Evaluating Pretrained models for Deployable Lifelong Learning

Kiran Lekkala, Eshan Bhargava, Yunhao Ge, Laurent Itti

`klekkala@usc.edu, ebhargav@usc.edu, yunhaoge@usc.edu, itti@usc.edu`

Thomas Lord Department of Computer Science
University of Southern California

First two authors contributed equally

Abstract

We create a novel benchmark for evaluating a Deployable Lifelong Learning system for Visual Reinforcement Learning (RL) that is pretrained on a curated dataset, and propose a novel Scalable Lifelong Learning system capable of retaining knowledge from the previously learnt RL tasks. Our benchmark measures the efficacy of a deployable Lifelong Learning system that is evaluated on scalability, performance and resource utilization. Our proposed system, once pretrained on the dataset, can be deployed to perform continual learning on unseen tasks. Our proposed method consists of a Few Shot Class Incremental Learning (FS-CIL) based task-mapper and an encoder/backbone trained entirely using the pretrain dataset. The policy parameters corresponding to the recognized task are then loaded to perform the task. We show that this system can be scaled to incorporate a large number of tasks due to the small memory footprint and fewer computational resources. We perform experiments on our DeLL (Deployment for Lifelong Learning) benchmark on the Atari games to determine the efficacy of the system.

1. Introduction

Humans have an innate ability to sequentially learn and perform new tasks without forgetting them, all while leveraging prior knowledge during this process. Continual learning is an imperative skill that needs to be acquired by any intelligent machine. This is especially true in the real-world, where environments keep evolving; thus, agents need to remember previously executed tasks in order to perform these tasks in the future without forgetting. Current continual learning methods use complex memory modules and data augmentations that become difficult to scale and deploy on real-world robotic systems. Furthermore, it's important that models are pretrained offline using large datasets so that when deployed they offer good inductive bias to warm-start the learning process.
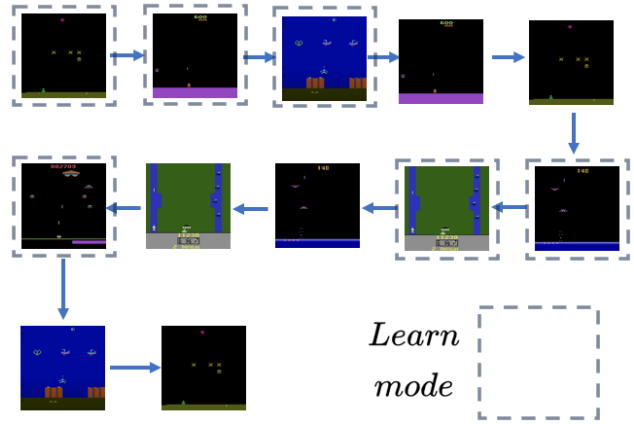

Figure 1. An example run for a typical Lifelong Learning system. The agent is given tasks sequentially with an objective of maximizing performance and minimizing total training time and resource utilization. Training time is counted only during the *Learn mode*.

As an attempt to solve the Lifelong Learning problem for Visual Reinforcement Learning (RL), especially those having more representation complexities than the control, we propose a simple, yet efficient Lifelong Learning system that can be pretrained on a large dataset offline and deployed on a real-world system. The core of our system consists of a meta task-mapper that learns to identify tasks, even when new tasks are given on the fly. Our method's primary novelty lies in the fact that the system is pretrained on a dataset and performs continual learning on a benchmark, with no overlap between both the data distributions.

