MODEL-FREE GENERATIVE REPLAY FOR LIFELONG REINFORCEMENT LEARNING: APPLICATION TO STARCRAFT-2

Zachary Daniels, Aswin Raghavan, Jesse Hostetler, Abrar Rahman, Indranil Sur
Michael Piacentino, **Ajay Divakaran**
SRI International, Princeton, NJ, USA
`zachary.daniel@sri.com, aswin.raghavan@sri.com, jesse.hostetler@sri.com`
`abrar.rahman@sri.com, indranil.sur@sri.com`
`michael.piacentino@sri.com, ajay.divakaran@sri.com`

ABSTRACT

One approach to meet the challenges of deep lifelong reinforcement learning (LRL) is careful management of the agent's learning experiences, in order to learn (without forgetting) and build internal meta-models (of the tasks, environments, agents, and world). Generative replay (GR) is a biologically-inspired replay mechanism that augments learning experiences with self-labelled examples drawn from an internal generative model that is updated over time. We present a version of GR for LRL that satisfies two desiderata: (a) Introspective density modelling of the latent representations of policies learned using deep RL, and (b) Model-free end-to-end learning. In this paper, we study three deep learning architectures for model-free GR, starting from a naïve GR and adding ingredients to achieve (a) and (b). We evaluate our proposed algorithms on three different scenarios comprising tasks from the *Starcraft 2* and Minigrid domains. We report several key findings showing the impact of the design choices on quantitative metrics that include transfer learning, generalization to unseen tasks, fast adaptation after task change, performance comparable to a task expert, and minimizing catastrophic forgetting. We observe that our GR prevents drift in the features-to-action mapping from the latent vector space of a deep RL agent. We also show improvements in established lifelong learning metrics. We find that a small random replay buffer significantly increases the stability of training when combined with the experience replay buffer and the generated replay buffer. Overall, we find that "hidden replay" (a well-known architecture for class-incremental classification) is the most promising approach that pushes the state-of-the-art in GR for LRL, and observe that the architecture of the sleep model might be more important for improving performance than the types of replay used. Our experiments required only 6% of training samples to achieve 80-90% of expert performance in most *Starcraft 2* scenarios.

1 INTRODUCTION

Lifelong Reinforcement Learning (LRL) involves training an agent to maximize its cumulative performance on a stream of changing *tasks* over a long lifetime. LRL agents must balance *plasticity vs. stability*: learning the current task while maintaining performance on previous tasks. Deep neural networks are especially prone to instability, often exhibiting *catastrophic forgetting* of previous tasks when trained on multiple tasks that are presented sequentially (Kirkpatrick et al., 2016). One approach to meet the challenges of deep LRL is by careful managing the agent's learning experiences, in order to learn (without forgetting) and build internal meta-models (of the tasks, environments, agents, and world). One strategy for managing experiences is to recall data from previous tasks and mix it with data from the current task when training, essentially transforming sequential LRL into batch multi-task reinforcement learning (RL) (Brunskill & Li, 2013). The technique of *experience replay* of past data is common in deep RL (e.g. Mnih et al., 2013) and has been studied in the lifelong learning literature (Isele & Cosgun, 2018; Rolnick et al., 2019; Hayes et al., 2021). However, storing sufficient examples from all previous tasks may be unreasonable in systems with limited resources or long lifetimes. Rather than store an ever-growing list of experiences, one can instead employ *generative replay* (GR) (Shin et al., 2017; van de Ven & Tolias, 2018), a biologically-inspired replay mechanism that augments learning experiences with self-labelled examples drawn from an internal generative model trained to approximate the distribution of past experiences.

In LRL (see Khetarpal et al. (2020) for a survey), each task is a Markov decision process (MDP), and the agent must learn a policy for interacting with its environment to maximize its lifetime performance (e.g., average reward over all tasks). While GR has been shown to be effective in lifelong supervised learning (van de Ven & Tolias, 2018), it has been less explored in LRL. The LRL setting is complicated by the fact that the agent's experiences are observation-

action trajectories whose distribution depends on the environment dynamics and the policy being executed. While prior work with world models (Ketz et al., 2019a) and pseudo-rehearsal (Atkinson et al., 2021) focused on generative replay of (partial) trajectories, this requires modeling the dynamics, which is difficult in complex environments.

In this paper, we present agent architectures for LRL that use GR for memory consolidation and satisfy two desiderata: (a) Introspective density modelling of the latent representations of policies learned using deep RL, and (b) Model-free end-to-end learning. The first property avoids the challenges of density modelling of complex high-dimensional perceptual inputs, whereas policy learning using deep RL works well with such perceptual inputs. The second property avoids the challenges of learning temporal dynamics and reward functions from few learning experiences with sparse rewards. Our contributions extend powerful model-free deep RL and do not learn the dynamics explicitly.

Our first contribution is a model-free approach to generative replay (Section 3) in lifelong RL. We empirically demonstrate that replay of independently sampled observation-action pairs is sufficient to mitigate forgetting. This is noteworthy because observation-action pairs are simpler to generate than trajectories, and because replay of observation-action pairs is directly analogous to replay of input-label pairs in supervised learning, thus unifying generative replay methods for lifelong learning.

Our second contribution is a mechanism for memory consolidation via wake-sleep cycles. During wake phases, our agent improves a policy for the current task(s) using an off-the-shelf RL algorithm while storing samples of experiences in a buffer. Periodically, the agent enters a sleep phase, during which the generative memory is trained to model the agent's new experiences, and the procedural knowledge embodied in the policy learned during the wake phase is consolidated into the agent's "skill set" via policy distillation (Rusu et al., 2015) with GR. In the next wake phase, a new exploration policy is seeded with knowledge from previously-learned skills, and the cycle repeats. We investigate three different replay architectures (Section 3) within the wake-sleep framework, starting from a naïve generative and adding ingredients to achieve the aforementioned introspective density modelling of the latent representations and model-free end-to-end learning.

We evaluate our algorithm and architectural contributions through experiments (Section 4) on three different scenarios comprising tasks from the *Starcraft 2* (SC-2) and Minigrid domains. We report several key findings showing the impact of the design choices on quantitative metrics that include transfer learning, generalization to unseen tasks, fast adaptation after task change, performance compared to task experts, and catastrophic forgetting. We observe that our GR prevents drift in the features-to-action mapping of a deep RL agent (that learns the features from observations). We find that hidden replay (replay of such features) outperforms other replay mechanisms, which agrees with past results in lifelong supervised learning (van de Ven et al., 2020). Since deep RL in SC-2 is computationally demanding, we release a dataset of trajectories of trained single-task expert policies for future research [1].

2 BACKGROUND

2.1 LIFELONG REINFORCEMENT LEARNING (LRL) SETUP

A lifelong RL syllabus is a sequence of tasks that the learning agent experiences one at a time for a fixed number of interactions. Each task is assumed to be formulated as a Partially Observable Markov Decision Processes (POMDPs) (Sutton & Barto, 2018), where each POMDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P(s'|s, a), R(s, a), \mathcal{O}, Z(o|s))$ consisting of set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition probability function between states $P(s'|s, a)$, a reward function $R(s, a)$, a set of observations $\mathcal{O}$, and conditional observation probability function $Z(o|s)$.

We assume that all tasks have distinct observation spaces of the same dimensionality, share a common action space (e.g., point-and-click, issue movement or attack commands), but differ in the transition and reward dynamics. For example, this setup can capture the use case of an embodied agent using the same set of sensors and actuators throughout their lifetime but with different reward functions or objectives.

In single-task RL, the objective is to learn a policy that maps the history of observations to actions, $\pi : \mathcal{O}^h \rightarrow \mathcal{A}$, such that the policy maximizes the expected discounted future reward, given by the value function $V^\pi(s) = E_{S_t \sim P, A_t \sim \pi}[\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, A_t)]$. We will refer to the task-optimal policy as a single task expert (STE).

Lifelong Learning involves training an agent to continually learn from a stream of tasks over long lifetimes. As the agent encounters new tasks, it must accumulate and leverage knowledge to learn novel tasks faster (positive forward transfer), while maintaining performance on previous tasks (minimizing catastrophic forgetting or negative backwards transfer). In this paper, we consider the task of lifelong reinforcement learning (LRL), which has been less explored than class-incremental classification Hayes et al. (2021) in the literature.

---

[1] https://github.com/sri-l2m/l2m_data

The objective in LRL is to learn a policy that maximizes average cumulative reward over all POMDPs in a task set while learning from limited interactions with a sequence of POMDPs drawn from the task set. This is challenging because the learning problem is non-stationary and the learner needs to accommodate an increasing number of tasks in a model with fixed capacity. The task changes after an unknown number of interactions with the environment. The agent has no control over the choice of the next task. In our setting the agent has no knowledge of the identity of the current task, or the transition points between tasks.

2.2 GENERATIVE REPLAY (GR)

Our work is focused on generative replay (GR), which can be broadly described as a form of data augmentation where real-world experiences are used to continually train a generative model over the input space of the problem. Subsequently, data sampled from this generator can be pseudo-labeled or self-labeled to generate paired training examples that may reflect previously trained tasks. Different types of replay were discussed by Hayes et al. (2021) along with its role in the brain and in current deep learning implementations. Most of the prior work on generative replay has focused on the problem of class-incremental classification van de Ven & Tolias (2019); Shin et al. (2017), or class-incremental generative modelling Cong et al. (2020) as a subroutine thereof, but there has been significantly less exploration of GR in LRL.

We use a Variational Auto-Encoder (VAE) (Kingma & Welling, 2014) as the generative model. The VAE consists of an encoder $q_\phi(z|o)$ mapping observations to a vector-valued latent space z, and a decoder $p_\theta(o|z)$ that is trained to reconstruct the original input. We use the Gaussian prior $z \sim N(0, \mathbf{I})$ with 0 mean and unit variance $\mathbf{I}$. To train the VAE, we minimize the standard loss $\mathcal{L}_{\text{VAE}}$,

$$\min E_{q_\phi(z|o)}[\ln p_\theta(o|z)] - D_{KL}[q_\phi(z|o)||p(z)] \tag{1}$$

World models (Ketz et al., 2019a) use model-based generative replay for LRL i.e., the replay generates rollouts or entire trajectories (e.g. Ketz et al., 2019b). Model-based replay works well in many domains, but it is challenging for domains such as SC-2 due to the complex dynamics and high dimensional observation and action spaces. The most relevant prior work is that of Atkinson et al. (Atkinson et al., 2021) who explored model-free generative replay for Deep Q-Learning (DQN). However, Atkinson et al. (2021) uses GANs in the raw perceptual space of the agent (aka veridical replay Hayes et al. (2021)). Others have explored sophisticated strategies for prioritized experience replay in DQN (e.g. Isele & Cosgun, 2018; Riemer et al., 2019; Wang et al., 2021; Caccia et al., 2020; Zhang & Sutton, 2017; Schaul et al., 2015) in LRL, but these are not generative in nature. We show that naïvely migrating the generative modelling (similar to Atkinson et al.) to the feature space of the policy causes instabilities and severe catastrophic forgetting. To the best of our knowledge, this is the first work on model-free GR for LRL in the feature space of generic deep RL agents.

3 HIDDEN REPLAY FOR LIFELONG RL

This section introduces our core contribution of wake-sleep training combined with model-free GR of latent representations. We also explore two other GR architectures along the way. The fundamental tradeoff in lifelong learning is between plasticity, i.e, learning the current task, and stability, i.e, remembering past tasks. We address this tradeoff in lifelong RL by employing an approach that alternates between two distinct phases (Raghavan et al., 2020). In the wake phase, a plastic wake policy π_w is optimized for the current task by interaction with the environment and using any off-the-shelf RL method. Transition tuples are collected during training and stored in a buffer (see Alg. 1); each tuple contains (o, r, a), the observation o, reward for the previous action r, and the policy output a (e.g., the policy logits or one-hot encoded action). The sleep policy π_s provides "advice" to explore the current task using the consolidated policy learned from all previous tasks. Wake phase actions are selected according to a mixture of π_w and π_s, the probability of choosing an action from π_s decaying to 0 over time. We use an off-policy RL algorithm such as VTrace (Espeholt et al., 2018) to accommodate this off-policy action selection in the optimization of π_w.

In the sleep phase, the stable sleep policy π_s is optimized with the objective of incorporating new knowledge (the action selection in the wake buffer) while retaining current knowledge. Similar to generative replay in supervised learning, augmented dataset(s) are created by combining wake transitions with tuples from the generative model g_s and pseudo-labelled by the sleep policy. This form of replay is model-free because it does not learn the POMDP dynamics and reward and does not generate and replay temporal trajectories. The sleep policy and the generative model are jointly trained. The sleep policy is trained to minimize $\mathcal{L}_{xent}$, a standard cross-entropy loss between the outputs (logits) of π_s, treating the replay augmented dataset as the ground truth. The generative model is trained using the combined dataset and the standard VAE loss (Eq. 1). The description so far is easily applied to veridical replay, i.e, when generative modelling and replay occurs in the original observation space of the MDP, but this is undesirable as discussed before. The design choice of whether the input or hidden space is replayed distinguishes the model implementations (Alg. 1 vs Alg. 2, and models in Fig. 1) from each other.
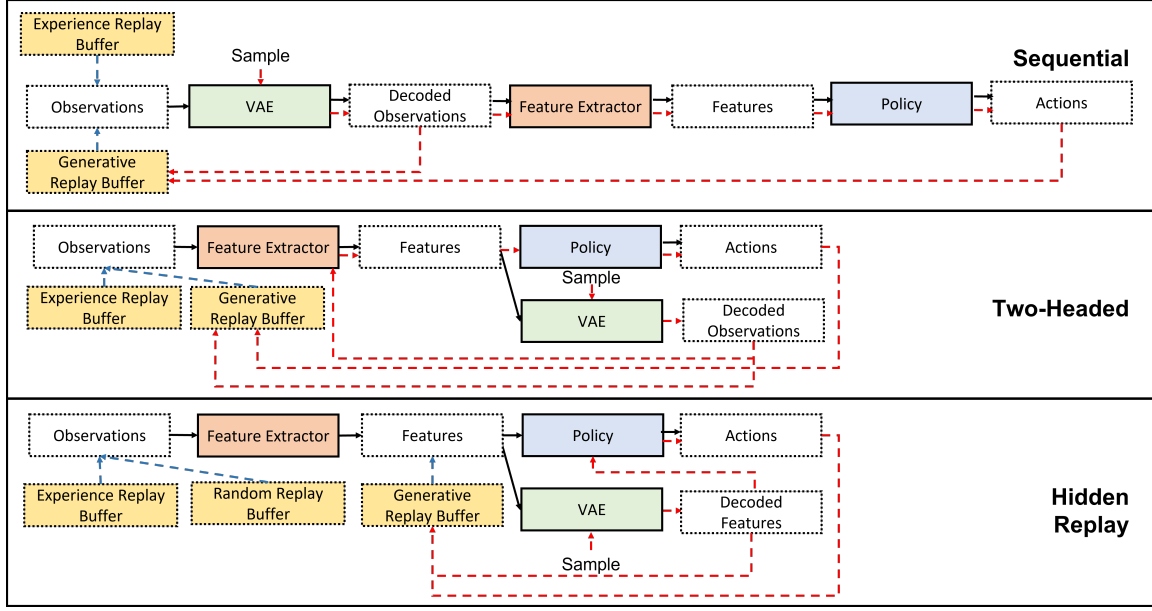
Figure 1: We investigate three replay architectures for model-free generative replay (sleep phase). The red dashed arrows denotes sampling steps starting from the VAE to generate labelled actions. The blue dashed arrows show how the various replay buffers are sampled and fed into the sleep model during training. The black arrows denote the forward pass. **Top:** Sequential Replay Architecture; **Middle:** Two-Headed Replay Architecture; **Bottom:** Hidden Replay Architecture.

