$, with $k \in [1, K]$. Let $h = f_e(s)$ be the latent representation of state s obtained from the encoder. The VGMM clusters the stream of latent vectors online, so each mixture component center becomes a prototype event state that summarizes a set of similar, informative states. For a VGMM with K components, let $p(k \mid h)$ be the probability that latent vector h belongs to component k; the event condition for table k can be defined as the indicator function

$$\omega_k(s) = \mathbf{1}\left\{ k = \arg \max_{j \in \{1,\ldots,K\}} p(j \mid f_e(s)) \right\}$$

which evaluates to 1 if and only if the latent representation $h = f_e(s)$ is most likely generated by the k-th component, and 0 otherwise. Over-sampling each prototype's table during training therefore reinforces those critical, cluster-representative transitions and mitigates forgetting. Manual event-table methods require careful tuning of sampling probabilities η_i to be effective, whereas ProtoCRL simply fixes a common sampling probability for all prototype tables, leaving finer schemes to future work. However, results show that ProtoCRL effectively manages to distribute states among tables to favor more frequent sampling of important states.

4.3 Training Procedure

Before starting the actual training, we collect data by performing a random policy on the first task in the sequence for $N_{init} = 200$ steps. The training of ProtoCRL then consists of three main phases: (1) VGMM warm start to initialize clustering, (2) alternating optimization between policy improvement and prototype refinement, and (3) prototypical event table construction and sampling. Algorithm 1 provides a detailed overview of the complete training procedure.

VGMM Warm Start To ensure initial reliable clustering in the latent space, the VGMM undergoes a warm start procedure. Although the agent has yet to see future tasks, this warm start builds a reasonable basis for prototype extraction. As training proceeds through subsequent tasks, the clustering continues to be updated in an online fashion to reflect data from the new tasks. The training objective used during this warm start procedure is made up of several components. The first one is the TD loss, which ensures that the learned representations are meaningful for the task at hand. The second term is the ELBO, that we need to maximize:

$$\mathcal{L}_{ELBO} = \alpha_{logp} log\ p(\mathcal{X}) - \alpha_{KL}\ D_{KL}\left[q(\theta, \lambda)||p(\theta, \lambda|\mathcal{X})\right]$$

The third component is an entropy term that encourages the use of diverse clusters. Let $r_i \in \mathbb{R}^K$ denote the cluster probability vector for the i-th sample, obtained by applying a softmax over the VGMM log-probabilities. We define the average probability of cluster k over a batch of N samples as:

$$m_k = \frac{1}{N}\sum_{i=1}^{N} r_{i,k}, \quad \text{for } k = 1, \dots, K$$

Then, we define the entropy loss as:

$$\mathcal{L}_E = -\sum_{k=1}^{K} m_k \log\big(m_k + \epsilon\big),$$

where ϵ is a small constant for numerical stability. Finally, we use Hoyer's sparsity loss (Hoyer, 2004) to encourage exactly one VGMM component to have high probability for each input observation. The Hoyer sparsity measure for the i-th input sample takes the form:

$$H(r_i) = \frac{\sqrt{K}\,\|r_i\|_1}{\|r_i\|_2 + \epsilon} - 1$$

where $\|r_i\|_1 = \sum_{k=1}^{K} |r_{i,k}|$ and $\|r_i\|_2 = \sqrt{\sum_{k=1}^{K} r_{i,k}^2}$. A perfectly one-hot vector achieves $H(r_i) = 1$, while a uniform distribution yields a lower value. The overall Hoyer sparsity loss over a batch of N samples is computed as the mean:

$$\mathcal{L}_H = \frac{1}{N}\sum_{i=1}^{N} H(r_i).$$

Minimizing this loss encourages each probability vector r_i to become more sparse. The overall training objective is:

$$\mathcal{L}_{ProtoCRL} = \mathcal{L}_{TD} - \lambda(\mathcal{L}_{ELBO} - \beta_E\,\mathcal{L}_E + \beta_H\,\mathcal{L}_H)$$

with λ, β_E and β_H being regularization parameters.

Alternating Training Objectives After the warm start procedure, we alternatively optimize two objectives: the TD loss, to sharpen the agent's policy, and the joint objective used during warm start, to establish a well-structured latent space and meaningful prototypes that form the basis for effective event table construction. This alternating training strategy is designed to find a balance between maintaining a robust and interpretable latent structure and optimizing the agent's decision-making capabilities. This design not only automates the discovery of significant event states but also improves the overall performance of the agent in continual learning environments.

5 Experiments

We test ProtoCRL on a sequence of 3 MiniGrid (Chevalier-Boisvert et al., 2018) tasks - *DoorKey*, *LavaCrossing*, and *SimpleCrossing* - using DDQN algorithm (Hasselt et al., 2016). The environments are episodic and terminate when the agent reaches the goal or when we reach the maximum episode length (set to 200 at evaluation time). The agent receives a reward of -0.05 for each environment step, $+1$ for reaching the goal, and -5 for falling into lava. Further details on the environments, architectures and hyperparameters needed for reproducibility are given in Appendix A. In our experiments, an epoch corresponds to 2000 environment steps. We train all agents for 3K