Lifelong Learning has gained massive popularity in the recent years. [8] proposes a self-improving Lifelong Learning framework for mobile robot navigation to improve behaviour purely based on its own experience and retain the learnt tasks, but such robots have to retain the experiences of the previous environments. Methods like CRIL [5] apply deep generative replay to alleviate *Catastrophic Forgetting* by generating pseudo-data to train new tasks. Deep
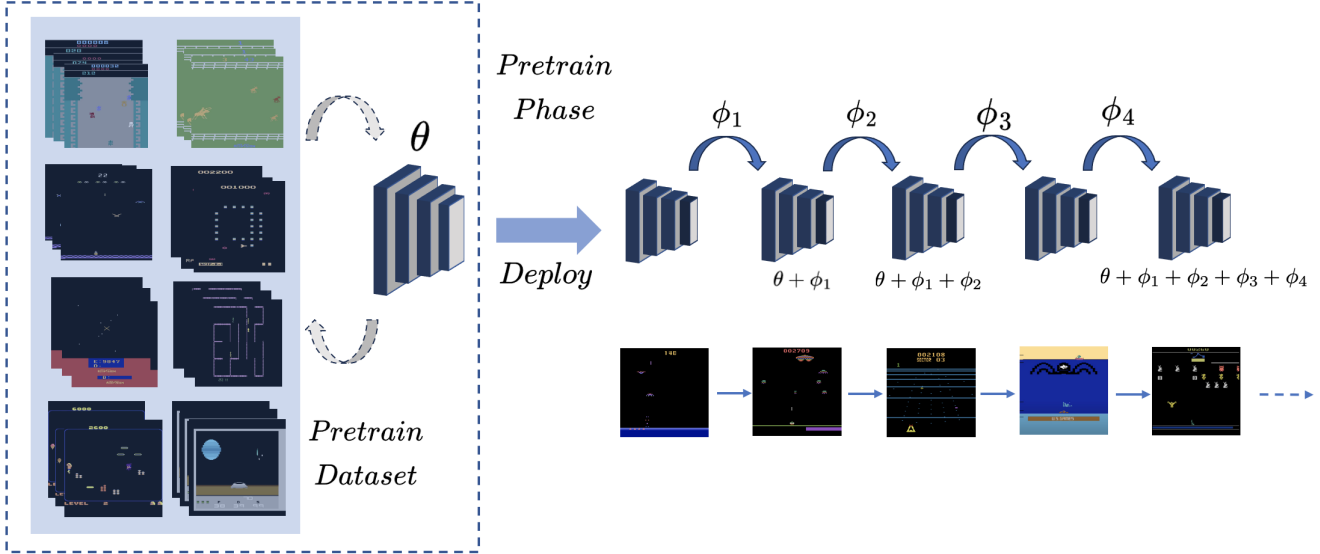
Figure 2. Evaluating *Deployable Lifelong Learning* system involves pretraining a system on a dataset and then deploying it. This process consists of freezing the model parameters and allowing the system to quickly learn and adapt to unseen tasks on the fly. As we can see in the above figure, during deployment, the model sequentially accommodates continual learning on a variable number of tasks by adding a few parameters or datapoints in the data-buffer.

Reinforcement learning (RL) amidst Lifelong Learning is explored by [2]. Progressive networks [13] start with a single column and a new column for new tasks, although this method is limited by parameters growing faster than the number of tasks. [2] introduces hypernetworks, a meta-model that generates the parameters of a target network that solves the task by using a trainable task embedding vector as an input.

[14] follows a similar setting as ours as it uses pretraining and then online learning, however only the policy and the critic are pretrained and data collected is mixed and filtered. Likewise, DARC [3] uses domain adaptation and transfer-learning. The model can overcome the difference between the source and target environment, including dynamics, by estimating δ_r using a pair of binary classifiers. Online adaption or forward transfers are explored in [6, 1]. Distillation-based methods [12, 16] are well suited for model/data compression, imitation and multi-task learning. [16] involves model distillation by keeping the policy closer to the behaviour policy that generated the data.

Offline pretraining is a fast growing field that involves using unlabeled, unorganized data that can be used to learn a pretrained representation model [1]. This model can be used to learn the inductive bias of tasks (like temporal sequencing), relationships of actions with states, and value function estimates corresponding to a state. Currently, the only forward transfer we have in our system is the priors that the backbone/encoder model learns during the offline pre-training, but we are currently working on improving trans-fer within the games pertaining to the same type.

Existing Lifelong Learning benchmarks evaluate many aspects of the system. [11] is one of the first few proposed benchmarks for RL. Compositional Lifelong Learning, like [9], evaluates on the functional aspects of Lifelong Learning. In the OpenAI Atari suite, Gym Retro [10] consists of a large-scale game emulator that has over 1000 games, which could be used to train RL agents. Unlike the above, our benchmark primarily focuses on scalability and resource utilization for deployable Lifelong Learning systems. *The following are the outlined contributions for this paper:*

1. We collect YouTube videos of Atari games (not in the OpenAI Atari suite) played by human experts and create a dataset that is used by a Lifelong Learning system for pretraining. The model is evaluated on sequential learning of unseen games that are based on OpenAI Gym, and have no overlap with games in the pretrain dataset.

2. To evaluate *Deployable Lifelong Learning* system on performance and resource utilization, we propose a novel benchmark. The code and leaderboard for using the benchmark are made available.