Algorithm 1 Wake-Sleep Training in the Sequential and Two-Headed models.

1: **Iterates:** Generator g_s^t, sleep policy π_s^t, wake policy π_w^t, wake buffer b_w^t
2: **for** $t = 1, 2, \ldots$ **do**
3:     **for** K times **do** {Wake Phase}
4:         Sample observation o and reward r from current task
5:         Sample $a \sim \pi_s^t(o)$ w.p. p_{advice}, else $a \sim \pi_w^t(o)$
6:         Add transition (o, r, a) to FIFO wake buffer b_w^t
7:         Update wake policy π_w^t on current task reward using b_w^t
8:         Decay p_{advice}
9:     **end for**
10:    **for** N iterations **do** {Sleep Phase}
11:       Sample batch B from b_w^t (c.f. Line 6).
12:       Sample batches $O_s^t \sim g_s^t$
13:       Pseudo-label $A_s^t = \pi_s^t(O_s^t)$
14:       (GR Buffer) $D = B \cup (O_s^t, A_s^t)$
15:       Minimize $\mathcal{L}_{VAE}(g_s^{t+1}) + \alpha \mathcal{L}_{xent}(g_s^{t+1}, \pi_s^{t+1})$ on D
16:    **end for**
17: **end for**

Algorithm 2 Wake-Sleep Training in the Hidden Replay model.

1: **Iterates:** Generator g_s^t, sleep policy π_s^t, wake policy π_w^t, wake buffer b_w^t, random replay buffer b_r^t
2: **for** $t = 1, 2, \ldots$ **do**
3:     **for** K times **do** {Wake Phase}
4:         Sample observation o and reward r from current task
5:         Sample $a \sim \pi_s^t(o)$ w.p. p_{advice}, else $a \sim \pi_w^t(o)$
6:         Add transition (o, r, a) to wake buffer b_w^t
7:         Update wake policy π_w^t on current task reward using b_w^t
8:         Decay p_{advice}
9:     **end for**
10:    **for** N iterations **do** {Sleep Phase}
11:       If ER: Batch $B \sim b_w^t$ (c.f. Line 6).
12:       If GR: Feature batches $H_s^t \sim g_s^t$
13:       If GR: Pseudo-label $A_s^t \sim \pi_s^t(H_s^t)$
14:       If RaR: $b_r^t = b_r^t \bigcup b_w^t.sample(K)$
15:       If RaR: $D_r \sim b_r^t$
16:       Minimize $\mathcal{L}_{VAE}(g_s^{t+1}) + \alpha \mathcal{L}_{xent}(g_s^{t+1}, \pi_s^{t+1})$ on (H_s^t, A_s^t) and end-to-end minimize (including feature extractor) the same loss on observation-action pairs in $B \bigcup D_r$
17:    **end for**
18: **end for**

We compare three replay architectures that we denote Sequential, Two-Headed, and Hidden Replay (shown in Fig. 1). Each architecture is a specific arrangement of the following three components: a feature extractor, a VAE, and

a policy network. The feature extractor maps raw image-like observations to feature vectors using a convolutional neural network (CNN) or a multi-layered perceptron. The policy network is dependent on the action space, e.g., in our experiments we utilize a convolutional architecture to output spatial actions.

In the **Sequential** architecture (Fig. 1, Top), the VAE is upstream of the policy network, and the sleep policy π_s operates on the VAE reconstruction rather than the true observation. This architecture is similar to the one proposed in Atkinson et al. (2021). During sleep (following the red dashes in Fig. 1, Top), sampled latent vectors are decoded to observations that are then labelled by the policy. We hypothesize that the sequential architecture tends to map observations for which the policy chooses similar actions to generic reconstructions. We found that this method produces blurry reconstructions for image-like observations.

The **Two-Headed** architecture (Fig. 1, Middle) decouples the decoder/generator from the policy network. Each observation (real or generated) is passed through the feature extractor. The features are then separately passed through the VAE and the policy network. Our experiments showed improved visual reconstructions (see Appendix A.15) due to separation of the influence of the policy on the VAE decoder. However, sampling in this architecture requires two forward passes as shown in the figure (once to decode an observation and once to label it).

Alg. 1 shows the data flow that is common between these two architectures that perform veridical replay, i.e, the VAE posterior is a density over raw and complex perceptual space. The training is wasteful because the wake policy π_w has already learned an effective feature extractor, and latent-to-action partition that works well for the current task (solved in wake phase). The sleep phase in veridical replay does not utilize these learned features and tries to re-learn them, whereas deep RL (as in wake phase) is effective at learning representations to map raw percepts to actions.

The final architecture we consider is inspired by the success of **Hidden Replay** in class incremental learning (van de Ven et al., 2020) and its connection to biology. This architecture (Fig. 1, Bottom) generates feature vectors directly rather than observations. Sampling features instead of image-like observations benefits from dimensionality reduction. Reconstructing or generating complex image observations is hard and requires increasingly complex generative models (Van Den Oord et al., 2017; Van den Oord et al., 2016; Razavi et al., 2019). In RL, especially when working with "symbolic" (semantic) (as opposed to "perceptual" (RGB)) observations, changing a small detail of the observation can change the optimal action drastically. Observations might be better separated in feature space, and thus, the sleep policy could be less sensitive to an imperfect decoder. In addition, in contrast to the two-headed architecture, the feature extractor does not need to be run when sampling.

For example, when faced with new tasks that require new actions to be taken, hidden replay can learn to add a new action partition to the feature space while rearranging the existing feature space in a manner that preserves partitioning needed to solve previous tasks. Empirically, we observe a clean partition of the feature space according to the optimal action. Even though samples and reconstructions are not perfect, they fall within the same action partition (Fig. 2) as the ground truth. These representations may allow for knowledge transfer between tasks because tasks requiring similar actions can potentially efficiently be mapped into the existing feature space, but validating this requires further studies.

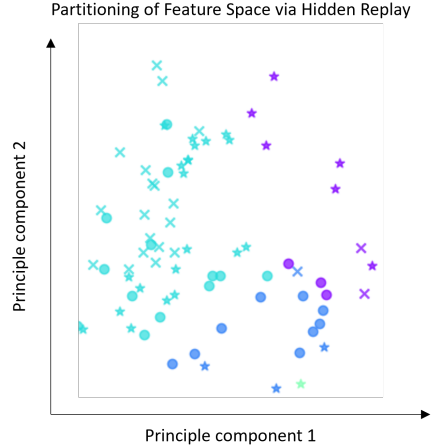Partitioning of Feature Space via Hidden Replay

Figure 2: Hidden replay prevents drift in the features-to-action mapping from the latent vector space of a deep RL agent (Minigrid tasks). Colors represent actions; ×s: drawn from the random replay buffer, ●: drawn from wake buffer; ⋆: sampled via generative replay.

Hidden replay is not without limitations because GR samples are in terms of features without the corresponding observations. This V-shape in the training graph (blue lines in Fig. 1, Bottom) can cause the feature extractor to drift over time or collapse to a mode (e.g., map all observations to the same feature vector). As new tasks are encountered, this would cause sampled features to not align with the features mapped by the policy and VAE components.

To limit feature drift, we store a small (typically $K < 1\%$ of the experience replay buffer) set of raw observations paired with actions in a separate buffer. We denote this RaR (for "random replay" buffer). These are accumulated over the lifetime of the agent from all wake replay buffers b_w^t (although a more sophisticated exemplar selection (Rebuffi et al., 2017; Mi et al., 2020) is also possible). We observed that replaying a few random examples results in less feature drift between task changes (See Appendix A.16 for an ablation). In our experiments, we found that saving 96 random samples per sleep was sufficient to alleviate feature drift in the *Starcraft 2* domain and saving 256 random samples per sleep was sufficient for the Minigrid domain. The complete data flow for hidden replay for LRL is shown in Alg. 2, where the design choice of combinations of ER, GR, and RaR buffers are ablated in our experimental ablations.

Remark: When applied to single task RL, our wake-sleep cycle resembles policy iteration interleaved with policy distillation steps, where each iteration t produces an improved π_w^t (nudge towards STE) that is then distilled into π_s^t. Wake experiences are gathered from the most recent sleep policy ("advice"), including from multiple sub-optimal policies similar in spirit to dataset aggregation in DAGGER (Ross et al., 2011). Similar to DAGGER, the consolidated policy is executed to sample next trajectories. Our supplementary experiments (Appendices A.11 and A.10) show stable convergence of the wake-sleep cycle to optimal policies in single task training (convergence to STE).

4 EXPERIMENTS

4.1 EVALUATION SCENARIOS

A lifelong learning *scenario* describes the order of task presentation and repetition of tasks. A concrete instantiation of a scenario with tasks (or POMDPs) is called a *syllabus*. Each scenario is used to evaluate the learning agent over all syllabi and average metrics. The three scenarios we consider are depicted in Figure 9. **Pairwise:** Two tasks are presented once each. This scenario evaluates knowledge transfer between two tasks. **Alternating:** Two tasks are presented with three repetitions of each pair. This scenario evaluates knowledge retention over longer lifetimes. **Condensed:** A random permutation of all tasks is presented. This is a useful setting for understanding how an agent performs across different task orderings.

Given a syllabus, an experiment run alternates between evaluation blocks (EBs) and learning blocks (LBs) (see New et al., 2022). We say that a task is seen wrt an EB if it has appeared in any LB preceeding it, otherwise it is unseen. During each EB, the average accumulated reward of the agent is evaluated on all tasks in the syllabus (including unseen tasks); during each LB, the agent learns on a single task for a fixed number of interactions. The STE for each task serves as a baseline and measures the relative performance of the learner wrt asymptotic optimal.

4.2 METRICS FOR LIFELONG RL

We run controlled and quantitative experiments and compare to multiple baselines along three sets of metrics. First, we compare learning behavior by examining the accumulated rewards captured by periodic evaluations of the agent, i.e., we look at the trend of EB rewards averaged over EBs (and tasks). Second, we aim to understand how the rewards achieved by an agent compares to a single task expert (STE). This forms the basis for our second set of metrics: relative reward (RR). Note that these metrics also focus on EBs and do not consider LBs. Finally, we compare algorithms using the lifelong learning metrics defined by (New et al., 2022), which take into account both LBs and EBs, and aim to frame metrics in a way that are agnostic to the domain.

We begin by describing the RR metrics. For each task, we train (offline) the STE using an off-the-shelf deep RL algorithm. RR metrics measure how well the agent has learned the seen tasks, how well the agent performs over all tasks in the syllabus, and how well the agent generalizes to unseen tasks. In general, RR is computed as

$$RR(\pi, T, S) = \frac{1}{|T|} \sum_{t \in T} \frac{1}{|S(t)|} \sum_{s \in S(t)} \frac{r_\pi(s)}{r_{\pi_s^*}(s)} \tag{2}$$

where π is the policy of the learner, T is a set EBs, $S(t)$ is the set of tasks at EB t, π_s^* is the STE for task s, and $r_\pi(s)$ is the accumulated reward (return) achieved on task s from using policy π. Different metrics are defined by setting S, T to different EBs and tasks.

$\mathbf{RR_\Omega}$: T only includes the final EB, and $S(t)$ is the set of seen tasks in the syllabus. RR_Ω measures how well the agent has learned the seen tasks in the syllabus "at the end of the day". To measure catastrophic forgetting, we denote $\mathbf{RR_\sigma}$ where T includes all EBs, and $S(t)$ is the set of seen tasks in the syllabus encountered in previous LBs up to and including EB t. $\mathbf{RR_v}$ is complementary in that it measures how well the agent generalizes to unseen tasks, T includes all EBs, and $S(t)$ is the set of unseen tasks not encountered in previous LBs up to and including EB t. To measure plasticity, we denote $\mathbf{RR_\alpha}$ where T is the set of all EBs, and $S(t)$ is only the task learned in LB t preceeding EB t. Note that in all cases, higher more-positive values are better for all metrics as all our STEs achieve positive return.

