Learn the Time to Learn: Replay Scheduling in Continual Learning

Marcus Klasson[1] **Hedvig Kjellström**[1,2] **Cheng Zhang**[3]

[1]KTH Royal Institute of Technology, Stockholm, Sweden, {mklas,hedvig}@kth.se
[2]Silo AI, Stockholm, Sweden
[3]Microsoft Research, Cambridge, United Kingdom, cheng.zhang@microsoft.com

Abstract

Replay methods have shown to be successful in mitigating catastrophic forgetting in continual learning scenarios despite having limited access to historical data. However, storing historical data is cheap in many real-world applications, yet replaying all historical data would be prohibited due to processing time constraints. In such settings, we propose learning the time to learn for a continual learning system, in which we learn replay schedules over which tasks to replay at different time steps. To demonstrate the importance of learning the time to learn, we first use Monte Carlo tree search to find the proper replay schedule and show that it can outperform fixed scheduling policies in terms of continual learning performance. Moreover, to improve the scheduling efficiency itself, we propose to use reinforcement learning to learn the replay scheduling policies that can generalize to new continual learning scenarios without added computational cost. In our experiments, we show the advantages of learning the time to learn, which brings current continual learning research closer to real-world needs.

1 Introduction

Many organizations deploying machine learning systems receive large volumes of data daily [5, 38]. Although all historical data are stored in the cloud in practice, retraining machine learning systems on a daily basis is prohibitive both in time and cost. In this setting, the systems often need to continuously adapt to new tasks while retaining the previously learned abilities. Continual learning (CL) methods [23, 70] address this challenge where, in particular, replay methods [18, 37] have shown to be very effective in achieving great prediction performance. Replay methods mitigate catastrophic forgetting by revisiting a small set of samples, which is feasible to process compared to the size of the historical data. In the traditional CL literature, replay memories are limited due to the assumption that historical data are not available. In the real-world setting where historical data are in fact always available, the requirement of small memory remains due to processing time and cost issues.

Recent research on replay-based CL has focused on the quality of memory samples [3, 9, 18, 20, 68, 73, 95] or data compression to increase the memory capacity [37, 44, 72]. Most previous methods allocate equal memory storage space for samples from old tasks, and replay the whole memory to mitigate catastrophic forgetting. However, in life-long learning settings, this simple strategy would be inefficient as the memory must store a large number of tasks. Furthermore, uniform selection policy of samples to revisit is commonly used which ignores the time of which tasks to learn again. This stands in contrast to human learning where education methods focus on scheduling of learning and rehearsal of previous learned knowledge. For example, spaced repetition [24, 28, 57], where the time interval between rehearsal increases, has been shown to enhance memory retention.

We argue that finding the proper schedule of which tasks to replay in the fixed memory setting is critical for CL. To demonstrate our claim, we perform a simple experiment on the Split MNIST [98] dataset where each task consists of learning the digits 0/1, 2/3, etc. arriving in sequence. The replay memory contains data from task 1 and can only be replayed at one point in time. Figure 1 shows how

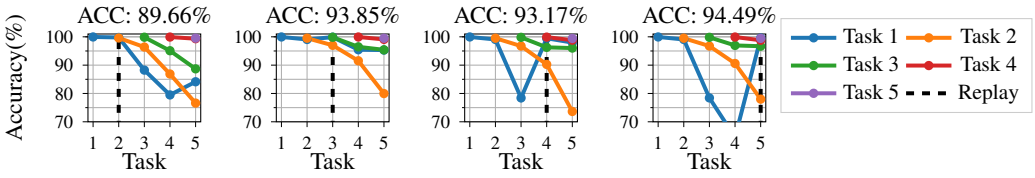


Figure 1: Task accuracies on Split MNIST [98] when replaying only 10 samples of classes $0/1$ at a single time step. The black vertical line indicates when replay is used. ACC denotes the average accuracy over all tasks after learning Task 5. Results are averaged over 5 seeds. These results show that the time to replay the previous task is critical for the final performance.

the task performances progress over time when the memory is replayed at different time steps. In this example, the best final performance is achieved when the memory is used when learning task 5. Note that choosing different time points to replay the same memory leads to noticeably different results in the final performance. These results indicate that scheduling the time when to apply replay can influence the final performance significantly of a CL system.

To this end, we propose learning the time to learn, in which we learn replay schedules of which tasks to replay at different times inspired from human learning [24]. To show the importance of replay scheduling, we first take an episodic-learning approach where a policy is learned from multiple trials selecting which tasks to replay in a CL scenario. In particular, we illustrate in single CL environments by using Monte Carlo tree search (MCTS) [22] as an example method that searches for good replay schedules. The replay schedules from MCTS are evaluated by measuring the final performance of a network trained on a sequence of CL tasks where the scheduled replay samples have been used for mitigating catastrophic forgetting. We use this way to show the importance of replay scheduling given an ideal environment to learn the policy which is infeasible for real-world large scale CL tasks. To enable replay scheduling in real-world CL scenarios, we also propose a framework using reinforcement learning (RL) [82] for learning general policies that can be applied in new CL scenarios without additional training at test time. In summary, our contributions are:

- We propose a new CL setting where historical data is available while the processing time is limited, in order to adjust current CL research closer to real-world needs (Section 3.1). In this new setting, we introduce replay scheduling where we learn the time of which tasks to replay (Section 3.2).
- We argue that learning the time to learn is essential for CL performance. We use MCTS as an example method to illustrate the benefits of replay scheduling in CL, where MCTS searches over finite sets of replay memory compositions at every task (Section 3.3). We show that the replay schedules from MCTS mitigate catastrophic forgetting efficiently across multiple CL benchmarks for various memory selection and replay methods, and in tiny memory settings (Section 4.1).
- To enable replay scheduling in real-world CL scenarios, we propose an RL-based framework for learning policies that generalize across different CL environments (Section 3.4). We show that the learned policies can efficiently mitigate catastrophic forgetting in CL scenarios with new task orders and datasets unseen during training without added computational cost (Section 4.2).

2 Related Work

In this section, we give a brief overview of various approaches in CL, especially replay methods. We provide more details on the related work, including spaced repetition in human CL [24, 28, 35, 57] and generalization in RL [41, 52, 99, 101], in Appendix E. Traditional CL can be divided into three main areas, namely regularization-based, architecture-based, and replay-based approaches. Regularization-based methods protect parameters influencing the performance on known tasks from wide changes and use the other parameters for learning new tasks [1, 49, 53, 59, 68, 77, 98]. Architecture-based methods mitigate catastrophic forgetting by maintaining task-specific parameters [29, 61, 76, 78, 93, 94, 96]. Replay methods mix samples from old tasks with the current dataset to mitigate catastrophic forgetting, where the replay samples are either stored in an external memory [2, 3, 9, 18, 20, 36, 37, 44, 45, 47, 60, 68, 72, 73, 75, 87, 95] or generated using a generative model [79, 85]. Selecting the time to replay old tasks has mostly been ignored in the literature, with an exception in [2] which replays memory samples that would most interfere with a foreseen parameter update. Our replay scheduling approach

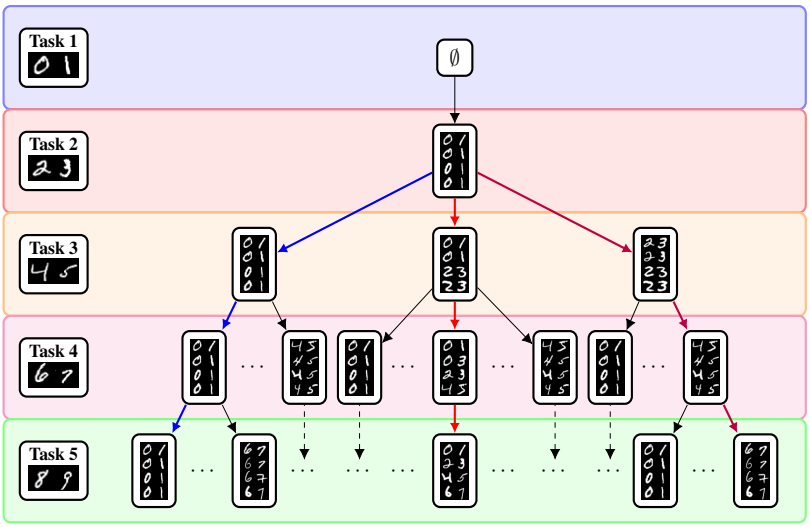


Figure 2: Tree-shaped action space of possible replay memory compositions with size $M = 8$ at every task from the discretization method described in Section 3.2 for Split MNIST.

differs from the above mentioned works since we focus on learning to select which tasks to replay. Nevertheless, our scheduling can be combined with any selection strategy and replay-based method.

3 Method

In this section, we describe our new problem setting of CL where historical data are available while the processing time is limited when learning new tasks. In Section 3.1 and 3.2, we present the considered problem setting, as well as our idea of learning schedules over which tasks to replay at different time steps to mitigate catastrophic forgetting. Section 3.3 describes how we use MCTS [22] for replay scheduling in single CL environments. In Section 3.4, we present a framework based on RL [82] for learning replay scheduling policies that generalize across different CL scenarios.

3.1 Problem Setting

We focus on a slightly new setting, considering the needs for CL in the real-world where all historical data can be available since data storage is cheap. However, as this data volume is typically huge, we are often prohibited from replaying all historical data due to processing time constraints. Therefore, the goal is to determine which historical tasks to revisit and sample a small replay memory from the selected tasks to mitigate catastrophic forgetting as efficiently as possible.

The notation of our problem setting resembles the traditional CL setting for image classification. We let the network $f_{\boldsymbol{\phi}}$, parameterized by $\boldsymbol{\phi}$, learn T tasks sequentially from the datasets $\mathcal{D}_1, \ldots, \mathcal{D}_T$ arriving one at a time. The t-th dataset $\mathcal{D}_t = \{(\boldsymbol{x}_t^{(i)}, y_t^{(i)})\}_{i=1}^{N_t}$ consists of N_t samples where $\boldsymbol{x}_t^{(i)}$ and $y_t^{(i)}$ are the i-th data point and class label respectively. Furthermore, each dataset is split into a training, validation, and test set, i.e., $\mathcal{D}_t = \{\mathcal{D}_t^{(train)}, \mathcal{D}_t^{(val)}, \mathcal{D}_t^{(test)}\}$. The objective at task t is to minimize the loss $\ell(f_{\boldsymbol{\phi}}(\boldsymbol{x}_t), y_t)$ where $\ell(\cdot)$ is the cross-entropy loss in our case. The challenge is for the network $f_{\boldsymbol{\phi}}$ to retain its performance on the previous tasks.

We assume that historical data from old tasks are accessible at any time step t. However, due to processing time constraints, we can only fill a small replay memory $\mathcal{M}$ with M historical samples for replay. The challenge then becomes how to select the M replay samples to efficiently retain knowledge of old tasks. We focus on selecting the samples on task-level by deciding on the task proportion $(p_1, \ldots, p_{t-1})$ of samples to fetch from each task, where $p_i \geq 0$ is the proportion of M samples from task i to place in $\mathcal{M}$ and $\sum_{i=1}^{t-1} p_i = 1$. To simplify the selection of which tasks to replay, we construct a discrete set of possible task proportions that can be used for constructing $\mathcal{M}$.

3.2 Replay Scheduling in Continual Learning

In this section, we describe our setup for enabling the scheduling for selecting replay memories at different time steps. We define a replay schedule as a sequence $S = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{T-1})$, where the task proportions $\boldsymbol{p}_i = (p_1, \ldots, p_{T-1})$ for $1 \leq i \leq T-1$ are used for determining how many samples from seen tasks with which to fill the replay memory at task i. We construct an action space with a discrete number of choices of task proportions that can be selected at each task: At task t, we have $t-1$ historical tasks that we can choose samples from. We create $t-1$ bins $\boldsymbol{b}_t = [b_1, \ldots, b_{t-1}]$ and sample a task index for each bin $b_i \in \{1, \ldots, t-1\}$. The bins are treated as interchangeable and we only keep the unique choices. For example, at task 3, we have seen task 1 and 2, so the unique choices of vectors are $[1,1], [1,2], [2,2]$, where $[1,1]$ indicates that all memory samples are from task 1, $[1,2]$ indicates that half memory is from task 1 and the other half are from task etc. We count the number of occurrences of each task index in $\boldsymbol{b}_t$ and divide by $t-1$ to obtain the task proportion, i.e., $\boldsymbol{p}_t = \texttt{bincount}(\boldsymbol{b}_t)/(t-1)$. We round the number of replay samples from task i, i.e., $p_i \cdot M$, up or down accordingly to keep the memory size M fixed when filling the memory. From this specification, we can build a tree of different replay schedules to evaluate with the network.

Figure 2 shows an example of a replay schedule tree with Split MNIST [98] where the memory size is $M = 8$. Each level corresponds to a task to learn and we show some examples of possible replay memories in the tree that can be evaluated at each task. A replay schedule is represented as a path traversal of different replay memory compositions from task 1 to task 5. At task 1, the memory $\mathcal{M}_1 = \emptyset$ is empty, while $\mathcal{M}_2$ is filled with samples from task 1 at task 2. The memory $\mathcal{M}_3$ can be composed with samples from either task 1 or 2, or equally fill $\mathcal{M}_3$ with samples from both tasks. All possible paths in the tree are valid replay schedules. We show three examples of possible schedules in Figure 2 for illustration: the blue path represents a replay schedule where only task 1 samples are replayed. The red path represents using memories with equally distributed tasks, and the purple path represents a schedule where the memory is only filled with samples from the most previous task.

3.3 Monte Carlo Tree Search for Replay Schedules

The tree-shaped action space of task proportions described in Section 3.2 grows fast with the number of tasks, which complicates studying replay scheduling in datasets with longer task-horizons. Thus, even for a single CL environment, the search space is too big for using exhaustive searches. To this end, we propose to use MCTS since it has been successful in applications with large action spaces [10, 19, 33, 80]. In our case, MCTS concentrates the search for replay schedules in directions with promising CL performance in the environment. We use MCTS in single CL environments for demonstration purposes to show that replay scheduling can be critical for the CL performance.

Each memory composition in the action space corresponds to a node that can be visited by MCTS. For example, in Figure 2, the nodes correspond to the possible memory examples which can be visited during the MCTS rollouts. One rollout corresponds to a tree traversal through all tree levels $1, \ldots, T$ to select the replay schedule S to use during the CL training. At level t, the node v_t is related to a task proportions $\boldsymbol{p}_t$ used for retrieving a replay memory from the historical data at task t. We store the task proportion $\boldsymbol{p}_t$ from every visited node v_t in the replay schedule S during the rollout. With the final replay schedule S, we start the CL training where S is used for constructing the replay memories at each task. Next, we briefly outline the MCTS steps for performing the replay schedule search (more details in Appendix D.1):

Selection. During a rollout, the current node v_t either moves randomly to unvisited children, or selects the next node by evaluating the Upper Confidence Tree (UCT) [54] if all children has been visited earlier. The child v_{t+1} with the highest UCT score is selected using the function from [19]:

$$UCT(v_t, v_{t+1}) = \max(q(v_{t+1})) + C\sqrt{\frac{2\log(n(v_t))}{n(v_{t+1})}}, \tag{1}$$

where $q(\cdot)$ is the reward function, C the exploration constant, and $n(\cdot)$ the number of node visits.
Expansion. Whenever the current node v_t has unvisited child nodes, the search tree is expanded with one of the unvisited child nodes v_{t+1} selected with uniform sampling.
Simulation and Reward. After expansion, the succeeding nodes are selected randomly until reaching a terminal node v_T. The task proportions from the visited nodes in the rollout constitutes the replay schedule S. After training the network using S for replay, we calculate the reward for the rollout given by $r = \frac{1}{T}\sum_{i=1}^{T} A_{T,i}^{(val)}$, where $A_{T,i}^{(val)}$ is the validation accuracy of task i at task T.

Backpropagation. Reward r is backpropagated from the expanded node v_t to the root v_1, where the reward function $q(\cdot)$ and number of visits $n(\cdot)$ are updated at each node.

3.4 Policy Learning Framework for Replay Scheduling

In real-world CL settings, learning the policy with multiple rollouts is infeasible. Thus, we need a general policy for memory scheduling. We now describe our RL-based framework for learning replay scheduling policies that generalize across different CL environments. Our intuition is that there may exist general patterns regarding the replay scheduling, e.g., tasks that are harder or have been forgotten should be replayed more often. Moreover, the policy may non-trivially take task properties into consideration. Therefore, we aim to learn policies that select which tasks to replay from states representing the current task performance in the CL environments. The policy can then be applied for mitigating catastrophic forgetting in new CL scenarios.

We model the CL environments as Markov Decision Processes [8] (MDPs) where each MDP is represented as a tuple $E_i = (\mathcal{S}_i, \mathcal{A}, P_i, R_i, \mu_i, \gamma)$ consisting of the state space $\mathcal{S}_i$, action space $\mathcal{A}$, state transition probability $P_i(s'|s,a)$, reward function $R_i(s,a)$, initial state distribution $\mu_i(s_1)$, and discount factor γ. We assume that we have access to a fixed set of training environments $\mathcal{E}^{(train)} = \{E_1, \dots, E_K\}$ sampled from a distribution of CL environments, i.e., $E_i \sim p(E)$ for $i = 1, ..., K$. Each environment E_i contains of network $f_{\boldsymbol{\phi}}$ and T datasets $\mathcal{D}_{1:T}$ where the t-th dataset is learned at time step t. To generate diverse CL environments, we get environments with different network initializations of $f_{\boldsymbol{\phi}}$ and shuffled task orders in the dataset when we sample environments from $p(E)$. We define the state s_t of the environment as the validation accuracies $A^{(val)}_{t,1:t}$ on each seen task $1, ..., t$ from $f_{\boldsymbol{\phi}}$ at task t, i.e., $s_t = [A_{t,1}, ..., A_{t,t}, 0, ..., 0]$, where we use zero-padding on future tasks. The action space $\mathcal{A}$ is constructed as described in Section 3.2, such that the $a_t \in \mathcal{A}$ corresponds to a task proportion $\boldsymbol{p}_t$ used for sampling the replay memory $\mathcal{M}_t$. We use a dense reward based on the average validation accuracies at task t, i.e., $r_t = \frac{1}{t}\sum_{i=1}^{t} A^{(val)}_{t,i}$. The state transition distribution $P_i(s'|s,a)$ represents the dynamics of the environment, which depend on the initialization of $f_{\boldsymbol{\phi}}$ and also the task order in the dataset.

The procedure for training the policy goes as follows: The state s_t is obtained by evaluating the network $f_{\boldsymbol{\phi}}$ on the validation sets $\mathcal{D}^{(val)}_{1:t}$ after learning the t-th task from $\mathcal{D}^{(train)}_{t}$. Action a_t is selected under the policy $\pi_{\boldsymbol{\theta}}(a|s_t)$ parameterized by $\boldsymbol{\theta}$. The action is converted into task proportion $\boldsymbol{p}_t$ for sampling the replay memory $\mathcal{M}_t$ from the historical datasets. We then train classifier $f_{\boldsymbol{\phi}}$ with $\mathcal{D}^{(train)}_{t+1}$ and $\mathcal{M}_t$, and obtain the reward r_{t+1} and the next state s_{t+1} by evaluating $f_{\boldsymbol{\phi}}$ on the validation sets $\mathcal{D}^{(val)}_{1:t+1}$. The collected transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ are used for updating the policy. This procedure is followed until the final task T after which we start a new episode.

We evaluate the learned policy by applying it to mitigate catastrophic forgetting in new CL environments at test time. To foster generalization across environments, we train the policy on multiple environments with different dynamics, e.g., task orders and datasets, to learn from diverse sets of training data. The goal for the agent is to maximize the sum of rewards in each training environment. At test time, the policy is applied on new CL classifiers and datasets in the test environments without added computational cost nor experience collection. In Section 4, we test the policies generalization capability to new CL environments where the task orders and datasets are unseen during training.

4 Experiments

The experiments with our replay scheduling methods are divided into two parts. Firstly, we evaluate the benefits of replay scheduling in single CL environments using MCTS for finding replay schedules. We perform extensive evaluation to show that the replay schedules from MCTS can outperform the baselines across several CL benchmarks and different backbones. Secondly, we evaluate our RL-based framework using DQN [65] and A2C [64] for learning policies that generalize to new CL scenarios. We show that the learned policies can efficiently mitigate catastrophic forgetting in CL environments with new task orders and datasets that are unseen during training. Full details on experimental settings and additional results are in Appendix B and C.