Algorithm 1 ProtoCRL Training Procedure

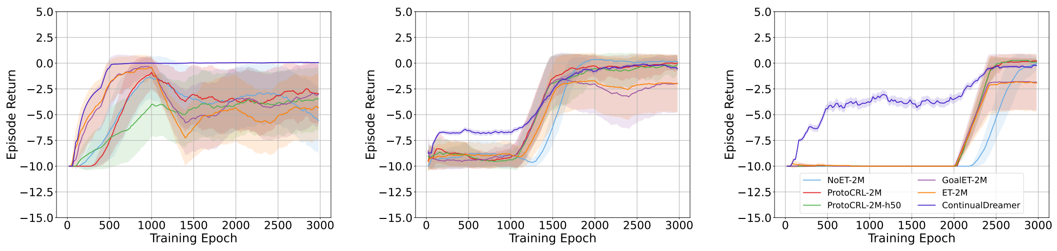
```
Require: Task sequence $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_T\}$, VGMM components $K$, start steps $N_{init}$
Ensure: Trained ProtoCRL network with prototypical event tables
 1: Initialize encoder $f_e$, VGMM parameters $\{\pi, \mu, \Sigma\}$, output layer $f_o$
 2: Initialize experience replay buffer partitions: $\{\mathcal{B}_i^0, \mathcal{B}_i^{\nu_k}\}$ for $i \in [1, T], k \in [1, K]$
 3: for task $i = 1$ to $T$ do
 4:     if $i = 1$ then                                              ▷ VGMM warm start for first task
 5:         for step $t = 1$ to $N_{warm}$ do
 6:             Collect experience $\langle s_t, a_t, r_t, s_{t+1} \rangle$ using random policy
 7:             Observe state $s_t$, compute $h_t = f_e(s_t)$
 8:             Determine prototype assignment: $k^* = \arg\max_j p(j|h_t)$
 9:             Store experience in event tables $\mathcal{B}^{\nu_{k^*}}$ and $\mathcal{B}^0$
10:         end for
11:         for epoch $e = 1$ to VGMM warm start epochs do
12:             Sample batch from all event tables
13:             Compute $\mathcal{L}_{ProtoCRL} = \mathcal{L}_{TD} - \lambda(\mathcal{L}_{ELBO} - \beta_E \mathcal{L}_E + \beta_H \mathcal{L}_H)$
14:             Update VGMM parameters $\pi_k, \mu_k, \Sigma_k$ for $k \in [1, K]$
15:         end for
16:     end if
17:     for epoch $e = 1$ to training epochs per task do
18:         Environment Interaction:
19:         for step $t = 1$ to steps per epoch do
20:             Observe state $s_t$, compute $h_t = f_e(s_t)$
21:             Select action $a_t$ using $\epsilon$-greedy policy based on $Q(h_t \oplus \mu_k, a)$
22:             Execute $a_t$, observe $r_t, s_{t+1}$
23:             Determine prototype assignment: $k^* = \arg\max_j p(j|h_t)$
24:             Store $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in event tables $\mathcal{B}_i^{\nu_{k^*}}$ and $\mathcal{B}^0$
25:         end for
26:         Alternating Training:
27:         if epoch $e$ is odd then                                 ▷ Policy improvement phase
28:             Sample batch from all event tables with probabilities $\{\eta_i\}$
29:             Optimize TD loss: $\mathcal{L}_{TD}$ using DDQN updates
30:         else                                                     ▷ Prototype refinement phase
31:             Sample batch from all event tables
32:             Optimize joint objective: $\mathcal{L}_{ProtoCRL}$
33:             Update VGMM parameters $\{\pi, \mu, \Sigma\}$ via SVI
34:         end if
35:         if $e$ % target network update frequency $= 0$ then
36:             Update target network parameters
37:         end if
38:     end for
39: end for
```

epochs (1K epochs for each task) and evaluate them with a frequency of 20 epochs, averaging results over 10 seeds. For each experiment, we report average performance and average forgetting CRL metrics (Woł czyk et al., 2021), whose formal definitions are provided in Appendix B. The former measures how well the agents performs on average on all tasks at the end of the training, while the latter captures the drop in the agent's performance on previous tasks after learning the entire task sequence.

A central goal of ProtoCRL is to eliminate the need for manually selecting event states while preserving performance in CRL settings. For this reason, we compare it to (1) a baseline using a single FIFO ERB with no event tables (NoET), (2) a simplified variant of event-based replay in which we introduce a single event table holding transitions related to the $goal$ state, (3) an event-based strategy using multiple event tables ($goal$, at_door, $pickup\_key$ for DoorKey, $goal$ and at_lava for LavaCrossing, and $goal$ and at_gap for SimpleCrossing), (4) ContinualDreamer (Kessler et al., 2023), which leverages both experience replay and world models. In particular, we expect ContinualDreamer to be a strong baseline, as its learned dynamic model can generate additional transitions and thus provide more frequent revisits of past tasks.

5.1 Performance Comparison

In this section, we compare the agents' performance when using an ERB with capacity of 2M. Note that event tables are partitions within the ERB, therefore the total memory usage remains fixed at 2M transitions, regardless of the number of event tables. Figure 2 shows episodic returns for all tasks across the whole training. In the DoorKey task (Figure 2a), all methods steadily improve over the training epochs, but experience different performance drops when encountering the second task in the sequence, at epoch 1K. While ProtoCRL, GoalET, and ET manage to recover from this drop, NoET keeps losing its ability to perform the DoorKey task, reaching the lowest final performance. In these experiments, we also show that ProtoCRL benefits from storing a longer history of transitions in the event tables, namely $\tau = 500$ compared to $\tau = 50$, which is the value used for GoalET and ET. ProtoCRL-2M and ProtoCRL-2M-h50 identify the variants using $\tau = 500$ and $\tau = 50$, respectively. ProtoCRL-2M-h50 obtains a lower final return than ProtoCRL-2M, as despite not experiencing forgetting, ProtoCRL-2M-h50 fails to reach a high performance on the DoorKey task. On the other hand, ContinualDreamer is the approach that converges faster and maintains the highest final return, indicating that the integration of world models and experience replay is well suited for CRL settings. This result is expected, as ContinualDreamer trains a single latent-dynamics network that is shared across tasks, then "imagines" roll-outs inside that latent space to generate synthetic transitions. This synthesis should allow more frequent and flexible revisits of earlier tasks than any buffer-only method. On the subsequent tasks, all methods obtain similar returns, except GoalET-2M and ET-2M, which reach the lowest final performances but do not experience much forgetting. Overall, ProtoCRL-2M achieves an average performance of -0.45 ± 1.11 and a near-zero forgetting of 0.03 ± 1.72, as reported in Table 1. This outcome is markedly better than the NoET and ET baselines, whose average performance stays around -2. Negative average forgetting (i.e., the final performance is higher at the end of the entire training than after training the individual tasks) appears for GoalET and ProtoCRL-h50 only because both variants start from a lower DoorKey score and then maintain roughly the same return throughout training.



(a) DoorKey task. (b) LavaCrossing task. (c) SimpleCrossing task.

Figure 2: Episodic return on 3 MiniGrid tasks using a 2M memory budget. The task ordering is *DoorKey-LavaCrossing-SimpleCrossing*. Each agent is trained on each task for 1K epochs and evaluated every 20 epochs across the entire training sequence. Shaded areas indicate standard deviations across 10 seeds.

5.2 Decreasing Buffer Capacity

Sample efficiency is a crucial requirement in CRL settings. A major drawback of using a fixed-size experience-replay buffer in continual learning is that, as new tasks arrive, the portion of the buffer

Table 1: Average performance (AvgP) and forgetting (AvgF) on three MiniGrid tasks using 2M memory budget.