As mentioned, we employ established lifelong learning metrics defined by New et al. (2022), which capture catastrophic forgetting, transfer between all pairs of tasks, and speed of learning. In all cases, more-positive values are better for all metrics. **Forward Transfer Ratio (FTR)** is the ratio of EB performance on an unseen task. **Backward Transfer Ratio (BTR)** is the ratio of EB performance on a seen task. A value greater than one indicates positive transfer. **Relative Performance (RP)** is the ratio of the Area Under the Learning Curve (AUC) between the lifelong learner and STE learning curve. A value greater than one indicates either faster learning by the lifelong learner and/or superior asymptotic performance. **Performance Maintenance (PM)** is the average difference between task performance in any EB that comes after the LB of a seen task. A value less than 0 indicates forgetting. Refer to (New et al., 2022) for more details about the metrics and the precise computation.
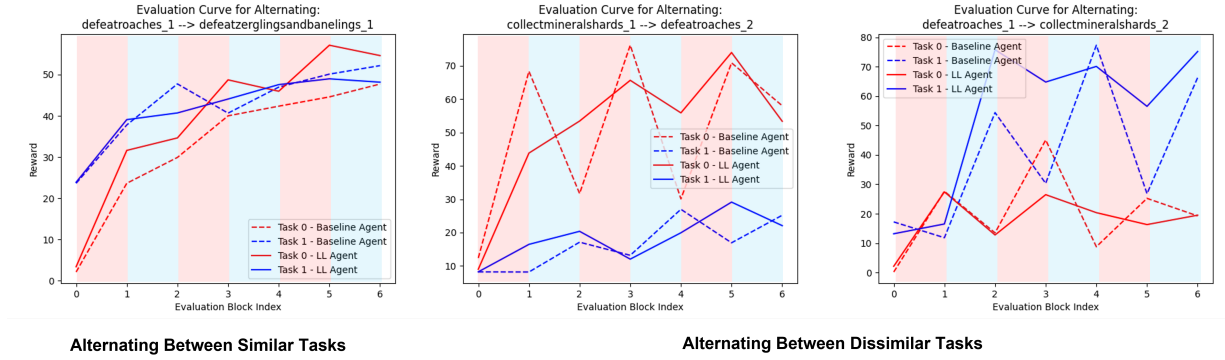
Figure 3: Illustrative examples showing EB rewards of baseline and hidden replay in alternating tasks scenario. Column color denotes which task is trained in LBs between EBs. Left: Example syllabus of two similar tasks; the lifelong learner (LL) performs similarly to the baseline. Right: Examples of two dissimilar tasks where the baseline exhibits catastrophic forgetting, our hidden replay does not.

4.3 DOMAINS

Our experiments use two RL domains: *Starcraft 2* minigames (Vinyals et al., 2017) and Minigrid (Chevalier-Boisvert et al., 2018). We evaluate on existing RL tasks (POMDPs) in these domains.

Starcraft 2: We selected 3 minigames namely DefeatRoaches, DefeatZerglingsAndBanelings, and CollectMineralShards. We created an additional task in each case as described in Appendix A.3. We used PySC2 (Vinyals et al., 2017) to interface with SC-2 game. The agent receives positive rewards for collecting resources and defeating enemies, and negative rewards for losing friendly units. As observations, we used the unit type, selection status, and unit density two-dimensional inputs. We follow the action space and convolutional policy network architecture defined in Vinyals et al. (2017). Single task experts were trained to convergence (~ 30M environment steps using the VTrace algorithm (Espeholt et al., 2018)). We consider the pairwise, alternating, and condensed scenarios where each LB consists of 2 million environment steps (about 6% of the STE samples), and each EB consists of 30 episodes per task. Complete details can be found in the Appendix A.4.1.

Minigrid: Due to the heavy computational load of experimentation in SC-2, we also show results in the Minigrid domain (Chevalier-Boisvert et al., 2018). We use the TELLA framework (Lifelong-Learning-Systems) to orchestrate the experiments ($\sim$100 repetitions per scenario). Tasks involve an agent that must navigate to a goal while interacting with various entities in the environment. We consider 5 tasks — SimpleCrossing, DistShift, a custom Fetch, a custom Unlock, and DoorKey — each with two variants (differing in the size of the grid and/or number of objects) for a total of 10 POMDPs. Descriptions of these tasks can be found in Chevalier-Boisvert et al. (2018). An observation is a 7×7 top-down view of the grid directly in front of the agent, with 3 channels containing information about the object type, object color, and object state in each tile. Rewards are sparse: -1 for running into obstacles and lava, 0 for not reaching the goal, and a reward in $[0, 1]$ for reaching the goal proportional to the number of steps taken. There are only six actions: turn left, turn right, move forward, pick up an object, drop an object, and interact with an object. STEs are trained on each task for 1M environment steps using PPO (Schulman et al., 2017). We consider the pairwise and condensed scenarios where the LB lengths vary between 100k and 700k (Appendix A.5) based on examining the smallest number of steps for the STE to converge to 80% of its peak performance averaged over ten runs. Each EB consists of 100 episodes per task. Further details can be found in Appendix A.4.2.

Baseline Agents: We compare against several non-lifelong baseline agents: 1) Random policy; 2) Baseline agent using off-the-shelf deep RL trained on tasks in sequence without any lifelong learning. We used the VTrace algorithm (Espeholt et al., 2018) for SC-2 and PPO (Schulman et al., 2017) for the Minigrid experiments.

Ablations: In order to compare different replay type, we perform ablations against the combination of buffers used for replay (see Fig. 1): 1) Hidden Replay (ER), 2) Hidden Replay (ER+RaR), 3) Hidden Replay (ER+GR), 4) Hidden Replay (ER+RaR+GR). In each case, the VAE, sleep policy, and wake policy are trained, but on different augmented datasets as shown in Alg. 2 (samples are drawn from VAE only if GR is enabled). Similarly, we perform ablations along architectures discussed in Fig. 1.

Single Task Experiments: We conducted experiments that demonstrated that the wake-sleep training phases successfully imitates the STE. These can be found in the Appendix A.10 for SC-2 and Minigrid.
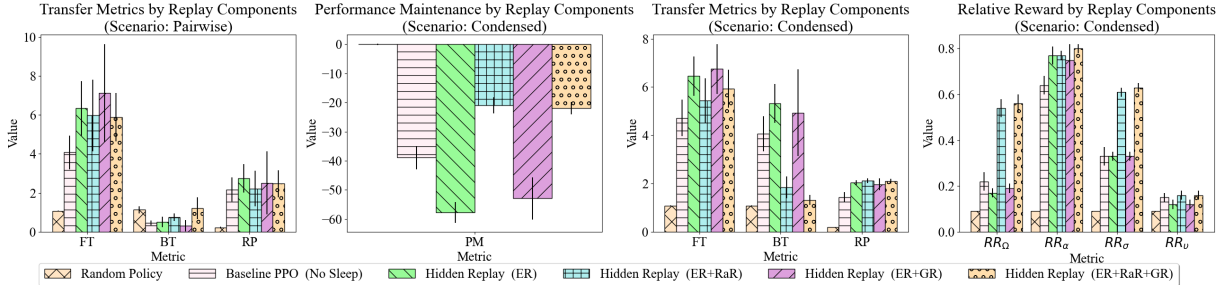
Figure 4: Evaluating different replay types on the hidden replay model for the Minigrid domain. Error bars denote 95% confidence interval.
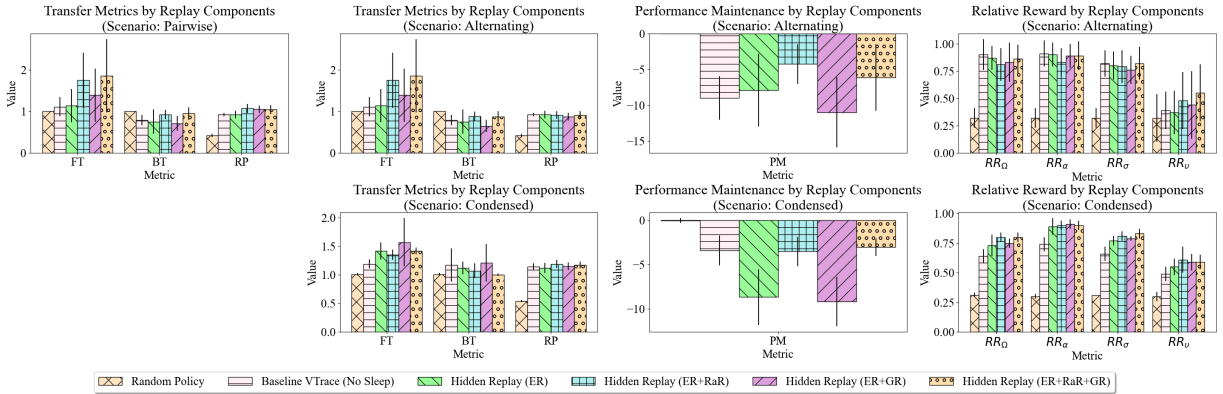

Figure 5: Evaluating different replay types on the hidden replay model for the *Starcraft 2* domain. Error bars denote 95% confidence interval.

4.4 EVIDENCE OF OVERCOMING CATASTROPHIC FORGETTING

We observed that different syllabi and scenarios can exhibit different levels of catastrophic forgetting due to varying degrees of similarity between the tasks in the syllabus. We first illustrate this issue before showing large scale results averaging over all syllabi.[2] In Figure 3, we illustrate examples that 1) the LL agent and non-LL agent perform similarly when syllabi are composed of similar tasks, and both agents exhibit positive FT and BT, and 2) the non-LL agent experiences catastrophic forgetting (zig-zag pattern) of one or both tasks whereas the LL agent shows significantly less forgetting when transitioning between two dissimilar tasks. We note that in the *Starcraft 2* domain, evidence of extreme catastrophic forgetting is somewhat rare, a possible weakness in the design of our task set. Nevertheless, LL agents clearly outperform non-LL on almost all the other metrics, signifying the need to look beyond catastrophic forgetting when evaluating LL.

4.5 COMPARISON OF DIFFERENT TYPES OF REPLAY FOR SLEEP

To understand the effect of different replay types on the hidden replay model: experience replay from the wake buffer (ER), generative replay (GR), and random replay (RaR). All approaches still train both the policy network and generator, even if GR is disabled. We summarize the important findings in Fig. 4 and Fig. 5 [3].

Transfer Learning and Generalization to Unseen Tasks: Looking at the pairwise scenario in Minigrid and *Starcraft 2*, we see that models that use hidden replay with ER+GR, ER+RaR or ER+RaR+GR show much higher FT compared to the baseline. Hidden replay with ER+GR, ER+RaR or ER+RaR+GR show higher RR_v and the effect is more pronounced for *Starcraft 2* condensed scenario, showing generalization to unseen tasks.

Mitigating Catastrophic Forgetting: Looking at PM across all scenarios in Minigrid and *Starcraft 2* and looking at RR_Ω of the Minigrid condensed scenario, we validate that the hidden replay model requires stabilization with RaR

---

[2]We attempt to quantify pairwise similarity between tasks in Appendix A.4.1 by evaluating the forward transfer of the STE for each task to all other tasks. Quantifying task similarity remains an open problem.

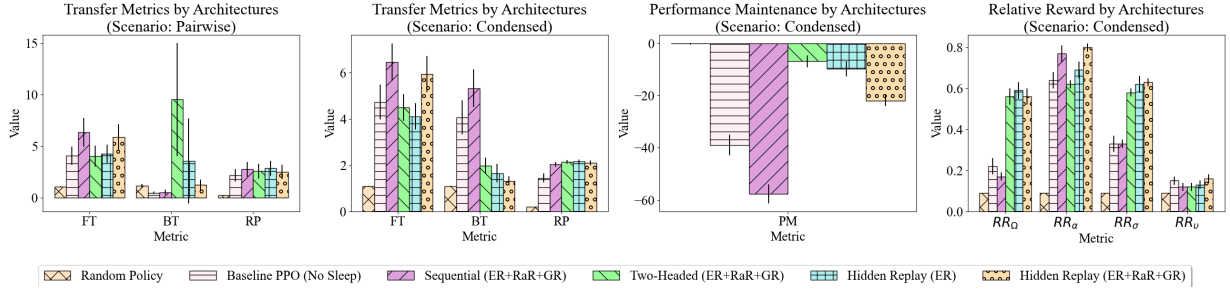[3]Further experiments can be found in Appendix A.13

Figure 6: Evaluating the performance of different architectures for the Minigrid domain. Error bars denote 95% confidence interval.
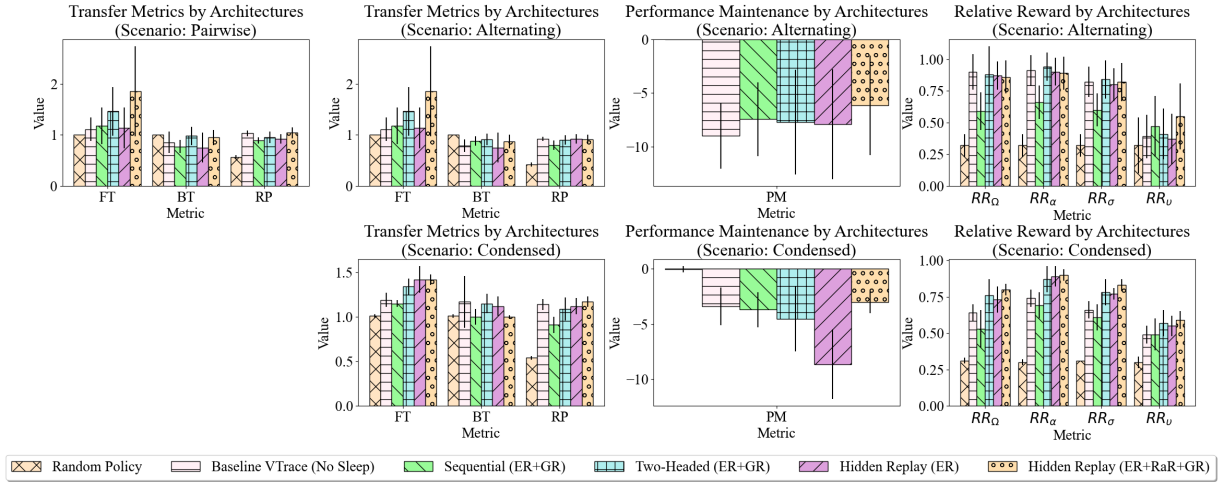

Figure 7: Evaluating the performance of different architectures for the *Starcraft 2* domain. Error bars denote 95% confidence interval.

in order to (significantly) reduce catastrophic forgetting, compared to ER+GR alone which is typically worse than ER+RaR and ER+RaR+GR.