Datasets and Network Architectures. We conduct experiments on several CL benchmark datasets: Split MNIST [58, 98], FashionMNIST [92], Split notMNIST [11], Permuted MNIST [34], Split

Table 1: Performance comparison with ACC between scheduling methods MCTS (Ours), Random, ETS, and Heuristic with various memory selection methods. Memory sizes are $M = 10$ and $M = 100$ for the 5-task and 10/20-task datasets respectively. We report the mean and standard deviation averaged over 5 seeds. Ours performs better or on par with the baselines on most datasets and selection methods, where MoF yields the best performance in general.

		5-task Datasets			10- and 20-task Datasets		
Selection	**Method**	S-MNIST	S-FashionMNIST	S-notMNIST	P-MNIST	S-CIFAR-100	S-miniImagenet
Uniform	Random	95.91 (± 1.56)	95.82 (± 1.45)	92.39 (± 1.29)	72.44 (± 1.15)	53.99 (± 0.51)	48.08 (± 1.36)
	ETS	94.02 (± 4.25)	95.81 (± 3.53)	91.01 (± 1.39)	71.09 (± 2.31)	47.70 (± 2.16)	46.97 (± 1.24)
	Heuristic	96.02 (± 2.32)	97.09 (± 0.62)	91.26 (± 3.99)	**76.68** (± 2.13)	**57.31** (± 1.21)	49.66 (± 1.10)
	Ours	**97.93** (± 0.56)	**98.27** (± 0.17)	**94.64** (± 0.39)	76.34 (± 0.98)	56.60 (± 1.13)	**50.20** (± 0.72)
k-means	Random	94.24 (± 3.20)	96.30 (± 1.62)	91.64 (± 1.39)	74.30 (± 1.43)	53.18 (± 1.66)	49.47 (± 2.70)
	ETS	92.89 (± 3.53)	96.47 (± 0.85)	**93.80** (± 0.82)	69.40 (± 1.32)	47.51 (± 1.14)	45.82 (± 0.92)
	Heuristic	96.28 (± 1.68)	95.78 (± 1.50)	91.75 (± 0.94)	75.57 (± 1.18)	54.31 (± 3.94)	49.25 (± 1.00)
	Ours	**98.20** (± 0.16)	**98.48** (± 0.26)	93.61 (± 0.71)	**77.74** (± 0.80)	**56.95** (± 0.92)	**50.47** (± 0.85)
k-center	Random	96.40 (± 0.68)	95.57 (± 3.16)	92.61 (± 1.70)	71.41 (± 2.75)	48.46 (± 0.31)	44.76 (± 0.96)
	ETS	94.84 (± 1.40)	97.28 (± 0.50)	91.08 (± 2.48)	69.11 (± 1.69)	44.13 (± 1.06)	41.35 (± 1.23)
	Heuristic	94.55 (± 2.79)	94.08 (± 3.72)	92.06 (± 1.20)	74.33 (± 2.00)	50.32 (± 1.97)	44.13 (± 0.95)
	Ours	**98.24** (± 0.36)	**98.06** (± 0.35)	**94.26** (± 0.37)	**76.55** (± 1.16)	**51.37** (± 1.63)	**46.76** (± 0.96)
MoF	Random	95.18 (± 3.18)	95.76 (± 1.41)	91.33 (± 1.75)	77.96 (± 1.84)	61.93 (± 1.05)	54.50 (± 1.33)
	ETS	97.04 (± 1.23)	96.48 (± 1.33)	92.64 (± 0.87)	77.62 (± 1.12)	60.43 (± 1.17)	56.12 (± 1.12)
	Heuristic	96.46 (± 2.41)	95.84 (± 0.89)	93.24 (± 0.77)	77.27 (± 1.45)	55.60 (± 2.70)	52.30 (± 0.59)
	Ours	**98.37** (± 0.24)	**97.84** (± 0.32)	**94.62** (± 0.42)	**81.58** (± 0.75)	**64.22** (± 0.65)	**57.70** (± 0.51)

CIFAR-10 and CIFAR-100 [56], and Split miniImagenet [88]. We randomly sample 15% of training data for each task to use for validation. We use multi-head output layers for most datasets and assume task labels are available at test time [86], except for Permuted MNIST where the network uses single-head output. We use a 2-layer MLP with 256 hidden units for Split MNIST, Split FashionMNIST, Split notMNIST, and Permuted MNIST. For Split CIFAR-10 and CIFAR-100, we use the ConvNet from [77, 88]. For Split miniImagenet, we apply the reduced ResNet-18 from [60].

Evaluation Protocol. We use the average test accuracy over all tasks after learning the final task, i.e., ACC $= \frac{1}{T}\sum_{i=1}^{T} A_{T,i}$ where $A_{T,i}$ is the test accuracy of task i after learning task T. We report results evaluated on the test set where the replay schedules are selected from evaluating the validation sets. To assess generalization capability, we use a ranking method based on the ACC between the methods in every test environment for comparison (see Appendix B.4 for more details).

4.1 Results on Replay Scheduling with Monte Carlo Tree Search

We show the importance of replay scheduling in single CL environments using MCTS. We perform extensive evaluation where MCTS is combined with different memory selection and replay methods, varying memory sizes, and show the efficiency of replay scheduling in a memory setting where only 1 example/class is available for replay. We compare MCTS to the following scheduling baselines:

- **Random.** Random policy that randomly selects task proportions from the action space on how to structure the replay memory at every task.
- **Equal Task Schedule (ETS).** Policy that selects equal task proportion such that the replay memory aims to fill the memory with an equal number of samples from every seen task.
- **Heuristic Scheduling (Heuristic).** Heuristic policy that replays tasks with validation accuracy below a certain threshold proportional to the best achieved validation accuracy on the task.

Heuristic scheduling is based on the intuition that forgotten tasks should be replayed. The replay memory is filled with M/k samples per task where k is the number of selected tasks. If $k = 0$, then replay is skipped at the current task.

Combine with Different Memory Selection Methods. We show that our method can be combined with any memory selection method for storing replay samples. In addition to uniform sampling, we apply various memory selection methods commonly used in the CL literature, namely k-means clustering, k-center clustering [68], and Mean-of-Features (MoF) [73]. The replay memory sizes are set to $M = 10$ for the 5-task datasets and $M = 100$ for the 10- and 20-task datasets. Table 1 shows the results across all datasets. We note that using the replay schedule from MCTS outperforms the baselines when using the alternative selection methods, where MoF performs the best on most datasets.

Table 2: Performance comparison with ACC between scheduling methods MCTS (Ours), Random, ETS, and Heuristic combined with replay-based methods HAL, MER, and DER++. Memory sizes are $M = 10$ and $M = 100$ for the 5-task and 10/20-task datasets respectively. We report the mean and standard deviation averaged over 5 seeds. * denotes results where some seed did not converge. Applying MCTS to each method can outperform the same method using the baseline schedules.

		5-task Datasets			10- and 20-task Datasets		
Method	**Schedule**	S-MNIST	S-FashionMNIST	S-notMNIST	P-MNIST	S-CIFAR-100	S-miniImagenet
HAL	Random	97.24 (± 0.70)	86.74 (± 6.05)	93.61 (± 1.31)	88.49 (± 0.99)	36.09 (± 1.77)	38.51 (± 2.22)
	ETS	94.02 (± 4.25)	95.81 (± 3.53)	91.01 (± 1.39)	88.46 (± 0.86)	34.90 (± 2.02)	38.13 (± 1.18)
	Heuristic	97.69 (± 0.19)	*74.16 (± 11.19)	93.64 (± 0.93)	*66.63 (± 28.50)	35.07 (± 1.29)	39.51 (± 1.49)
	Ours	**97.93** (± 0.56)	**98.27** (± 0.17)	**94.64** (± 0.39)	**89.14** (± 0.74)	**40.22** (± 1.57)	**41.39** (± 1.15)
MER	Random	93.07 (± 0.81)	85.53 (± 3.30)	91.13 (± 0.86)	75.90 (± 1.34)	42.96 (± 1.70)	31.48 (± 1.65)
	ETS	92.89 (± 3.53)	96.47 (± 0.85)	**93.80** (± 0.82)	73.01 (± 0.96)	43.38 (± 1.81)	33.58 (± 1.53)
	Heuristic	94.30 (± 2.79)	96.91 (± 0.62)	90.90 (± 1.30)	**83.86** (± 3.19)	40.90 (± 1.70)	**34.22** (± 1.93)
	Ours	**98.20** (± 0.16)	**98.48** (± 0.26)	93.61 (± 0.71)	79.72 (± 0.71)	**44.29** (± 0.69)	32.74 (± 1.29)
DER++	Random	97.90 (± 0.52)	97.10 (± 1.03)	93.29 (± 1.43)	87.89 (± 1.10)	58.49 (± 1.44)	48.40 (± 0.69)
	ETS	97.98 (± 0.52)	98.12 (± 0.40)	94.53 (± 1.02)	85.25 (± 0.88)	52.54 (± 1.06)	41.36 (± 2.90)
	Heuristic	92.35 (± 2.42)	*67.31 (± 21.20)	93.88 (± 1.33)	79.17 (± 2.44)	56.70 (± 1.27)	45.73 (± 0.84)
	Ours	**98.84** (± 0.21)	**98.38** (± 0.43)	**94.73** (± 0.20)	**89.84** (± 0.22)	**59.23** (± 0.83)	**49.45** (± 0.68)

Replay Schedule Visualization. We visualize a learned replay schedule from Split CIFAR-100 with memory size $M = 100$ to gain insights into the behavior of the scheduling policy from MCTS. Figure 3 shows a bubble plot of the task proportions that are used for filling the replay memory at every task. Each circle color corresponds to a historical task and its size represents the proportion of replay samples at the current task. The sum of points in all circles at each column is fixed at all current tasks. We see that the task proportions vary dynamically over time in a sophisticated nonlinear way which would be hard to replace by a heuristic method. Moreover, we can observe space repetition-style scheduling on many tasks, e.g., task 1-3 are replayed with similar proportion at the initial tasks but eventually starts varying the time interval between replay. Also, task 4 and 6 need less replay in their early stages, which could potentially be that they are simpler or correlated with other tasks. We provide a similar visualization for Split MNIST in Figure 7 in Appendix C.2 to bring more insights to the benefits of replay scheduling.

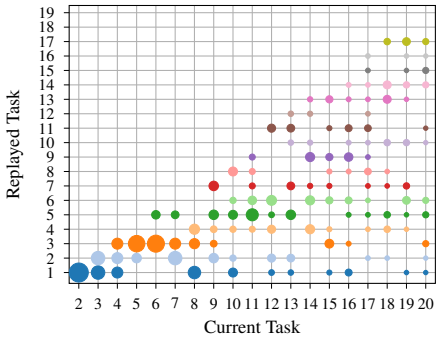


Figure 3: Replay schedule from MCTS on Split CIFAR-100 visualized as bubble plot.

Varying Memory Size. We show that our method can improve the CL performance across different memory sizes. In Figure 4, we observe that MCTS obtains better task accuracies than ETS, especially for small memory sizes. Both MCTS and ETS perform better than Heuristic as M increases, which shows that Heuristic requires careful tuning of the validation accuracy threshold. These results show that replay scheduling can outperform the baselines, especially for small M, on both small and large datasets across different backbone choices.

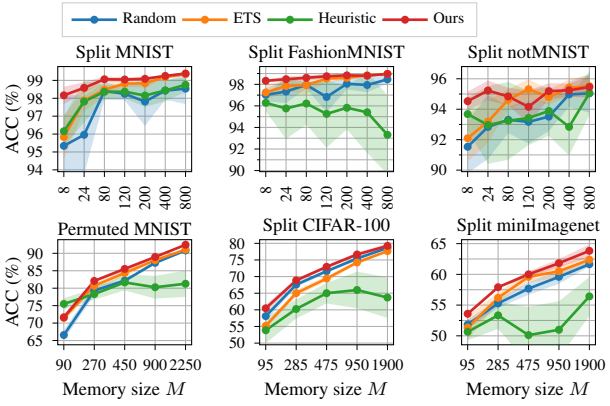


Figure 4: Performace comparison over memory sizes M for MCTS (Ours) and the Random, ETS, and Heuristic baselines. All results have been averaged over 5 seeds.

Applying Scheduling to Recent Replay Methods. In this experiment, we show that replay scheduling can be combined with any replay method to enhance the CL performance. We combine MCTS with Hindsight Anchor Learning (HAL) [16], Meta-Experience Replay (MER) [74], Dark Experience Replay++ (DER++) [12]. We provide the hyperparameter settings in Appendix C.4. Table 2 shows the performance comparison between our the MCTS scheduling

Table 3: Accuracy comparison in the 1 sample/class memory setting evaluated across all datasets. MCTS has memory size $M = 2$ and $M = 50$ for the 5-task and 10/20-task datasets respectively. The baselines replay all available memory samples. We report the mean and standard deviation averaged over 5 seeds. MCTS performs on par with the best baselines on all datasets except S-CIFAR-100.

	5-task Datasets			10- and 20-task Datasets		
Method	S-MNIST	S-FashionMNIST	S-notMNIST	P-MNIST	S-CIFAR-100	S-miniImagenet
Random	92.56 (± 2.90)	92.70 (± 3.78)	89.53 (± 3.96)	70.02 (± 1.76)	48.62 (± 1.02)	48.85 (± 1.38)
A-GEM	94.97 (± 1.50)	94.81 (± 0.86)	92.27 (± 1.16)	64.71 (± 1.78)	42.22 (± 2.13)	32.06 (± 1.83)
ER-Ring	94.94 (± 1.56)	95.83 (± 2.15)	91.10 (± 1.89)	69.73 (± 1.13)	**53.93** (± 1.13)	49.82 (± 1.69)
Uniform	95.77 (± 1.12)	97.12 (± 1.57)	92.14 (± 1.45)	69.85 (± 1.01)	52.63 (± 1.62)	50.56 (± 1.07)
RS-MCTS (Ours)	**96.07** (± 1.60)	**97.17** (± 0.78)	**93.41** (± 1.11)	**72.52** (± 0.54)	51.50 (± 1.19)	**50.70** (± 0.54)

against using Random, ETS, and Heuristic schedules for each method. The results confirm that replay scheduling is important for the final performance given the same memory constraints and it can benefit any existing CL framework.

Efficiency of Replay Scheduling. We illustrate the efficiency of replay scheduling with comparisons to several common replay-based CL baselines when only 1 sample/class is available for replay. We consider CL scenarios where the memory size is even smaller than the number of classes. To this end, we set the replay memory size for our method to $M = 2$ for the 5-task datasets, such that only 2 samples can be selected for replay at all times. For the 10- and 20-task datasets which have 100 classes, we set $M = 50$. We then compare against the memory efficient CL baselines A-GEM [17] and ER-Ring [18], which have shown promising results with 1 sample per class for replay, as well as uniform memory selection as reference. We visualize the memory usage for our method and the baselines in Appendix F. Table 3 shows the ACC for each method across all datasets. Despite using significantly fewer samples for replay, MCTS performs better or on par with the best baselines on most datasets. These results indicate that replay scheduling is an important research direction in CL as storing 1 sample/class in the memory could be inefficient in settings with large number of tasks.

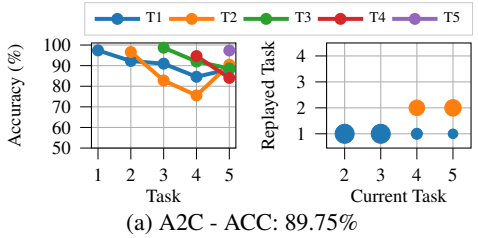
4.2 Policy Generalization to New Continual Learning Scenarios

We show that the policies learned with our RL-based framework using DQN and A2C are capable of generalizing across CL environments with new task orders and datasets unseen during training. In addition to the baselines in Section 4.1, we add two more heuristic scheduling baselines:

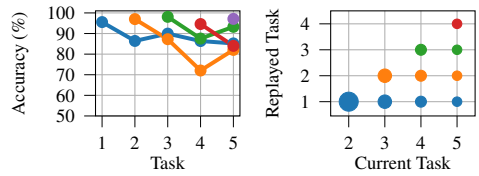
- **Heuristic Local Drop (Heur-LD).** Heuristic policy that replays tasks with validation accuracy below a threshold proportional to the previous achieved validation accuracy on the task.
- **Heuristic Accuracy Threshold (Heur-AT).** Heuristic policy that replays tasks with validation accuracy below a fixed threshold.

Here, we name the Heuristic from Section 4.1 as Heuristic Global Drop (Heur-GD).

Generalization to New Task Orders. We show that the learned replay scheduling policies can generalize to CL environments with previously unseen task orders. We generate training and test environments with unique task orders for three datasets, namely, Split MNIST, FashionMNIST, and CIFAR-10. Table 4 shows the average ranking for the DQN, A2C, and the baselines when being applied in 10 test environments. Our learned policies obtain the best average ranking across the datasets, where the DQN performs best for Split MNIST and FashionMNIST while A2C outperforms all methods on Split CIFAR-10. We provide further insights in the benefits of learning the replay scheduling policy by visualizing the replay schedule and task accuracy progress from A2C in one Split CIFAR-10 test environment in Figure 5. Additionally, we show the replay schedule and task accuracies from the ETS baseline in the same



(a) A2C - ACC: 89.75%



(b) ETS - ACC: 88.32%

Figure 5: Task accuracies and replay schedules for A2C and ETS for a Split CIFAR-10 environment.

Table 4: Average ranking (lower is better) for experiments on generalizing policies to environments with new task orders or a new dataset. We average the results over 10 test environments.

	New Task Order			New Dataset	
Method	S-MNIST	S-FashionMNIST	S-CIFAR-10	S-notMNIST	S-FashionMNIST
Random	4.23	3.60	5.03	3.87	3.95
ETS	3.80	4.57	5.37	4.27	3.63
Heur-GD	4.48	4.25	3.98	4.58	**2.77**
Heur-LD	4.65	3.65	3.75	4.97	5.10
Heur-AT	4.33	3.87	3.43	4.28	3.72
DQN (Ours)	**3.23**	**3.55**	3.68	3.20	4.37
A2C (Ours)	3.27	4.52	**2.75**	**2.83**	4.47

environment. The replay schedules are visualized with bubble plots showing the selected task proportion to use for composing the replay memories at each task. In Figure 5a, we observe that A2C decides to replay task 2 more than task 1 as the performance on task 2 decreases, which results in a slightly better ACC metric achieved by A2C than ETS. These results show that the learned policy can flexibly consider replaying forgotten tasks to enhance the CL performance.

Generalization to New Datasets. We show that the learned replay scheduling policy is capable of generalizing to CL environments with new datasets unseen in the training environments. We perform two sets of experiments, 1) train with environments generated with Split MNIST and FashionMNIST and test on environments generated with Split notMNIST, and 2) train with environments generated with Split MNIST and notMNIST and test on environments generated with Split FashionMNIST. Table 4 shows the average ranking for DQN, A2C, and the baselines when generalization to test environments with new datasets. We observe that both A2C and DQN successfully generalize to Split notMNIST environments outperforming all baselines. However, the learned policies have difficulties to generalize to Split FashionMNIST environments, which could be due to high variations in the dynamics between training and test environments. The policy may exhibit state transition dynamics which has not been experienced during training, which makes generalization difficult for both DQN and A2C. Potentially, the performance could be improved by generating more training environments for the agent to exhibit more variations in the CL scenarios or by using other advanced RL methods which may generalize better [41].

5 Conclusions

We proposed learning the time to learn, i.e., in a real-world CL context, learning schedules of which tasks to replay at different times. To the best of our knowledge, we are the first to consider the time to learn in CL inspired by human learning techniques. We demonstrated the benefits with replay scheduling by showing the performance improvements with the MCTS schedules on several CL benchmarks in single CL environments. To improve the scheduling efficiency, we further proposed an RL-based framework that allows learning policies that can generalize across different CL environments with unseen task orders and datasets without additional computational cost or training in the test environment. Moreover, the learned policies are capable of considering replaying forgotten tasks which can mitigate catastrophic forgetting more efficiently than fixed scheduling policies. The proposed problem setting and replay scheduling approach brings CL research closer to real-world needs, especially in scenarios where CL is applied under limited processing times but with rich amounts of historical data available for replay.