3. Lastly, we propose a novel method that uses the above dataset and benchmark. Our method is based on Few Shot Class Incremental Learning (FSCIL) to learn task differences from the pretrain dataset collected offline and quickly generalize to unseen games.
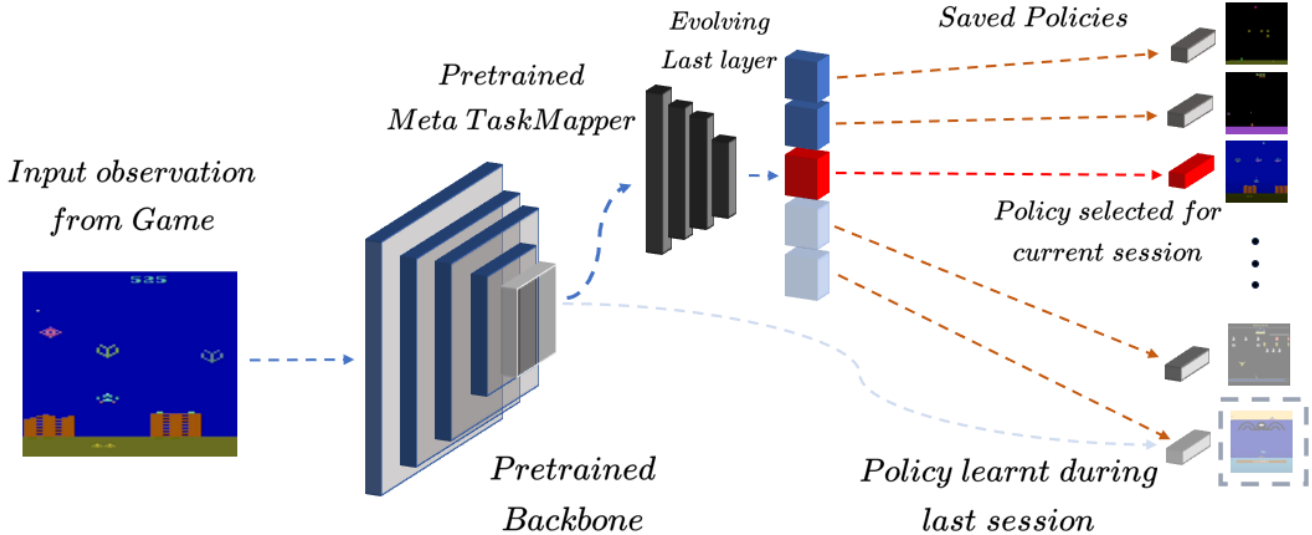
Figure 3. Proposed architecture of our system. Our system contains an encoder and a task-mapper that are pretrained on a large offline dataset. When deployed, our system can identify the previously learned task using the observation from the game. By detecting the task, the appropriate policy can be loaded. On the top, we have the task-mapper, whose last layer is adapted based on the policies the model is currently learning. Arrows and the red modules represent the current policy, selected and loaded, by the task-mapper.

2. Dataset and Benchmark

Before describing our proposed method for Scalable and Deployable Lifelong Learning, we first detail our dataset and benchmark for evaluation. These could be used by researchers to evaluate their models to asses the ability when deployed on real-world systems.

2.1. Dataset for Pretraining

To pretrain the system, we collected a dataset using expert-played YouTube videos, such that every video was extracted at 10fps to obtain a sequence of observations. All of these games are different from the games in OpenAI Atari suite. A total of 1,116,275 images with a dimension of 360×480 were collected as part of the pretrain dataset. All the images were cropped and resized to 84×84. The list of the games and the format of the dataset can be found here [1]

For each video frame, we also extract associated rewards directly from the frame. Every Atari game consists of the score that is awarded from the start of the game. We use Tesseract OCR engine [7] by providing the bounding box of the reward location for each video in order to obtain the reward value for each frame. All the rewards are normalized across both the video and the game by only computing the difference of rewards of the frames as given in the equation below:

$$r^t_{norm} = \begin{cases} 0, & r^t - r^{t-1} = 0, \\ 1, & r^t - r^{t-1} > 0. \end{cases} \quad (1)$$

The dataset consists of 101 folders, each corresponding to a specific game. Each folder consists of a set of Numpy files that consists of a sequence of observations along with the normalized reward value.

Along with the pretrain dataset, we also provide a meta file describing each of the games in the pretrain dataset. The meta file consists of the following attributes

1. **Game Name**: Name of the specific game.

2. **Game Type**: Genre of the game. Various Atari games are classified under genres like *Shoot'em up*, *Maze* etc.

3. **Input Text**: A brief description of the game's objective. This would enable the agent to understand the game and reuse any previously learnt skills.

4. **Minimum Reward**: Minimum reward required by the agent to not switch to learn mode. This reward is computed by evaluating a specific game using a frozen, randomly initialized model.