Adaptation to New Tasks: Interestingly, we find that RR_α for the condensed scenarios of Minigrid and *Starcraft 2*, shows that simply performing distillation during sleep phases produces a beneficial "jumpstart" effect when learning new tasks, compared to the baseline which is also able to learn the new task but requires more learning experiences.

Backward Transfer: The BT metric seems to vary across the board based on the type of replay used. This is in part because when computing the BT metric from task 2 to task 1, it is very sensitive to how well the model learns on task 1. If the model learns task 1 to near optimal performance, it is very difficult to achieve $BT > 1$ whereas if task 1 is not well-learned, even a weak model on a similar task 2 can potentially easily achieve $BT > 1$. However, if we look at RR_σ instead, we see a different story where the models that learn best (ER+RaR and ER+RaR+GR) often exhibit the best performance on seen tasks in opposition to the BT metric.

General Observations: Adding a small amount of random replay (ER+RaR+GR) fixes many issues with standard generative hidden replay, producing noticeably better results. We find that ER+RaR, even without GR, achieves significant improvement over the baseline agent, and the model further improves when all types of replay (ER+RaR+GR) are combined, albeit by a small amount. This suggests that RaR of a small number of samples is often powerful and complementary (i.e., often helpful, rarely hurtful) to GR (n.b. all methods train the VAE, but only GR samples from it). This leads us to suspect that the replay architecture is more important than the sampling strategy that we investigate next.

9

4.6 COMPARISON OF DIFFERENT ARCHITECTURES FOR SLEEP

We aim to understand the strengths and weaknesses of the model architectures introduced in this paper (Sequential, Two-Headed, Hidden Replay) and understand how these models improve over simple baselines. We summarize the important findings in Fig. 6 and Fig. 7 [4].

Transfer Learning and Generalization to Unseen Tasks: Forward transfer seems to be a strength of the hidden replay model. Looking at FT across all domains and scenarios and looking at the RR_v of the *Starcraft 2* scenarios, we see that the hidden replay model noticeably outperforms the other models.

Avoiding Catastrophic Forgetting: Looking at PM and RR_Ω across all scenarios, the hidden replay model often outperforms the other models.

Adaptation to New Tasks: We see that the hidden replay model achieves consistently high (and often the best) RP and RR_α across most scenarios in both domains. This suggests that the hidden replay model is not interfering with new tasks and may be providing a helpful "jumpstart" effect.

Positive Backward Transfer: As before, it can be difficult to interpret the BT metric. In general, the BT of the hidden replay model is on par with other models that achieve comparable terminal performance, and the hidden replay model shows strong performance in terms of RR_σ.

Sample Efficiency Compared to STE, our experiments using the hidden replay model required only 6% of training samples to achieve 80-90% of expert performance in most *Starcraft 2* scenarios.

General Observations: In general, across both domains and many scenarios, the hidden replay model seems to outperform the other model architectures tested in most of the metrics of interest. This suggests that the hidden replay model is a potentially powerful model for addressing lifelong RL. We observe (in the case of the Minigrid experiments where all types of replay are used for all architectures), architecture seems to play a more significant role in improving lifelong learning performance than the type of replay.

5 DISCUSSION

Our experiments demonstrate LRL via positive FT and BT, high performance relative to an STE, reduced catastrophic forgetting, and fast adaptation to new tasks. Our work clearly advances the domain-specific state-of-the-art in *Starcraft 2* by reducing the sample complexity of deep RL to master the minigames. Compared to STE, our experiments required only 6% of training samples to achieve 80-90% of expert performance. Existing lifelong RL implementations (e.g., (Mendez et al., 2020)) have been tested on benchmark problems, but proved to be challenging to scale up to the complex observation and action space of SC-2. Future work can leverage our techniques in the full SC-2 game and reduce the astronomical sample complexity of current deep RL agents in SC-2 (e.g., AlphaStar (Vinyals et al., 2019)).

While the wake-sleep mechanism and model-free GR show promising results, there is room for improvement, and the approach is easily extensible. We highlight several potential extensions. Our current model triggers sleep on a set schedule. Instead, it is possible to self-trigger sleep via recent work in lifelong changepoint detection (Faber et al., 2021). Currently, we store full observations in the replay buffer, which due to memory constraints, limits the number of samples that can be saved. It would be prudent to intelligently compress experiences in the replay buffer (Riemer et al., 2019; Wang et al., 2021; Caccia et al., 2020; Zhang & Sutton, 2017; Schaul et al., 2015; Hayes et al., 2020). Past work has looked at improving replay buffers via prioritized replay (Schaul et al., 2015) based on detecting dead-end states. We further hypothesize that increasing the lifetime of an agent by avoiding dead end states is a useful bias. Our methods operate over hand-crafted representations of the observation space; the system could be extended by introducing representation learning from RGB-observations via unsupervised (STAM (Smith et al., 2019a;b)) and self-supervised learning (Gallardo et al., 2021). We can further extend our models using mixtures of VAEs (e.g., the Eigentask framework Raghavan et al. (2020)) to dynamically grow and shrink the model, leading towards agents that build implicit models of the tasks they experience. Finally, we used a fixed neural architecture and loss functions throughout the agent's lifetime. GR is a flexible approach that can include progress in other areas of lifelong learning, e.g., dynamically changing the underlying neural architectures (Schwarz et al., 2018), neuromodulation (Brna et al., 2019), and regularization (Kolouri et al., 2019). Further, interesting avenues include the application of GR to multi-agent RL (Nekoei et al., 2021) and federated learning (Rostami et al., 2017), where communication between agents can enable better coordination between agents and collective acquisition of knowledge. Due to the inherent plastic-yet-stable nature of lifelong learners, our methods can be more robust to data drift, concept drift, or task changes, reducing system downtime and the overhead of retraining, and used to address model obsolescence and reduce the technical debt of machine learning models (Sculley et al., 2015; Alahdab & Çalıklı, 2019; Bogner et al., 2021).

---

[4]Further experiments can be found in Appendix A.14)

REFERENCES

Mohannad Alahdab and Gül Çalıklı. Empirical analysis of hidden technical debt patterns in machine learning software. In International Conference on Product-Focused Software Process Improvement, pp. 195–202. Springer, 2019.

Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting. Neurocomputing, 428:291–307, 2021.

Justus Bogner, Roberto Verdecchia, and Ilias Gerostathopoulos. Characterizing technical debt and antipatterns in ai-based systems: A systematic mapping study. In 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 64–73. IEEE, 2021.

Andrew P Brna, Ryan C Brown, Patrick M Connolly, Stephen B Simons, Renee E Shimizu, and Mario Aguilar-Simon. Uncertainty-based modulation for lifelong learning. Neural Networks, 120:129–142, 2019.

Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. arXiv preprint arXiv:1309.6821, 2013.

Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with adaptive quantization modules. In International Conference on Machine Learning, pp. 1240–1250. PMLR, 2020.

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

Yulai Cong, Miaoyun Zhao, Jianqiao Li, Sijia Wang, and Lawrence Carin. Gan Memory with No Forgetting. Advances in Neural Information Processing Systems, 33:16481–16494, 2020.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In ICML, 2018.

Kamil Faber, Roberto Corizzo, Bartlomiej Sniezynski, Michael Baron, and Nathalie Japkowicz. Watch: Wasserstein change point detection for high-dimensional time series data. In 2021 IEEE International Conference on Big Data (Big Data), pp. 4450–4459. IEEE, 2021.

Jhair Gallardo, Tyler L Hayes, and Christopher Kanan. Self-supervised training enhances online continual learning. arXiv preprint arXiv:2103.14010, 2021.

Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In ECCV, 2020.

Tyler L Hayes, Giri P Krishnan, Maxim Bazhenov, Hava T Siegelmann, Terrence J Sejnowski, and Christopher Kanan. Replay in deep learning: Current approaches and missing biological elements. Neural Computation, 33(11):2908–2950, 2021.

David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In AAAI, 2018.

Nicholas Ketz, Soheil Kolouri, and Praveen Pilly. Continual learning using world models for pseudo-rehearsal. arXiv preprint arXiv:1903.02647, 2019a.

Nicholas Ketz, Soheil Kolouri, and Praveen Pilly. Using World Models for Pseudo-Rehearsal in Continual Learning. arXiv preprint arXiv:1903.02647, 2019b.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. arXiv preprint arXiv:2012.13490, 2020.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. arXiv:1312.6114 [cs, stat], 2014. URL arxiv.org/abs/1312.6114.

James Kirkpatrick, Razvan Pascanu, Neil 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. arXiv:1612.00796 [cs, stat], 2016. URL arxiv.org/abs/1612.00796.

Soheil Kolouri, Nicholas A Ketz, Andrea Soltoggio, and Praveen K Pilly. Sliced cramer synaptic consolidation for preserving deeply learned representations. In International Conference on Learning Representations, 2019.

Lifelong-Learning-Systems. Lifelong-learning-systems/tella: Framework for training & evaluating lifelong learning agents (tella). URL https://github.com/lifelong-learning-systems/tella.

Jorge Mendez, Boyu Wang, and Eric Eaton. Lifelong policy gradient learning of factored policies for faster training without forgetting. Advances in Neural Information Processing Systems, 33:14398–14409, 2020.

Fei Mi, Xiaoyu Lin, and Boi Faltings. Ader: Adaptively distilled exemplar replay towards continual learning for session-based recommendation. In Fourteenth ACM Conference on Recommender Systems, pp. 408–413, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

Hadi Nekoei, Akilesh Badrinaaraayanan, Aaron Courville, and Sarath Chandar. Continuous coordination as a realistic scenario for lifelong learning. In International Conference on Machine Learning, pp. 8016–8024. PMLR, 2021.

Alexander New, Megan Baker, Eric Nguyen, and Gautam Vallabha. Lifelong learning metrics. arXiv preprint arXiv:2201.08278, 2022.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Aswin Raghavan, Jesse Hostetler, Indranil Sur, Abrar Rahman, and Ajay Divakaran. Lifelong learning using eigen-tasks: Task separation, skill acquisition, and selective transfer. arXiv preprint arXiv:2007.06918, 2020.

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. Advances in neural information processing systems, 32, 2019.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 2001–2010, 2017.

Matthew Riemer, Tim Klinger, Djallel Bouneffouf, and Michele Franceschini. Scalable recollections for continual lifelong learning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pp. 1352–1359, 2019.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In Advances in Neural Information Processing Systems, 2019.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Mohammad Rostami, Soheil Kolouri, Kyungnam Kim, and Eric Eaton. Multi-agent distributed lifelong learning for collective knowledge acquisition. arXiv preprint arXiv:1709.05412, 2017.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. arXiv preprint arXiv:1511.06295, 2015.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In International Conference on Machine Learning, pp. 4528–4537. PMLR, 2018.

David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. Advances in neural information processing systems, 28, 2015.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. Advances in neural information processing systems, 30, 2017.

James Smith, Seth Baer, Zsolt Kira, and Constantine Dovrolis. Unsupervised continual learning and self-taught associative memory hierarchies. 2019a.

James Smith, Cameron Taylor, Seth Baer, and Constantine Dovrolis. Unsupervised progressive learning and the stam architecture. arXiv preprint arXiv:1904.02021, 2019b.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.

Gido M van de Ven and Andreas S Tolias. Generative replay with feedback connections as a general strategy for continual learning. arXiv preprint arXiv:1809.10635, 2018.

Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. arXiv:1904.07734 [cs, stat], 2019. URL arxiv.org/abs/1904.07734.

Gido M van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. Nature communications, 11(1):1–14, 2020.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. Advances in neural information processing systems, 29, 2016.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. Advances in neural information processing systems, 30, 2017.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, 575(7782):350–354, 2019.

Kai Wang, Joost van de Weijer, and Luis Herranz. Acae-remind for online continual learning with compressed feature replay. Pattern Recognition Letters, 150:122–129, 2021.

Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. arXiv preprint arXiv:1712.01275, 2017.

A APPENDIX

A.1 POTENTIAL NEGATIVE AND POSITIVE SOCIETAL IMPACTS

This approach presents a means of training a multi-agent system to play real-time strategy games using lifelong reinforcement learning. Such a tool could be used by malicious foreign actors to train models for strategic military actions, which could negatively impact the national security of benign nations.

However, this tool can also be used for societal benefits including training intelligent robot swarms to perform useful tasks over long lifetimes (e.g., applications in agricultural and manufacturing robotics). Similarly, it could be used in domains which require incorporation of feedback into the learning (i.e. necessitates RL) and where distribution shift is a significant problem; e.g., for planning treatments in healthcare.

A.2   LIMITATIONS

The model is not flawless. The following are some limitations we have observed w.r.t. our proposed approach:

- While catastrophic forgetting is reduced using our proposed models, the agent still fails to perfectly learn and remember tasks, and thus, there is still room for improvement in terms of many of the LL metrics. Specifically, we've observed that the model can struggle to remember over very long periods when trained on long syllabi of tasks.

- Training the agent is non-trivial, requiring carefully defining reasonable tasks and performing hyperparameter tuning.

- The sleep model is dependent on having a good quality single-task expert for the wake model, and in some cases, the wake model fails to properly learn a task, disrupting the learning of the sleep model.

- We have also seen some fragility in the learning of the hidden space for hidden replay and imperfect generated samples in the sequential and two-headed models. We have observed cases where the sampled data drifts enough that it mis-aligns with the true distribution of data and/or the quality of the sampled data degrades significantly. In such cases, the sleep model can become unstable and the proposed replay mechanisms may hurt the model's learning.

- While we would like to rely mostly on generative replay and experience replay from the last wake cycle for training the model, we've found that the hidden replay model requires a small buffer of random samples across tasks to maintain the hidden space. This introduces sub-optimality in the memory usage of the model. We have also observed that this "random replay" has an out-sized effect on improving lifelong learning compared to the generative replay.

A.3   STARCRAFT 2 TASK VARIANTS

The task variants we consider are as follows:

- **Collect Mineral Shards – No Fog of War**: A map with 2 Marines and an endless supply of Mineral Shards. Rewards are earned by moving the Marines to collect the Mineral Shards, with optimal collection requiring both Marine units to be split up and moved independently. Whenever all 20 Mineral Shards have been collected, a new set of 20 Mineral Shards are spawned at random locations (at least 2 units away from all Marines). Fog of war is disabled.