Limitations and Future Work. Generalization in RL is a challenging research topic by itself. With the current method, large amounts of diverse data and training time is required to enable the learned policy to generalize well. This can be costly due to generating the CL environments is expensive since each state transition involves training the classifier on a CL task. Moreover, we are currently considering a discrete action space which is hard to construct especially when the number of tasks is large. Thus, in future work, we would explore more advanced RL methods which can handle continuous action spaces and generalize well.

Acknowledgments

This research was funded by the Promobilia Foundation and the Swedish e-Science Research Centre. We would like to thank Christian Pek, Sofia Broomé, Ruibo Tu, and Truls Nyberg (in alphabetical order) for feedback on the manuscript. We also thank Sam Devlin for early discussions on the reinforcement learning part of the paper. Finally, we would like to thank the GPU cluster admins at RPL (KTH) of Joonatan Mänttäri, Federico Baldassarre, Matteo Gamba, and Yonk Shi.

References

[1] Tameem Adel, Han Zhao, and Richard E Turner. Continual learning with adaptive weights (claw). *arXiv preprint arXiv:1911.09514*, 2019.

[2] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. *Advances in neural information processing systems*, 32, 2019.

[3] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *arXiv preprint arXiv:1903.08671*, 2019.

[4] Hadi Amiri, Timothy Miller, and Guergana Savova. Repeat before forgetting: Spaced repetition for efficient and effective training of neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2401–2410, 2017.

[5] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 541–556, 2017.

[6] Philip J Ball, Yingzhen Li, Angus Lamb, and Cheng Zhang. A study on efficiency in continual learning inspired by human learning. *arXiv preprint arXiv:2010.15187*, 2020.

[7] Philip J Ball, Cong Lu, Jack Parker-Holder, and Stephen Roberts. Augmented world models facilitate zero-shot dynamics generalization from a single offline environment. In *International Conference on Machine Learning*, pages 619–629. PMLR, 2021.

[8] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[9] Zalán Borsos, Mojmír Mutnỳ, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *arXiv preprint arXiv:2006.03875*, 2020.

[10] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[11] Yaroslav Bulatov. The notMNIST dataset. http://yaroslavvb.com/upload/notMNIST/, 2011.

[12] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

[13] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for reinforcement learning. In *International conference on machine learning*, pages 941–950. PMLR, 2019.

[14] Yash Chandak, Georgios Theocharous, Chris Nota, and Philip Thomas. Lifelong learning with a changing action set. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3373–3380, 2020.

[15] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[16] Arslan Chaudhry, Albert Gordo, Puneet Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6993–7001, 2021.

[17] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[18] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[19] Muhammad Umar Chaudhry and Jee-Hyong Lee. Feature selection for high dimensional data using monte carlo tree search. *IEEE Access*, 6:76036–76048, 2018.

[20] Aristotelis Chrysakis and Marie-Francine Moens. Online continual learning from imbalanced data. In *ICML*, 2020.

[21] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[22] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[23] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[24] Frank N Dempster. Spacing effects and their implications for theory and practice. *Educational Psychology Review*, 1(4):309–330, 1989.

[25] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[26] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *Computer Vision – ECCV 2020*, pages 86–102. Springer International Publishing, 2020.

[27] John Dunlosky, Katherine A. Rawson, Elizabeth J. Marsh, Mitchell J. Nathan, and Daniel T. Willingham. Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest*, 14(1):4–58, 2013.

[28] Hermann Ebbinghaus. Memory: A contribution to experimental psychology. *Annals of neurosciences*, 20(4):155, 2013.

[29] Sayna Ebrahimi, Franziska Meier, Roberto Calandra, Trevor Darrell, and Marcus Rohrbach. Adversarial continual learning. *arXiv preprint arXiv:2003.09553*, 2020.

[30] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[31] Kanyin Feng, Xiao Zhao, Jing Liu, Ying Cai, Zhifang Ye, Chuansheng Chen, and Gui Xue. Spaced learning enhances episodic memory by increasing neural pattern similarity across repetitions. *Journal of Neuroscience*, 39(27):5351–5360, 2019.

[32] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[33] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of uct with patterns in monte-carlo go. Technical Report RR-6062, INRIA, 2006. inria-00117266v3f.

[34] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[35] Karri S Hawley, Katie E Cherry, Emily O Boudreaux, and Erin M Jackson. A comparison of adjusted spaced retrieval versus a uniform expanded retrieval schedule for learning a name–face association in older adults with probable alzheimer's disease. *Journal of Clinical and Experimental Neuropsychology*, 30(6):639–649, 2008.

[36] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9769–9776. IEEE, 2019.

[37] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pages 466–483. Springer, 2020.

[38] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629. IEEE, 2018.

[39] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[40] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR, 2017.

[41] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *Advances in neural information processing systems*, 32, 2019.

[42] Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

[43] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[44] Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *European Conference on Computer Vision*, pages 699–715. Springer, 2020.

[45] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32:1, 2018.

[46] Ayush Jain, Andrew Szot, and Joseph J Lim. Generalization to new actions in reinforcement learning. *arXiv preprint arXiv:2011.01928*, 2020.

[47] Xisen Jin, Arka Sadhu, Junyi Du, and Xiang Ren. Gradient based memory editing for task-free continual learning. *arXiv preprint arXiv:2006.15294*, 2020.

[48] KJ Joseph and Vineeth N Balasubramanian. Meta-consolidation for continual learning. *arXiv preprint arXiv:2010.00352*, 2020.

[49] Ta-Chu Kao, Kristopher Jensen, Gido van de Ven, Alberto Bernacchia, and Guillaume Hennequin. Natural continual learning: success is a journey, not (just) a destination. *Advances in Neural Information Processing Systems*, 34, 2021.

[50] Samuel Kessler, Jack Parker-Holder, Philip Ball, Stefan Zohren, and Stephen J Roberts. Same state, different task: Continual reinforcement learning without interference. *arXiv preprint arXiv:2106.02940*, 2021.

[51] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[52] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

[53] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[54] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[55] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[56] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[57] T. Landauer and Robert Bjork. Optimum rehearsal patterns and name learning. *Practical aspects of memory*, 1, 11 1977.

[58] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[59] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[60] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *arXiv preprint arXiv:1706.08840*, 2017.

[61] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[62] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*, 2020.

[63] Seyed Iman Mirzadeh and Hassan Ghasemzadeh. Cl-gym: Full-featured pytorch library for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3621–3627, June 2021.

[64] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[65] 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.

[66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[67] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[68] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[69] Pingbo Pan, Siddharth Swaroop, Alexander Immer, Runa Eschenhagen, Richard E Turner, and Mohammad Emtiyaz Khan. Continual deep learning by functional regularisation of memorable past. *arXiv preprint arXiv:2004.14070*, 2020.

[70] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[71] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[72] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. *arXiv preprint arXiv:1912.01100*, 2019.

[73] 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*, pages 2001–2010, 2017.

[74] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[75] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P Lillicrap, and Greg Wayne. Experience replay for continual learning. *arXiv preprint arXiv:1811.11682*, 2018.

[76] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[77] 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*, pages 4528–4537. PMLR, 2018.

[78] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, 2018.

[79] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*, 2017.

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

[81] Paul Smolen, Yili Zhang, and John H Byrne. The right time to learn: mechanisms and optimization of spaced learning. *Nature Reviews Neuroscience*, 17(2):77, 2016.

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

[83] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[84] 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.

[85] 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.

[86] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[87] Eli Verwimp, Matthias De Lange, and Tinne Tuytelaars. Rehearsal revealed: The limits and merits of revisiting samples in continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9385–9394, 2021.

[88] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080*, 2016.

[89] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[90] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

[91] Judy Willis. Review of research: Brain-based teaching strategies for improving students' memory, learning, and test-taking success. *Childhood Education*, 83(5):310–315, 2007.

[92] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[93] Ju Xu and Zhanxing Zhu. Reinforced continual learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[94] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. *arXiv preprint arXiv:1902.09432*, 2019.

[95] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. *arXiv preprint arXiv:2106.01085*, 2021.

[96] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[97] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

[98] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[99] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[100] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[101] Chenyang Zhao, Olivier Sigaud, Freek Stulp, and Timothy M Hospedales. Investigating generalisation in continuous deep reinforcement learning. *arXiv preprint arXiv:1902.07015*, 2019.

Appendix

This supplementary material is structured as follows:

- Appendix A: Potential negative societal impacts.
- Appendix B: Full details of the experimental settings.
- Appendix C: Additional experimental results.
- Appendix D: Additional methodology of the MCTS method and our RL-based framework for policy learning.
- Appendix E: Extended related work.
- Appendix F: Includes additional figures and tables.

A Potential Negative Societal Impacts

Privacy is one of the main concerns when storing raw input samples for replay in continual learning (CL). However, our replay scheduling framework could store compressed features or use synthetic

data from a generative model for replay rather than the raw samples to mitigate the privacy risks. Moreover, as mentioned in the Limitations (see Section 5), substantial amounts of data and training time are required for learning policies that generalize which increases the need for secure data storage solutions. Our policy learning framework only requires metrics for representing task performances since the policy is incentivized to replay tasks that are to be forgotten. Hence, the policy can be trained on stored state transitions representing how the CL task performances progresses based on selected actions which circumvents the potential privacy issues. Requirements on training time could potentially be reduced with more advanced RL methods that generalize well which we will explore in future work.

B Experimental Settings

In this section, we describe the full details of the experimental settings used in this paper. The experimental settings are divided into two parts where we first describe the settings for single CL environment experiments with MCTS in Section B.1 and then describe the settings for the RL-based framework with DQN [65, 66] and A2C [64] in Section B.2.

B.1 Experimental Settings in Single CL Environments

Here, we provide details on the experimental settings for the experiments with MCTS in single CL environments.

Datasets. We conduct experiments on six datasets commonly used in the CL literature. Split MNIST [98] is a variant of the MNIST [58] dataset where the classes have been divided into 5 tasks incoming in the order 0/1, 2/3, 4/5, 6/7, and 8/9. Split FashionMNIST [92] is of similar size to MNIST and consists of grayscale images of different clothes, where the classes have been divided into the 5 tasks T-shirt/Trouser, Pullover/Dress, Coat/Sandals, Shirt/Sneaker, and Bag/Ankle boots. Similar to MNIST, Split notMNIST [11] consists of 10 classes of the letters A-J with various fonts, where the classes are divided into the 5 tasks A/B, C/D, E/F, G/H, and I/J. We use training/test split provided by [29] for Split notMNIST. Permuted MNIST [34] dataset consists of applying a unique random permutation of the pixels of the images in original MNIST to create each task, except for the first task that is to learn the original MNIST dataset. We reduce the original MNIST dataset to 10k samples and create 9 unique random permutations to get a 10-task version of Permuted MNIST. In Split CIFAR-100 [56], the 100 classes are divided into 20 tasks with 5 classes for each task [60, 73]. Similarly, Split miniImagenet [88] consists of 100 classes randomly chosen from the original Imagenet dataset where the 100 classes are divided into 20 tasks with 5 classes per task.

CL Network Architectures. We use a 2-layer MLP with 256 hidden units and ReLU activation for Split MNIST, Split FashionMNIST, Split notMNIST, and Permuted MNIST. We use a multi-head output layer for each dataset except Permuted MNIST where the network uses single-head output layer. For Split CIFAR-100, we use a multi-head CNN architecture built according to the CNN in [1, 77, 88], which consists of four 3x3 convolutional blocks, i.e. convolutional layer followed by batch normalization [43], with 64 filters, ReLU activations, and 2x2 Max-pooling. For Split miniImagenet, we use the reduced ResNet-18 from [60] with multi-head output layer.

CL Hyperparameters. We train all networks with the Adam optimizer [51] with learning rate $\eta = 0.001$ and hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Note that the learning rate for Adam is not reset before training on a new task. Next, we give details on number of training epochs and batch sizes specific for each dataset:

- Split MNIST: 10 epochs/task, batch size 128.
- Split FashionMNIST: 30 epochs/task, batch size 128.
- Split notMNIST: 50 epochs/task, batch size 128.
- Permuted MNIST: 20 epochs/task, batch size 128.
- Split CIFAR-100: 25 epochs/task, batch size 256.
- Split miniImagenet: 1 epoch/task (task 1 trained for 5 epochs as warm up), batch size 32.

Monte Carlo Tree Search. We run RS-MCTS for 100 iterations in all experiments. The replay schedules used in the reported results on the held-out test sets are from the replay schedule that gave the highest reward on the validation sets. The exploration constant for UCT in Equation 1 is set to $C = 0.1$ in all experiments [19].

Computational Cost. All experiments were performed on one NVIDIA GeForce RTW 2080Ti on an internal GPU cluster. The wall clock time for ETS on Split MNIST was around 1.5 minutes, and RS-MCTS and BFS takes 40 seconds on average to run one iteration, where BFS runs 1050 iterations in total for Split MNIST.

Implementations. We adapted the implementation released by [9] for the memory selection strategies Uniform sampling, k-means clustering, k-center clustering [68], and Mean-of-Features [73]. For HAL [16], MER [74], DER [12], and DER++, we follow the implementations released by [12] for each method to apply them to our replay scheduling methods. Furthermore, we follow the implementations released by [18] and [63] for A-GEM [17] and ER-Ring [18]. For MCTS, we adapted the implementation from `https://github.com/int8/monte-carlo-tree-search` to search for replay schedules.

Experimental Settings for Single Task Replay Memory Experiment. We motivated the need for replay scheduling in CL with Figure 1 in Section 1. This simple experiment was performed on Split MNIST where the replay memory only contains samples from the first task, i.e., learning the classes 0/1. Furthermore, the memory can only be replayed at one point in time and we show the performance on each task when the memory is replayed at different time steps. We set the memory size to $M = 10$ samples such that the memory holds 5 samples from both classes. We use the same network architecture and hyperparameters as described above for Split MNIST. The ACC metric above each subfigure corresponds to the ACC for training a network with the single task memory replay at different tasks. We observe that choosing different time points to replay the same memory leads to noticeably different results in the final performance, and in this example, the best final performance is achieved when the memory is used when learning task 5. Therefore, we argue that finding the proper schedule of what tasks to replay at what time in the fixed memory situation can be critical for CL.

B.2 Experimental Settings for RL-Based Framework

Here, we provide details on the experimental settings for the experiments with our RL-based framework where we use multiple CL environments for learning replay scheduling policies that generalize.

Datasets. We conduct experiments on CL environments with four datasets common CL benchmarks, namely, Split MNIST [98], Split Fashion-MNIST [92], Split notMNIST [11], and Split CIFAR-10 [56]. All datasets consists of 5 tasks with 2 classes/task.

CL Network Architectures. We use a 2-layer MLP with 256 hidden units and ReLU activation for Split MNIST, Split FashionMNIST, and Split notMNIST. For Split CIFAR-10, we use the same ConvNet architecture as used for Split CIFAR-100 in Appendix B.1. We use a multi-head output layer for each dataset and assume task labels are available at test time for selecting the correct output head related to the task.

CL Hyperparameters. We train all networks with the Adam optimizer [51] with learning rate $\eta = 0.001$ and hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Note that the learning rate for Adam is not reset before training on a new task. Next, we give details on number of training epochs and batch sizes specific for each dataset:

- Split MNIST: 10 epochs/task, batch size 128.
- Split FashionMNIST: 10 epochs/task, batch size 128.
- Split notMNIST: 20 epochs/task, batch size 128.
- Split CIFAR-100: 20 epochs/task, batch size 256.

Generating CL Environments. We generate multiple CL environments with pre-set random seeds for initializing the network parameters ϕ and shuffling the task order. The pre-set random seeds are in the range $0 - 49$, such that we have 50 environments for each dataset. We shuffle the task order by permuting the class order and then split the classes into 5 pairs (tasks) with 2 classes/pair. For

environments with seed 0, we keep the original task order in the dataset. Taking a step at task t in the CL environments involves training the CL network on the t-th dataset with a replay memory $\mathcal{M}_t$ from the discrete action space described in Section 3.2. Therefore, to speed up the experiments with the RL algorithms, we run a breadth-first search (BFS) through the discrete action space and save the classification results for re-use during policy learning. Note that the action space has 1050 possible paths of replay schedules for the datasets with $T = 5$ tasks, which makes the environment generation time-consuming. Hence, we only generate environments where the replay memory size $M = 10$ have been used, and leave analysis of different memory sizes as future work.

DQN and A2C Architectures. The input layer has size $T - 1$ where each unit is inputting the task performances since the states are represented by the validation accuracies $s_t = [A_{t,1}^{(val)}, ..., A_{t,t}^{(val)}, 0, ..., 0]$. The current task can therefore be determined by the number of non-zero state inputs. The output layer has 35 units representing the possible actions at $T = 5$ with the discrete action space we have constructed in Section 3.2. We use action masking on the output units to prevent the network from selection invalid actions for constructing the replay memory at the current task. The DQN is a 2-layer MLP with 512 hidden units and ReLU activations. For A2C, we use separate networks for parameterizing the policy and the value function, where both networks are 2-layer MLPs with 64 hidden units of Tanh activations.

DQN and A2C Hyperparameters. We provide the hyperparameters for the both DQN and A2C in Table 5-8. Table 5 and 6 includes the hyperparameters on the New Task Order experiment for DQN and A2C respectively, while Table 7 and 8 includes the hyperparameters on the New Dataset experiment for DQN and A2C respectively. Regarding the training environments in Table 7 and 8, we use two different datasets in the training environments to increase the diversity. When Spit notMNIST is for testing, half the amount of training environments are using Split MNIST and the other half uses Split FashionMNIST. For example, in Table 8, A2C uses 10 training environments which means that there are 5 Split MNIST environments and 5 Split FashionMNIST environments. Similarly, half the amount of training environments are using Split MNIST and the other half uses Split notMNIST when the testing environments uses Split FashionMNIST.

Computational Cost. All experiments were performed on one NVIDIA GeForce RTW 2080Ti on an internal GPU cluster. Generating a CL environment for one seed with Split MNIST took on around 9.5 hours averaged over 10 runs of BFS. Similarly for Split CIFAR-10, generating one CL environment took on average 16.1 hours.

Implementations. The implementation for DQN was adapted from OpenAI baselines [25] and the PyTorch [71] tutorial on DQN https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html. For A2C, we followed the implementations released by Kostrikov [55] and Igl et al. [42].

Table 5: DQN hyperparameters for the experiments on **New Task Orders** in Section 4.2.

Hyperparameters	**Split MNIST**	**Split FashionMNIST**	**Split CIFAR-10**
Training Environments	30	20	10
Learning Rate	0.0001	0.0003	0.0003
Optimizer	Adam	Adam	Adam
Buffer Size	10k	10k	10k
Target Update per step	500	500	500
Batch Size	32	32	32
Discount Factor γ	1.0	1.0	1.0
Exploration Start ϵ_{start}	1.0	1.0	1.0
Exploration Final ϵ_{final}	0.02	0.02	0.02
Exploration Annealing (episodes)	2.5k	2.5k	2.5k
Training Episodes	10k	10k	10k

Table 6: A2C hyperparameters for the experiments on **New Task Orders** in Section 4.2.

Hyperparameters	**Split MNIST**	**Split FashionMNIST**	**Split CIFAR-10**
Training Environments	10	10	10
Learning Rate	0.0001	0.0003	0.00003
Optimizer	RMSProp	RMSProp	RMSProp
Gradient Clipping	0.5	0.5	0.5
GAE parameter λ	0.95	0.95	0.95
VF coefficient	0.5	0.5	0.5
Entropy coefficient	0.01	0.01	0.01
Number of steps n_{steps}	5	5	5
Discount Factor γ	1.0	1.0	1.0
Training Episodes	100k	100k	100k

Table 7: DQN hyperparameters for the experiments on **New Dataset** in Section 4.2. Split notMNIST and Split FashionMNIST indicate the dataset used in the test environments.

Hyperparameters	**Split notMNIST**	**Split FashionMNIST**
Training Environments	30	30
Learning Rate	0.0001	0.0001
Optimizer	Adam	Adam
Buffer Size	10k	10k
Target Update per step	500	500
Batch Size	32	32
Discount Factor γ	1.0	1.0
Exploration Start ϵ_{start}	1.0	1.0
Exploration Final ϵ_{final}	0.02	0.02
Exploration Annealing (episodes)	2.5k	2.5k
Training Episodes	10k	10k

Table 8: A2C hyperparameters for the experiments on **New Dataset** in Section 4.2. Split notMNIST and Split FashionMNIST indicate the dataset used in the test environments.