	AvgP	**AvgF**
NoET	-2.35 ± 3.79	2.32 ± 3.28
GoalET	-1.75 ± 0.01	$\mathbf{-0.02 \pm 0.02}$
ET	-2.92 ± 1.36	1.78 ± 2.35
ContinualDreamer	$\mathbf{0.19 \pm 0.07}$	0.05 ± 0.09
ProtoCRL-h50	-0.82 ± 1.64	$\mathbf{-0.28 \pm 0.61}$
ProtoCRL	$\mathbf{-0.45 \pm 1.11}$	0.03 ± 1.72

devoted to earlier tasks shrinks, effectively diluting the agent's memory of past experiences. It is therefore necessary for CRL methods to perform well even in the presence of memory constraints. For this reason, we test the effectiveness of NoET, GoalET, ET, and ProtoCRL with different buffer capacities.

In Table 2, we compare the effectiveness of these methods in terms of average performance. In Appendix C, we additionally provide the average forgetting and plots showing the episodic returns for the three environments across the entire training. Overall, ET and GoalET are the approaches that suffer from the greatest performance drops (i.e., increased forgetting) when decreasing the memory budget, reaching performances that are much lower than the NoET baseline with 1M and 100K memory budgets. This might be due to manually defined tables oversampling a narrow subset of transitions, thus limiting the agent's exposure to broader context. If event definitions or table sampling probabilities are not well tuned, uniform replay (NoET) might maintain a more diverse set of experiences, resulting in better overall performance. On the contrary, ProtoCRL, with its ability to automatically build event tables, manages to achieve high performance with very low memory budgets.

Table 2: Average performance over 3 MiniGrid tasks with different memory budgets. All metrics are an average and standard deviation over 10 seeds. We highlight in bold the best performing approaches.

	100K	**500K**	**1M**	**2M**
NoET	-5.36 ± 3.79	-4.82 ± 4.22	-4.24 ± 4.10	-2.35 ± 3.79
GoalET	-8.62 ± 0.98	-6.90 ± 1.46	-7.82 ± 1.55	-1.75 ± 0.01
ET	-9.66 ± 0.49	-5.87 ± 2.23	-8.54 ± 1.03	-2.92 ± 1.36
ProtoCRL	$\mathbf{-1.74 \pm 2.24}$	$\mathbf{-2.84 \pm 1.43}$	$\mathbf{-0.83 \pm 1.63}$	$\mathbf{-0.45 \pm 1.11}$

5.3 Robustness To Task Order

To assess whether the superior performance of ProtoCRL was due to task ordering, we ran all six permutations of the three MiniGrid tasks, using the same experimental setup (seeds, hyperparameters) of the previous experiments. Table 3 reports the average performance and forgetting over all 6 permutations, together with standard deviations across permutations, using 100K and 2M memory budgets. In Appendix D, we provide details on the exact permutations, and report per-permutation average metrics and episodic return.

ProtoCRL achieves the best average performance in both memory regimes. With only 100K transitions it outperforms the strongest baseline (NoET) by 0.29 points. The gap widens to more than 1.25 when the buffer is enlarged to 2M. At the same time, ProtoCRL records the lowest forgetting, despite using a *single* sampling schedule for all permutations. NoET shows a slightly lower average forgetting with the 100 K buffer, but at the cost of a lower return, indicating it avoids forgetting mainly by learning less. The small standard deviations confirm that each method's behavior is similar across curricula. The results demonstrate that automatically constructed prototype tables make ProtoCRL more robust to task order than manual event-table variants, a desirable property for continual learning agents deployed in uncontrolled settings.

Table 3: Average performance (AvgP) and average forgetting (AvgF) across all 6 task-order permutations, reported for each method with buffer size 100K and 2M. We report the standard deviation across task-order permutations. Bold denotes the best value in a column, underlined the second best.

Method	**100K**		**2M**	
	AvgP	**AvgF**	**AvgP**	**AvgF**
NoET	$\underline{-6.20 \pm 0.84}$	$\mathbf{-0.48 \pm 0.93}$	-5.16 ± 0.41	-0.43 ± 1.00
GoalET	-8.43 ± 0.76	1.49 ± 1.04	$\underline{-3.41 \pm 2.25}$	$\underline{-0.60 \pm 1.39}$
ET	-8.74 ± 0.72	1.41 ± 1.54	-4.98 ± 2.58	-0.03 ± 1.81
ProtoCRL	$\mathbf{-5.91 \pm 0.62}$	$\underline{0.03 \pm 0.63}$	$\mathbf{-2.15 \pm 1.47}$	$\mathbf{-1.24 \pm 1.00}$

5.4 Prototypes Analysis

To shed light on what types of events each prototype, i.e., each VGMM's mixture component, is representing, we inspect which states or transitions predominantly fall into each cluster. To this aim, we took the agent trained on the entire task sequence and made it perform 1 episode in each environment following the learned policy, and 10 episodes by choosing a random action with probability 70% in order to collect more diverse states, thus collecting 2959 states in total.

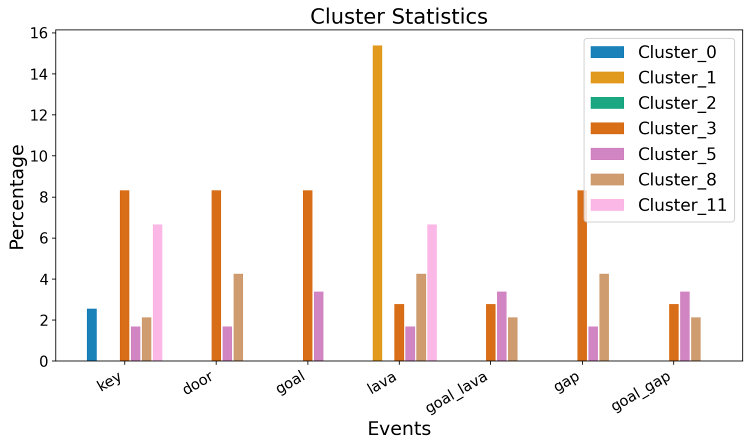


Figure 3: Distribution of environment-defined event types within each discovered prototype cluster. The automatically learned prototypes frequently match to key environment events, demonstrating ProtoCRL's ability to partition states in an event-centric manner.