- **Collect Mineral Shards – Fog of war:** A map with 2 Marines and an endless supply of Mineral Shards. Rewards are earned by moving the Marines to collect the Mineral Shards, with optimal collection requiring both Marine units to be split up and moved independently. Whenever all 20 Mineral Shards have been collected, a new set of 20 Mineral Shards are spawned at random locations (at least 2 units away from all Marines). Fog of war is enabled, meaning the agent must be able to learn without full knowledge of the current state of the environment.

- **DefeatZerglingsAndBanelings – One Group:** A map with 9 Marines on the opposite side from a group of 6 Zerglings and 4 Banelings. Rewards are earned by using the Marines to defeat Zerglings and Banelings. Whenever all Zerglings and Banelings have been defeated, a new group of 6 Zerglings and 4 Banelings is spawned and the player is awarded 4 additional Marines at full health, with all other surviving Marines retaining their existing health (no restore). Whenever new units are spawned, all unit positions are reset to opposite sides of the map

- **DefeatZerglingsAndBanelings – Two Groups:** A map with 9 Marines in the center with two groups consisting of 9 Zerglings one one side and 6 Banelings on the other side. Rewards are earned by using the Marines to defeat Zerglings and Banelings. Whenever a group has been defeated, a new group of 9 Zerglings and 6 Banelings is spawned and the player is awarded 6 additional Marines at full health, with all other surviving Marines retaining their existing health (no restore). Whenever new units are spawned, all unit positions are reset to opposite sides of the map.

- **DefeatRoaches – One Group:** A map with 9 Marines and a group of 4 Roaches on opposite sides. Rewards are earned by using the Marines to defeat Roaches, with optimal combat strategy requiring the Marines to perform focus fire on the Roaches. Whenever all 4 Roaches have been defeated, a new group of 4 Roaches is spawned and the player is awarded 5 additional Marines at full health, with all other surviving Marines retaining their existing health (no restore). Whenever new units are spawned, all unit positions are reset to opposite sides of the map.

- **DefeatRoaches – Two Groups:** A map with 9 Marines in the center and two groups consisting of 6 total Roaches on opposite sides (3 on each side). Rewards are earned by using the Marines to defeat Roaches, with optimal combat strategy requiring the Marines to perform focus fire on the Roaches. Whenever all 6 Roaches have been defeated, a new group of 6 Roaches is spawned and the player is awarded 7 additional Marines at full health, with all other surviving Marines retaining their existing health (no restore). Whenever new units are spawned, all unit positions are reset to starting areas of the map.

A.4 Implementation Details

A.4.1 Starcraft 2

In this section we describe the implementation details for our *Starcraft 2* agent and experiments. The wake agent is a VTrace agent (Espeholt et al., 2018). The feature extractor component of the wake agent consists of an architecture that is a slight modification of the FullyConv model of Vinyals et al. (2017). We further split this into a feature extractor component and a policy network component to facilitate easy weight copying between the wake and sleep agents. For the hidden replay model, we add a LayerNormalization layer to the output of the feature extractor to constrain the space of the features, which helps to improve feature reconstructions and limit feature drift. The architecture produces features of dimension 8192.

The sleep agent feature extractor and policy network architectures are identical to the wake model. We use a five-layer convolutional neural network architecture with for the encoder and a five-layer deconvolutional architecture for the decoder for the VAEs for the sequential and two-headed models. In both cases, we use ReLU activations between the layers and add batch normalization before the ReLUs. We use two-layered multi-layered perceptrons with ReLU activation functions for the encoder and decoder for the VAE for the hidden replay model. To improve numerical stability, the logvar predicted by the encoder is bounded to take values in $[-5, 5]$ using a scaled-hyperbolic tangent layer.

Three styles of replay are used during sleep for the hidden replay architecture: experience replay on a buffer of 10,000 observation-action pairs from a FIFO queue collected during the wake phase, generative replay on the feature vectors, and random replay. Only experience replay and generative replay is used to train the sleep agent for the sequential and two-headed architectures.

For the sequential architecture: the weight of the imitation loss (cross-entropy on action logits) is 50.0, the weight of the reconstruction loss is 1.0, and the weight of the KL loss of the VAE is 3.0. For the two-headed architecture: the weight of the imitation loss (cross-entropy on action logits) is 30.0, the weight of the reconstruction loss is 1.0, and the weight of the KL loss of the VAE is 2.0. For the hidden replay architecture: the weight of the imitation loss (cross-entropy on action logits) is 50.0, the weight of the reconstruction loss is 200.0, and the weight of the KL loss of the VAE is 5.0. These were determined via hand-tuning.

The agent enters sleep three times per task at even intervals for the sequential and two-headed architectures. The agent enters sleep twice per task at even intervals for the hidden replay architecture. Each training iteration of the wake model is trained on 32 trajectories. Each sleep consists of 4,500 iterations of training using batch sizes of 64 (64 wake samples, 64 generated samples, and 64 random observations from the random replay buffer). Generative replay is not used until the second sleep. 96 observations are selected randomly from the wake buffer every sleep to be added to the random replay buffer for the hidden replay architecture. When the agent wakes up, it resets the PPO model and offers advice with a 80% probability linearly decaying to 0% after for one-half the the duration between sleeps. No weight copying is used for the sequential variant, but weight copying (running a mini-evaluation block after sleep and copying the best Eigentask's parameters into the wake agent) and advice are both used for the two-headed and hidden replay variants.

The Adam optimizer is used to train both the wake and sleep agents with a learning rate of 1.0e-3.

Evaluation: We selected 3 minigames involving battles (DefeatRoaches and DefeatZerglingsAndBanelings tasks) or resource collection (CollectMineralShards). We created two variants of each task, differentiated by the starting locations of units, the presence of "fog-of-war", and/or the number of friendly and enemy units. Task variants are described in Section A.3. We used PySC2 (Vinyals et al., 2017) to interface with SC-2. We used a subset of the available observation maps namely the unit type, selection status, and unit density two-dimensional observations. The action space is factored into functions and arguments, such as move(x, y) or stop$()$. Following Vinyals et al. (2017), our policy networks output probabilities for the different action factors independently. The agent receives positive rewards for collecting resources and defeating enemy units, and negative rewards for losing friendly units. Single task experts (STEs) were trained to convergence (~ 30 million environment steps using the VTrace algorithm (Espeholt

et al., 2018)). We consider the pairwise, alternating, and condensed scenarios where each LB consists of 2 million environment steps (about 6% of the STE samples), and each EB consists of 30 episodes per task.

Each lifetime of a condensed scenario takes approximately 3-4 days running on systems with RTX2080 GPUs.

	collectmineralshards_1	collectmineralshards_2	defeatroaches_1	defeatroaches_2	defeatzerglingsandbanelings_1	defeatzerglingsandbanelings_2
collectmineralshards_1	1	0.777125457	0.400111228	0.145920706	0.092597378	0.036750716
collectmineralshards_2	0.704313295	1	0.353735948	0.140707388	-0.206531382	-0.033803937
defeatroaches_1	-0.208179913	0.037697357	1	0.199487033	0.294108741	-0.001298364
defeatroaches_2	-0.24852037	-0.242845206	0.57786301	1	0.269604282	0.129954049
defeatzerglingsandbanelings_1	-0.150499921	-0.077760498	0.624702523	0.303168748	1	0.103322863
defeatzerglingsandbanelings_2	-0.193706044	-0.121581342	0.316105349	0.188948396	0.220628812	1

Figure 8: Similarity between tasks measured in terms of forward transfer using single task experts

A.4.2 MINIGRID

In this section, we describe the implementation details for our Minigrid agent and experiments. The wake agent is based on the Stable Baselines 3 PPO agent (Schulman et al., 2017; Raffin et al., 2021). There are separate networks for computing the value and policy. Both networks are simple multi-layered perceptrons that first map to a 256-dimensional feature space and then apply a densely connected layer to select one of six actions or estimate the value of a state. ReLUs are used as activation functions between all of the layers. Observations are scaled so the values of every cell are in $[0, 1]$ before being input to the network. For exact parameters for the wake agent, please refer to the appendix.

The sleep agent uses one Eigentask. The feature extractor and expected inputs are the same as those of the wake agent's policy network. A variational autoencoder is used as generative model. The VAE is a four-layer bottleneck-style MLP architecture with a latent dimension size: 128. To improve numerical stability, the logvar predicted by the encoder is bounded to take values in $[-5, 5]$.

Three styles of replay are used during sleep: experience replay on a buffer of 20,000 observation-action pairs from a FIFO queue collected during the wake phase, generative replay on the feature vectors, and random replay (256 random observations added per sleep). The weight of the imitation loss (cross-entropy on discrete actions) is 3.0, the weight of the reconstruction loss is 1.0, and the weight of the KL loss of the VAE is 0.03. The agent enters sleep at the end of each task. Each sleep consists of 20,000 iterations of training using batch sizes of 32 (32 wake samples, 32 generated samples, and 32 random examples drawn from the random replay buffer). Generative replay and random replay are not used until the second sleep. 256 observations are selected randomly from the wake buffer every sleep to be added to the random replay buffer with a maximum buffer size of 4096. When the agent wakes up, it resets the PPO model and offers advice with a 90% probability linearly decaying to 0% after 100,000 environment steps for the condensed scenario and 33,333 environment steps for the dispersed scenario. No weight copying is used.

The Adam optimizer is used to train both the wake and sleep agents.

Evaluation: We report results using the Minigrid domain (Chevalier-Boisvert et al., 2018), in which an agent must navigate to a goal while interacting with various entities in the environment. We consider five tasks — SimpleCrossing, DistShift, a custom Fetch, a custom Unlock, and DoorKey — each with two variants (differing in the size of the grid and/or number of objects) for a total of ten POMDPs. Descriptions of these tasks can be found in Chevalier-Boisvert et al. (2018). An observation consists of a 7×7 top-down view of the grid directly in front of the agent, with three channels containing information about the object type, object color, and object state in each tile. Rewards are sparse and consist of -1 for running into obstacles and lava, 0 for not reaching the goal, and a positive reward in $[0, 1]$ for reaching the goal that is proportional to the number of steps taken. Unlike SC-2, there are only six actions: turn left, turn right, move forward, pick up an object, drop an object, and interact with an object. STEs are trained on each task for 1M environment steps using PPO (Schulman et al., 2017). We consider the pairwise, alternating, and condensed scenarios where the LB lengths vary between 100k and 700k (Section A.5) based on examining the smallest number of steps for the STE to converge to 80% of its peak performance averaged over ten runs. Each EB consists of 100 episodes per task.

Each lifetime of a condensed scenario takes approximately 4-6 hours to run on systems with RTX2080 GPUs.

A.5 MINIGRID TASK LENGTHS

We use the following number of environment steps per task in our minigrid experiments:

- SimpleCrossingS9N1: 400,000

- SimpleCrossingS9N2: 500,000

- DistShiftR2: 200,000

- DistShiftR3: 200,000

- CustomFetchS5T1N2: 500,000

- CustomFetchS8T1N2: 700,000

- CustomUnlockS5: 200,000

- CustomUnlockS7: 300,000

- DoorKeyS5: 200,000

- DoorKeyS6: 300,000

A.6 PARAMETERS OF THE PPO AGENT

These are the parameters used for the stable baselines 3 PPO agent for the Minigrid tasks:

- n_steps: 512

- batch_size: 32

- gae_lambda: 0.95

- gamma: 0.99

- n_epochs: 10

- ent_coef: 5.0e-5

- learning_rate: 2.5e-4

- clip_range: 0.3

- vf_coef: 0.75

- max_grad_norm: 5.0

- policy_kwargs: dict(activation_fn=nn.ReLU, net_arch=[dict(pi=[256, 256], vf=[256, 256])])

A.7 EVALUATION SCENARIOS VISUALIZATION

Figure 9: Lifelong Learning Scenarios. Grey evaluation blocks (EBs) evaluate the agent on all tasks. Learning blocks (LBs) are color coded by task, and subscripts denote task variants.
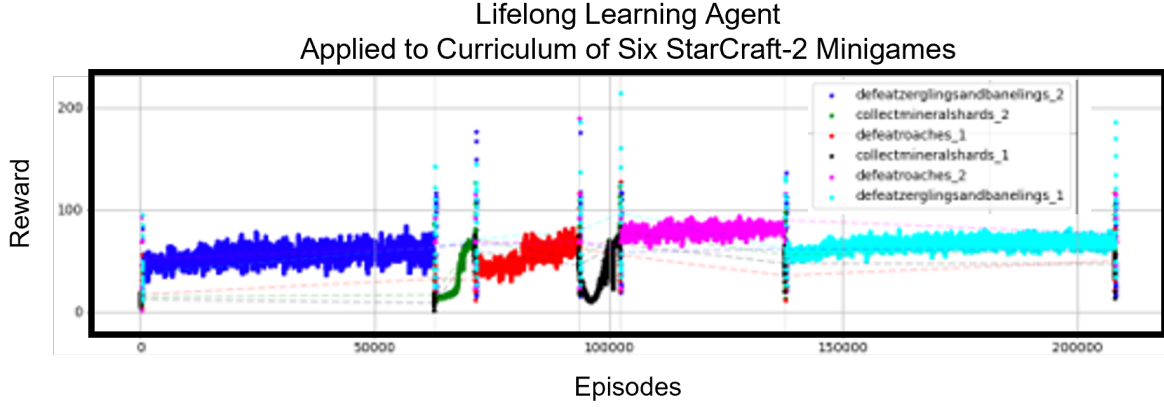
A.8   EXAMPLE LEARNING CURVES FOR LIFELONG CURRICULA ON *Starcraft 2*

Figure 10: Learning curve for our lifelong learning agent applied to a six minigame scenario in the *Starcraft 2* domain. Performance comparable to the converged single-task expert is ˜100.