Hyperparameters	**Split notMNIST**	**Split FashionMNIST**
Training Environments	10	10
Learning Rate	0.0001	0.0003
Optimizer	RMSProp	RMSProp
Gradient Clipping	0.5	0.5
GAE parameter λ	0.95	0.95
VF coefficient	0.5	0.5
Entropy coefficient	0.01	0.01
Number of steps n_{steps}	5	5
Discount Factor γ	1.0	1.0
Training Episodes	100k	100k

B.3 Heuristic Scheduling Baselines

We implemented three heuristic scheduling baselines to compare against our proposed methods. These heuristics are based on the intuition of re-learning tasks when they have been forgotten. We keep a validation set for each task to determine whether any task should be replayed by comparing the validation accuracy against a hand-tuned threshold. If the validation accuracy is below the threshold, then the corresponding task is replayed. Let $A_{t,i}^{(val)}$ be the validation accuracy for task t evaluated at time step i. The threshold is set differently in each of the baselines:

- **Heuristic Global Drop (Heur-GD).** Heuristic policy that replays tasks with validation accuracy below a certain threshold proportional to the best achieved validation accuracy on the task. The best achieved validation accuracy for task i is given by $A_{t,i}^{(best)} = \max\{(A_{1,i}^{(val)}, \ldots, A_{t,i}^{(val)})\}$. Task i is replayed if $A_{t,i}^{(val)} < \tau A_{t,i}^{(best)}$ where $\tau \in [0, 1]$ is a ratio representing the degree of how much the validation accuracy of a task is allowed to drop. Note that Heur-GD (denoted as Heuristic) is the only one used in the experiments with MCTS in single CL environments in Section 4.1.
- **Heuristic Local Drop (Heur-LD).** Heuristic policy that replays tasks with validation accuracy below a threshold proportional to the previous achieved validation accuracy on the task. Task i is replayed if $A_{t,i}^{(val)} < \tau A_{t-1,i}^{(val)}$ where τ again represents the degree of how much the validation accuracy of a task is allowed to drop.
- **Heuristic Accuracy Threshold (Heur-AT).** Heuristic policy that replays tasks with validation accuracy below a fixed threshold. Task i is replayed if if $A_{t,i}^{(val)} < \tau$ where $\tau \in [0, 1]$ represents the least tolerated accuracy before we need to replay the task.

The replay memory is filled with M/k samples from each selected task, where k is the number of tasks that need to be replayed according to their decrease in validation accuracy. We skip replaying any tasks if no tasks are selected for replay, i.e., $k = 0$.

Grid search for τ in Single CL Environments. We performed a coarse-to-fine grid search for the parameter τ on each dataset to compare against the MCTS replay schedules. The best value for τ is selected according to the highest mean accuracy on the validation set averaged over 5 seeds. The validation set consists of 15% of the training data and is the same for MCTS. We use the same experimental settings as described in Appendix B. The memory sizes are set to $M = 10$ and $M = 100$ for the 5-task datasets and the 10/20-task datasets respectively, and we apply uniform sampling as the memory selection method. We provide the ranges for τ that was used on each dataset and put the best value in **bold**:

- Split MNIST: $\tau = \{0.9, 0.93, 0.95, \mathbf{0.96}, 0.97, 0.98, 0.99\}$
- Split FashionMNIST: $\tau = \{0.9, 0.93, 0.95, 0.96, \mathbf{0.97}, 0.98, 0.99\}$
- Split notMNIST: $\tau = \{0.9, 0.93, 0.95, 0.96, 0.97, \mathbf{0.98}, 0.99\}$
- Permuted MNIST: $\tau = \{0.5, 0.55, 0.6, 0.65, 0.7, \mathbf{0.75}, 0.8, 0.9, 0.95, 0.97, 0.99\}$
- Split CIFAR-100: $\tau = \{0.3, 0.4, 0.45, \mathbf{0.5}, 0.55, 0.6, 0.65, 0.7, 0.8, 0.9, 0.95, 0.97, 0.99\}$
- Split miniImagenet: $\tau = \{0.5, 0.6, 0.65, 0.7, \mathbf{0.75}, 0.8, 0.85, 0.9, 0.95, 0.97, 0.99\}$

Note that we use these values for τ on all experiments with Heuristic for the corresponding datasets. The performance for this heuristic highly depends on careful tuning for the ratio τ when the memory size or memory selection method changes, as can be seen in in Figure 4 and Table 1. Note that this heuristic corresponds to Heur-GD described above.

Grid search for τ in Multiple CL Environments. We performed a grid search for the parameter τ for the three heuristic scheduling baselines for each experiment to compare against the learned replay scheduling policies. We select the parameter based on ACC scores achieved in the same number of training environments used by either DQN or A2C. The search range we use is $\tau \in \{0.90, 0.95, 0.999\}$. In Table 9, we show the selected parameter value of τ and the number of environments used for selecting the value for each method and experiment in Section 4.2. The same parameters are used to generate the results on the heuristics in Table 4.

Table 9: The threshold parameter τ used in the heuristic scheudling baselines Heuristic Global Drop (Heur-GD), Heuristic Local Drop (Heur-LD), and Heuristic Accuracy Threshold (Heur-AT). The search range is $\tau \in \{0.90, 0.95, 0.999\}$ for all methods and we display the number of environments used for selecting the parameter used at test time.

	New Task Order						**New Dataset**			
	S-MNIST		**S-FashionMNIST**		**S-CIFAR-10**		**S-notMNIST**		**S-FashionMNIST**	
Method	τ	#Envs	τ	#Envs	τ	#Envs	τ	#Envs	τ	#Envs
Heur-GD	0.9	10	0.95	20	0.9	10	0.9	10	0.9	10
Heur-LD	0.9	10	0.999	20	0.999	10	0.95	10	0.999	10
Heur-AT	0.9	10	0.999	20	0.9	10	0.9	10	0.95	10

B.4 Assessing Generalization with Ranking Method

We use a ranking method based on the CL performance in every test environment for performance comparison between the methods in Section 4.2. We use rankings because the performances can vary greatly between environments with different task orders and datasets. To measure the CL performance in the environments, we use the average test accuracy over all tasks after learning the final task, i.e.,

$$\text{ACC} = \frac{1}{T}\sum_{i=1}^{T} A_{T,i}^{(test)},$$

where $A_{t,i}^{(test)}$ is the test accuracy of task i after learning task t. Each method are ranked in descending order based on the ACC achieved in an environment. For example, assume that we want to compare the CL performance from using learned replay scheduling policies with DQN and A2C against a Random scheduling policy in one environment. The CL performances achieved for each method are given by

$$[\text{ACC}_{\text{Random}}, \text{ACC}_{\text{DQN}}, \text{ACC}_{\text{A2C}}] = [90\%, 99\%, 95\%].$$

We get the following ranking order between the methods based on their corresponding ACC:

$$\texttt{ranking}([\text{ACC}_{\text{Random}}, \text{ACC}_{\text{DQN}}, \text{ACC}_{\text{A2C}}]) = [3, 1, 2],$$

where DQN is ranked in 1st place, A2C in 2nd, and Random in 3rd. When there are multiple environments for evaluation, we compute the average ranking across the ranking positions in every environment for each method to compare.

The average ranking for DQN and A2C are computed over the seed for initializing the network parameters as well as the seed of the environment. Similarly, the Random baseline is affected by the seed setting the random selection of actions and the environment seed. However, the performance of the ETS and Heuristic baselines are affected by the seed of the environment as these policies are fixed. We use copied values of the performance in environments for the ETS and Heuristic baselines when we need to compare across different random seeds for Random, DQN, and A2C. We show an example of such ranking calculation for ETS, a Heuristic baseline, DQN, and A2C. Consider the following performances for one environment:

$$\begin{bmatrix} \text{ACC}_{\text{ETS}}^1 & \text{ACC}_{\text{Heur}}^1 & \text{ACC}_{\text{DQN}}^1 & \text{ACC}_{\text{A2C}}^1 \\ \text{ACC}_{\text{ETS}}^2 & \text{ACC}_{\text{Heur}}^2 & \text{ACC}_{\text{DQN}}^2 & \text{ACC}_{\text{A2C}}^2 \end{bmatrix} = \begin{bmatrix} 90\% & 95\% & 95\% & 99\% \\ * & * & 97\% & 98\% \end{bmatrix},$$

where $*$ denotes a copy of the ACC value in the first row. The subscript on ACC denotes the method and the superscript the seed used for initializing the policy network $\boldsymbol{\theta}$. Therefore, we copy the values for ETS and Heur such that the $\text{ACC}_{\text{DQN}}^2$ for seed 2 can be compared against ETS and Heur. Note that there is a tie between $\text{ACC}_{\text{Heur}}^1$ and $\text{ACC}_{\text{DQN}}^1$ as they have ACC 95%. We handle ties by assigning tied methods the average of their ranks, such that the ranks for both seeds will be

$$
\begin{aligned}
&\texttt{ranking}\left(\begin{bmatrix} \text{ACC}^1_{\text{ETS}} & \text{ACC}^1_{\text{Heur}} & \text{ACC}^1_{\text{DQN}} & \text{ACC}^1_{\text{A2C}} \\ \text{ACC}^2_{\text{ETS}} & \text{ACC}^2_{\text{Heur}} & \text{ACC}^2_{\text{DQN}} & \text{ACC}^2_{\text{A2C}} \end{bmatrix}, \texttt{axis=-1, keepdim=True}\right) \\
=&\texttt{ranking}\left(\begin{bmatrix} 90\% & 95\% & 95\% & 99\% \\ 90\% & 95\% & 97\% & 98\% \end{bmatrix}, \texttt{axis=-1, keepdim=True}\right) \\
=&\begin{bmatrix} 4 & 2.5 & 2.5 & 1 \\ 4 & 3 & 2 & 1 \end{bmatrix},
\end{aligned}
$$

where we inserted the copied values, such that $\text{ACC}^1_{\text{ETS}} = \text{ACC}^2_{\text{ETS}} = 90\%$ and $\text{ACC}^1_{\text{Heur}} = \text{ACC}^2_{\text{Heur}} = 95\%$. The mean ranking across the seeds thus becomes

$$
\texttt{mean}\left(\begin{bmatrix} 4 & 2.5 & 2.5 & 1 \\ 4 & 3 & 2 & 1 \end{bmatrix}, \texttt{axis=0}\right) = [4 \quad 2.75 \quad 2.25 \quad 1]
$$

where A2C comes in 1st place, DQN in 2nd, Heur. in 3rd, and ETS on 4th place. We average across seeds and environments to obtain the final ranking score for each method for comparison.

C Additional Experimental Results

In this section, we bring more insights to the benefits of replay scheduling in Section C.2 as well as provide metrics for catastrophic forgetting in Section C.3.

C.1 Performance Progress of MCTS

In the first experiments, we show that the replay schedules from MCTS yield better performance than replaying an equal amount of samples per task. The replay memory size is fixed to $M = 10$ for Split MNIST, FashionMNIST, and notMNIST, and $M = 100$ for Permuted MNIST, Split CIFAR-100, and Split miniImagenet. Uniform sampling is used as the memory selection method for all methods in this experiment. For the 5-task datasets, we provide the optimal replay schedule found from a breadth-first search (BFS) over all 1050 possible replay schedules in our action space (which corresponds to a tree with depth of 4) as an upper bound for MCTS. As the search space grows fast with the number of tasks, BFS becomes computationally infeasible when we have 10 or more tasks.

Figure 6 shows the progress of ACC over iterations by MCTS for all datasets. We also show the best ACC metrics for Random, ETS, Heuristic, and BFS (where appropriate) as straight lines. Furthermore, we include the ACC achieved by training on all seen datasets jointly at every task (Joint) for the 5-task datasets. We observe that MCTS outperforms Random and ETS successively with more iterations. Furthermore, MCTS approaches the upper limit of BFS on the 5-task datasets. For Permuted MNIST and Split CIFAR-100, the Heuristic baseline and MCTS perform on par after 50 iterations. This shows that Heuristic with careful tuning of the validation accuracy threshold can be a strong baseline when comparing replay scheduling methods. The top row of Table 1 shows the ACC for each method for this experiment. We note that MCTS outperforms ETS significantly on most datasets and performs on par with Heuristic.

C.2 Replay Schedule Visualization for Split MNIST

In Figure 7, we show the progress in test classification performance for each task when using ETS and MCTS with memory size $M = 10$ on Split MNIST. For comparison, we also show the performance from a network that is fine-tuning on the current task without using replay. Both ETS and MCTS overcome catastrophic forgetting to a large degree compared to the fine-tuning network. Our method MCTS further improves the performance compared to ETS with the same memory, which indicates that learning the time to learn can be more efficient against catastrophic forgetting. Especially, Task 1 and 2 seems to be the most difficult task to remember since it has the lowest final performance using the fine-tuning network. Both ETS and MCTS manage to retain their performance on Task 1 using replay, however, MCTS remembers Task 2 better than ETS by around 5%.

To bring more insights to this behavior, we have visualized the task proportions of the replay examples using a bubble plot showing the corresponding replay schedule from MCTS in Figure 7(right). At Task 3 and 4, we see that the schedule fills the memory with data from Task 2 and discards replaying

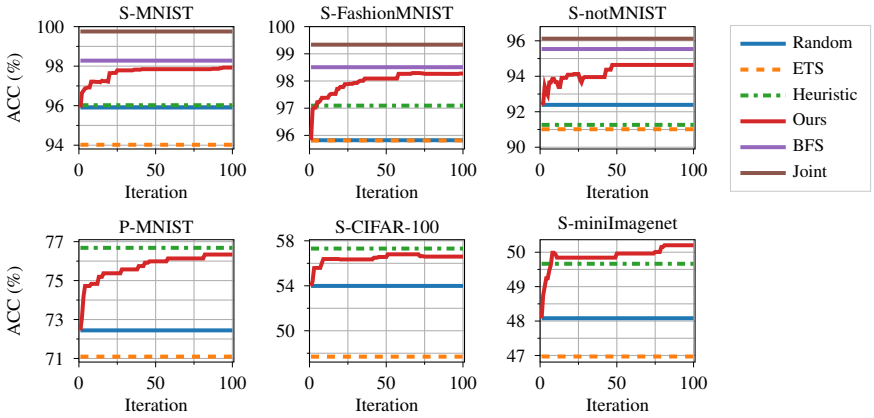


Figure 6: Average test accuracies over tasks after learning the final task (ACC) over the MCTS simulations for all datasets, where 'S' and 'P' are used as short for 'Split' and 'Permuted'. We compare performance for MCTS (Ours) against random replay schedules (Random), Equal Task Schedule (ETS), and Heuristic Scheduling (Heuristic) baselines. For the first three datasets, we show the best ACC found from a breadth-first search (BFS) as well as the ACC achieved by training on all seen datasets jointly at every task (Joint). All results have been averaged over 5 seeds. These results show that replay scheduling can improve over ETS and outperform or perform on par with Heuristic across different datasets and network architectures.

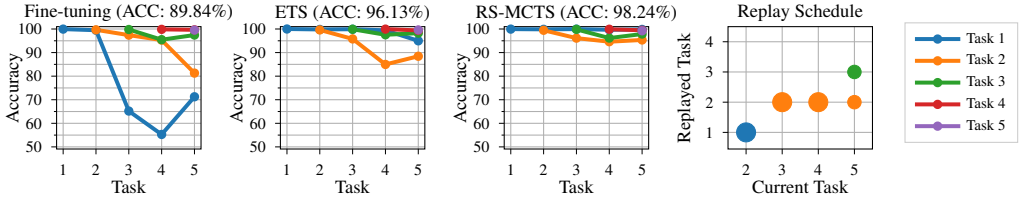


Figure 7: Comparison of test classification accuracies for Task 1-5 on Split MNIST from a network trained without replay (Fine-tuning), ETS, and MCTS. The ACC metric for each method is shown on top of each figure. We also visualize the replay schedule found by MCTS as a bubble plot to the right. The memory size is set to $M = 10$ with uniform memory selection for ETS and MCTS. Results are shown for 1 seed.

Task 1. This helps the network to retain knowledge about Task 2 better than ETS at the cost of forgetting Task 3 slightly when learning Task 4. This shows that the learned policy has considered the difficulty level of different tasks. At the next task, the MCTS schedule has decided to rehearse Task 3 and reduces replaying Task 2 when learning Task 5. This behavior is similar to spaced repetition, where increasing the time interval between rehearsals helps memory retention. We emphasize that even on datasets with few tasks, using learned replay schedules can overcome catastrophic forgetting better than standard ETS approaches.

C.3 Analysis of Catastrophic Forgetting

We have compared the degree of catastrophic forgetting for our method against the baselines by measuring the backward transfer (BWT) metric from [60], which is given by

$$\text{BWT} = \frac{1}{T-1} \sum_{i=1}^{T-1} A_{T,i} - A_{i,i}, \tag{2}$$

where $A_{t,i}$ is the test accuracy for task t after learning task i. Table 11 shows the ACC and BWT metrics for the experiments in Section 4.1. In general, the BWT metric is consistently better when the corresponding ACC is better. We find an exception in Table 11 on Split CIFAR-100 and Split miniImagenet between Ours and Heuristic with uniform selection method, where Heuristic has better BWT while its mean of ACC is slightly lower than ACC for Ours. Table 12 shows the ACC and

Table 10: Hyperparameters for replay-based methods HAL, MER, DER and DER++ used in experiments on applying MCTS to recent replay-based methods in Section 4.1.

		5-task Datasets			10- and 20-task Datasets		
Method	**Hyperparam.**	S-MNIST	S-FashionMNIST	S-notMNIST	P-MNIST	S-CIFAR-100	S-miniImagenet
HAL	η	0.1	0.1	0.1	0.1	0.03	0.03
	λ	0.1	0.1	0.1	0.1	1.0	0.03
	γ	0.5	0.1	0.1	0.1	0.1	0.1
	β	0.7	0.5	0.5	0.5	0.5	0.5
	k	100	100	100	100	100	100
MER	γ	1.0	1.0	1.0	1.0	1.0	1.0
	β	1.0	0.01	1.0	1.0	0.1	0.1
DER	α	0.2	0.2	0.1	1.0	1.0	0.1
DER++	α	0.2	0.2	0.1	1.0	1.0	0.1
	β	1.0	1.0	1.0	1.0	1.0	1.0

BWT metrics for the experiments on efficiency of replay scheduling (also in Section 4.1), where we see a similar pattern that better ACC yields better BWT. The BWT of MCTS is on par with the other baselines except on Split CIFAR-100 where the ACC on our method was a bit lower than the best baselines.

C.4 Applying Scheduling to Recent Replay Methods

In Section 4.1, we showed that MCTS can be applied to any replay method. We combined MCTS together with four recent replay methods, namely Hindsight Anchor Learning (HAL) [16], Meta Experience Replay (MER) [74], and Dark Experience Replay (DER) [12]. Table 13 shows the ACC and BWT for all methods combined with the scheduling from Random, ETS, Heuristic, and MCTS. We observe that MCTS can further improve the performance for each of the replay methods across the different datasets. We present the hyperparameters used for each method in Table 10. The hyperparameters for each method are denoted as

- **HAL.** η: learning rate, λ: regularization, γ: mean embedding strength, β: decay rate, k: gradient steps on anchors
- **MER.** γ: across batch meta-learning rate, β: within batch meta-learning rate
- **DER.** α: loss coefficient for memory logits
- **DER++.** α: loss coefficient for memory logits, β: loss coefficient for memory labels

For the experiments, we used the same architectures and hyperparameters as described in Appendix B for all datasets if not mentioned otherwise. We used the Adam optimizer with learning rate $\eta = 0.001$ for MER, DER, and DER++. For HAL, we used the SGD optimizer since using Adam made the model diverge in our experiments.

C.5 Complementary Experimental Results for Policy Generalization

Here, we provide complementary results for the policy generalization experiments in Section 4.2. Figure 8-11 shows the performance progress measured in ACC in the test environments during training for the DQN and A2C. We also show the ACC achieved by the scheduling baselines as straight lines as these policies are fixed. Figure 8 and 9 shows the performance progress for test environments with Split MNIST and Split CIFAR-10 respectively in the New Task Orders experiment. Figure 10 and 11 shows the performance progress for test environments with Split notMNIST and Split FashionMNIST respectively in the New Dataset experiment. In general, we observe that DQN exhibits noisier progress of the achieved ACC in the test environments than A2C.