In Figure 3, we show the percentage of states assigned to each cluster that are related to specific event states. Specifically, for *goal*, *key*, and *lava* we consider all states in which the agent is in the position of the object or in an adjacent position but oriented towards the object. For *door* and *gap*, instead, we consider only states in which the agent is exactly at the door and gap positions. Interestingly, around 25% of the states in Cluster 3 represent *key*, *door*, and *goal* states relevant for the DoorKey task. Additionally, the same cluster has an important coverage of the *gap* state, which is similar to the *door* state, together with smaller percentages of *lava* states and *goal* states for the LavaCrossing and SimpleCrossing tasks. Overall, Cluster 3 appears to be an event cluster, as it captures many state transitions that are important to the agent's task progression. Similarly, around 17% of the states in Cluster 5 seem to be capturing event-related states. Lava-related events, instead, are predominantly collected in Cluster 1, although there was 1 state among the 2959 we collected that was the only one assigned to Cluster 9. Because no cluster stands out with more than 50% of its states representing a particular event, it appears that the VGMM's components might be capturing more nuanced distinctions (e.g., each cluster is partially capturing key states under different local configurations, such as the agent's orientation or position near the key) or sub-trajectories leading to overcoming the obstacles or reaching the goal. This might seem the case by looking at Figure 4, which shows an overlay of the states visited by the agent's policy that are assigned to each cluster. In accordance with our previous analysis, Cluster 3 collects the states corresponding to door opening and the subtrajectory leading to the goal in the DoorKey task. Clusters 5 and 8, instead, seem more focused on subtrajectories that allow overcoming obstacles and reaching the goal in the other two environments. Finally, Cluster 11 collects the state relative to key collection in the DoorKey task. These results confirm that ProtoCRL effectively identifies and groups pivotal states for each environment, supporting more efficient experience replay and removing the need for manually defined event tables.

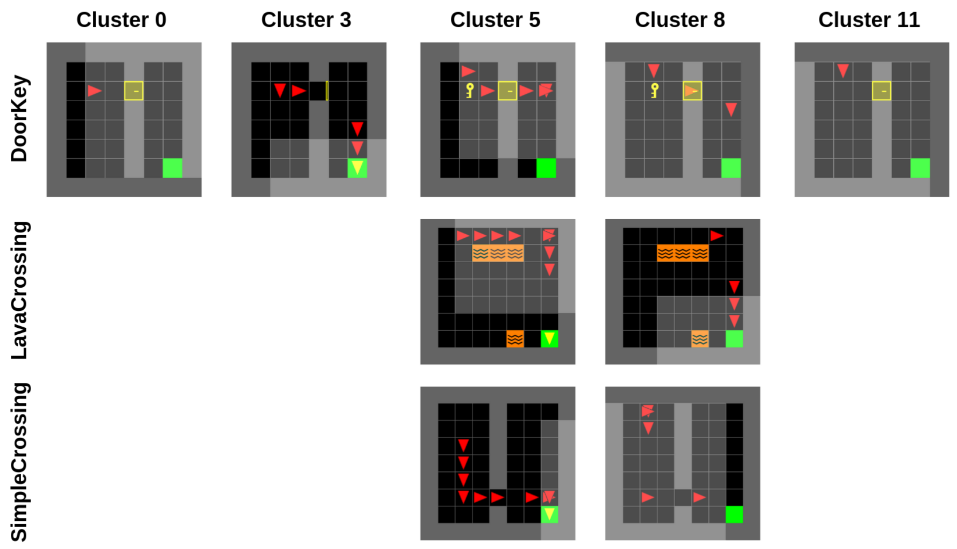


Figure 4: Overlayed states visited by the agent's policy that are assigned to each cluster.

6 Conclusions

In this work, we introduce ProtoCRL, a prototype-based network for CRL that automatically discovers event states via a VGMM. By replacing the manual event definitions with an adaptive, data-driven mechanism for organizing the ERB, ProtoCRL allows the agent to effectively mitigate forgetting. Our experiments demonstrate that under stringent memory constraints and shuffled task sequences, ProtoCRL outperforms conventional baselines that rely on manually defined event tables or uniform sampling. While ProtoCRL shows promising sample efficiency and robustness, some limitations remain. The effectiveness of the VGMM-based clustering can be highly sensitive to hyperparameters and may capture nuanced distinctions across subtrajectories rather than isolating single dominant event states. Additionally, the results ProtoCRL achieves are still lower than state-of-the-art CRL approaches integrating experience replay with world models (Kessler et al., 2023). Future work should explore tighter integration between ProtoCRL and world-model-based approaches. Moreover, investigating adaptive hyperparameter schemes that allow learning other sensitive hyperparameters, such as the sampling probabilities for the automatically built event tables, could further enhance the stability and performance of ProtoCRL in more complex settings.

Acknowledgments

We thank Varun Kompella and Thomas J. Walsh for answering our questions on event tables and for valuable discussions.

References

Caterina Borzillo, Alessio Ragno, and Roberto Capobianco. Understanding deep rl agent decisions: a novel interpretable approach with trainable prototypes. In *XAI.it@AI*IA*, 2023. URL https://api.semanticscholar.org/CorpusID:264491932.

Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. *This looks like that: deep learning for interpretable image recognition*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Adrian Corduneanu and Christopher M Bishop. Variational bayesian model selection for mixture distributions. In *Artificial intelligence and Statistics*, volume 2001, pp. 27–34. Morgan Kaufmann Waltham, MA, 2001.

Fei Deng, Ingook Jang, and Sungjin Ahn. DreamerPro: Reconstruction-free model-based reinforcement learning with prototypical representations. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4956–4975. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/deng22a.html.

Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. ISSN 1364-6613. DOI: https://doi.org/10.1016/S1364-6613(99)01294-2. URL https://www.sciencedirect.com/science/article/pii/S1364661399012942.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *J. Mach. Learn. Res.*, 14(1):1303–1347, may 2013. ISSN 1532-4435.

Patrik O. Hoyer. Non-negative matrix factorization with sparseness constraints. *J. Mach. Learn. Res.*, 5:1457–1469, December 2004. ISSN 1532-4435.

Eoin M. Kenny, Mycal Tucker, and Julie Shah. Towards interpretable deep reinforcement learning with human-friendly prototypes. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hWwY_Jq0xsN.

Samuel Kessler, Jack Parker-Holder, Philip Ball, Stefan Zohren, and Stephen J. Roberts. Same state, different task: Continual reinforcement learning without interference. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):7143–7151, Jun. 2022. DOI: 10.1609/aaai.v36i7.20674. URL https://ojs.aaai.org/index.php/AAAI/article/view/20674.

Samuel Kessler, Mateusz Ostaszewski, MichałPaweł Bortkiewicz, Mateusz Żarski, Maciej Wolczyk, Jack Parker-Holder, Stephen J Roberts, Piotr Mi, et al. The effectiveness of world models for continual reinforcement learning. In *Conference on Lifelong Learning Agents*, pp. 184–204. PMLR, 2023.

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

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521 – 3526, 2016. URL https://api.semanticscholar.org/CorpusID:4704285.

Varun Raj Kompella, Thomas Walsh, Samuel Barrett, Peter R. Wurman, and Peter Stone. Event tables for efficient experience replay. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=XejzjAjKjv.