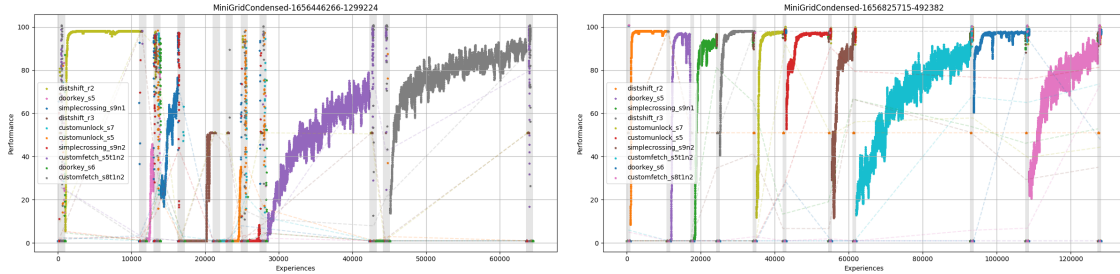A.9   COMPARISON OF LEARNING CURVES FOR LIFELONG CURRICULA ON MINIGRID

Figure 11: Learning curves for a condensed scenario in the Minigrid domain. Left: learning curve of non-LL agent; Right: learning curve of LL agent on the same scenario. Maximum reward is 100. The LL agent is capable of learning more tasks without interference and exhibits jump starts in performance for similar tasks due to the advice mechanism.

A.10   A SLEEP POLICY TRAINED WITH POLICY DISTILLATION CAN LEARN TO IMITATE A CONVERGED SINGLE TASK EXPERT

Single Task Experiments:

SC-2: We demonstrate that a sleep policy can be trained with policy distillation to mimic the performance of an STE's policy. We consider STEs identical to the wake agents trained for 30M environment steps for each of the six SC-2 variants. We act in accordance with the following procedure: i) Train an STE on each of the six SC-2 POMDPs for 30M steps. ii) Measure the final reward for each STE. iii) Load the trained STE and build a buffer of 10K trajectories. iv) Train the sleep model until convergence on randomly sampled batches of observation-action pairs (batch size: 32). v) Periodically evaluate the sleep model on SC-2 simulations, each consisting of 30 episodes of the task-of-interest. We show results in Figs. 12 and 13. The two-headed and hidden replay architectures architectures typically match or exceed the STE performance in terms of rewards. The sequential model learns non-trivial policies, but generally performs worse than the STE. This supports our hypothesis that decoupling the generator from the policy network of the VAE results in better-performing policies.

18

Figure 12: Examining whether sleep policies can be trained to mimic the performance of single task experts for the SC-2 POMDPs with the sequential, two-headed, and hidden replay model without a random replay buffer. The blue curves denote the performance (in terms of rewards) of the sleep policy interpolated along evaluation points. The red line denotes terminal performance of the STE after convergence. Error bars denote standard deviation of rewards.

Figure 13: Examining whether the sleep policy can be trained to mimic the performance of single task experts for the SC-2 POMDPs with the hidden replay model without a random replay buffer. The blue curves denote the performance (in terms of rewards) of the sleep policy interpolated along evaluation points. The red line denotes terminal performance of the STE after convergence. Error bars denote standard deviation of rewards.
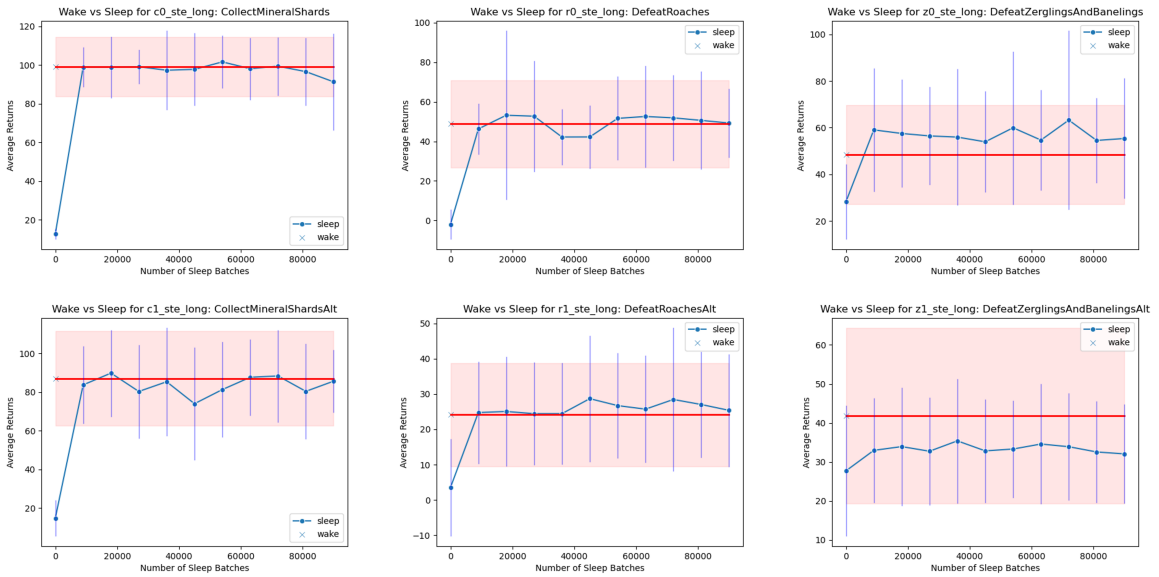
Minigrid: We also quantitatively validate that the sleep policy using the hidden replay model is able to distill the an STE's converged policy after 1M environment steps for the Minigrid domain using an experience buffer of the STE's last 20K observation-action pairs (repeated 10 times per task). We find that the average difference in rewards between the STE and sleep policy's performance over all task variants is 0.14 with a standard deviation of 0.11, suggesting that the sleep policy imitates the STE to an acceptable level.

A.11 A SLEEP POLICY TRAINED WITH POLICY DISTILLATION CAN LEARN TO IMITATE THE WAKE POLICY ON A SINGLE TASK

We conduct experiments to show that the converged performance of a wake policy can be achieved by our agent via imitating the sequence of intermediate wake policies over multiple wake-sleep phases. This is an important experiment needed to show stable convergence of the wake-sleep mechanism even when distilling sequences of sub-optimal policies. These experiments involve training an agent on a single task, periodically entering sleep at set intervals. We observed that for most tasks, the curves formed by evaluating the sleep agent and wake agents at periodic evaluation blocks closely track one another, demonstrating stable convergence of the sleep agent's policy.

SC-2: We consider the case where the system performs multiple wake-sleep phases over the course of a single task, requiring the sleep policy to consolidate the increasingly improving policies of the wake agent. Using the SC-2 simulator, each wake policy is trained for 6M environment steps, entering sleep at even intervals ten times over the entire run. Every time the system exits sleep, the wake and sleep policies are evaluated on thirty episodes of the task. We show the wake versus sleep performance (in terms of rewards) of a hidden replay-based sleep policy in Fig. 14. The sleep policy's performance typically closely mirrors the wake policy's performance, but there are cases where the agent learns a non-trivial policy, but under-performs the wake policy. While not pictured here, similar behavior was observed for the sequential and two-headed architectures as well.

Figure 14: Examining whether the sleep policy can be trained to mimic the performance of wake non-lifelong learner for the SC-2 POMDPs as the non-lifelong learners improve. This experiment examined the performance of the hidden replay model without using observations from the random replay buffer. The blue curves denote the performance (in terms of rewards) of the sleep policy interpolated along evaluation points. The red line denotes performance of the wake policy interpolated along evaluation points. Error bars denote standard deviation of rewards.

Minigrid: We also conducted a post-hoc analysis of our condensed scenario runs in the Minigrid domain to quantify the sleep policy's capability to imitate the wake policy after learning the first task.

Sequential Model: The mean absolute difference between the wake policy's average reward for the last 100 episodes and the sleep policy's average reward for the evaluation block was 0.589 with a standard deviation of 0.320, suggesting the sleep policy when using the sequential architecture struggles to accurately imitate the wake policy in many, but not

all runs. In cases where the wake model converges to a policy with average reward > 0.9 for the last 100 episodes, the mean difference metric decreases to 0.591 ± 0.384.

Two-Headed Model: The mean absolute difference between the wake policy's average reward for the last 100 episodes and the sleep policy's average reward for the evaluation block was 0.422 with a standard deviation of 0.263, suggesting the sleep policy when using the two-headed model can struggle to accurately imitate the wake policy in some runs. In cases where the wake model converges to a policy with average reward > 0.9 for the last 100 episodes, the mean difference metric decreases to 0.341 ± 0.297.

Hidden Replay Model: The mean absolute difference between the wake policy's average reward for the last 100 episodes and the sleep policy's average reward for the evaluation block was 0.209 with a standard deviation of 0.227, suggesting the sleep policy accurately imitates the wake policy in many, but not all runs. In cases where the wake model converges to a policy with average reward > 0.9 for the last 100 episodes, the mean difference metric decreases to 0.110 ± 0.145.

These results suggest that the hidden replay model has an advantage when it comes to imitating a wake model when learning a task from scratch. The sequential model struggles with imitating the wake model when learning a task from scratch; however, our main results in the paper suggest that as the sequential model encounters additional tasks, this issue of successfully imitating the wake policy is somewhat alleviated as the relative reward post learning block metric ultimately is similar to the two-headed model when deployed over an entire syllabus of tasks.

A.12 EFFECT OF VARYING THE NUMBER OF WAKE-SLEEP CYCLES

we consider the case where the system encounters multiple tasks in sequence, and we evaluate the effect of the number of wake-sleep phases per task on the lifelong learning metrics. We trained the two-headed replay architecture for 4 lifetimes, each consisting of alternating between an *Starcraft 2* combat and collection task with 2M environment steps per task. We compared the effect of one, two, and three sleeps per task and report results in Table 1. Results suggest that multiple sleep phases result in improved or maintained performance in most metrics, but may negatively affect FT.

Agent	PM	FT	BT	RP	RR_Ω	RR_α	RR_σ	RR_υ
One Sleep Cycle Per Task	-8.76 (± 4.39)	1.68 (± 2.16)	0.76 (± 0.06)	0.78 (± 0.22)	0.58 (± 0.29)	0.7 (± 0.22)	0.58 (± 0.22)	0.3 (± 0.38)
Two Sleep Cycles Per Task	-10.47 (± 6.41)	1.16 (± 0.67)	0.93 (± 0.11)	0.87 (± 0.11)	0.79 (± 0.47)	0.95 (± 0.25)	0.81 (± 0.31)	0.27 (± 0.31)
Three Sleep Cycles Per Task	-8.63 (± 7.13)	1.06 (± 0.86)	0.78 (± 0.25)	0.84 (± 0.13)	0.67 (± 0.19)	0.7 (± 0.13)	0.61 (± 0.1)	0.19 (± 0.19)

Table 1: Comparing the effect of sleeping multiple times per task on a subset of four alternating scenarios involving transitioning between combat and collection *Starcraft 2* tasks. We see in many cases that sleeping multiple times per task can improve metrics.

A.13 UNDERSTANDING THE EFFECT OF DIFFERENT REPLAY TYPES

In Tables 2 and 3, we investigate the effects of turning on and off different replay mechanisms in the hidden replay model. In general, we validate that combining experience replay of the wake buffer, generative replay, and random replay, results in an improved agent; although experience replay in combination with random replay performs surprisingly well.

Scenario	Agent	PM	FT	BT	RP	RR_Ω	RR_α	RR_σ	RR_υ
Pairwise	Random Policy	0.37 (± 0.81)	1.06 (± 0.02)	1.13 (± 0.18)	0.22 (± 0.04)	0.09 (± 0.02)	0.09 (± 0.02)	0.09 (± 0.02)	0.09 (± 0.00)
	Baseline PPO	-48.2 (± 12.04)	4.07 (± 0.87)	0.46 (± 0.12)	2.17 (± 0.61)	0.5 (± 0.1)	0.78 (± 0.08)	0.64 (± 0.08)	0.16 (± 0.02)
	Hidden Replay (ER)	-45.25 (± 13.54)	6.34 (± 1.4)	0.51 (± 0.26)	2.75 (± 0.73)	0.47 (± 0.06)	0.73 (± 0.08)	0.6 (± 0.06)	0.11 (± 0.02)
	Hidden Replay (ER + RaR)	-21.11 (± 11.06)	5.97 (± 1.84)	0.75 (± 0.19)	2.22 (± 0.91)	0.56 (± 0.08)	0.68 (± 0.08)	0.62 (± 0.08)	0.11 (± 0.02)
	Hidden Replay (ER + GR)	-62.32 (± 25.2)	7.12 (± 2.51)	0.31 (± 0.29)	2.51 (± 1.61)	0.4 (± 0.11)	0.74 (± 0.13)	0.57 (± 0.11)	0.11 (± 0.02)
	Hidden Replay (ER + RaR + GR)	-15.77 (± 10.09)	5.87 (± 1.24)	1.21 (± 0.55)	2.47 (± 0.69)	0.6 (± 0.06)	0.69 (± 0.08)	0.64 (± 0.06)	0.11 (± 0.02)
Condensed	Random Policy	-0.02 (± 0.26)	1.08 (± 0.02)	1.08 (± 0.02)	0.19 (± 0.00)	0.09 (± 0.00)	0.09 (± 0.00)	0.09 (± 0.00)	0.09 (± 0.00)
	Baseline PPO	-38.92 (± 3.98)	4.72 (± 0.75)	4.06 (± 0.73)	1.44 (± 0.2)	0.22 (± 0.04)	0.64 (± 0.04)	0.33 (± 0.04)	0.15 (± 0.02)
	Hidden Replay (ER)	-57.71 (± 3.61)	6.45 (± 0.81)	5.32 (± 0.81)	2.04 (± 0.1)	0.17 (± 0.02)	0.77 (± 0.04)	0.33 (± 0.02)	0.12 (± 0.02)
	Hidden Replay (ER + RaR)	-20.96 (± 2.81)	5.44 (± 0.93)	1.85 (± 0.45)	2.11 (± 0.1)	0.54 (± 0.04)	0.77 (± 0.02)	0.61 (± 0.02)	0.16 (± 0.02)
	Hidden Replay (ER + GR)	-52.91 (± 7.26)	6.75 (± 1.03)	4.93 (± 1.8)	1.97 (± 0.24)	0.19 (± 0.02)	0.75 (± 0.07)	0.33 (± 0.02)	0.12 (± 0.02)
	Hidden Replay (ER + RaR + GR)	-21.94 (± 2.09)	5.93 (± 0.79)	1.32 (± 0.2)	2.1 (± 0.1)	0.56 (± 0.04)	0.8 (± 0.02)	0.63 (± 0.02)	0.16 (± 0.02)

Table 2: Effect of different replay mechanisms during sleep on lifelong learning metrics on the Minigrid domain. Results are collected using the hidden replay model on thirty-six syllabi. We report means and 95% confidence intervals. Hidden replay using only experience replay (ER) and generative replay (GR) is insufficient. Random replay (RaR) is needed to stablize the learning of the feature space, and combining all three types of replay yield the best results.