In Table 14-18, we display the ACC and rank in every test environment that was used for generating the average rankings in Table 4 (see Section 4.2). Table 14, 15, and 16 shows the ACCs and ranks for the New Task Orders experiments with datasets Split MNIST, FashionMNIST, and CIFAR-10 respectively. Table 17 and 18 shows the ACCs and ranks for the New Dataset experiments with datasets Split notMNIST and FashionMNIST respectively. The numbers for the average rankings in Table 4 are computed by averaging over the ranks in each test environment for every method separately.

Table 11: Performance comparison with ACC and BWT metrics for all datasets between MCTS (Ours) and the baselines with various memory selection methods. We provide the metrics for training on all seen task datasets jointly (Joint) as an upper bound. Furthermore, we include the results from a breadth-first search (BFS) with Uniform memory selection for the 5-task datasets. The memory size is set to $M = 10$ and $M = 100$ for the 5-task and 10/20-task datasets respectively. We report the mean and standard deviation of ACC and BWT, where all results have been averaged over 5 seeds. MCTS performs better or on par than the baselines on most datasets and selection methods, where MoF yields the best performance in general.

		Split MNIST		**Split FashionMNIST**		**Split notMNIST**	
Selection	**Method**	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑
–	Joint	99.75 (± 0.06)	0.01 (± 0.06)	99.34 (± 0.08)	-0.01 (± 0.14)	96.12 (± 0.57)	-0.21 (± 0.71)
Uniform	BFS	98.28 (± 0.49)	-1.84 (± 0.63)	98.51 (± 0.23)	-1.03 (± 0.28)	95.54 (± 0.67)	-1.04 (± 0.87)
Uniform	Random	95.91 (± 1.56)	-4.79 (± 1.95)	95.82 (± 1.45)	-4.35 (± 1.79)	92.39 (± 1.29)	-4.56 (± 1.29)
	ETS	94.02 (± 4.25)	-7.22 (± 5.33)	95.81 (± 3.53)	-4.45 (± 4.34)	91.01 (± 1.39)	-6.16 (± 1.82)
	Heuristic	96.02 (± 2.32)	-4.64 (± 2.90)	97.09 (± 0.62)	-2.82 (± 0.84)	91.26 (± 3.99)	-6.06 (± 4.70)
	Ours	97.93 (± 0.56)	-2.27 (± 0.71)	98.27 (± 0.17)	-1.29 (± 0.20)	94.64 (± 0.39)	-1.47 (± 0.79)
k-means	Random	94.24 (± 3.20)	-6.88 (± 4.00)	96.30 (± 1.62)	-3.77 (± 2.05)	91.64 (± 1.39)	-5.64 (± 1.77)
	ETS	92.89 (± 3.53)	-8.66 (± 4.42)	96.47 (± 0.85)	-3.55 (± 1.07)	93.80 (± 0.82)	-2.84 (± 0.81)
	Heuristic	96.28 (± 1.68)	-4.32 (± 2.11)	95.78 (± 1.50)	-4.46 (± 1.87)	91.75 (± 0.94)	-5.60 (± 2.07)
	Ours	98.20 (± 0.16)	-1.94 (± 0.22)	98.48 (± 0.26)	-1.04 (± 0.31)	93.61 (± 0.71)	-3.11 (± 0.55)
k-center	Random	96.40 (± 0.68)	-4.21 (± 0.84)	95.57 (± 3.16)	-7.20 (± 3.93)	92.61 (± 1.70)	-4.14 (± 2.37)
	ETS	94.84 (± 1.40)	-6.20 (± 1.77)	97.28 (± 0.50)	-2.58 (± 0.66)	91.08 (± 2.48)	-6.39 (± 3.46)
	Heuristic	94.55 (± 2.79)	-6.47 (± 3.50)	94.08 (± 3.72)	-6.59 (± 4.57)	92.06 (± 1.20)	-4.70 (± 2.09)
	Ours	98.24 (± 0.36)	-1.93 (± 0.44)	98.06 (± 0.35)	-1.59 (± 0.45)	94.26 (± 0.37)	-1.97 (± 1.02)
MoF	Random	95.18 (± 3.18)	-5.73 (± 3.95)	95.76 (± 1.41)	-4.41 (± 1.75)	91.33 (± 1.75)	-5.94 (± 1.92)
	ETS	97.04 (± 1.23)	-3.46 (± 1.50)	96.48 (± 1.33)	-3.55 (± 1.73)	92.64 (± 0.87)	-4.57 (± 1.59)
	Heuristic	96.46 (± 2.41)	-4.09 (± 3.01)	95.84 (± 0.89)	-4.39 (± 1.15)	93.24 (± 0.77)	-3.48 (± 1.37)
	Ours	98.37 (± 0.24)	-1.70 (± 0.28)	97.84 (± 0.32)	-1.81 (± 0.39)	94.62 (± 0.42)	-1.80 (± 0.56)

		Permuted MNIST		**Split CIFAR-100**		**Split miniImagenet**	
Selection	**Method**	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑
–	Joint	95.34 (± 0.13)	0.17 (± 0.18)	84.73 (± 0.81)	-1.06 (± 0.81)	74.03 (± 0.83)	9.70 (± 0.68)
Uniform	Random	72.44 (± 1.15)	-25.56 (± 1.39)	53.99 (± 0.51)	-34.19 (± 0.66)	48.08 (± 1.36)	-15.98 (± 1.08)
	ETS	71.09 (± 2.31)	-27.39 (± 2.59)	47.70 (± 2.16)	-41.68 (± 2.37)	46.97 (± 1.24)	-18.32 (± 1.34)
	Heuristic	76.68 (± 2.13)	-20.82 (± 2.41)	57.31 (± 1.21)	-30.76 (± 1.45)	49.66 (± 1.10)	-12.04 (± 0.59)
	Ours	76.34 (± 0.98)	-21.21 (± 1.16)	56.60 (± 1.13)	-31.39 (± 1.11)	50.20 (± 0.72)	-13.46 (± 1.22)
k-means	Random	74.30 (± 1.43)	-23.50 (± 1.64)	53.18 (± 1.66)	-35.15 (± 1.61)	49.47 (± 2.70)	-14.10 (± 2.71)
	ETS	69.40 (± 1.32)	-29.23 (± 1.47)	47.51 (± 1.14)	-41.77 (± 1.30)	45.82 (± 0.92)	-19.53 (± 1.10)
	Heuristic	75.57 (± 1.18)	-22.11 (± 1.22)	54.31 (± 3.94)	-33.80 (± 4.24)	49.25 (± 1.00)	-12.92 (± 1.22)
	Ours	77.74 (± 0.80)	-19.66 (± 0.95)	56.95 (± 0.92)	-30.92 (± 0.83)	50.47 (± 0.85)	-13.31 (± 1.24)
k-center	Random	71.41 (± 2.75)	-26.73 (± 3.11)	48.46 (± 0.31)	-39.89 (± 0.27)	44.76 (± 0.96)	-18.72 (± 1.17)
	ETS	69.11 (± 1.69)	-29.58 (± 1.81)	44.13 (± 1.06)	-45.28 (± 1.04)	41.35 (± 0.96)	-23.71 (± 1.45)
	Heuristic	74.33 (± 2.00)	-23.45 (± 2.27)	50.32 (± 1.97)	-37.99 (± 2.14)	44.13 (± 0.95)	-18.26 (± 1.05)
	Ours	76.55 (± 1.16)	-21.06 (± 1.32)	51.37 (± 1.63)	-37.01 (± 1.62)	46.76 (± 0.96)	-16.56 (± 0.90)
MoF	Random	77.96 (± 1.84)	-19.44 (± 2.13)	61.93 (± 1.05)	-25.89 (± 1.07)	54.50 (± 1.33)	-8.64 (± 1.26)
	ETS	77.62 (± 1.12)	-20.10 (± 1.26)	60.43 (± 1.17)	-28.22 (± 1.26)	56.12 (± 1.12)	-8.93 (± 0.83)
	Heuristic	77.27 (± 1.45)	-20.15 (± 1.63)	55.60 (± 2.70)	-32.57 (± 2.77)	52.30 (± 0.59)	-9.61 (± 0.67)
	Ours	81.58 (± 0.75)	-15.41 (± 0.86)	64.22 (± 0.65)	-23.48 (± 1.02)	57.70 (± 0.51)	-5.31 (± 0.55)

Table 12: Performance comparison with ACC and BWT metrics for all datasets between MCTS and the baselines in the setting where only 1 sample per class can be replayed. The memory sizes are set to $M = 10$ and $M = 100$ for the 5-task and 10/20-task datasets respectively. We report the mean and standard deviation of ACC and BWT, where all results have been averaged over 5 seeds. MCTS performs on par with the best baselines for both metrics on all datasets except S-CIFAR-100.

	Split MNIST		**Split FashionMNIST**		**Split notMNIST**	
Method	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑
Random	92.56 (± 2.90)	-8.97 (± 3.62)	92.70 (± 3.78)	-8.24 (± 4.75)	89.53 (± 3.96)	-8.13 (± 5.02)
A-GEM	94.97 (± 1.50)	-6.03 (± 1.87)	94.81 (± 0.86)	-5.65 (± 1.06)	92.27 (± 1.16)	-4.17 (± 1.39)
ER-Ring	94.94 (± 1.56)	-6.07 (± 1.92)	95.83 (± 2.15)	-4.38 (± 2.59)	91.10 (± 1.89)	-6.27 (± 2.35)
Uniform	95.77 (± 1.12)	-5.02 (± 1.39)	97.12 (± 1.57)	-2.79 (± 1.98)	92.14 (± 1.45)	-4.90 (± 1.41)
MCTS (Ours)	96.07 (± 1.60)	-4.59 (± 2.01)	97.17 (± 0.78)	-2.64 (± 0.99)	93.41 (± 1.11)	-3.36 (± 1.56)

	Permuted MNIST		**Split CIFAR-100**		**Split miniImagenet**	
Method	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑
Random	70.02 (± 1.76)	-28.22 (± 1.92)	48.62 (± 1.02)	-39.95 (± 1.10)	48.85 (± 1.38)	-14.55 (± 1.86)
A-GEM	64.71 (± 1.78)	-34.41 (± 2.05)	42.22 (± 2.13)	-46.90 (± 2.21)	32.06 (± 1.83)	-30.81 (± 1.79)
ER-Ring	69.73 (± 1.13)	-28.87 (± 1.29)	53.93 (± 1.13)	-34.91 (± 1.18)	49.82 (± 1.69)	-14.38 (± 1.57)
Uniform	69.85 (± 1.01)	-28.74 (± 1.17)	52.63 (± 1.62)	-36.43 (± 1.81)	50.56 (± 1.07)	-13.52 (± 1.34)
MCTS (Ours)	72.52 (± 0.54)	-25.43 (± 0.65)	51.50 (± 1.19)	-37.01 (± 1.08)	50.70 (± 0.54)	-12.60 (± 1.13)

Table 13: Performance comparison with ACC and BWT metrics between scheduling methods MCTS (Ours), Random, ETS, and Heuristic when combining them with replay-based methods Hindsight Anchor Learning (HAL), Meta Experience Replay (MER), Dark Experience Replay (DER), and DER++. Replay memory sizes are $M = 10$ and $M = 100$ for the 5-task and 10/20-task datasets respectively. We report the mean and standard deviation averaged over 5 seeds. Results on Heuristic where some seed did not converge is denoted by *. Applying MCTS to each method can enhance the performance compared to using the baseline schedules.

		Split MNIST		**Split FashionMNIST**		**Split notMNIST**	
Method	**Schedule**	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑
HAL	Random	97.24 (± 0.70)	-2.77 (± 0.90)	86.74 (± 6.05)	-15.54 (± 7.58)	93.61 (± 1.31)	-2.73 (± 1.33)
	ETS	97.21 (± 1.25)	-2.80 (± 1.59)	96.75 (± 0.50)	-2.84 (± 0.75)	92.16 (± 1.82)	-5.04 (± 2.24)
	Heuristic	97.69 (± 0.19)	-2.22 (± 0.24)	*74.16 (± 11.19)	-31.26 (± 14.00)	93.64 (± 0.93)	-2.80 (± 1.20)
	Ours	97.96 (± 0.15)	-1.85 (± 0.18)	97.56 (± 0.51)	-2.02 (± 0.63)	94.47 (± 0.82)	-1.67 (± 0.64)
MER	Random	93.07 (± 0.81)	-8.36 (± 0.99)	85.53 (± 3.30)	-0.56 (± 3.36)	91.13 (± 0.86)	-5.32 (± 0.95)
	ETS	92.97 (± 1.73)	-8.52 (± 2.15)	84.88 (± 3.85)	-3.34 (± 5.59)	90.56 (± 0.83)	-6.11 (± 1.06)
	Heuristic	94.30 (± 2.79)	-6.46 (± 3.50)	96.91 (± 0.62)	-1.34 (± 0.76)	90.90 (± 1.30)	-6.24 (± 1.96)
	Ours	96.44 (± 0.72)	-4.14 (± 0.94)	86.67 (± 4.09)	0.85 (± 3.85)	92.44 (± 0.77)	-3.63 (± 1.06)
DER	Random	98.23 (± 0.53)	-1.89 (± 0.65)	96.56 (± 1.79)	-3.48 (± 2.22)	92.89 (± 0.86)	-3.75 (± 1.16)
	ETS	98.17 (± 0.35)	-2.00 (± 0.42)	97.69 (± 0.58)	-2.05 (± 0.71)	94.74 (± 1.05)	-1.94 (± 1.17)
	Heuristic	94.57 (± 1.71)	-6.08 (± 2.09)	*72.49 (± 19.32)	-20.88 (± 11.46)	*77.88 (± 12.58)	-12.66 (± 4.17)
	Ours	99.02 (± 0.10)	-0.91 (± 0.13)	98.33 (± 0.51)	-1.26 (± 0.63)	95.02 (± 0.33)	-0.97 (± 0.81)
DER++	Random	97.90 (± 0.52)	-2.32 (± 0.67)	97.10 (± 1.03)	-2.77 (± 1.29)	93.29 (± 1.43)	-3.11 (± 1.60)
	ETS	97.98 (± 0.52)	-2.24 (± 0.66)	98.12 (± 0.40)	-1.59 (± 0.52)	94.53 (± 1.02)	-1.82 (± 1.02)
	Heuristic	92.35 (± 2.42)	-8.83 (± 2.99)	*67.31 (± 21.20)	-24.86 (± 16.34)	93.88 (± 1.33)	-2.86 (± 1.49)
	Ours	98.84 (± 0.21)	-1.14 (± 0.26)	98.38 (± 0.43)	-1.17 (± 0.51)	94.73 (± 0.20)	-1.21 (± 1.12)

		Permuted MNIST		**Split CIFAR-100**		**Split miniImagenet**	
Method	**Schedule**	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑	ACC(%)↑	BWT(%)↑
HAL	Random	88.49 (± 0.99)	-7.03 (± 1.05)	36.09 (± 1.77)	-17.49 (± 1.78)	38.51 (± 2.22)	-6.65 (± 1.43)
	ETS	88.46 (± 0.86)	-7.26 (± 0.90)	34.90 (± 2.02)	-18.92 (± 0.91)	38.13 (± 1.18)	-8.19 (± 1.73)
	Heuristic	*66.63 (± 28.50)	-29.68 (± 27.90)	35.07 (± 1.29)	-24.76 (± 2.41)	39.51 (± 1.49)	-5.65 (± 0.77)
	Ours	89.14 (± 0.74)	-6.29 (± 0.74)	40.22 (± 1.57)	-12.77 (± 1.30)	41.39 (± 1.15)	-3.69 (± 1.86)
MER	Random	75.90 (± 1.34)	-21.69 (± 1.47)	42.96 (± 1.70)	-34.01 (± 2.07)	31.48 (± 1.65)	-6.99 (± 1.27)
	ETS	73.01 (± 0.96)	-25.19 (± 1.10)	43.38 (± 1.81)	-34.84 (± 1.98)	33.58 (± 1.53)	-6.80 (± 1.46)
	Heuristic	83.86 (± 3.19)	-12.48 (± 3.60)	40.90 (± 1.70)	-44.10 (± 2.03)	34.22 (± 1.93)	-7.57 (± 1.63)
	Ours	79.72 (± 0.71)	-17.42 (± 0.78)	44.29 (± 0.69)	-32.73 (± 0.88)	32.74 (± 1.29)	-5.77 (± 1.04)
DER	Random	87.51 (± 1.10)	-8.81 (± 1.28)	56.83 (± 0.76)	-27.34 (± 0.63)	42.19 (± 0.67)	-10.60 (± 1.28)
	ETS	85.71 (± 0.75)	-11.15 (± 0.87)	52.58 (± 1.49)	-32.93 (± 2.04)	35.50 (± 2.84)	-10.94 (± 2.21)
	Heuristic	81.56 (± 2.28)	-15.06 (± 2.51)	55.75 (± 1.08)	-31.27 (± 1.02)	43.62 (± 0.88)	-8.18 (± 1.16)
	Ours	90.11 (± 0.18)	-5.89 (± 0.23)	58.99 (± 0.98)	-24.95 (± 0.64)	43.46 (± 0.95)	-9.32 (± 1.37)
DER++	Random	87.89 (± 1.10	-8.35 (± 1.33)	58.49 (± 1.44)	-25.48 (± 1.55)	48.40 (± 0.69)	-4.20 (± 0.86)
	ETS	85.25 (± 0.88)	-11.60 (± 1.03)	52.54 (± 1.06)	-33.22 (± 1.51)	41.36 (± 2.90)	-4.07 (± 2.28)
	Heuristic	79.17 (± 2.44)	-17.68 (± 2.68)	56.70 (± 1.27)	-30.33 (± 1.41)	45.73 (± 0.84)	-6.09 (± 1.24)
	Ours	89.84 (± 0.22)	-6.13 (± 0.29)	59.23 (± 0.83)	-24.61 (± 0.91)	49.45 (± 0.68)	-3.12 (± 0.89)

Figure 8: Performance progress measured in ACC (%) for all methods in the 10 test environments for **Split MNIST** in the New Task Orders experiment. We plot the performance progress for A2C and DQN for 100 evaluation steps equidistantly distributed over the training episodes. The performance for the Random, ETS, and Heuristic scheduling baselines are plotted as straight lines.

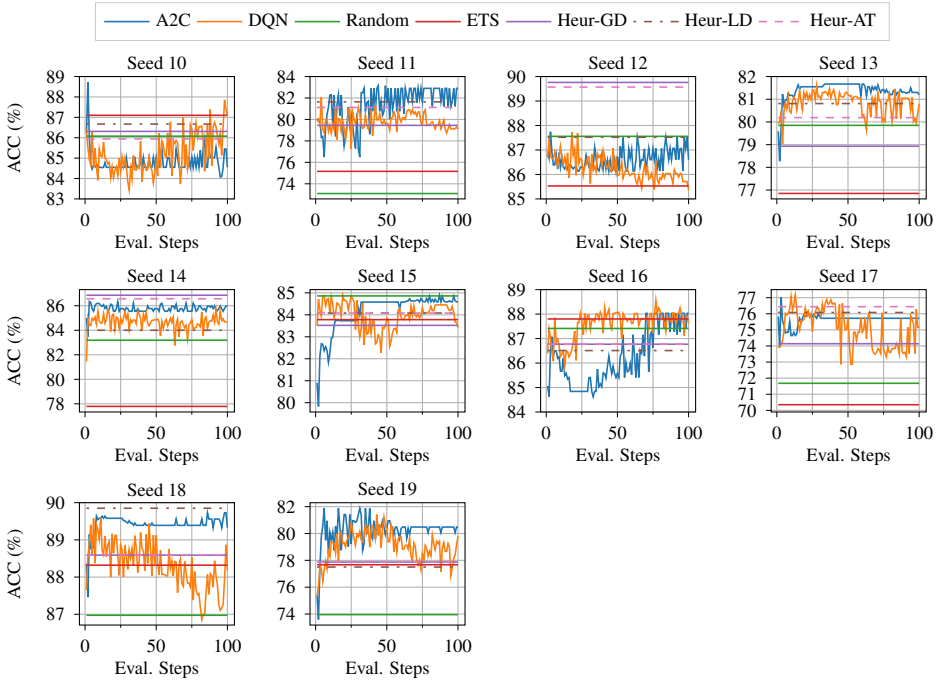


Figure 9: Performance progress measured in ACC (%) for all methods in the 10 test environments for **Split CIFAR-10** in the New Task Orders experiment. We plot the performance progress for A2C and DQN for 100 evaluation steps equidistantly distributed over the training episodes. The performance for the Random, ETS, and Heuristic scheduling baselines are plotted as straight lines.