Chunmao Li, Yang Li, Yinliang Zhao, Peng Peng, and Xupeng Geng. Sler: Self-generated long-term experience replay for continual reinforcement learning. *Applied Intelligence*, 51(1):185–201, 2021.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321, 1992.

Pierre Liotet, Francesco Vidaich, Alberto Maria Metelli, and Marcello Restelli. Lifelong hyper-policy optimization with multiple importance sampling regularization. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 7525–7533. AAAI Press, 2022. URL https://doi.org/10.1609/aaai.v36i7.20717.

Xin Liu, Yaran Chen, Haoran Li, Boyu Li, and Dongbin Zhao. Cross-domain random pre-training with prototypes for reinforcement learning. *arXiv preprint arXiv:2302.05614*, 2023.

Jun Lu. A survey on bayesian inference for gaussian mixture model. *arXiv preprint arXiv:2108.11753*, 2021.

Yongle Luo, Yuxin Wang, Kun Dong, Qiang Zhang, Erkang Cheng, Zhiyong Sun, and Bo Song. Relay hindsight experience replay: Self-guided continual reinforcement learning for sequential object manipulation tasks with sparse rewards. *Neurocomputing*, 557:126620, 2023.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2017. URL https://api.semanticscholar.org/CorpusID:35249701.

Bogdan Mazoure, Ahmed M Ahmed, Patrick MacAlpine, R Devon Hjelm, and Andrey Kolobov. Cross-trajectory representation learning for zero-shot generalization in rl. *International Conference on Learning Representations*, 2022.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24(C):109–165, January 1989. ISSN 0079-7421. DOI: 10.1016/S0079-7421(08)60536-8.

Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, 4, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. URL https://api.semanticscholar.org/CorpusID:205242740.

Nikolaos Nasios and Adrian G Bors. Variational learning for gaussian mixture models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(4):849–862, 2006.

Ronilo Ragodos, Tong Wang, Qihang Lin, and Xun Zhou. Explaining a reinforcement learning agent via prototyping. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=nyBJcnhjAoy.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019a. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019b.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *ArXiv*, abs/1606.04671, 2016. URL https://api.semanticscholar.org/CorpusID:15350923.

Dawid Rymarczyk, Łukasz Struski, Michał Górszczak, Koryna Lewandowska, Jacek Tabor, and Bartosz Zieliński. Interpretable image classification with differentiable prototypes assignment. In *European Conference on Computer Vision*, pp. 351–368. Springer, 2022.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Junjie Wang, Qichao Zhang, Yao Mu, Dong Li, Dongbin Zhao, Yuzheng Zhuang, Ping Luo, Bin Wang, and Jianye Hao. Prototypical context-aware dynamics for generalization in visual control with model-based reinforcement learning. *IEEE Transactions on Industrial Informatics*, pp. 1–11, 2024. DOI: 10.1109/TII.2024.3396525.

Maciej Woł czyk, MichałZając, Razvan Pascanu, Ł ukasz Kuciński, and Piotr Mił oś. Continual world: A robotic benchmark for continual reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 28496–28510. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/ef8446f35513a8d6aa2308357a268a7e-Paper.pdf.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pp. 11920–11931. PMLR, 2021.

Supplementary Materials

The following content was not necessarily subject to peer review.

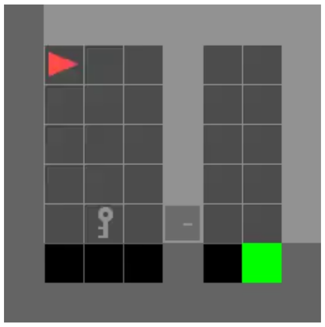
A Experimental Setup

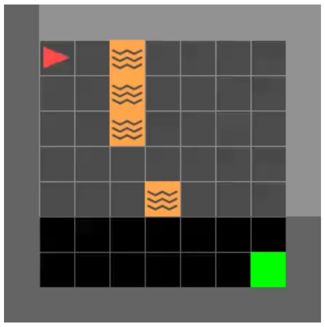
A.1 Environments

In our experiments, we evaluate ProtoCRL on the *DoorKey*, *LavaCrossing*, and *SimpleCrossing* MiniGrid tasks (Chevalier-Boisvert et al., 2018). Figure 5 shows a random instantiations of each environment. The exact positions of lava, gap in the wall, door, key, and the color of the key-door combination are randomly set in each episode. The goal of the agent is to reach the green square starting from the top-left corner of the grid. The agent's action space includes 5 actions: *forward*, *left*, *right*, *pickup key*, and *toggle door*. In each environment, the agent receives an observations consisting of:

- An egocentric 9x9 localized forward-view image (highlighted in Figure 5).
- A boolean flag indicating whether the agent is carrying an object.
- A 2D representation $(category, color)$ of the object it is carrying. The default value is $(-1, -1)$.
- The agent's 3D grid position and orientation (x, y, θ).

The agent gets a reward of -0.05 for each environment step, $+1$ for reaching the goal, and -5 for falling into lava. The episode terminates when the agents reaches the goal, falls into lava, or we have reached the maximum episode length. The latter is set to 2000 while collecting experiences and 200 at evaluation time for all environments. During training, instead, the maximum episode length is set to 500 for *LavaCrossing* and *SimpleCrossing*, and to 1000 for *DoorKey*.


(a) DoorKey task


(b) LavaCrossing task

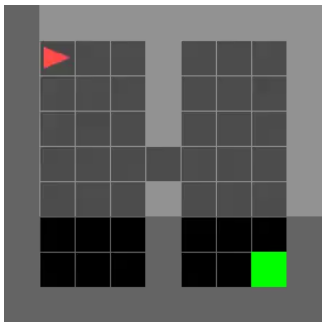

(c) SimpleCrossing task

Figure 5: Random instantiations of the used MiniGrid environments.

A.2 Learning Parameters

Tables 4, 5, 7, and 6 list the parameters used in our experiments. Event conditions check whether the agent has reached the goal ($done$), has picked up a key ($pickup\_key$), has opened the door (at_door), has fallen into lava (at_lava), or has traversed the gap in the wall (at_gap). In order to produce results for ContinualDreamer (Kessler et al., 2023), we have used the author's original implementation. The only changes we made concerned the reward function, updated to match the one described in Appendix A.1, the number of environment interactions and the evaluation frequency.