Scenario	Agent	PM	FT	BT	RP	RR$_\Omega$	RR$_\alpha$	RR$_\sigma$	RR$_\upsilon$
	Random Policy	0.0 (±0.00)	1.0 (±0.00)	1.0 (±0.00)	0.42 (±0.04)	0.32 (±0.09)	0.32 (±0.09)	0.32 (±0.09)	0.32 (±0.22)
	Baseline VTrace	−8.99 (±3.06)	1.11 (±0.23)	0.79 (±0.12)	0.92 (±0.04)	0.9 (±0.14)	0.91 (±0.12)	0.82 (±0.12)	0.39 (±0.17)
	Hidden Replay(ER)	−7.9 (±5.11)	1.14 (±0.4)	0.75 (±0.29)	0.92 (±0.09)	0.87 (±0.11)	0.9 (±0.11)	0.8 (±0.13)	0.37 (±0.2)
Pairwise	Hidden Replay(ER + RaR)	−3.7 (±4.18)	1.75 (±0.66)	0.92 (±0.11)	1.07 (±0.11)	0.78 (±0.18)	0.81 (±0.18)	0.79 (±0.18)	0.48 (±0.26)
	Hidden Replay(ER + GR)	−9.35 (±7.46)	1.39 (±0.64)	0.71 (±0.18)	1.05 (±0.09)	0.71 (±0.15)	0.82 (±0.11)	0.77 (±0.13)	0.44 (±0.31)
	Hidden Replay (ER + RaR + GR)	−0.57 (±2.93)	1.85 (±0.88)	0.95 (±0.15)	1.04 (±0.11)	0.77 (±0.18)	0.82 (±0.13)	0.8 (±0.15)	0.55 (±0.26)
	Random Policy	0.0 (±0.00)	1.0 (±0.00)	1.0 (±0.00)	0.42 (±0.04)	0.32 (±0.09)	0.32 (±0.09)	0.32 (±0.09)	0.32 (±0.22)
	Baseline VTrace	−8.99 (±3.06)	1.11 (±0.23)	0.79 (±0.12)	0.92 (±0.04)	0.9 (±0.14)	0.91 (±0.12)	0.82 (±0.12)	0.39 (±0.17)
Alternating	Hidden Replay(ER)	−7.9 (±5.11)	1.14 (±0.4)	0.75 (±0.29)	0.92 (±0.09)	0.87 (±0.11)	0.9 (±0.11)	0.8 (±0.13)	0.37 (±0.2)
	Hidden Replay(ER + RaR)	−4.25 (±2.77)	1.75 (±0.66)	0.88 (±0.11)	0.91 (±0.09)	0.81 (±0.15)	0.83 (±0.13)	0.79 (±0.15)	0.48 (±0.26)
	Hidden Replay(ER + GR)	−10.99 (±4.89)	1.39 (±0.64)	0.65 (±0.15)	0.87 (±0.09)	0.83 (±0.18)	0.89 (±0.11)	0.76 (±0.13)	0.44 (±0.31)
	Hidden Replay (ER + RaR + GR)	−6.13 (±4.64)	1.85 (±0.88)	0.87 (±0.13)	0.91 (±0.09)	0.86 (±0.13)	0.89 (±0.13)	0.82 (±0.15)	0.55 (±0.26)
	Random Policy	−0.03 (±0.28)	1.01 (±0.02)	1.01 (±0.02)	0.54 (±0.02)	0.31 (±0.02)	0.3 (±0.02)	0.31 (±0.00)	0.3 (±0.04)
	Baseline VTrace	−3.41 (±1.7)	1.19 (±0.29)	1.17 (±0.29)	1.14 (±0.06)	0.64 (±0.06)	0.74 (±0.06)	0.66 (±0.06)	0.49 (±0.06)
Condensed	Hidden Replay(ER)	−8.65 (±3.15)	1.42 (±0.15)	1.12 (±0.11)	1.12 (±0.09)	0.73 (±0.09)	0.89 (±0.07)	0.77 (±0.04)	0.55 (±0.07)
	Hidden Replay(ER + RaR)	−3.53 (±1.67)	1.35 (±0.09)	1.07 (±0.13)	1.19 (±0.07)	0.8 (±0.04)	0.9 (±0.04)	0.81 (±0.04)	0.61 (±0.11)
	Hidden Replay(ER + GR)	−9.17 (±2.8)	1.57 (±0.42)	1.21 (±0.33)	1.15 (±0.07)	0.75 (±0.04)	0.91 (±0.04)	0.79 (±0.02)	0.59 (±0.07)
	Hidden Replay (ER + RaR + GR)	−3.05 (±0.97)	1.42 (±0.06)	1.0 (±0.02)	1.17 (±0.06)	0.8 (±0.04)	0.9 (±0.04)	0.83 (±0.04)	0.59 (±0.06)

Table 3: Effect of different replay mechanisms during sleep on lifelong learning metrics on the *Starcraft 2* domain. Results are collected using the hidden replay model on twelve syllabi. We report means and 95% confidence intervals. Hidden replay using only experience replay (ER) and generative replay (GR) is insufficient. Random replay (RaR) is needed to stablize the learning of the feature space, and combining all three types of replay yield the best results.

A.14 Understanding the Effect of Different Model Architectures

In Tables 4 and 5, we compare the different replay architectures introduced in this paper with several simple benchmark models. We observe that the hidden replay model generally outperforms the sequential and two-headed architectures as well as the simple benchmark models.

Scenario	Agent	PM ●	FT	BT	RP	RR$_\Omega$	RR$_\alpha$	RR$_\sigma$	RR$_\upsilon$
	Random Policy	0.37 (±0.81) ●	1.06 (±0.02) ○	1.13 (±0.18) ●	0.22 (±0.04) ○	0.09 (±0.02) ○	0.09 (±0.02) ○	0.09 (±0.02) ○	0.09 (±0.00) ○
	Baseline PPO	−48.2 (±12.04)	4.07 (±0.87)	0.46 (±0.12)	2.17 (±0.61)	0.5 (±0.1)	0.78 (±0.08)	0.64 (±0.08)	0.16 (±0.02)
Pairwise	Hidden Replay (ER)	−45.25 (±13.54)	6.34 (±1.4)	0.51 (±0.26)	2.75 (±0.73)	0.47 (±0.06)	0.73 (±0.08)	0.6 (±0.06)	0.11 (±0.02) ○
	Sequential (ER + RaR + GR)	13.58 (±9.06) ●	4.01 (±1.02)	9.52 (±5.5) ●	2.57 (±0.72)	0.45 (±0.08)	0.37 (±0.08) ○	0.41 (±0.08) ○	0.08 (±0.02) ○
	Two-Headed (ER + RaR + GR)	6.73 (±6.19) ●	4.24 (±0.88)	3.55 (±4.12) ●	2.85 (±0.72)	0.49 (±0.1)	0.46 (±0.08) ○	0.48 (±0.08)	0.1 (±0.02) ○
	Hidden Replay (ER + RaR + GR)	−15.77 (±10.09)	5.87 (±1.24)	1.21 (±0.55)	2.47 (±0.69)	0.6 (±0.06)	0.69 (±0.08)	0.64 (±0.06)	0.11 (±0.02) ○
	Random Policy	−0.02 (±0.26) ●	1.08 (±0.02) ○	1.08 (±0.02) ○	0.19 (±0.00) ○	0.09 (±0.00) ○	0.09 (±0.00) ○	0.09 (±0.00) ○	0.09 (±0.00) ○
	Baseline PPO	−38.92 (±3.98)	4.72 (±0.75)	4.06 (±0.73)	1.44 (±0.2)	0.22 (±0.04)	0.64 (±0.04)	0.33 (±0.04)	0.15 (±0.02)
Condensed	Hidden Replay (ER)	−57.71 (±3.61)	6.45 (±0.81)	5.32 (±0.81)	2.04 (±0.1) ●	0.17 (±0.02)	0.77 (±0.04) ●	0.33 (±0.02)	0.12 (±0.02)
	Sequential (ER + RaR + GR)	−6.8 (±2.29) ●	4.49 (±0.57)	1.98 (±0.35) ○	2.13 (±0.08) ●	0.56 (±0.04) ●	0.62 (±0.02)	0.58 (±0.02) ●	0.12 (±0.02)
	Two-Headed (ER + RaR + GR)	−9.69 (±2.87) ●	4.1 (±0.57)	1.65 (±0.41) ○	2.15 (±0.1) ●	0.59 (±0.04) ●	0.69 (±0.04)	0.62 (±0.04) ●	0.13 (±0.02)
	Hidden Replay (ER + RaR + GR)	−21.94 (±2.09)	5.93 (±0.79)	1.32 (±0.2) ○	2.1 (±0.1) ●	0.56 (±0.04) ●	0.8 (±0.02) ●	0.63 (±0.02) ●	0.16 (±0.02)

Table 4: Comparison of the different model-free generative replay architectures and the baseline on two different Minigrid scenarios. The metrics are averaged over 36 lifetimes per scenario and the 95% confidence interval is shown. ● denotes statistically significant improvement compared to the baseline with a Kruskal-Wallis test followed by Dunn's test with Bon Ferroni correction w/ a p-value of 0.05. ○ denotes the metric is statistically significantly worse than the baseline. We see that the LL agents significantly outperform the non-LL agent interms of performance maintainence and RR$_\Omega$, suggesting the LL agents significantly reduce catastrophic forgetting. The hidden replay model achieves significantly higher RP and RR$_\alpha$ suggesting that it enables the model to learn new tasks with minimal interference from learning previous tasks.

Scenario	Agent	PM	FT	BT	RP	RR_Ω	RR_α	RR_σ	RR_v
Pairwise	Random Policy	0.0 (±0.00)	1.0 (±0.00)	1.0 (±0.00)	0.56 (±0.04) ○	0.33 (±0.06) ○	0.33 (±0.06) ○	0.33 (±0.06) ○	0.34 (±0.14)
	Baseline VTrace (No Sleep)	−8.2 (±6.54)	1.11 (±0.23) •	0.85 (±0.21)	1.03 (±0.06)	0.74 (±0.14)	0.82 (±0.12)	0.78 (±0.12)	0.39 (±0.17)
	Sequential (ER + GR)	−5.51 (±3.41)	1.18 (±0.36)	0.77 (±0.13)	0.89 (±0.06)	0.54 (±0.15)	0.61 (±0.15)	0.57 (±0.15)	0.47 (±0.24)
	Two-Headed (ER + GR)	−2.35 (±7.31)	1.46 (±0.48)	0.98 (±0.18)	0.95 (±0.11)	0.76 (±0.18)	0.79 (±0.18)	0.77 (±0.18)	0.41 (±0.2)
	Hidden Replay (ER Only)	−7.9 (±5.11)	1.14 (±0.4)	0.75 (±0.29)	0.92 (±0.09)	0.87 (±0.11)	0.9 (±0.11)	0.8 (±0.13)	0.37 (±0.2)
	Hidden Replay (ER + RaR + GR)	−0.57 (±2.93)	1.85 (±0.88)	0.95 (±0.15)	1.04 (±0.11)	0.77 (±0.18)	0.82 (±0.13)	0.8 (±0.15)	0.55 (±0.26)
Alternating	Random Policy	0.0 (±0.00) •	1.0 (±0.00)	1.0 (±0.00) •	0.42 (±0.04) ○	0.32 (±0.09) ○	0.32 (±0.09) ○	0.32 (±0.09) ○	0.32 (±0.22)
	Baseline VTrace (No Sleep)	−8.99 (±3.06)	1.11 (±0.23)	0.79 (±0.12)	0.92 (±0.04)	0.9 (±0.14)	0.91 (±0.12)	0.82 (±0.12)	0.39 (±0.17)
	Sequential (ER + GR)	−7.44 (±3.43)	1.18 (±0.36)	0.88 (±0.09)	0.8 (±0.09)	0.59 (±0.15)	0.66 (±0.13)	0.6 (±0.13)	0.47 (±0.24)
	Two-Headed (ER + GR)	−7.7 (±4.89)	1.46 (±0.48)	0.91 (±0.11)	0.9 (±0.09)	0.88 (±0.22)	0.94 (±0.11)	0.84 (±0.15)	0.41 (±0.2)
	Hidden Replay (ER Only)	−7.9 (±5.11)	1.14 (±0.4)	0.75 (±0.29)	0.92 (±0.09)	0.87 (±0.11)	0.9 (±0.11)	0.8 (±0.13)	0.37 (±0.2)
	Hidden Replay (ER + RaR + GR)	−6.13 (±4.64)	1.85 (±0.88)	0.87 (±0.13)	0.91 (±0.09)	0.86 (±0.13)	0.89 (±0.13)	0.82 (±0.15)	0.55 (±0.26)
Condensed	Random	−0.03 (±0.28) •	1.01 (±0.02)	1.01 (±0.02)	0.54 (±0.02) ○	0.31 (±0.02) ○	0.3 (±0.02) ○	0.31 (±0.00) ○	0.3 (±0.04) ○
	Baseline VTrace (No Sleep)	−3.41 (±1.7)	1.19 (±0.08)	1.17 (±0.29)	1.14 (±0.06)	0.64 (±0.06)	0.74 (±0.06)	0.66 (±0.06)	0.49 (±0.06)
	Sequential (ER + GR)	−3.7 (±1.58)	1.15 (±0.04)	1.0 (±0.09)	0.91 (±0.09) ○	0.53 (±0.13)	0.69 (±0.09)	0.61 (±0.09)	0.49 (±0.11)
	Two-Headed (ER + GR)	−4.53 (±2.97)	1.34 (±0.09)	1.15 (±0.11)	1.09 (±0.13)	0.76 (±0.11)	0.87 (±0.09)	0.78 (±0.09)	0.57 (±0.09)
	Hidden Replay (ER Only)	−8.65 (±3.15) ○	1.42 (±0.15) •	1.12 (±0.11)	1.12 (±0.09)	0.73 (±0.09)	0.89 (±0.07)	0.77 (±0.04)	0.55 (±0.07)
	Hidden Replay (ER + RaR + GR)	−3.05 (±0.97)	1.42 (±0.06) •	1.0 (±0.02)	1.17 (±0.06)	0.8 (±0.04)	0.9 (±0.04)	0.83 (±0.04) •	0.59 (±0.06)

Table 5: Comparison of the different model-free generative replay architectures and the baseline on three different *Starcraft 2* scenarios. The metrics are averaged over 12 lifetimes per scenario and the 95% confidence interval is shown. • denotes statistically significant improvement compared to the baseline with a Kruskal-Wallis test followed by Dunn's test with Bon Ferroni correction w/ a p-value of 0.05. ○ denotes the metric is statistically significantly worse than the baseline. In general, the hidden replay model performs very well, showing significant gains in FT. Interestingly, the baseline agent performs well in learning new tasks, but the continual learners exhibit better performance on unseen tasks, indicating that they exhibit better generalization. RR_Ω also shows the continual learners are able to learn without forgetting.