5. **Maximum Reward**: Reward obtained by an end-to-end trained agent. We used a CNN Encoder and the PPO Algorithm for training.
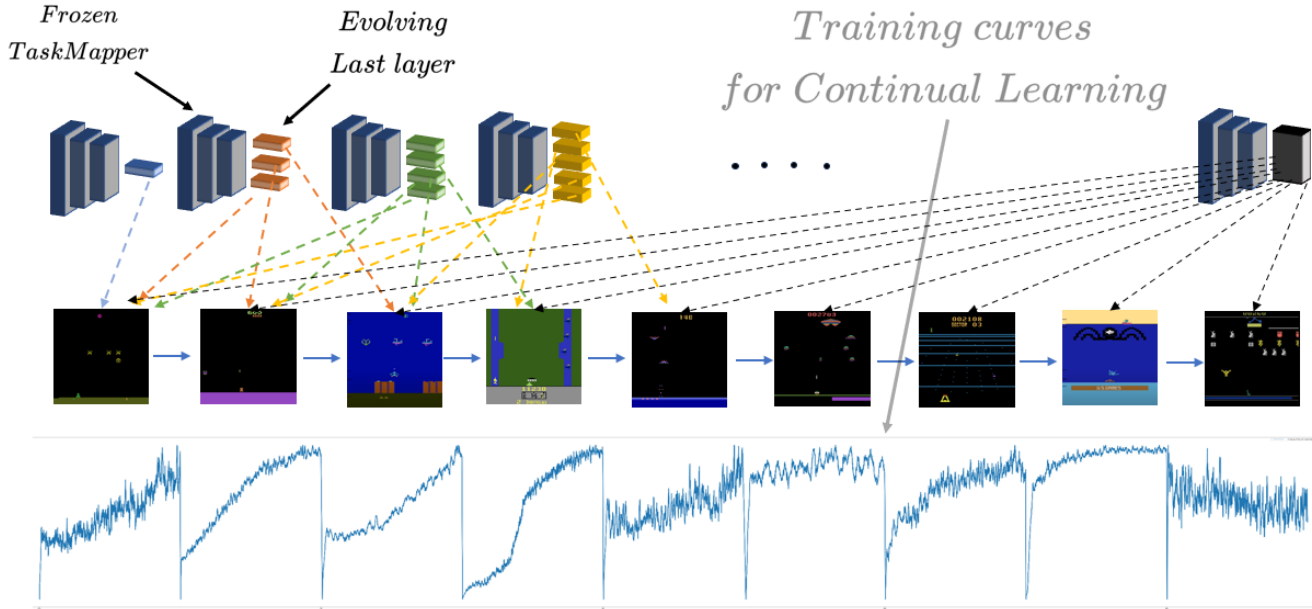
Figure 4. Pictorial overview of our method. With every new game the agent learns, the task-mapper needs to adapt the last layer parameters such that it's able to include the new game in the class prediction. Blue training curves represent the model training on a specific game.

2.2. Benchmarking Deployable Lifelong Learning

Once a model is trained on the above dataset during the pretraining phase, we then evaluate the model's performance on the $DeLL$ benchmark. Note, that we use the term "model" not only for the checkpoint but also as a program for the entire system that takes in an input from the benchmark. The benchmark loads the pretrained model and performs evaluation.

A specific Benchmark $DeLL$ is parameterized by α and β. α corresponds to the total number of unique games present in the benchmark, and β corresponds to the total number of games the agent is given one after the other. Note that for all cases, $\beta > \alpha$. A specific $DeLL$ benchmark consists of *.yaml* file that has a list of games and the specific game type. Its always assumed that all the game types present in the benchmark are also present in the pretrain dataset, although none of the game themselves are present in the pretrain dataset. For example, the games $Demon Attack$ and $SpaceInvaders$ are part of the benchmark and $Xevious$ and $Galaga$ are part of the pretrain dataset, although all of them fall under *Shoot up* games.

We urge the reader to take note of some terminology that is beneficial for understanding the benchmark. Firstly, we use the term *task* and *game* interchangeably since we are currently concerned with Atari games. A specific benchmark $DeLL(\alpha, \beta)$ consists of sequentially evaluating the model on β games, which we call a run. In a run of β games, there are α unique games. A model learns or performs inference on any game in the β games during a *ses-*

sion. A high-level overview of a run is presented in Figure 1.

Any method/model that is evaluated on a specific benchmark yields 4 different metrics. The following metrics are employed to evaluate the Lifelong Learning system that corresponds to a specific benchmark $DeLL(\alpha, \beta)$. α and β correspond to the number of unique games and the total number of games the agent is given one after the other, respectively.

1. **Model Size (MS)** (In *MB*): The size of the model when deployed.

2. **Model Inference time (MI)** (In *ms*): Mean inference time of the model on all the β games in a run.

3. **Learn Switches (LS)**: Number of times the agent switches to learn mode. If the agent obtains a score lower than the minimum reward, then the agent switches to learn mode.

4. **Model Growth (MG)**: Every time the model switches to learn mode, and thereby learns a task, there is an increase in buffer size, model size or both. This value estimates the average percentage increase (in *MB*) of the model after every learn switch.