The values reported in Tables 4–6 are the best performers from a small, systematic grid search that we applied to every agent variant (NoET, GoalET, ET and ProtoCRL). Our search started from the ranges recommended in Kompella et al. (2023) and explored the following settings:

- **Value-function network**: {[128,128], [64,64], [128,128,128], [128,128,64]}
- **Learning rate**: {0.01, 0.005, 0.001, 0.0005}
- **Batch size** {32, 64}
- **Epsilon-greedy** (ϵ): {0.3, 0.4}
- **Phase-1 steps ratio** (ProtoCRL only): {0.5, 0.7, 0.8}
- **# VGMM components** (ProtoCRL only): $K \in \{3, 4, 6, 8, 10, 12, 15\}$
- **VGMM warm-start epochs** (ProtoCRL only): {0, 1, 5, 10}

For ProtoCRL, the additional loss coefficients $(\alpha_{\text{logp}},\ \alpha_{\text{KL}},\ \beta_E,\ \beta_H)$ were set once per experiment by matching the typical magnitude of each term to that of the TD loss during a short pilot run; they were not tuned further.

Table 4: Learning parameters for experiments with no event tables (NoET).

Parameter	**Value**
Number of tasks (T)	3
Value function networks	3 hidden layers of 128, 128, and 64 ReLU units
Learning rate	0.0005
Batch size	64
Epsilon-greedy (ϵ)	0.3
Stale network refresh rate	0.01
Discount factor	0.99

Table 5: Learning parameters for experiments with just goal event tables (GoalET).

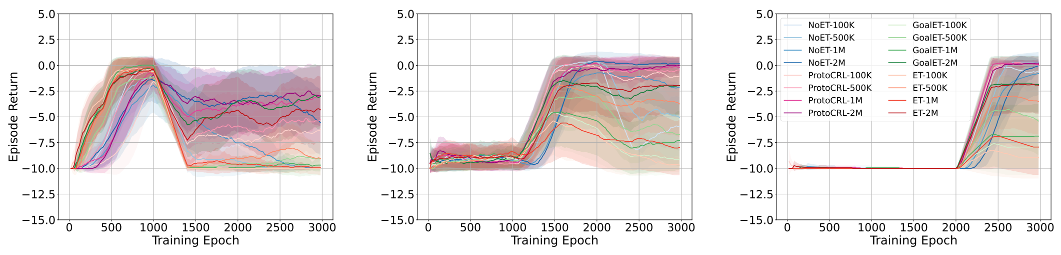
Parameter	**Value**
Number of tasks (T)	3
Event conditions (ω_i)	$done_t$ with $t = [0, T]$
Event history length (τ)	50
Event sampling probabilities (η_i)	$\mathcal{B}_0^0$: 0.3529, $\mathcal{B}_1^0$: 0.1765, $\mathcal{B}_2^0$: 0.2941, $done_t$: 0.0588 with $t = [0, T]$
Value function networks	3 hidden layers of 128, 128, and 64 ReLU units
Learning rate	0.0005
Batch size	64
Epsilon-greedy (ϵ)	0.3
Stale network refresh rate	0.01
Discount factor	0.99

Table 6: Learning parameters for experiments with multiple event tables per task (ET).

Parameter	**Value**
Number of tasks (T)	3
Event conditions (ω_i)	$done_t$ with $t = [0, T]$, $pickup\_key$, at_door, at_lava
Event history length (τ)	50
Event sampling probabilities (η_i)	$\mathcal{B}_0^0$: 0.34, $done_0$: 0.03, at_door : 0.07, $pickup\_key$: 0.03, $\mathcal{B}_1^0$: 0.16, $done_1$: 0.02, at_lava : 0.05, $\mathcal{B}_2^0$: 0.23, $done_2$: 0.02, at_gap : 0.05
Value function networks	3 hidden layers of 128, 128, and 64 ReLU units
Learning rate	0.0005
Batch size	64
Epsilon-greedy (ϵ)	0.3
Stale network refresh rate	0.01
Discount factor	0.99

Table 7: Learning parameters for experiments with ProtoCRL.

Parameter	**Value**
Number of tasks (T)	3
Number of VGMM components (K)	12
Event conditions (ω_i)	$\mathbf{1}\left\{ k = \arg\max_{j \in \{1,\ldots,K\}} p(j \mid f_e(s)) \right\}$
Event history length (τ)	500
Event sampling probabilities (η_i)	$\mathcal{B}_0^0$: 0.6, $\mathcal{B}_1^0$: 0.3, $\mathcal{B}_2^0$: 0.6, $prototype_{k,t}$: 0.1 with $k \in [0, K-1]$, $t \in [1, T]$
Value function networks	3 hidden layers of 128, 128, and 64 LeakyReLU units
Phase 1 ($\mathcal{L}_{TD}$) learning rate	0.001
Phase 1 steps ratio	0.5
Phase 2 ($\mathcal{L}_{ProtoCRL}$) learning rate	0.001
Phase 2 regularization parameter (λ)	0.1
VGMM warm start epochs	10
α_{logp}	0.1
α_{KL}	0.001
β_E	1000
β_H	1.0
Batch size	64
Epsilon-greedy (ϵ)	0.3
Stale network refresh rate	0.01
Discount factor	0.99



(a) DoorKey task. (b) LavaCrossing task. (c) SimpleCrossing task.

Figure 6: Episode return on 3 MiniGrid tasks using decreasing memory budget. The task ordering is *DoorKey-LavaCrossing-SimpleCrossing*. Each agent is trained on each task for 1K epochs and evaluated every 20 epochs across the entire training sequence.

B Continual Reinforcement Learning Metrics

In order to evaluate the CRL performance of the trained agents, we use two popular metrics: the average performance and the average forgetting.

The average performance metric estimates the average performance of the agent on all tasks at the end of the task sequence and is computed as:

$$p(t_f) = \frac{1}{T}\sum_{i=1}^{T} p_i(t_f)$$

where t_f is the final timestep and $p_i(t_f)$ is the performance of the agent on the i-th task at timestep t_f.

The average forgetting, instead, is measured as the average difference between the performance of each task after its own training and at the end of the task sequence:

$$F = \frac{1}{T}\sum_{i=1}^{T} F_i \text{ with } F_i = p_i(i \times N) - p_i(t_f)$$

where N is the number of steps per task. The forgetting for the last task in the sequence is $F_T = 0$. Finally, forgetting can also take negative values if the performance of a task i is higher at the end of the task sequence compared to after task i training.