Figure 10: Performance progress measured in ACC (%) for all methods in the 10 test environments for **Split notMNIST** in the New Dataset experiment. We plot the performance progress for A2C and DQN for 100 evaluation steps equidistantly distributed over the training episodes. The performance for the Random, ETS, and Heuristic scheduling baselines are plotted as straight lines.

Figure 11: Performance progress measured in ACC (%) for all methods in the 10 test environments for **Split FashionMNIST** in the New Dataset experiment. We plot the performance progress for A2C and DQN for 100 evaluation steps equidistantly distributed over the training episodes. The performance for the Random, ETS, and Heuristic scheduling baselines are plotted as straight lines.

Table 14: The ACC and rank for each seed (10-19) in every test environment with **Split MNIST** for New Task Order experiment. Under each column named 'Seed X', we show the ACC (%) and, in parenthesis, the rank for the corresponding method for the specific seed. We also show the average rank of each method in the test environment under 'Rank'. Note that the ACC for ETS, Heur-GD, Heur-LD, and Heur-AT are copied across the seeds, since the seed only affects the policy initialization in DQN and A2C and the random selection in Random.

	Test Env. Seed 10				**Test Env. Seed 11**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	93.2 (6.0)	93.7 (6.0)	96.2 (1.0)	4.33	92.2 (3.0)	90.8 (7.0)	90.5 (7.0)	5.67
ETS	89.9 (7.0)	89.9 (7.0)	89.9 (7.0)	7	93.8 (2.0)	93.8 (3.0)	93.8 (3.0)	2.67
Heur-GD	94.6 (3.5)	94.6 (4.5)	94.6 (5.5)	4.5	91.3 (6.0)	91.3 (5.0)	91.3 (5.0)	5.33
Heur-LD	95.6 (1.0)	95.6 (1.0)	95.6 (3.0)	1.67	91.3 (6.0)	91.3 (5.0)	91.3 (5.0)	5.33
Heur-AT	94.6 (3.5)	94.6 (4.5)	94.6 (5.5)	4.5	91.3 (6.0)	91.3 (5.0)	91.3 (5.0)	5.33
DQN	93.2 (5.0)	95.5 (2.0)	95.7 (2.0)	3	91.8 (4.0)	96.0 (2.0)	95.7 (2.0)	2.67
A2C	95.3 (2.0)	95.3 (3.0)	95.3 (4.0)	3	96.5 (1.0)	96.5 (1.0)	96.5 (1.0)	1
	Test Env. Seed 12				**Test Env. Seed 13**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	90.7 (7.0)	88.8 (7.0)	95.2 (1.0)	5	93.4 (5.0)	94.4 (6.0)	89.9 (7.0)	6
ETS	91.7 (6.0)	91.7 (6.0)	91.7 (7.0)	6.33	95.0 (2.0)	95.0 (3.0)	95.0 (3.0)	2.67
Heur-GD	94.8 (1.5)	94.8 (1.5)	94.8 (2.5)	1.83	94.7 (3.5)	94.7 (4.5)	94.7 (4.5)	4.17
Heur-LD	91.9 (5.0)	91.9 (5.0)	91.9 (6.0)	5.33	93.2 (7.0)	93.2 (7.0)	93.2 (6.0)	6.67
Heur-AT	94.8 (1.5)	94.8 (1.5)	94.8 (2.5)	1.83	94.7 (3.5)	94.7 (4.5)	94.7 (4.5)	4.17
DQN	94.2 (3.0)	94.5 (3.0)	92.8 (5.0)	3.67	93.2 (6.0)	96.6 (1.0)	96.2 (1.0)	2.67
A2C	93.6 (4.0)	93.6 (4.0)	93.6 (4.0)	4	95.6 (1.0)	95.6 (2.0)	95.6 (2.0)	1.67
	Test Env. Seed 14				**Test Env. Seed 15**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	84.4 (3.0)	80.5 (7.0)	86.6 (3.0)	4.33	93.8 (7.0)	95.4 (4.0)	91.9 (7.0)	6
ETS	87.3 (2.0)	87.3 (2.0)	87.3 (2.0)	2	95.3 (4.0)	95.3 (5.0)	95.3 (4.0)	4.33
Heur-GD	81.2 (6.5)	81.2 (5.5)	81.2 (6.5)	6.17	95.9 (2.5)	95.9 (2.5)	95.9 (2.5)	2.5
Heur-LD	81.2 (6.5)	81.2 (5.5)	81.2 (6.5)	6.17	96.1 (1.0)	96.1 (1.0)	96.1 (1.0)	1
Heur-AT	82.4 (5.0)	82.4 (4.0)	82.4 (5.0)	4.67	95.9 (2.5)	95.9 (2.5)	95.9 (2.5)	2.5
DQN	92.8 (1.0)	86.8 (3.0)	88.1 (1.0)	1.67	94.8 (6.0)	94.8 (7.0)	94.3 (6.0)	6.33
A2C	83.4 (4.0)	95.3 (1.0)	83.4 (4.0)	3	94.8 (5.0)	94.8 (6.0)	94.8 (5.0)	5.33
	Test Env. Seed 16				**Test Env. Seed 17**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	80.5 (6.0)	87.5 (6.0)	83.0 (6.0)	6	96.4 (1.0)	96.1 (3.0)	96.1 (1.0)	1.67
ETS	79.4 (7.0)	79.4 (7.0)	79.4 (7.0)	7	95.7 (3.0)	95.7 (4.0)	95.7 (3.0)	3.33
Heur-GD	91.2 (3.0)	91.2 (4.0)	91.2 (3.0)	3.33	93.5 (6.0)	93.5 (6.0)	93.5 (6.0)	6
Heur-LD	91.2 (3.0)	91.2 (4.0)	91.2 (3.0)	3.33	93.5 (6.0)	93.5 (6.0)	93.5 (6.0)	6
Heur-AT	91.2 (3.0)	91.2 (4.0)	91.2 (3.0)	3.33	93.5 (6.0)	93.5 (6.0)	93.5 (6.0)	6
DQN	93.9 (1.0)	92.9 (1.0)	94.5 (1.0)	1	95.7 (4.0)	96.2 (2.0)	95.7 (4.0)	3.33
A2C	90.3 (5.0)	92.3 (2.0)	90.3 (5.0)	4	96.1 (2.0)	96.5 (1.0)	96.1 (2.0)	1.67
	Test Env. Seed 18				**Test Env. Seed 19**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	92.7 (2.0)	94.2 (1.0)	93.2 (1.0)	1.33	97.3 (2.0)	95.9 (2.0)	91.6 (2.0)	2
ETS	92.9 (1.0)	92.9 (2.0)	92.9 (2.0)	1.67	97.4 (1.0)	97.4 (1.0)	97.4 (1.0)	1
Heur-GD	87.8 (6.0)	87.8 (6.0)	87.8 (6.0)	6	88.3 (5.0)	88.3 (6.0)	88.3 (4.0)	5
Heur-LD	87.8 (6.0)	87.8 (6.0)	87.8 (6.0)	6	88.3 (5.0)	88.3 (6.0)	88.3 (4.0)	5
Heur-AT	87.8 (6.0)	87.8 (6.0)	87.8 (6.0)	6	88.3 (5.0)	88.3 (6.0)	88.3 (4.0)	5
DQN	89.5 (4.0)	89.4 (4.0)	90.5 (4.0)	4	89.4 (3.0)	90.5 (3.0)	87.5 (6.0)	4
A2C	91.6 (3.0)	91.6 (3.0)	91.6 (3.0)	3	86.1 (7.0)	89.9 (4.0)	86.1 (7.0)	6

Table 15: The ACC and rank for each seed (10-19) in every test environment with **Split FashionMNIST** for New Task Order experiment. Under each column named 'Seed X', we show the ACC (%) and, in parenthesis, the rank for the corresponding method for the specific seed. We also show the average rank of each method under 'Rank' in the corresponding test environment. Note that the ACC for ETS, Heur-GD, Heur-LD, and Heur-AT are copied across the seeds, since the seed only affects the policy initialization in DQN and A2C and the random selection in Random.

	Test Env. Seed 10				**Test Env. Seed 11**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	98.4 (1.0)	90.8 (7.0)	98.0 (1.0)	3	88.9 (5.0)	93.4 (3.0)	89.2 (5.0)	4.33
ETS	96.1 (6.0)	96.1 (5.0)	96.1 (5.0)	5.33	88.8 (6.0)	88.8 (6.0)	88.8 (6.0)	6
Heur-GD	98.0 (2.5)	98.0 (1.5)	98.0 (2.5)	2.17	93.2 (3.0)	93.2 (4.0)	93.2 (4.0)	3.67
Heur-LD	98.0 (2.5)	98.0 (1.5)	98.0 (2.5)	2.17	94.7 (1.0)	94.7 (1.0)	94.7 (1.0)	1
Heur-AT	92.0 (7.0)	92.0 (6.0)	92.0 (7.0)	6.67	93.5 (2.0)	93.5 (2.0)	93.5 (2.0)	2
DQN	96.9 (4.5)	97.0 (4.0)	95.4 (6.0)	4.83	91.3 (4.0)	93.2 (5.0)	93.3 (3.0)	4
A2C	96.9 (4.5)	97.1 (3.0)	96.9 (4.0)	3.83	85.3 (7.0)	86.9 (7.0)	85.3 (7.0)	7
	Test Env. Seed 12				**Test Env. Seed 13**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	97.6 (1.0)	96.8 (1.0)	93.4 (4.0)	2	94.0 (6.0)	88.9 (7.0)	93.4 (6.0)	6.33
ETS	91.2 (6.0)	91.2 (6.0)	91.2 (5.0)	5.67	91.2 (7.0)	91.2 (6.0)	91.2 (7.0)	6.67
Heur-GD	95.0 (5.0)	95.0 (4.0)	95.0 (3.0)	4	95.1 (5.0)	95.1 (5.0)	95.1 (5.0)	5
Heur-LD	96.1 (2.0)	96.1 (2.0)	96.1 (1.0)	1.67	96.2 (4.0)	96.2 (4.0)	96.2 (4.0)	4
Heur-AT	95.8 (3.0)	95.8 (3.0)	95.8 (2.0)	2.67	98.1 (1.0)	98.1 (2.0)	98.1 (1.0)	1.33
DQN	95.0 (4.0)	94.8 (5.0)	88.6 (6.0)	5	96.6 (3.0)	98.4 (1.0)	97.4 (2.0)	2
A2C	87.7 (7.0)	87.7 (7.0)	87.7 (7.0)	7	98.0 (2.0)	98.0 (3.0)	96.7 (3.0)	2.67
	Test Env. Seed 14				**Test Env. Seed 15**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	94.2 (5.0)	96.0 (2.0)	93.0 (5.0)	4	94.3 (1.0)	92.6 (2.0)	93.8 (1.0)	1.33
ETS	90.0 (6.0)	90.0 (6.0)	90.0 (6.0)	6	93.5 (2.0)	93.5 (1.0)	93.5 (2.0)	1.67
Heur-GD	95.4 (1.0)	95.4 (3.0)	95.4 (2.0)	2	79.3 (7.0)	79.3 (7.0)	79.3 (7.0)	7
Heur-LD	95.1 (3.0)	95.1 (4.0)	95.1 (3.0)	3.33	92.6 (3.0)	92.6 (3.0)	92.6 (3.0)	3
Heur-AT	82.0 (7.0)	82.0 (7.0)	82.0 (7.0)	7	88.5 (5.0)	88.5 (5.0)	88.5 (4.0)	4.67
DQN	95.3 (2.0)	96.1 (1.0)	95.5 (1.0)	1.33	89.6 (4.0)	90.1 (4.0)	88.4 (5.0)	4.33
A2C	94.7 (4.0)	94.7 (5.0)	94.7 (4.0)	4.33	81.0 (6.0)	81.0 (6.0)	80.5 (6.0)	6
	Test Env. Seed 16				**Test Env. Seed 17**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	90.0 (2.0)	88.1 (3.0)	92.3 (3.0)	2.67	99.1 (3.0)	98.6 (5.0)	99.2 (4.0)	4
ETS	94.4 (1.0)	94.4 (1.0)	94.4 (1.0)	1	98.1 (7.0)	98.1 (7.0)	98.1 (7.0)	7
Heur-GD	73.8 (7.0)	73.8 (7.0)	73.8 (7.0)	7	99.4 (1.0)	99.4 (1.0)	99.4 (1.0)	1
Heur-LD	80.4 (6.0)	80.4 (6.0)	80.4 (6.0)	6	99.3 (2.0)	99.3 (2.0)	99.3 (2.0)	2
Heur-AT	89.2 (4.0)	89.2 (2.0)	89.2 (4.0)	3.33	98.7 (4.0)	98.7 (4.0)	98.7 (6.0)	4.67
DQN	87.3 (5.0)	85.7 (4.0)	92.3 (2.0)	3.67	98.7 (5.0)	99.1 (3.0)	99.0 (5.0)	4.33
A2C	89.3 (3.0)	83.2 (5.0)	87.7 (5.0)	4.33	98.7 (6.0)	98.5 (6.0)	99.2 (3.0)	5
	Test Env. Seed 18				**Test Env. Seed 19**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	87.2 (7.0)	87.2 (7.0)	94.2 (4.0)	6	96.2 (5.0)	97.8 (1.0)	97.7 (1.0)	2.33
ETS	93.6 (4.0)	93.6 (4.0)	93.6 (5.0)	4.33	97.5 (1.0)	97.5 (3.0)	97.5 (2.0)	2
Heur-GD	92.9 (5.0)	92.9 (5.0)	92.9 (6.0)	5.33	95.8 (6.0)	95.8 (5.0)	95.8 (5.0)	5.33
Heur-LD	92.1 (6.0)	92.1 (6.0)	92.1 (7.0)	6.33	93.4 (7.0)	93.4 (7.0)	93.4 (7.0)	7
Heur-AT	94.2 (2.0)	94.2 (2.0)	94.2 (3.0)	2.33	96.6 (4.0)	96.6 (4.0)	96.6 (4.0)	4
DQN	95.3 (1.0)	94.7 (1.0)	96.0 (1.0)	1	96.9 (3.0)	95.7 (6.0)	95.7 (6.0)	5
A2C	93.8 (3.0)	93.8 (3.0)	95.0 (2.0)	2.67	97.0 (2.0)	97.7 (2.0)	96.9 (3.0)	2.33

Table 16: The ACC and rank for each seed (10-19) in every test environment with **Split CIFAR-10** for New Task Order experiment. Under each column named 'Seed X', we show the ACC (%) and, in parenthesis, the rank for the corresponding method for the specific seed. We also show the average rank of each method in the test environment under 'Rank'. Note that the ACC for ETS, Heur-GD, Heur-LD, and Heur-AT are copied across the seeds, since the seed only affects the policy initialization in DQN and A2C and the random selection in Random.

	Test Env. Seed 10				**Test Env. Seed 11**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	87.3 (2.0)	85.2 (6.0)	85.7 (6.0)	4.67	75.3 (6.0)	71.1 (7.0)	72.8 (7.0)	6.67
ETS	87.1 (3.0)	87.1 (1.0)	87.1 (1.0)	1.67	75.2 (7.0)	75.2 (6.0)	75.2 (6.0)	6.33
Heur-GD	86.3 (5.0)	86.3 (3.0)	86.3 (4.0)	4	79.5 (4.0)	79.5 (5.0)	79.5 (4.0)	4.33
Heur-LD	86.7 (4.0)	86.7 (2.0)	86.7 (3.0)	3	81.6 (2.0)	81.6 (2.0)	81.6 (2.0)	2
Heur-AT	85.9 (6.0)	85.9 (4.0)	85.9 (5.0)	5	81.1 (3.0)	81.1 (3.0)	81.1 (3.0)	3
DQN	88.9 (1.0)	85.5 (5.0)	86.9 (2.0)	2.67	78.2 (5.0)	79.9 (4.0)	79.3 (5.0)	4.67
A2C	84.6 (7.0)	84.6 (7.0)	84.6 (7.0)	7	82.9 (1.0)	82.9 (1.0)	82.9 (1.0)	1
	Test Env. Seed 12				**Test Env. Seed 13**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	86.2 (4.0)	88.9 (3.0)	87.6 (3.0)	3.33	77.3 (6.0)	81.3 (2.0)	80.9 (2.0)	3.33
ETS	85.5 (7.0)	85.5 (6.0)	85.5 (7.0)	6.67	76.9 (7.0)	76.9 (7.0)	76.9 (7.0)	7
Heur-GD	89.8 (1.0)	89.8 (1.0)	89.8 (1.0)	1	78.9 (5.0)	78.9 (6.0)	78.9 (6.0)	5.67
Heur-LD	87.5 (3.0)	87.5 (4.0)	87.5 (5.0)	4	80.8 (3.0)	80.8 (4.0)	80.8 (3.0)	3.33
Heur-AT	89.6 (2.0)	89.6 (2.0)	89.6 (2.0)	2	80.2 (4.0)	80.2 (5.0)	80.2 (5.0)	4.67
DQN	86.0 (6.0)	84.4 (7.0)	85.7 (6.0)	6.33	80.8 (2.0)	81.3 (1.0)	80.2 (4.0)	2.33
A2C	86.1 (5.0)	86.1 (5.0)	87.5 (4.0)	4.67	81.3 (1.0)	81.0 (3.0)	81.3 (1.0)	1.67
	Test Env. Seed 14				**Test Env. Seed 15**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	81.7 (6.0)	84.8 (5.0)	83.0 (6.0)	5.67	83.8 (4.0)	85.0 (2.0)	85.8 (1.0)	2.33
ETS	77.8 (7.0)	77.8 (7.0)	77.8 (7.0)	7	83.8 (5.0)	83.8 (6.0)	83.8 (5.0)	5.33
Heur-GD	86.9 (1.0)	86.9 (1.0)	86.9 (1.0)	1	83.5 (6.0)	83.5 (7.0)	83.5 (6.0)	6.33
Heur-LD	84.0 (5.0)	84.0 (6.0)	84.0 (4.0)	5	84.1 (2.5)	84.1 (4.5)	84.1 (3.5)	3.5
Heur-AT	86.6 (2.0)	86.6 (2.0)	86.6 (2.0)	2	84.1 (2.5)	84.1 (4.5)	84.1 (3.5)	3.5
DQN	85.0 (4.0)	85.2 (4.0)	83.8 (5.0)	4.33	83.2 (7.0)	85.4 (1.0)	81.7 (7.0)	5
A2C	86.2 (3.0)	86.2 (3.0)	85.5 (3.0)	3	84.6 (1.0)	84.6 (3.0)	84.6 (2.0)	2
	Test Env. Seed 16				**Test Env. Seed 17**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	85.2 (7.0)	89.4 (1.0)	87.6 (4.0)	4	68.9 (7.0)	72.1 (6.0)	74.0 (6.0)	6.33
ETS	87.8 (3.0)	87.8 (3.0)	87.8 (2.0)	2.67	70.3 (5.0)	70.3 (7.0)	70.3 (7.0)	6.33
Heur-GD	86.8 (4.5)	86.8 (5.5)	86.8 (5.5)	5.17	74.1 (4.0)	74.1 (5.0)	74.1 (5.0)	4.67
Heur-LD	86.5 (6.0)	86.5 (7.0)	86.5 (7.0)	6.67	76.1 (2.0)	76.1 (3.0)	76.1 (3.0)	2.67
Heur-AT	86.8 (4.5)	86.8 (5.5)	86.8 (5.5)	5.17	76.4 (1.0)	76.4 (1.0)	76.4 (2.0)	1.33
DQN	87.9 (2.0)	87.4 (4.0)	87.6 (3.0)	3	69.9 (6.0)	76.4 (2.0)	79.2 (1.0)	3
A2C	88.0 (1.0)	88.0 (2.0)	88.0 (1.0)	1.33	75.7 (3.0)	75.7 (4.0)	75.7 (4.0)	3.67
	Test Env. Seed 18				**Test Env. Seed 19**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	86.9 (7.0)	87.6 (7.0)	86.4 (7.0)	7	71.2 (7.0)	77.5 (7.0)	73.2 (7.0)	7
ETS	88.3 (6.0)	88.3 (6.0)	88.3 (5.0)	5.67	77.7 (5.0)	77.7 (5.0)	77.7 (5.0)	5
Heur-GD	88.6 (4.5)	88.6 (4.5)	88.6 (3.5)	4.17	77.9 (3.5)	77.9 (3.5)	77.9 (3.5)	3.5
Heur-LD	89.8 (1.0)	89.8 (2.0)	89.8 (1.0)	1.33	77.5 (6.0)	77.5 (6.0)	77.5 (6.0)	6
Heur-AT	88.6 (4.5)	88.6 (4.5)	88.6 (3.5)	4.17	77.9 (3.5)	77.9 (3.5)	77.9 (3.5)	3.5
DQN	89.1 (2.0)	88.6 (3.0)	86.8 (6.0)	3.67	80.5 (2.0)	80.5 (1.5)	78.5 (2.0)	1.83
A2C	88.7 (3.0)	89.9 (1.0)	89.4 (2.0)	2	80.5 (1.0)	80.5 (1.5)	80.5 (1.0)	1.17

Table 17: The ACC and rank for each seed (0-9) in every test environment with **Split notMNIST** for New Dataset experiment. Under each column named 'Seed X', we show the ACC (%) and, in parenthesis, the rank for the corresponding method for the specific seed. We also show the average rank of each method in the test environment under 'Rank'. Note that the ACC for ETS, Heur-GD, Heur-LD, and Heur-AT are copied across the seeds, since the seed only affects the policy initialization in DQN and A2C and the random selection in Random.