5. **Buffer Size (BS)**: A metric (in *KB*) the model uses to avoid *Catastrophic Forgetting*. This could be any form of data, including images, embeddings, rewards and actions.
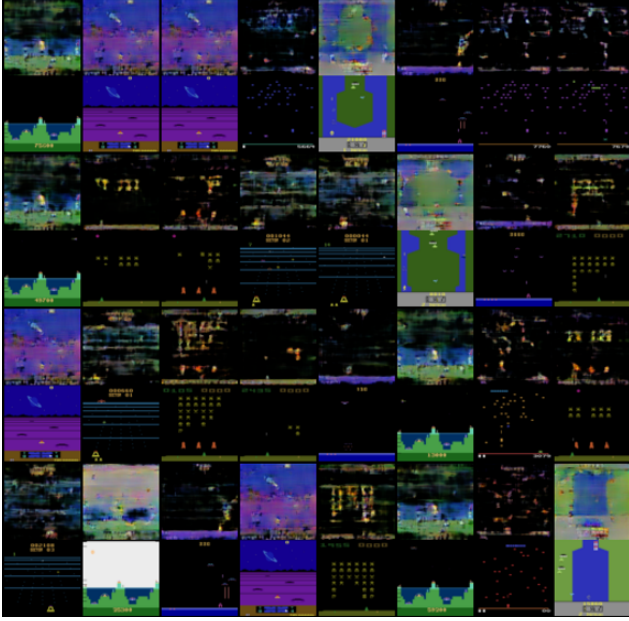
Figure 5. Reconstructions obtained from the trained VAE model on the test-dataset. *Note that the model has not been trained on any of the above games.* Odd rows correspond to the reconstructions of the subsequent even rows.

6. **Buffer Growth (BG)** The average percentage increase of the data buffer increase after every learn switch.

7. **Mean Avg Reward (MAR)** Measured during evaluation of a game. This is an array metric, where the length of the array corresponds to the total number of unique games (α) in a run. This excludes the sessions that the agent used for learning the game.

8. **Total Normalized Mean Reward (TNMR)**: We normalize all the values in the MAR array using the minimum and the maximum reward of the corresponding game, and then compute the mean of the array to get TNMR.

3. Proposed Method

We propose a Lifelong Learning system designed to identify its originating task using minimal resources, making it well-suited for real-time systems. One of the most important features of our method is that, unlike other methods, it scales well when the number of tasks increases. The most significant novelty of our method is that the entire system is pretrained on a dataset that is different from the dataset used in deployment.

3.1. Pretrained Encoder

We use a pretrained encoder to extract the relevant features from the observations required for task identification

and downstream policy execution. During the deployment phase, the pretrained encoder is frozen and used to infer the embeddings, which are then utilized by the task-mapper. Currently, we use a VAE-based encoder that is trained on the pretrain dataset. The reconstructions obtained using the encoder model is presented in Figure 5.

3.2. Meta Task-mapper

One of the major challenges in Lifelong Learning, apart from reusing prior knowledge, is to continually adapt and remember previous tasks. We tackle *Catastrophic Forgetting* by transforming *Continual Learning* into a task identification problem. By predicting the appropriate Task ID during classification, we can load the appropriate policy that was previously learnt, and perform the task. Since the policies are a 1 layer neural network, the number of tasks can be scaled easily.

We utilize a meta task-mapper that is also trained offline, along with the backbone, to recognize task differences. Given a few observations, the task-mapper learns to identify which of the previously learnt tasks the current task falls into. The task-mapper, denoted by, g_ϕ parameterized by ϕ is trained on a large pretrain dataset. Since the task-mapper has already recognized the differences in the tasks during the pretraining phase, it only needs to adapt to the new tasks using a few-shot learning setting.

In many real-world instances, the agent needs to keep track of a diverse number of tasks that may keep increasing. In this case, the task-mapper must also accommodate the newer predictions. To allow this, we apply CEC-FSCIL [15], which was originally proposed to solve class-incremental continual learning. This method uses a trained graph neural network to learn the correspondences and relations of the classes. During the training phase, the method uses pseudo-incremental training by simulating sequences of different classes in the pretrain dataset. This would mimic how each class would be included in every session during test time. Furthermore the graph model allows a trained task-mapper to be extended to indefinite number of classes by aggregating a list of last-layer parameters corresponding to each class. In the below equation, N corresponds to the N-way classification:

$$\mathbf{W}_{last} = \{w_0, w_1, w_2, w_3, ..., w_N\} \tag{2}$$

Once the task-mapper receives the data embeddings for the new sessions, the learnt classifiers in the current session and previous sessions are fed to the graph model for adaptation. The adaptation is done using the support data for N way K shot classification. *At any given point during deployment, the data buffer would consist of $N * K$ datapoints*, where N is the number of learnt tasks (tasks that made the model switch to learn mode). Finally, the updated

classifiers can be used for evaluation. As a baseline comparison, we also use a Meta learning [4] based task-mapper, which unlike our FSCIL based task-mapper has no plasticity. Nonetheless, it can still perform the task-mapping for unseen data during deployment. In which case, for every N, there needs to be a different task-mapper. Although this is a naive approach, compared to our task-mapper, we show that even at the expense of more parameters, the CEC-FSCIL based task-mapper outperforms Meta learning based task-mapper for larger values of N.