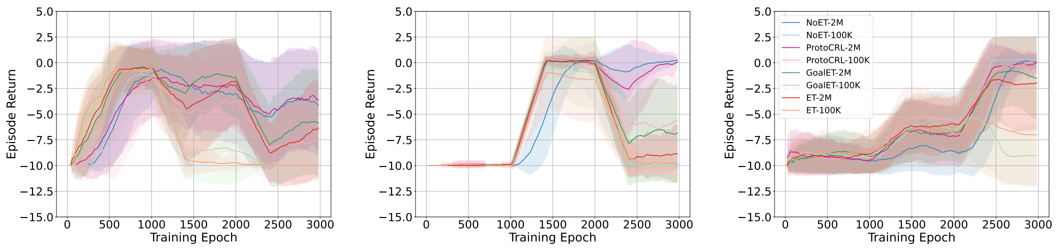
C Episodic Return with Decreasing Buffer Capacity

Figure 6 shows the episodic return for each MiniGrid task in the tested sequence (DoorKey, LavaCrossing, SimpleCrossing) obtained by ProtoCRL and the NoET, GoalET, and ET baselines. All baselines experience severe performance drop after encountering the second task at epoch 1K, with NoET-100K/500K/1M, GoalET-100K/1M, and ET-100K/1M obtaining the lowest return of -10 at the end of the task sequence. On the other hand, ProtoCRL maintains a performance that is higher than all baselines with memory budget up to 1M transitions. Similar observations can be made for the performance on subsequent tasks, in which ProtoCRL obtains episodic returns that are slightly lower than 0 for the LavaCrossing task and slightly above 0 (highest achievable) for the SimpleCrossing task.

Table 8 reports the average forgetting achieved by ProtoCRL and the baselines under different memory budgets. ProtoCRL is the approach experiencing the lowest forgetting with stricter memory constraints.

Table 8: Average forgetting over 3 MiniGrid tasks with different memory budgets. All metrics are an average and standard deviation over 10 seeds. We highlight in bold the best performing approaches.

	100K	**500K**	**1M**	**2M**
NoET	4.63 ± 3.43	4.76 ± 3.73	3.16 ± 3.63	2.32 ± 3.28
GoalET	3.40 ± 4.11	3.78 ± 3.40	3.65 ± 4.69	$\mathbf{-0.02 \pm 0.02}$
ET	4.44 ± 4.27	2.71 ± 3.14	3.36 ± 4.18	1.78 ± 2.35
ProtoCRL	$\mathbf{1.26 \pm 2.72}$	$\mathbf{2.08 \pm 1.62}$	$\mathbf{0.46 \pm 0.62}$	0.03 ± 1.72



(a) DoorKey task. (b) SimpleCrossing task. (c) LavaCrossing task.

Figure 7: Episode return on *DoorKey-SimpleCrossing-LavaCrossing* task sequence using memory buffers with size 2M and 100K. We report the average and standard deviation across 10 seeds. Each agent is trained on each task for 1K epochs and evaluated every 20 epochs across the entire training sequence.

D Ablation Study: Robustness to Task Order

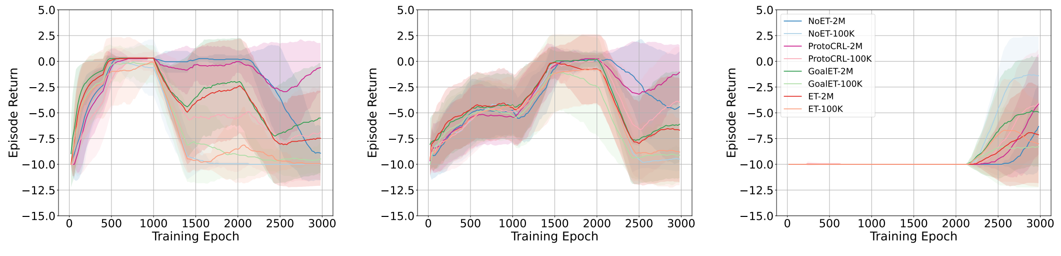
Table 9 details the average performance (AvgP) and average forgetting (AvgF) scores for each of the five remaining permutations of the SimpleCrossing (SC), LavaCrossing (LC) and DoorKey (DK) tasks. Results (means and standard deviations over 10 seeds) are reported for both the tight 100K buffer and the larger 2M buffer.

ProtoCRL delivers the highest AvgP in four of the five permutations when memory is scarce and remains either best or a close second with the 2M buffer. Additionally, ProtoCRL consistently keeps forgetting low, or even negative, while maintaining its AvgP advantage. NoET occasionally records the numerically smallest forgetting value, but this always coincides with a lower AvgP (except for LCßDKßSC, 100K), indicating that it "forgets less" largely because it never learns the later tasks as well. GoalET and ET frequently exhibit positive forgetting, indicating that the manual event tables needs much more careful tuning to work with each task order.

Overall, these permutation-specific findings, together episodic return curves for every permutation (see Figures 7–11), reinforce the aggregate analysis in Section 5.3: automatically constructed prototype event tables allow ProtoCRL to sustain high return regardless of task order, even with low memory budgets, a property that hand-crafted event tables fail to guarantee.

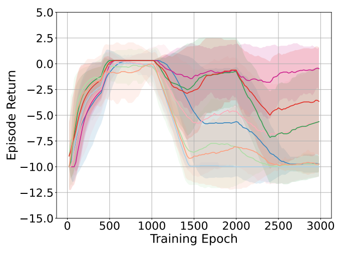
Table 9: Average performance (AvgP) and average forgetting (AvgF) on the remaining task-order permutations of the three MiniGrid tasks: SimpleCrossing (SC), LavaCrossing (LC), and DoorKey (DK). Values are means and standard deviations over 10 seeds. Values in bold identify the best values for AvgP and AvgF, while the underlined values are the second best.