	Test Env. Seed 0				**Test Env. Seed 1**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	93.8 (1.0)	94.9 (1.0)	93.2 (2.0)	1.33	93.5 (2.0)	92.0 (4.0)	91.0 (7.0)	4.33
ETS	91.1 (5.0)	91.1 (7.0)	91.1 (6.0)	6	92.5 (4.0)	92.5 (3.0)	92.5 (3.0)	3.33
Heur-GD	92.4 (3.0)	92.4 (5.0)	92.4 (4.0)	4	91.2 (6.0)	91.2 (6.0)	91.2 (5.0)	5.67
Heur-LD	92.4 (3.0)	92.4 (5.0)	92.4 (4.0)	4	91.2 (6.0)	91.2 (6.0)	91.2 (5.0)	5.67
Heur-AT	92.4 (3.0)	92.4 (5.0)	92.4 (4.0)	4	91.2 (6.0)	91.2 (6.0)	91.2 (5.0)	5.67
DQN	90.7 (6.0)	94.5 (2.0)	85.3 (7.0)	5	94.2 (1.0)	94.6 (1.0)	94.5 (1.0)	1
A2C	90.6 (7.0)	92.7 (3.0)	93.3 (1.0)	3.67	93.5 (3.0)	93.9 (2.0)	93.9 (2.0)	2.33
	Test Env. Seed 2				**Test Env. Seed 3**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	82.1 (7.0)	90.5 (3.0)	88.5 (4.0)	4.67	94.2 (5.0)	89.5 (6.0)	85.2 (7.0)	6
ETS	91.2 (3.0)	91.2 (2.0)	91.2 (3.0)	2.67	89.3 (7.0)	89.3 (7.0)	89.3 (6.0)	6.67
Heur-GD	82.6 (5.0)	82.6 (6.0)	82.6 (6.0)	5.67	95.3 (2.0)	95.3 (2.0)	95.3 (2.0)	2
Heur-LD	82.6 (5.0)	82.6 (6.0)	82.6 (6.0)	5.67	95.3 (2.0)	95.3 (2.0)	95.3 (2.0)	2
Heur-AT	82.6 (5.0)	82.6 (6.0)	82.6 (6.0)	5.67	95.3 (2.0)	95.3 (2.0)	95.3 (2.0)	2
DQN	93.9 (1.0)	94.2 (1.0)	92.0 (1.0)	1	93.3 (6.0)	92.8 (5.0)	92.3 (5.0)	5.33
A2C	91.7 (2.0)	89.7 (4.0)	91.7 (2.0)	2.67	95.0 (4.0)	95.0 (4.0)	95.0 (4.0)	4
	Test Env. Seed 4				**Test Env. Seed 5**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	90.2 (7.0)	93.2 (1.0)	90.1 (7.0)	5	90.7 (7.0)	93.7 (1.0)	91.4 (6.0)	4.67
ETS	90.5 (6.0)	90.5 (7.0)	90.5 (6.0)	6.33	90.7 (6.0)	90.7 (7.0)	90.7 (7.0)	6.67
Heur-GD	91.6 (3.5)	91.6 (4.5)	91.6 (4.5)	4.17	91.8 (4.0)	91.8 (5.0)	91.8 (4.0)	4.33
Heur-LD	92.7 (2.0)	92.7 (3.0)	92.7 (2.0)	2.33	91.8 (4.0)	91.8 (5.0)	91.8 (4.0)	4.33
Heur-AT	91.6 (3.5)	91.6 (4.5)	91.6 (4.5)	4.17	91.8 (4.0)	91.8 (5.0)	91.8 (4.0)	4.33
DQN	91.3 (5.0)	91.2 (6.0)	92.0 (3.0)	4.67	93.2 (1.0)	92.1 (3.0)	93.0 (1.0)	1.67
A2C	93.1 (1.0)	93.1 (2.0)	93.0 (1.0)	1.33	92.6 (2.0)	93.1 (2.0)	92.8 (2.0)	2
	Test Env. Seed 6				**Test Env. Seed 7**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	91.6 (4.0)	87.2 (6.0)	91.9 (3.0)	4.33	95.0 (1.0)	94.3 (1.0)	93.3 (3.0)	1.67
ETS	92.0 (3.0)	92.0 (2.0)	92.0 (2.0)	2.33	94.0 (2.0)	94.0 (2.0)	94.0 (1.0)	1.67
Heur-GD	91.6 (5.5)	91.6 (3.5)	91.6 (4.5)	4.5	88.8 (7.0)	88.8 (7.0)	88.8 (7.0)	7
Heur-LD	87.0 (7.0)	87.0 (7.0)	87.0 (7.0)	7	90.6 (6.0)	90.6 (6.0)	90.6 (6.0)	6
Heur-AT	91.6 (5.5)	91.6 (3.5)	91.6 (4.5)	4.5	93.1 (4.0)	93.1 (4.0)	93.1 (4.0)	4
DQN	93.1 (2.0)	95.2 (1.0)	93.8 (1.0)	1.33	94.0 (3.0)	92.6 (5.0)	93.7 (2.0)	3.33
A2C	93.3 (1.0)	90.0 (5.0)	90.0 (6.0)	4	92.6 (5.0)	93.7 (3.0)	92.6 (5.0)	4.33
	Test Env. Seed 8				**Test Env. Seed 9**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	92.3 (5.0)	92.6 (4.0)	92.4 (4.0)	4.33	90.7 (3.0)	91.2 (2.0)	93.4 (2.0)	2.33
ETS	89.0 (6.0)	89.0 (6.0)	89.0 (6.0)	6	94.3 (1.0)	94.3 (1.0)	94.3 (1.0)	1
Heur-GD	93.6 (3.5)	93.6 (2.5)	93.6 (2.5)	2.83	79.1 (6.0)	79.1 (5.0)	79.1 (6.0)	5.67
Heur-LD	88.2 (7.0)	88.2 (7.0)	88.2 (7.0)	7	79.1 (6.0)	79.1 (5.0)	79.1 (6.0)	5.67
Heur-AT	93.6 (3.5)	93.6 (2.5)	93.6 (2.5)	2.83	79.1 (6.0)	79.1 (5.0)	79.1 (6.0)	5.67
DQN	94.6 (1.0)	91.6 (5.0)	92.1 (5.0)	3.67	82.7 (4.0)	79.1 (7.0)	82.9 (4.0)	5
A2C	93.9 (2.0)	93.9 (1.0)	93.9 (1.0)	1.33	90.8 (2.0)	90.8 (3.0)	90.8 (3.0)	2.67

Table 18: The ACC and rank for each seed (0-9) in every test environment with **Split FashionMNIST** for New Dataset experiment. Under each column named 'Seed X', we show the ACC (%) and, in parenthesis, the rank for the corresponding method for the specific seed. We also show the average rank of each method in the test environment under 'Rank'. Note that the ACC for ETS, Heur-GD, Heur-LD, and Heur-AT are copied across the seeds, since the seed only affects the policy initialization in DQN and A2C and the random selection in Random.

	Test Env. Seed 0				**Test Env. Seed 1**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	94.7 (7.0)	97.7 (2.0)	91.5 (7.0)	5.33	92.8 (5.0)	95.1 (1.0)	93.7 (6.0)	4
ETS	97.1 (6.0)	97.1 (6.0)	97.1 (6.0)	6	94.1 (2.0)	94.1 (3.0)	94.1 (3.0)	2.67
Heur-GD	97.9 (1.0)	97.9 (1.0)	97.9 (1.0)	1	95.0 (1.0)	95.0 (2.0)	95.0 (1.0)	1.33
Heur-LD	97.6 (3.0)	97.6 (3.0)	97.6 (3.0)	3	90.0 (7.0)	90.0 (7.0)	90.0 (7.0)	7
Heur-AT	97.4 (4.0)	97.4 (5.0)	97.4 (5.0)	4.67	94.1 (3.0)	94.1 (4.0)	94.1 (4.0)	3.67
DQN	97.7 (2.0)	97.4 (4.0)	97.6 (4.0)	3.33	94.0 (4.0)	90.1 (6.0)	94.6 (2.0)	4
A2C	97.3 (5.0)	89.1 (7.0)	97.7 (2.0)	4.67	90.9 (6.0)	90.9 (5.0)	94.0 (5.0)	5.33
	Test Env. Seed 2				**Test Env. Seed 3**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	94.6 (6.0)	95.1 (6.0)	94.8 (5.0)	5.67	89.5 (6.0)	99.4 (1.5)	93.9 (4.0)	3.83
ETS	86.7 (7.0)	86.7 (7.0)	86.7 (7.0)	7	89.4 (7.0)	89.4 (6.0)	89.4 (6.0)	6.33
Heur-GD	97.4 (2.0)	97.4 (3.0)	97.4 (3.0)	2.67	96.7 (3.0)	96.7 (3.0)	96.7 (2.0)	2.67
Heur-LD	97.3 (3.0)	97.3 (4.0)	97.3 (4.0)	3.67	90.6 (5.0)	90.6 (5.0)	90.6 (5.0)	5
Heur-AT	97.7 (1.0)	97.7 (1.0)	97.7 (1.0)	1	99.4 (1.0)	99.4 (1.5)	99.4 (1.0)	1.17
DQN	96.6 (4.0)	95.9 (5.0)	94.3 (6.0)	5	97.3 (2.0)	96.6 (4.0)	96.0 (3.0)	3
A2C	96.2 (5.0)	97.6 (2.0)	97.6 (2.0)	3	93.8 (4.0)	88.6 (7.0)	89.2 (7.0)	6
	Test Env. Seed 4				**Test Env. Seed 5**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	88.6 (4.0)	84.6 (6.0)	80.3 (7.0)	5.67	88.7 (5.0)	93.9 (1.0)	88.0 (5.0)	3.67
ETS	87.1 (5.0)	87.1 (4.0)	87.1 (4.0)	4.33	91.5 (1.0)	91.5 (2.0)	91.5 (1.0)	1.33
Heur-GD	91.3 (3.0)	91.3 (3.0)	91.3 (2.0)	2.67	90.3 (3.0)	90.3 (4.0)	90.3 (3.0)	3.33
Heur-LD	86.8 (6.0)	86.8 (5.0)	86.8 (5.0)	5.33	88.0 (6.0)	88.0 (5.0)	88.0 (4.0)	5
Heur-AT	83.3 (7.0)	83.3 (7.0)	83.3 (6.0)	6.67	90.5 (2.0)	90.5 (3.0)	90.5 (2.0)	2.33
DQN	91.3 (2.0)	92.2 (1.0)	90.5 (3.0)	2	89.2 (4.0)	87.1 (7.0)	86.1 (7.0)	6
A2C	91.8 (1.0)	91.8 (2.0)	91.8 (1.0)	1.33	87.9 (7.0)	87.9 (6.0)	87.9 (6.0)	6.33
	Test Env. Seed 6				**Test Env. Seed 7**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	95.3 (2.0)	94.0 (2.0)	96.3 (1.0)	1.67	93.3 (4.0)	94.0 (4.0)	94.4 (5.0)	4.33
ETS	95.5 (1.0)	95.5 (1.0)	95.5 (2.0)	1.33	95.3 (2.0)	95.3 (2.0)	95.3 (3.0)	2.33
Heur-GD	90.0 (3.0)	90.0 (4.0)	90.0 (4.0)	3.67	91.8 (5.0)	91.8 (5.0)	91.8 (6.0)	5.33
Heur-LD	85.4 (4.0)	85.4 (5.0)	85.4 (5.0)	4.67	95.1 (3.0)	95.1 (3.0)	95.1 (4.0)	3.33
Heur-AT	76.0 (7.0)	76.0 (7.0)	76.0 (7.0)	7	96.8 (1.0)	96.8 (1.0)	96.8 (1.0)	1
DQN	80.1 (6.0)	84.3 (6.0)	81.6 (6.0)	6	87.4 (7.0)	88.8 (7.0)	87.4 (7.0)	7
A2C	81.9 (5.0)	91.7 (3.0)	91.5 (3.0)	3.67	91.7 (6.0)	91.3 (6.0)	96.1 (2.0)	4.67
	Test Env. Seed 8				**Test Env. Seed 9**			
Method	Seed 1	Seed 2	Seed 3	Rank	Seed 1	Seed 2	Seed 3	Rank
Random	88.6 (6.0)	87.9 (6.0)	95.8 (1.0)	4.33	97.2 (1.0)	97.0 (1.0)	97.7 (1.0)	1
ETS	94.8 (2.0)	94.8 (3.0)	94.8 (3.0)	2.67	96.7 (3.0)	96.7 (2.0)	96.7 (2.0)	2.33
Heur-GD	95.0 (1.0)	95.0 (1.0)	95.0 (2.0)	1.33	95.3 (4.0)	95.3 (3.0)	95.3 (4.0)	3.67
Heur-LD	87.9 (7.0)	87.9 (7.0)	87.9 (7.0)	7	90.0 (7.0)	90.0 (7.0)	90.0 (7.0)	7
Heur-AT	93.9 (3.0)	93.9 (4.0)	93.9 (4.0)	3.67	90.3 (6.0)	90.3 (6.0)	90.3 (6.0)	6
DQN	91.8 (5.0)	95.0 (2.0)	91.0 (6.0)	4.33	96.9 (2.0)	93.7 (4.0)	95.9 (3.0)	3
A2C	93.6 (4.0)	93.6 (5.0)	93.6 (5.0)	4.67	90.4 (5.0)	92.6 (5.0)	90.4 (5.0)	5

D Additional Methodology

In this section, we provide pseudo-code for MCTS to search for replay schedules in single CL environments in Section D.1 as well as pseudo-code for the RL-based framework for learning the replay scheduling policies in Section D.2.

D.1 Monte Carlo Tree Search Algorithm for Replay Scheduling

We provide pseudo-code in Algorithm 1 outlining the steps for our method using Monte Carlo tree search (MCTS) to find replay schedules described in the main paper (Section 3.3). The MCTS procedure selects actions over which task proportions to fill the replay memory with at every task, where the selected task proportions are stored in the replay schedule S. The schedule is then passed to EVALUATEREPLAYSCHEDULE$(\cdot)$ where the continual learning part executes the training with replay memories filled according to the schedule. The reward for the schedule S is the average validation accuracy over all tasks after learning task T, i.e., ACC, which is backpropagated through the tree to update the statistics of the selected nodes. The schedule S_{best} yielding the best ACC score is returned to be used for evaluation on the held-out test sets.

The function GETREPLAYMEMORY$(\cdot)$ is the policy for retrieving the replay memory $\mathcal{M}$ from the historical data given the task proportion $\boldsymbol{p}$. The number of samples per task determined by the task proportions are rounded up or down accordingly to fill $\mathcal{M}$ with M replay samples in total. The function GETTASKPROPORTION$(\cdot)$ simply returns the task proportion that is related to given node.

The following steps are performed during one MCTS rollout (or iteration):

1. **Selection** involves either selecting an unvisited node randomly, or selecting the next node by evaluating the UCT score (see Equation 1) if all children has been visited already. In Algorithm 1, TREEPOLICY$(\cdot)$ appends the task proportions $\boldsymbol{p}_t$ to the replay schedule S at every selected node.
2. **Expansion** involves expanding the search tree with one of the unvisited child nodes v_{t+1} selected with uniform sampling. EXPANSION$(\cdot)$ in Algorithm 1 appends the task proportions $\boldsymbol{p}_t$ to the replay schedule S of the expanded node.
3. **Simulation** involves selecting the next nodes randomly until a terminal node v_T is reached. In Algorithm 1, DEFAULTPOLICY$(\cdot)$ appends the task proportions $\boldsymbol{p}_t$ to the replay schedule S at every randomly selected node until reaching the terminal node.
4. **Reward** The reward for the rollout is given by the ACC of the validation sets for each task. In Algorithm 1, EVALUATEREPLAYSCHEDULE$(\cdot)$ involves learning the tasks $t = 1, \dots, T$ sequentially and using the replay schedule to sample the replay memories to use for mitigating catastrophic forgetting when learning a new task. The reward r for the rollout is calculated after task T has been learnt.
5. **Backpropagation** involves updating the reward function $q(\cdot)$ and number of visits $n(\cdot)$ from the expansion node up to the root node. See BACKRPROPAGATE$(\cdot)$ in Algorithm 1.

D.2 RL Framework Algorithm

We provide pseudo-code for the RL-based framework for learning the replay scheduling policy with either DQN [65] or A2C [64] in Algorithm 2. The procedure collects experience from all training environments in $\mathcal{E}^{(train)}$ at every time step t. The datasets and classifiers are specific for each environment $E_i \in \mathcal{E}^{(train)}$. At $t = 1$, we obtain the initial state $s_1^{(i)}$ by evaluating the classifier on the validation set $\mathcal{D}_1^{(val)}$ after training the classifier on the task 1. Next, we get the replay memory for mitigating catastrophic forgetting when learning the next task $t + 1$ by 1) taking action $a_t^{(i)}$ under policy $\pi_{\boldsymbol{\theta}}$, 2) converting action $a_t^{(i)}$ into the task proportion $\boldsymbol{p}_t$, and 3) sampling the replay memory $\mathcal{M}_t$ from the historical datasets given the selected proportion. We then obtain the reward r_t and the next state s_{t+1} by evaluating the classifier on the validation sets $\mathcal{D}_{1:t+1}^{(val)}$ after learning task $t + 1$. The collected experience from each time step is stored in the experience buffer $\mathcal{B}$ for both DQN and A2C. In UPDATEPOLICY$(\cdot)$, we outline the steps for updating the policy parameters $\boldsymbol{\theta}$ with either DQN or A2C.