Algorithm 1 System Pre-training

Require: Offline dataset $\mathcal{D}_{Train}$ containing only sequence of observations $\{o_1^i, o_2^i, .., o_T^i\}_{i=1}^M$
1: Train a ResNet VAE using $\mathcal{D}_{Train}$
2: Freeze and obtain Encoder f_θ from ResNet VAE
3: Initialize Task-mapper g_ϕ
4: **while** training not done **do**
5:     Select train tasks $\tau_i \sim p_{Train}$
6:     Obtain (Image, class) pairs $(o_1, c_1), (o_2, c_2), ..(o_K, c_K)$ corresponding to τ_i
7:     Estimate and apply gradients using Meta learning or FSCIL loss on g_ϕ
8: **end while**

Algorithm 2 System Evaluation

Require: Pretrained Backbone f_θ
Require: Pretrained task-mapper g_ϕ
Require: List of learnt policies $\mathcal{P}$ of size N.
Require: Initialize Task-mapper output to N
Require: Initialize buffer data $\mathcal{D}_{buf}$ containing $N*K$ datapoints
1: **while** not done **do**
2:     Request task τ for evaluation
3:     **if** mode = *TRAIN* **then**
4:         Obtain data O and trained policy p through $O, p \leftarrow RL\text{-}Procedure(\tau)$
5:         *Selective-Sample* Offline data O into $\hat{O}$
6:         Update list of learnt policies $\mathcal{P} \leftarrow \mathcal{P} \cup p$
7:         Update buffer data $\mathcal{D}_{buf} \leftarrow \mathcal{D}_{buf} \cup \hat{O}$
8:         Update output size for the task-mapper g_ϕ to $N+1$
9:         Perform N way, K shot adaptation on g_ϕ
10:     **else**
11:         Obtain test observation(s) $\mathbf{o} \sim \tau$
12:         Infer Task ID i using the task-mapper $g_\phi(o)$
13:         Load Policy i from $\mathcal{P}$ and perform task τ
14:     **end if**
15: **end while**

Once the backbone and the task-mapper are pretrained on the offline data, using the procedure mentioned in Algorithm 1, they are then deployed on a real-world system and the last layer parameters of the task-mapper are updated using K shot adaptation through the support set. In order to obtain a maximum performance gain using the support set, it's important that we are selective about these K shot support samples from the copious amount of data generated during the RL procedure for learning the new task. Furthermore, by eliminating all the redundant data, we can minimize the computational overhead and accommodate more tasks. Currently, we choose K random samples in a task and store them in a buffer that gets propagated with new sessions, although more optimal methods for selecting the support set may exist. We will be exploring this in our future work. We also show the evaluation/deployment of the system in Algorithm 2.

3.3. Policy

To perform a specific task, we employ a 1-Layer policy that receives the feature embedding for action. Using a *Float16* quantized format, we store each policy in under 1.5 MB, tagged with its class-id from the task-mapper.

4. Evaluation and Results

We use a ResNet-based architecture for the VAE Encoder. The encoder and the decoder block are built for a 84×84 image and result in a latent vector of size of 512. The entire network was end-to-end trained for 100 epochs which took about 27 hours on a NVIDIA V100 GPU. The task-mapper consists of a Feed forward Neural Network that has a variable last layer based on the number of learnt tasks. The last layer weights are the only parameters that are continuously updated as the agent keeps learning unseen tasks, and the remaining parameters stay frozen.

4.1. Evaluation of the system

We perform experiments by evaluating our system on a set of 11 Atari games from OpenAI Gym. The list of games and their corresponding results are outlined in Table 2. Note that for all these evaluations, the encoder was trained on the pretrain dataset and frozen, and only the policy is trained on the respective game. We also provide the results obtained using our benchmark on our proposed Lifelong Learning system and 3 other baselines (Random encoder, E2E (End to end trained) and Meta learning based task-mapper) in Table 3.

4.2. Ablation experiments for task-mapper

The results obtained from the CEC-FSCIL task-mapper are presented in Table 1, alongside a baseline comparison from the Meta learning-based task-mapper. From these tables, we can see that the CEC-FSCIL task-mapper results

Table 1. On the left we see the accuracy of a Meta learning based task-mapper. On the right, we see the accuracy of a CEC-FSCIL based task-mapper. The disadvantage of using a Meta learning based task-mapper is the need for separate last layer for each of the row, whereas in the CEC-FSCIL based task-mapper a single pretrained task-mapper can be used for sequential adaptation.