Task order	**Method**	**100K**		**2M**	
		AvgP	**AvgF**	**AvgP**	**AvgF**
DK→SC→LC	NoET	$\underline{-6.55 \pm 4.89}$	$\underline{1.60 \pm 6.48}$	$\mathbf{-2.07 \pm 3.42}$	$\underline{-0.76 \pm 6.77}$
	GoalET	-9.31 ± 0.49	3.36 ± 4.14	-4.15 ± 2.13	$\mathbf{-1.12 \pm 3.90}$
	ET	-8.96 ± 1.47	3.04 ± 4.37	-5.54 ± 2.97	0.46 ± 4.50
	ProtoCRL	$\mathbf{-5.14 \pm 3.09}$	$\mathbf{0.20 \pm 4.74}$	$\underline{-2.08 \pm 2.72}$	-0.39 ± 4.65
SC→LC→DK	NoET	$\underline{-6.79 \pm 4.22}$	$\mathbf{0.24 \pm 7.78}$	$\underline{-5.22 \pm 2.91}$	$\underline{-0.99 \pm 5.47}$
	GoalET	-8.62 ± 0.48	1.73 ± 3.99	-5.54 ± 0.49	0.72 ± 4.24
	ET	-8.97 ± 0.83	2.08 ± 4.46	-7.59 ± 0.50	1.39 ± 3.52
	ProtoCRL	$\mathbf{-5.88 \pm 0.83}$	$\underline{0.36 \pm 4.63}$	$\mathbf{-1.07 \pm 1.30}$	$\mathbf{-2.38 \pm 3.40}$
SC→DK→LC	NoET	$\underline{-6.54 \pm 4.89}$	$\mathbf{-2.97 \pm 4.20}$	-5.87 ± 3.38	$\underline{-3.33 \pm 2.46}$
	GoalET	-9.66 ± 0.49	0.47 ± 0.66	-6.55 ± 2.44	-0.92 ± 1.30
	ET	-8.27 ± 1.76	-1.38 ± 1.96	$\underline{-5.86 \pm 2.24}$	-1.78 ± 1.83
	ProtoCRL	$\mathbf{-5.03 \pm 1.45}$	$\underline{-2.47 \pm 1.77}$	$\mathbf{-1.24 \pm 0.75}$	$\mathbf{-4.88 \pm 3.46}$
LC→DK→SC	NoET	$\mathbf{-5.24 \pm 3.49}$	$\mathbf{-1.53 \pm 2.17}$	-5.53 ± 2.11	-1.03 ± 2.23
	GoalET	-7.59 ± 1.75	$\mathbf{0.35 \pm 0.49}$	$\mathbf{-1.05 \pm 1.96}$	$\underline{-2.06 \pm 2.91}$
	ET	-7.93 ± 1.47	-0.35 ± 0.49	$\underline{-2.43 \pm 0.99}$	$\mathbf{-2.07 \pm 2.91}$
	ProtoCRL	$\underline{-6.55 \pm 2.44}$	$\underline{-0.69 \pm 0.98}$	-3.82 ± 2.22	-0.50 ± 1.91
LC→SC→DK	NoET	$\underline{-6.42 \pm 4.75}$	$\mathbf{-2.24 \pm 5.87}$	-9.16 ± 0.88	2.27 ± 4.73
	GoalET	-9.32 ± 0.48	$\underline{2.08 \pm 3.68}$	$\underline{-3.12 \pm 0.50}$	$\mathbf{-1.37 \pm 4.32}$
	ET	-10.00 ± 0.00	2.42 ± 3.42	-3.49 ± 0.95	$\underline{-1.33 \pm 4.27}$
	ProtoCRL	$\mathbf{-4.84 \pm 0.01}$	$\underline{0.01 \pm 4.22}$	$\mathbf{-2.70 \pm 2.74}$	-0.78 ± 1.93

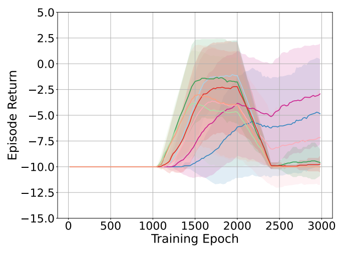


(a) SimpleCrossing task. (b) LavaCrossing task. (c) DoorKey task.

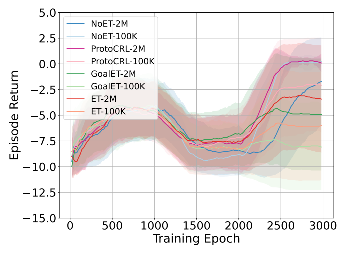
Figure 8: Episode return on *SimpleCrossing-LavaCrossing-DoorKey* task sequence using memory buffers with size 2M and 100K. We report the average and standard deviation across 10 seeds. Each agent is trained on each task for 1K epochs and evaluated every 20 epochs across the entire training sequence.



(a) SimpleCrossing task.

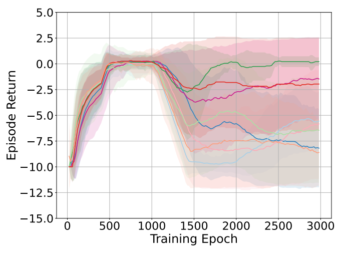


(b) DoorKey task.

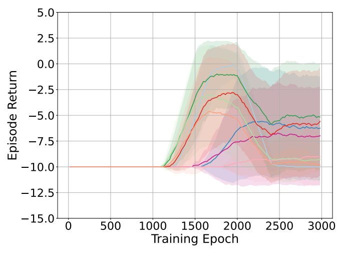


(c) LavaCrossing task.

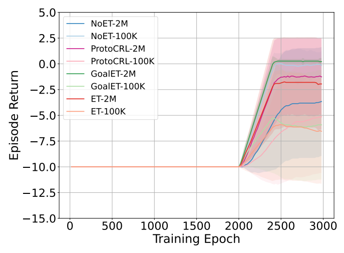
Figure 9: Episode return on *SimpleCrossing*-*DoorKey*-*LavaCrossing* task sequence using memory buffers with size 2M and 100K. We report the average and standard deviation across 10 seeds. Each agent is trained on each task for 1K epochs and evaluated every 20 epochs across the entire training sequence.



(a) LavaCrossing task.

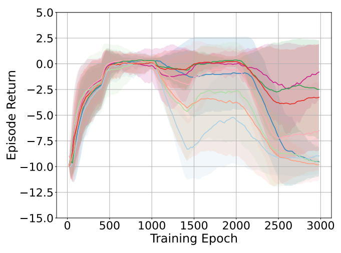


(b) DoorKey task.

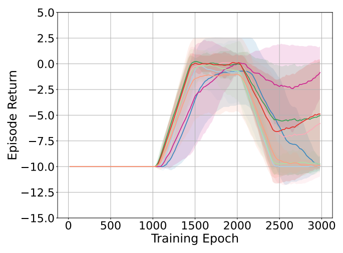


(c) SimpleCrossing task.

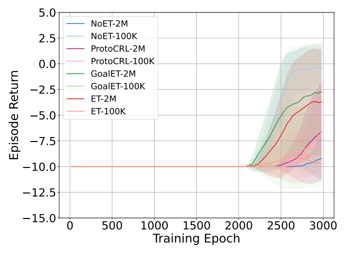
Figure 10: Episode return on *LavaCrossing*-*DoorKey*-*SimpleCrossing* task sequence using memory buffers with size 2M and 100K. We report the average and standard deviation across 10 seeds. Each agent is trained on each task for 1K epochs and evaluated every 20 epochs across the entire training sequence.



(a) LavaCrossing task.



(b) SimpleCrossing task.



(c) DoorKey task.

Figure 11: Episode return on *LavaCrossing*-*SimpleCrossing*-*DoorKey* task sequence using memory buffers with size 2M and 100K. We report the average and standard deviation across 10 seeds. Each agent is trained on each task for 1K epochs and evaluated every 20 epochs across the entire training sequence.