Algorithm 1 Monte Carlo Tree Search for Replay Scheduling

Require: Tree nodes $v_{1:T}$, Datasets $\mathcal{D}_{1:T}$, Learning rate η
Require: Replay memory size M
ACC$_{best} \leftarrow 0$, $S_{best} \leftarrow ()$
while within computational budget **do**
  $S \leftarrow ()$
  $v_t, S \leftarrow$ TREEPOLICY(v_1, S)
  $v_T, S \leftarrow$ DEFAULTPOLICY(v_t, S)
  ACC $\leftarrow$ EVALUATEREPLAYSCHEDULE$(\mathcal{D}_{1:T}, S, M)$
  BACKPROPAGATE$(v_t,$ ACC$)$
  if ACC $>$ ACC$_{best}$ **then**
    ACC$_{best} \leftarrow$ ACC
    $S_{best} \leftarrow S$
return ACC$_{best}, S_{best}$

function TREEPOLICY(v_t, S)
  while v_t is non-terminal **do**
    if v_t not fully expanded **then**
      return EXPANSION(v_t, S)
    else
      $v_t \leftarrow$ BESTCHILD(v_t)
      S.append$(\boldsymbol{p}_t)$, where $\boldsymbol{p}_t \leftarrow$ GETTASKPROPORTION(v_t)
  return v_t, S

function EXPANSION(v_t, S)
  Sample v_{t+1} uniformly among unvisited children of v_t
  S.append$(\boldsymbol{p}_{t+1})$, where $\boldsymbol{p}_{t+1} \leftarrow$ GETTASKPROPORTION(v_{t+1})
  Add new child v_{t+1} to node v_t
  return v_{t+1}, S

function BESTCHILD(v_t)
  $v_{t+1} = \underset{v_{t+1} \in \text{ children of } v}{\arg\max} \max(Q(v_{t+1})) + C\sqrt{\frac{2\log(N(v_t))}{N(v_{t+1})}}$
  return v_{t+1}

function DEFAULTPOLICY(v_t, S)
  while v_t is non-terminal **do**
    Sample v_{t+1} unIFormly among children of v_t
    S.append$(\boldsymbol{p}_{t+1})$, where $\boldsymbol{p}_{t+1} \leftarrow$ GETTASKPROPORTION(v_{t+1})
    Update $v_t \leftarrow v_{t+1}$
  return v_t, S

function EVALUATEREPLAYSCHEDULE$(\mathcal{D}_{1:T}, S, M)$
  Initialize neural network $f_{\boldsymbol{\theta}}$
  for $t = 1, \ldots, T$ **do**
    $\boldsymbol{p} \leftarrow S[t-1]$
    $\mathcal{M} \leftarrow$ GETREPLAYMEMORY$(\mathcal{D}_{1:t-1}^{(train)}, \boldsymbol{p}, M)$
    for $\mathcal{B} \sim \mathcal{D}_t^{(train)}$ **do**
      $\boldsymbol{\theta} \leftarrow SGD(\mathcal{B} \cup \mathcal{M}, \boldsymbol{\theta}, \eta)$
  $A_{1:T}^{(val)} \leftarrow$ EVALUATEACCURACY$(f_{\boldsymbol{\theta}}, \mathcal{D}_{1:T}^{(val)})$
  ACC $\leftarrow \frac{1}{T}\sum_{i=1}^{T} A_{T,i}^{(val)}$
  return ACC

function BACKPROPAGATE(v_t, R)
  while v_t is not root **do**
    $N(v_t) \leftarrow N(v_t) + 1$
    $Q(v_t) \leftarrow R$
    $v_t \leftarrow$ parent of v_t

Algorithm 2 RL Framework for Learning Replay Scheduling Policy

Require: $\mathcal{E}^{(train)}$: Training environments, $\boldsymbol{\theta}$: Policy parameters, γ: Discount factor
Require: η: Learning rate, $n_{episodes}$: Number of episodes, M: Replay memory size
Require: n_{steps}: Number of steps for A2C

$\mathcal{B} = \{\}$ ▷ Initialize experience buffer
for $i = 1, \ldots, n_{\text{episodes}}$ **do**
  for $t = 1, \ldots, T-1$ **do**
    for $E_i \in \mathcal{E}^{(train)}$ **do**
      $\mathcal{D}_{1:t+1} = \textsc{GetDatasets}(E_i, t)$ ▷ Get datasets from environment E_i
      $f_{\boldsymbol{\phi}}^{(i)} = \textsc{GetClassifier}(E_i)$ ▷ Get classifier from environment E_i
      if $t == 1$ **then**
        $\textsc{Train}(f_{\boldsymbol{\phi}}^{(i)}, \mathcal{D}_t^{(train)})$ ▷ Train classifier $f_{\boldsymbol{\phi}}^{(i)}$ on task 1
        $A_{1:t}^{(val)} = \textsc{Eval}(f_{\boldsymbol{\phi}}^{(i)}, \mathcal{D}_{1:t}^{(val)})$ ▷ Evaluate classifier $f_{\boldsymbol{\phi}}^{(i)}$ on task 1
        $s_t^{(i)} = A_{1:t}^{(val)} = [A_{1,1}^{(val)}, 0, ..., 0]$ ▷ Get initial state
      $a_t^{(i)} \sim \pi_{\boldsymbol{\theta}}(a, s_t^{(i)})$ ▷ Take action under policy $\pi_{\boldsymbol{\theta}}$
      $\boldsymbol{p}_t = \textsc{GetTaskProportion}(a_t^{(i)})$
      $\mathcal{M}_t \sim \textsc{GetReplayMemory}(\mathcal{D}_{1:t}^{(train)}, \boldsymbol{p}_t, M)$
      $\textsc{Train}(f_{\boldsymbol{\phi}}^{(i)}, \mathcal{D}_{t+1}^{(train)} \cup \mathcal{M}_t)$ ▷ Train classifier $f_{\boldsymbol{\phi}}^{(i)}$
      $A_{1:t+1}^{(val)} = \textsc{Eval}(f_{\boldsymbol{\phi}}^{(i)}, \mathcal{D}_{1:t+1}^{(val)})$ ▷ Evaluate classifier $f_{\boldsymbol{\phi}}^{(i)}$
      $s_{t+1}^{(i)} = A_{1:t+1}^{(val)} = [A_{t+1,1}^{(val)}, ..., A_{t+1,t+1}^{(val)}, 0, ..., 0]$ ▷ Get next state
      $r_t^{(i)} = \frac{1}{t+1} \sum_{j=1}^{t+1} A_{1:t+1}^{(val)}$ ▷ Compute reward
      $\mathcal{B} = \mathcal{B} \cup \{(s_t^{(i)}, a_t^{(i)}, r_t^{(i)}, s_{t+1}^{(i)})\}$ ▷ Store transition in buffer
      if time to update policy **then**
        $\boldsymbol{\theta}, \mathcal{B} = \textsc{UpdatePolicy}(\boldsymbol{\theta}, \mathcal{B}, \gamma, \eta, n_{steps})$ ▷ Update policy with experience
return $\boldsymbol{\theta}$ ▷ Return policy

function $\textsc{UpdatePolicy}(\boldsymbol{\theta}, \mathcal{B}, \gamma, \eta, n_{steps})$
  if DQN **then**
    $(s_j, a_j, r_j, s'_j) \sim \mathcal{B}$ ▷ Sample mini-batch from buffer
    $y_j = \begin{cases} r_j & \text{if } s'_j \text{ is terminal} \\ r_j + \gamma \max_a Q_{\boldsymbol{\theta}^-}(s'_j, a) & \text{else} \end{cases}$ ▷ Compute y_j with target net $\boldsymbol{\theta}^-$
    $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} (y_j - Q_{\boldsymbol{\theta}}(s_j, a_j))^2$ ▷ Update Q-function
  else if A2C **then**
    $s_t = \mathcal{B}[n_{steps}]$ ▷ Get last state in buffer
    $R = \begin{cases} 0 & \text{if } s_t \text{ is terminal} \\ V_{\boldsymbol{\theta}_v}(s_t) & \text{else} \end{cases}$ ▷ Bootstrap from last state
    for $j = n_{steps} - 1, ..., 0$ **do**
      $s_j, a_j, r_j = \mathcal{B}[j]$ ▷ Get state, action, and reward at step j
      $R = r_j + \gamma R$
      $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_j, s_j)(R - V_{\boldsymbol{\theta}_v}(s_j))$ ▷ Update policy
      $\boldsymbol{\theta}_v = \boldsymbol{\theta}_v - \eta \nabla_{\boldsymbol{\theta}_v} (R - V_{\boldsymbol{\theta}_v}(s_j))^2$ ▷ Update value function
    $\mathcal{B} = \{\}$ ▷ Reset experience buffer
  return $\boldsymbol{\theta}, \mathcal{B}$

E Extended Related Work

In this section, we give a brief overview of various approaches in CL, especially replay methods as well as spaced repetition techniques for human CL and generalization in RL.

Continual Learning. Traditional CL can be divided into three main areas, namely regularization-based, architecture-based, and replay-based approaches. Regularization-based methods protect parameters influencing the performance on known tasks from wide changes and use the other parameters for learning new tasks [1, 49, 53, 59, 68, 77, 98]. Architecture-based methods mitigate catastrophic forgetting by maintaining task-specific parameters [29, 61, 76, 78, 93, 94, 96]. Replay methods mix samples from old tasks with the current dataset to mitigate catastrophic forgetting, where the replay samples are either stored in an external memory [2, 3, 9, 18, 20, 36, 37, 44, 45, 47, 60, 68, 72, 73, 75, 87, 95] or generated using a generative model [79, 85]. Regularization-based approaches and dynamic architectures have been combined with replay-based approaches to methods to overcome their limitations [12, 15, 17, 16, 26, 29, 48, 62, 68, 69, 72, 75, 90]. Our work relates most to replay-based methods with external memory which we spend more time on describing in the next paragraph.

Replay-based Continual Learning. Much research effort in replay- or memory-based CL has focused on selecting higher quality samples to store in memory [3, 9, 18, 20, 36, 45, 60, 68, 73, 95]. In [18], several selection strategies are reviewed in scenarios with tiny memory capacity, such as reservoir sampling [89], first-in first-out buffer [60], k-Means, and Mean-of-Features [73]. However, elaborate selection strategies have been shown to give little benefit over random selection for image classification problems [15, 37]. More recently, there has been work on compressing raw images to feature representations to increase the number of memory examples for replay [37, 44, 72]. Selecting the time to replay old tasks has mostly been ignored in the literature, with an exception in [2] which replays memory samples that would most interfere with a foreseen parameter update. Our replay scheduling approach differs from the above mentioned works since we focus on learning to select which tasks to replay. Nevertheless, our scheduling can be combined with any selection strategy and replay-based method.

Human Continual Learning. Humans are CL systems in the sense of learning tasks and concepts sequentially. The timing of learning and rehearsal is essential for humans to memorize better [24, 27, 91]. An example technique is spaced repetition where time intervals between rehearsal are gradually increased to improve long-term memory retention [24, 28], which has been shown to improve memory retention better uniformly spaced rehearsal times [35, 57]. Several works in CL with neural networks are inspired by human learning techniques, including spaced repetition [4, 31, 81], sleep mechanisms [6, 61, 77], and memory reactivation [37, 84]. Replay scheduling is also inspired by spaced repetition, where we learn schedules of which tasks to replay at different times.

Generalization in Reinforcement Learning. Generalization is an active research topic in RL [52] as RL agents tend to overfit to their training environments [39, 99, 101, 100]. The goal is often to transfer learned policies to environments with new tasks [32, 50, 40] and action spaces [13, 46, 14]. Some approaches aim to improve generalization capabilities by generating more diverse training data [21, 83, 99], using network regularization or inductive biases [30, 41, 97], or learning dynamics models [7, 67]. In this paper, we use RL for learning policies for selecting which tasks a CL network should replay. The goal is to learn policies that can be applied in new CL environments for replay scheduling on unseen task orders and datasets without additional computational cost.

F Additional Figures and Tables

In this section, we show figures and tables that provides additional information to the experiments in Section 4.

F.1 Memory Usage in Experiments on Replay Scheduling Efficiency

We visualize the memory usage in the experiment on efficiency of replay scheduling in Section 4.1. For the 5-task datasets, the replay memory size for MCTS is set to $M = 2$, such that only 2 samples can be selected for replay at all times. Similarly, we set $M = 50$ for the 10- and 20-task datasets which have 100 classes to learn in total. The baselines A-GEM [17], ER-Ring [18], and Uniform use an incremental memory in order to replay 1 sample/class at all tasks. We visualize the memory usage for our method and the baselines for the 5-task datasets in Figure 12. Here, the memory capacity is reached at task 2, while the baselines must increment their memory size. Figure 13 shows the memory usage for Permuted MNIST and the 20-task datasets Split CIFAR-100 and Split miniImagenet. The memory size capacity for our method is reached after learning task 6 and task 11 on the Permuted MNIST and the 20-task datasets respectively, while the baselines continue incrementing their replay memory size.

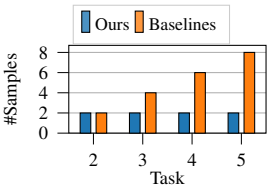


Figure 12: Number of replayed samples per task for the 5-task datasets in the tiny memory setting.

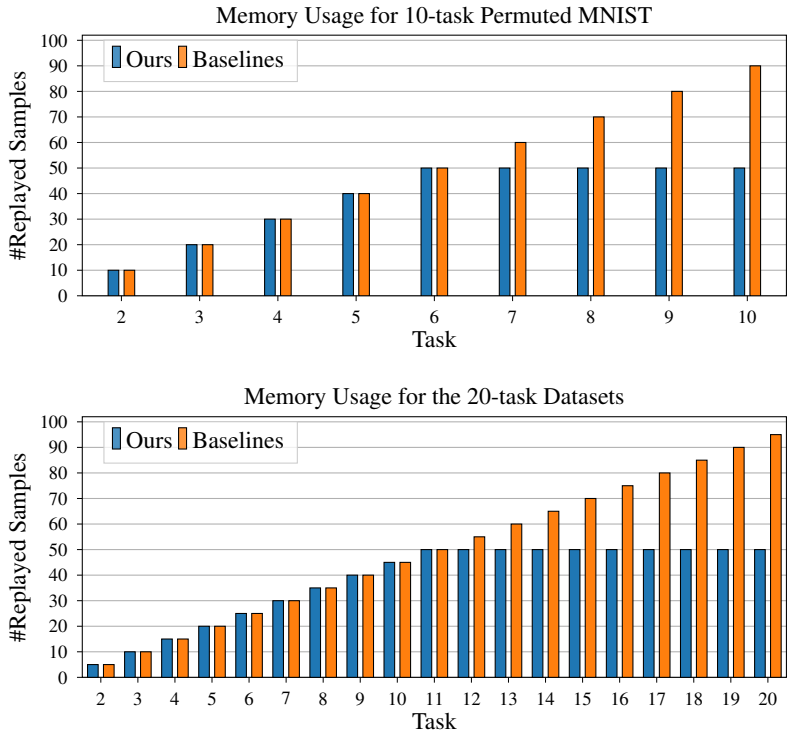


Figure 13: Number of replayed samples per task for 10-task Permuted MNIST (top) and the 20-task datasets in the experiment in Section 4.1. The fixed memory size $M = 50$ for our method is reached after learning task 6 and task 11 on the Permuted MNIST and the 20-task datasets respectively, while the baselines continue incrementing their number of replay samples per task.

F.2 Task Splits in Test Environments in Policy Generalization Experiments

Here, we provide the task splits of the test environments used in the policy generalization experiments in Section 4.2. We evaluated all methods using 10 test environments in all experiments. The test environments in the New Task Order experiments were generated with seeds 10-19. We show the

task splits for the Split MNIST, Split FashionMNIST, and Split CIFAR-10 environments in Table 19, 20, and 21 respectively. The test environments in the New Dataset experiments were generated with seeds 0-9. We show the task splits for the Split notMNIST and Split FashionMNIST environments in Table 22 and 23 respectively.

Table 19: Task splits with their corresponding seed for test environments of Split MNIST datasets in the **New Task Orders** experiments in Section 4.2.

Seed	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**
10	8, 2	5, 6	3, 1	0, 7	4, 9
11	7, 8	2, 6	4, 5	1, 3	0, 9
12	5, 8	7, 0	4, 9	3, 2	1, 6
13	3, 5	6, 1	4, 7	8, 9	0, 2
14	3, 9	0, 5	4, 2	1, 7	6, 8
15	2, 6	1, 3	7, 0	9, 4	5, 8
16	6, 2	0, 7	8, 4	3, 1	5, 9
17	7, 2	5, 3	4, 0	9, 8	6, 1
18	7, 9	0, 4	2, 1	6, 5	8, 3
19	1, 7	9, 6	8, 4	3, 0	2, 5

Table 20: Task splits with their corresponding seed for test environments of Split FashionMNIST datasets in the **New Task Orders** experiments in Section 4.2.

Seed	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**
10	Bag, Pullover	Sandal, Shirt	Dress, Trouser	T-shirt/top, Sneaker	Coat, Ankle boot
11	Sneaker, Bag	Pullover, Shirt	Coat, Sandal	Trouser, Dress	T-shirt/top, Ankle boot
12	Sandal, Bag	Sneaker, T-shirt/top	Coat, Ankle boot	Dress, Pullover	Trouser, Shirt
13	Dress, Sandal	Shirt, Trouser	Coat, Sneaker	Bag, Ankle boot	T-shirt/top, Pullover
14	Dress, Ankle boot	T-shirt/top, Sandal	Coat, Pullover	Trouser, Sneaker	Shirt, Bag
15	Pullover, Shirt	Trouser, Dress	Sneaker, T-shirt/top	Ankle boot, Coat	Sandal, Bag
16	Shirt, Pullover	T-shirt/top, Sneaker	Bag, Coat	Dress, Trouser	Sandal, Ankle boot
17	Sneaker, Pullover	Sandal, Dress	Coat, T-shirt/top	Ankle boot, Bag	Shirt, Trouser
18	Sneaker, Ankle boot	T-shirt/top, Coat	Pullover, Trouser	Shirt, Sandal	Bag, Dress
19	Trouser, Sneaker	Ankle boot, Shirt	Bag, Coat	Dress, T-shirt/top	Pullover, Sandal

Table 21: Task splits with their corresponding seed for test environments of Split CIFAR-10 datasets in the **New Task Orders** experiments in Section 4.2.

Seed	Task 1	Task 2	Task 3	Task 4	Task 5
10	Ship, Bird	Dog, Frog	Cat, Automobile	Airplane, Horse	Deer, Truck
11	Horse, Ship	Bird, Frog	Deer, Dog	Automobile, Cat	Airplane, Truck
12	Dog, Ship	Horse, Airplane	Deer, Truck	Cat, Bird	Automobile, Frog
13	Cat, Dog	Frog, Automobile	Deer, Horse	Ship, Truck	Airplane, Bird
14	Cat, Truck	Airplane, Dog	Deer, Bird	Automobile, Horse	Frog, Ship
15	Bird, Frog	Automobile, Cat	Horse, Airplane	Truck, Deer	Dog, Ship
16	Frog, Bird	Airplane, Horse	Ship, Deer	Cat, Automobile	Dog, Truck
17	Horse, Bird	Dog, Cat	Deer, Airplane	Truck, Ship	Frog, Automobile
18	Horse, Truck	Airplane, Deer	Bird, Automobile	Frog, Dog	Ship, Cat
19	Automobile, Horse	Truck, Frog	Ship, Deer	Cat, Airplane	Bird, Dog

Table 22: Task splits with their corresponding seed for test environments of Split notMNIST datasets in the **New Dataset** experiments in Section 4.2.

Seed	Task 1	Task 2	Task 3	Task 4	Task 5
0	A, B	C, D	E, F	G, H	I, J
1	C, J	G, E	A, D	B, H	I, F
2	E, B	F, A	H, C	D, G	J, I
3	F, E	B, C	J, G	H, A	D, I
4	D, I	E, J	C, G	A, B	F, H
5	J, F	C, E	H, B	A, I	G, D
6	I, B	H, A	G, F	C, E	D, J
7	I, F	A, C	B, J	H, D	G, E
8	I, G	J, A	C, F	H, B	E, D
9	I, E	H, C	B, J	D, A	G, F

Table 23: Task splits with their corresponding seed for test environments of Split FashionMNIST datasets in the **New Dataset** experiments in Section 4.2.

Seed	Task 1	Task 2	Task 3	Task 4	Task 5
0	T-shirt/top, Trouser	Pullover, Dress	Coat, Sandal	Shirt, Sneaker	Bag, Ankle boot
1	Pullover, Ankle boot	Shirt, Coat	T-shirt/top, Dress	Trouser, Sneaker	Bag, Sandal
2	Coat, Trouser	Sandal, T-shirt/top	Sneaker, Pullover	Dress, Shirt	Ankle boot, Bag
3	Sandal, Coat	Trouser, Pullover	Ankle boot, Shirt	Sneaker, T-shirt/top	Dress, Bag
4	Dress, Bag	Coat, Ankle boot	Pullover, Shirt	T-shirt/top, Trouser	Sandal, Sneaker
5	Ankle boot, Sandal	Pullover, Coat	Sneaker, Trouser	T-shirt/top, Bag	Shirt, Dress
6	Bag, Trouser	Sneaker, T-shirt/top	Shirt, Sandal	Pullover, Coat	Dress, Ankle boot
7	Bag, Sandal	T-shirt/top, Pullover	Trouser, Ankle boot	Sneaker, Dress	Shirt, Coat
8	Bag, Shirt	Ankle boot, T-shirt/top	Pullover, Sandal	Sneaker, Trouser	Coat, Dress
9	Bag, Coat	Sneaker, Pullover	Trouser, Ankle boot	Dress, T-shirt/top	Shirt, Sandal