	1-shot	2-shot	5-shot	10-shot	15-shot
1-way	0.76	0.77	0.77	0.78	**0.79**
2-way	0.74	0.78	0.78	**0.79**	0.78
3-way	0.76	**0.78**	0.77	0.77	0.77
4-way	0.77	0.77	0.77	0.77	**0.78**
5-way	0.78	0.76	**0.79**	0.79	0.78
6-way	**0.79**	0.77	0.79	0.79	0.78
7-way	0.79	0.77	0.79	0.77	**0.81**
8-way	0.76	0.78	**0.80**	0.78	0.78
9-way	0.78	**0.79**	0.77	0.78	0.79
10-way	0.77	0.79	0.80	**0.81**	0.77
11-way	0.74	0.80	0.78	**0.81**	0.79
12-way	0.77	0.78	0.78	**0.80**	0.80
13-way	0.76	0.77	0.77	0.78	**0.79**
14-way	0.78	0.78	0.78	0.78	**0.79**
15-way	0.81	0.78	**0.82**	0.82	0.78
16-way	**0.80**	0.79	0.79	0.77	0.77
17-way	**0.81**	0.80	0.81	0.78	0.78
18-way	0.75	0.76	**0.81**	0.78	x
19-way	0.77	**0.82**	0.77	0.78	x
20-way	0.77	0.80	0.81	**0.82**	x

	1-shot	2-shot	5-shot	10-shot
5-way	0.80	0.84	0.93	0.94
10-way	0.65	0.77	0.84	0.88
20-way	0.55	0.62	0.75	0.80
30-way	0.42	0.51	0.56	0.57

Table 2. Performance of the proposed system when evaluated on the games sequentially. Note that the reward values are formatted with the respective **mean/median** value. Random and Trained correspond to the random or pretrained encoders that are frozen. Net-mean total reward corresponds to the average value of the reward after multiplying the CEC accuracy to the Trained-agent reward value. This value estimates the performance of our system during sequential evaluation of the games. Note that for all the sequentially executed tasks, the CEC based task-mapper performs better than the baseline (Meta learning based task-mapper).

Name	Random	ML	CIFL	Trained	Net-mean
AirRaid-v4	605.0/605.0	1.0	1.0	750.0/712.5	750.0
Assault-v4	272.0/268.0	0.8	0.76	300.3/315.0	228.2
BeamRider-v4	420.0/425.0	0.8	0.74	440.0/440.0	325.6
Carnival-v4	2123.0/2242.0	0.8	0.76	2639.0/2810.0	2005.64
DemonAttack-v4	240.0/255.0	0.8	0.77	276.0/257.5	212.5
NameThisGame-v4	3923.0/4310.0	0.65	0.78	4052.0/4230.0	3160.56
Pooyan-v4	1020.0/1120.0	0.65	0.79	1106.5/997.5	874.13
Gopher-v4	732.1/733.0	0.65	0.76	746.0/620.0	566.96
Riverraid-v4	2723.0/2740.0	0.65	0.78	2886.0/2815.0	2251.08
Solaris-v4	1001.0/1101.0	0.65	0.77	1094.0/840.0	842.38
SpaceInvaders-v4	401.0/435.0	0.55	0.74	427.0/385.0	315.98

stay consistent throughout different N way, whereas for the Meta learning based task-mapper performs well with a low N but the performance significantly drops with increasing N.

5. Discussion and Conclusion

We propose a dataset and benchmark that allows for evaluating deployable Lifelong Learning systems. The dataset consists of sequences of games and rewards obtained from human expert played YouTube videos. A model, when pretrained on the offline dataset, is used as a warm-start to

Table 3. Performance of each baseline on our benchmark with $\alpha = 5$ and $\beta = 10$ parameters. Please refer to Page-5 for the definitions of the metrics. Note that Random and E2E backbones don't have any buffer.

	MS	MG	BS	BG	TNMR
Random	**235**	0	0	0	0.41
E2E	236	0	0	0	0.48
ML	257	0.51	1.2	5.6	0.67
CEC	242	**0.13**	1.2	5.7	**0.71**

adapt to unseen tasks. Furthermore, we propose a simple, yet scalable framework for Lifelong Learning that involves a *task-mapper* and a backbone that are both pretrained using an offline dataset. The task-mapper evolves with each new task and learns to identify a new task based on previously seen tasks. The entire system is evaluated on a suite of test tasks. Although our method is simple, it scales well, even with a large number of tasks.