A.15 Qualitative Analysis of the Generative Model

In this section, we qualitatively evaluate the capabilities of each replay architecture with respect to its generative modeling capabilities. In Fig. 15, we show reconstructions of the first channel of the "CollectMineralShards" SC-2 task using the sequential architecture after 11K sleep batches, imitating a converged wake model. These observations are challenging to reconstruct and generate due to the random distribution of shards, each represented by a few pixels. The sequential architecture produces blurry reconstructions that resemble density maps over the shard locations. In Fig. 16, we show reconstructions on the same task for the two-headed architecture. By separating the generative model from the policy network, the model reconstructs the observations with sharper detail; however, the two-headed model still struggles with generating realistic observations, generating observations that degrade to density maps. While the generated observations are sub-optimal in terms of realism, experiments suggest they still provide useful information for training the policy network. We show additional ground-truth, reconstructions, and generated samples for the "DefeatRoaches" task using the two-headed architecture in Fig. 17. In this case, there is much more structural regularity in the observations, so the reconstructions and generated samples better resemble ground truth data.
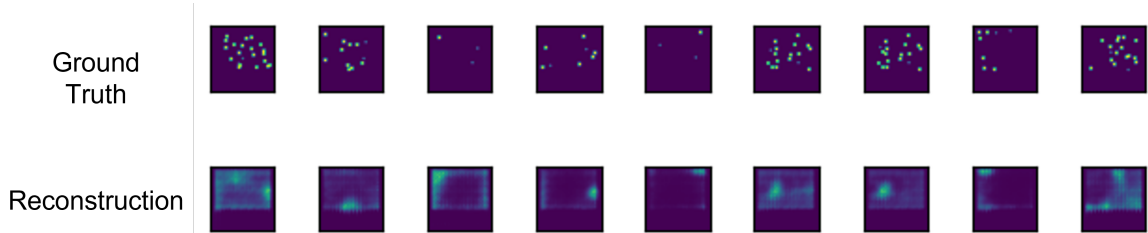

Figure 15: Reconstructions of the first channel of the "CollectMineralShards" SC-2 task using the sequential architecture after 11,000 sleep batches, imitating a converged wake model.
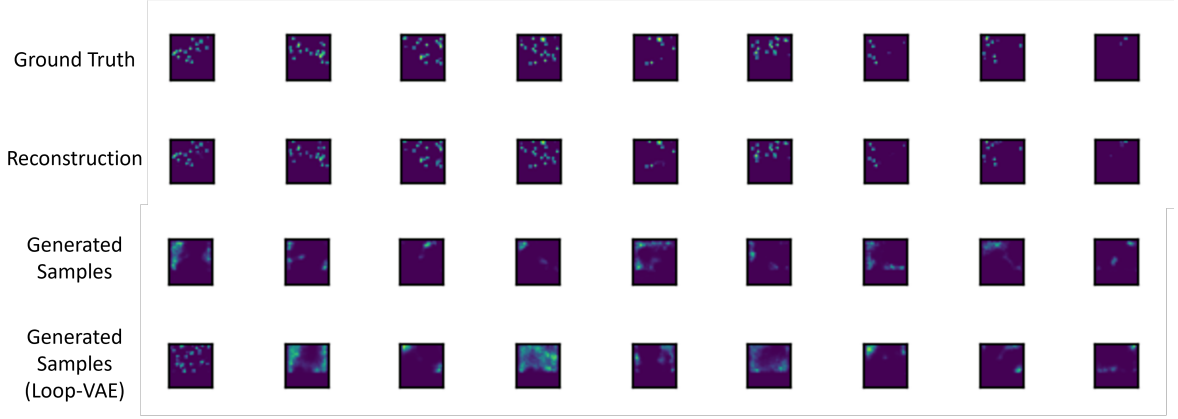
Figure 16: Reconstructions of the first channel of the "CollectMineralShards" SC-2 task using the two-headed archi-tecture after 11,000 sleep batches, imitating a converged wake model. We also show some generated samples and the effect of passing generating via the "loop-vae" procedure.
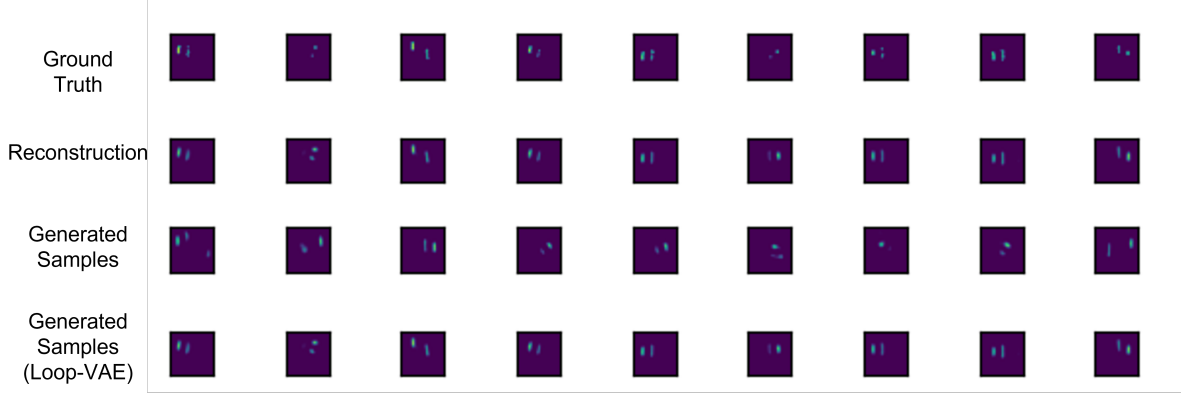

Figure 17: Reconstructions and generated samples of the first channel of the "DefeatRoaches" SC-2 task using the two-headed architecture.

Visualizing the reconstructions and generated samples of the hidden replay model is more challenging. To do so, we use a two-dimensional principle components analysis to compare "ground-truth" features (i.e., the output of the feature extractor) with reconstructions and generated features. On the left side of Fig. 18, the VAE is able to closely recon-struct the ground-truth features for the "DefeatRoaches" task. On the right side, we overlay the feature distribution of "DefeatZerglingsAndBanelings" and the generated samples for "DefeatRoaches" after learning the task sequence "DefeatRoaches" → "DefeatZerglingsAndBanelings". Since these tasks are similar, we see an overlap between the generated features of "DefeatRoaches" from the previous sleep with the new features of "DefeatZerglingsAndBanel-ings", suggesting the generator is generating meaningful features from the task seen during the previous sleep phase.
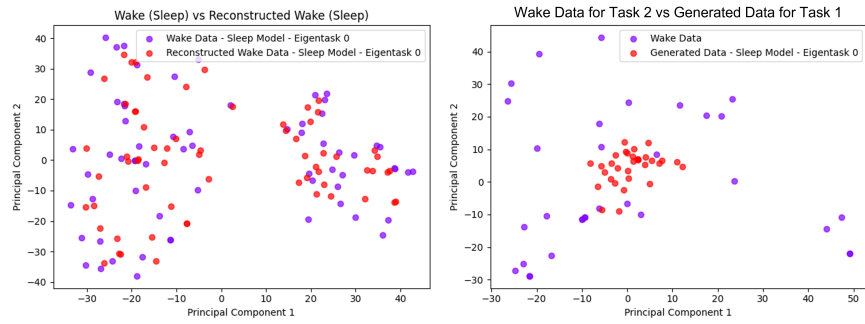
Figure 18: To evaluate reconstructions and generated samples for the hidden replay architecture, we project data in the feature space via its principle components. Left: Reconstructions of the features for the "DefeatRoaches" task. Right: Overlaying the feature distribution of "DefeatZerglingsAndBanelings" and the generated samples for "DefeatRoaches" after learning the task sequence "DefeatRoaches" → "DefeatZerglingsAndBanelings".

A.16 ADDITIONAL RESULTS: UNDERSTANDING THE INTERPLAY BETWEEN DIFFERENT REPLAY MECHANISMS

We've observed that generative replay alone is not sufficient for the hidden replay architecture. This is because there is no constraint limiting the drift in the feature space between sleeps, so the feature extractor can completely change between two time steps if the tasks are perceptually dissimilar or dissimilar in policy. To help to overcome this issue, we employ "random replay" where we save a small batch of random observation-action pairs in the original observation space after every sleep, and randomly replay them during every sleep. In Fig. 19, we show the necessity of combining hidden replay with random replay. On the left, we see that using hidden replay without random replay results in the feature space for task 0 dramatically changing after training on task 1, and PM is low. On the right, we see that the distribution of task 0 before and after training on task 1 is significantly better aligned when hidden replay is used in conjunction with random replay, and PM significantly improves. Additional visualizations for pairs of dissimilar tasks are seen in Fig. 20. In Table 6, we see using random replay generally improves PM and FTR, which agrees with our hypothesis that it helps to maintain the feature space, leading to less forgetting and potentially features that generalize better to other tasks. These results are inline with similar observations made from the main experiments of this paper.

Scenario	Agent	PM	FT	BT	RP
Pairwise	No Random Replay	−6.28 (±12.82)	1.35 (±0.24)	0.80 (±0.37)	1.08 (±0.19)
	Random Replay	−3.03 (±4.92)	1.65 (±0.28)	0.92 (±0.13)	1.04 (±0.17)
Condensed	No Random Replay	−9.35 (±3.69)	1.37 (±0.15)	1.20 (±0.24)	1.16 (±0.16)
	Random Replay	−3.05 (±1.76)	1.42 (±0.11)	1.00 (±0.03)	1.17 (±0.11)

Table 6: Comparison of the hidden replay model with and without random replay. Results are averaged over multiple lifetimes, and we also record the standard deviation of the metrics.
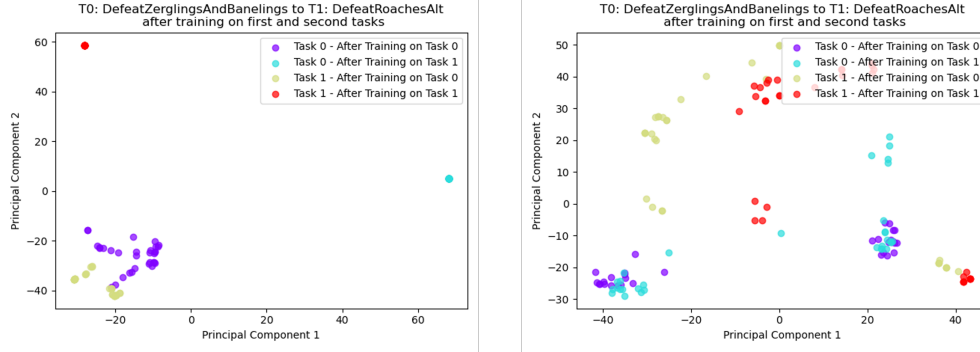
Figure 19: We show the necessity of combining hidden replay with random replay. Left: Using hidden replay without random replay results in the feature space for task 0 dramatically changing after training on task 1, and the feature space for both classes converge to a point. PM is -10.68. Right: The distribution of task 0 before and after training on task 1 are significantly better aligned when hidden replay is used in conjunction with random replay. PM is -1.93.
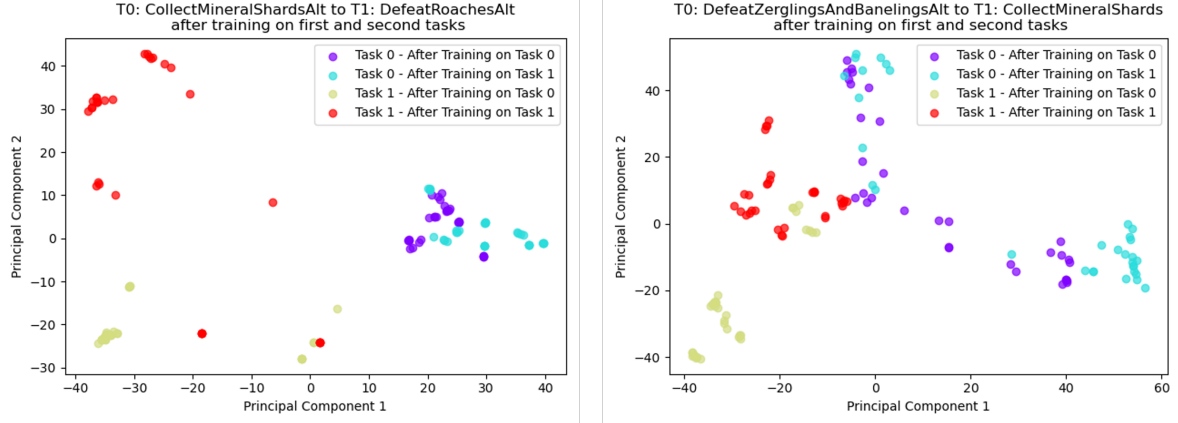

Figure 20: We show two examples where hidden + random replay maintains the feature space of the previous task after learning a new dissimilar SC-2 task ("Collect" → "Defeat" and vice versa).