In this paper, we use a representation bottleneck that learns embeddings of the observations solely based on appearance. Even when there are differences in appearances, if the skills learnt are similar, the games can still be played using a single set of parameters. For example, even though *Phoenix* and *AirRaid* have different appearances, they share the same action space and are both *Shoot Up* games. We are currently working on incorporating representations that are skill/dynamic-aware, as opposed to those based solely on appearance, like VAEs. This would not only identify previously learnt tasks, but also help reuse existing policy parameters to play a new game sharing existing skills.

References

[1] Maruan Al-Shedivat et al. "Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=Sk2u1g-0-.

[2] Sayantan Auddy et al. "Continual Learning from Demonstration of Robotic Skills". In: *CoRR* abs/2202.06843 (2022). arXiv: 2202.06843. URL: https://arxiv.org/abs/2202.06843.

[3] Benjamin Eysenbach et al. "Off-Dynamics Reinforcement Learning: Training for Transfer with Domain Classifiers". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=eqBwg3AcIAK.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1126–1135. URL: http://proceedings.mlr.press/v70/finn17a.html.

[5] Chongkai Gao et al. "CRIL: Continual Robot Imitation Learning via Generative and Prediction Model". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*. IEEE, 2021, pp. 6747–5754. DOI: 10.1109/IROS51168.2021.9636069. URL: https://doi.org/10.1109/IROS51168.2021.9636069.

[6] Rituraj Kaushik, Timothée Anne, and Jean-Baptiste Mouret. "Fast Online Adaptation in Robotics through Meta-Learning Embeddings of Simulated Priors". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 5269–5276. DOI: 10.1109/IROS45743.2020.9341462. URL: https://doi.org/10.1109/IROS45743.2020.9341462.

[7] Anthony Kay. "Tesseract: An Open-Source Optical Character Recognition Engine". In: *Linux J.* 2007.159 (2007), p. 2. ISSN: 1075-3583.

[8] Bo Liu, Xuesu Xiao, and Peter Stone. "A Lifelong Learning Approach to Mobile Robot Navigation". In: *IEEE Robotics Autom. Lett.* 6.2 (2021), pp. 1090–1096. DOI: 10.1109/LRA.2021.3056373. URL: https://doi.org/10.1109/LRA.2021.3056373.

[9] Jorge A. Mendez et al. "CompoSuite: A Compositional Reinforcement Learning Benchmark". In: *Conference on Lifelong Learning Agents, CoLLAs 2022, 22-24 August 2022, McGill University, Montréal, Québec, Canada*. Ed. by Sarath Chandar, Razvan Pascanu, and Doina Precup. Vol. 199. Proceedings of Machine Learning Research. PMLR, 2022, pp. 982–1003. URL: https://proceedings.mlr.press/v199/mendez22a.html.

[10] Alex Nichol et al. "Gotta Learn Fast: A New Benchmark for Generalization in RL". In: *CoRR* abs/1804.03720 (2018). arXiv: 1804.03720. URL: http://arxiv.org/abs/1804.03720.

[11] Sam Powers et al. "CORA: Benchmarks, Baselines, and Metrics as a Platform for Continual Reinforcement Learning Agents". In: *Conference on Lifelong Learning Agents, CoLLAs 2022, 22-24 August 2022, McGill University, Montréal, Québec, Canada*. Ed. by Sarath Chandar, Razvan Pascanu, and Doina Precup. Vol. 199. Proceedings of Machine Learning Research. PMLR, 2022, pp. 705–743. URL: https://proceedings.mlr.press/v199/powers22b.html.

[12] Andrei A. Rusu et al. "Policy Distillation". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1511.06295.

[13] Andrei A. Rusu et al. "Progressive Neural Networks". In: *CoRR* abs/1606.04671 (2016). arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671.

[14] Annie Xie and Chelsea Finn. "Lifelong Robotic Reinforcement Learning by Retaining Experiences". In: *Conference on Lifelong Learning Agents, CoLLAs 2022, 22-24 August 2022, McGill University, Montréal, Québec, Canada*. Ed. by Sarath Chandar, Razvan Pascanu, and Doina Precup. Vol. 199. Proceedings of Machine Learning Research. PMLR, 2022, pp. 838–855. URL: https://proceedings.mlr.press/v199/xie22a.html.

[15] Chi Zhang et al. "Few-Shot Incremental Learning With Continually Evolved Classifiers". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 12455–12464. DOI: 10.1109/CVPR46437.2021.01227. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Zhang_Few-Shot_Incremental_Learning_With_Continually_Evolved_Classifiers_CVPR_2021_paper.html.

[16] Wenxuan Zhou et al. "Offline Distillation for Robot Lifelong Learning with Imbalanced Experience". In: *CoRR* abs/2204.05893 (2022). DOI: 10.48550/arXiv.2204.05893. arXiv: 2204.05893. URL: https://doi.org/10.48550/arXiv.2204.05893.