Chameleon: Dual Memory Replay for Online Continual Learning on Edge Devices

Shivam Aggarwal*, Kuluhan Binici*†, Tulika Mitra*
*School of Computing, National University of Singapore
†Institute for Infocomm Research, A*STAR, Singapore
{shivam, kuluhan, tulika}@comp.nus.edu.sg

***Abstract*—Once deployed on edge devices, a deep neural network model should dynamically adapt to newly discovered environments and personalize its utility for each user. The system must be capable of continual learning, i.e., learning new information from a temporal stream of data *in situ* without forgetting previously acquired knowledge. However, the prohibitive intricacies of such a personalized continual learning framework stand at odds with limited compute and storage on edge devices. Existing continual learning methods rely on massive memory storage to preserve the past data while learning from the incoming data stream. We propose *Chameleon*, a hardware-friendly continual learning framework for user-centric training with dual replay buffers. The proposed strategy leverages the hierarchical memory structure available on most edge devices, introducing a short-term replay store in the on-chip memory and a long-term replay store in the off-chip memory to acquire new information while retaining past knowledge. Extensive experiments on two large-scale continual learning benchmarks demonstrate the efficacy of our proposed method, achieving better or comparable accuracy than existing state-of-the-art techniques while reducing the memory footprint by roughly $16\times$. Our method achieves up to $7\times$ speedup and energy efficiency on edge devices such as ZCU102 FPGA, NVIDIA Jetson Nano and Google's EdgeTPU. Our code is available at https://github.com/ecolab-nus/Chameleon.**

I. Introduction

Edge devices continuously interact with novel, unpredictable environments. Real-world applications on edge devices such as image recognition are driven by high variation, both in terms of the number of observed instances per object and the characteristics of these instances. In such circumstances, the deployed model needs to adapt to new domains (such as under different lighting, background, and environmental conditions, also called domain-shift [1]) of already learned objects on the fly, as illustrated in Figure 1. Therefore, it is imperative to continuously learn and accustom to the dynamics of the data distribution on-device and build user-specific models.

Fig. 1. Instances of same object under varying domains in CORe50 [2] dataset.

However, learning from such a time-varying data stream is a non-trivial task. Conventional machine learning (ML) algorithms perform well over fixed datasets with iterative gradient-based methods [3]. Such methods usually loop over the entire dataset multiple times, assuming the data is static and available beforehand. These multi-epoch (joint) training methods incur very high computational and memory costs, making them unsuitable for edge devices.

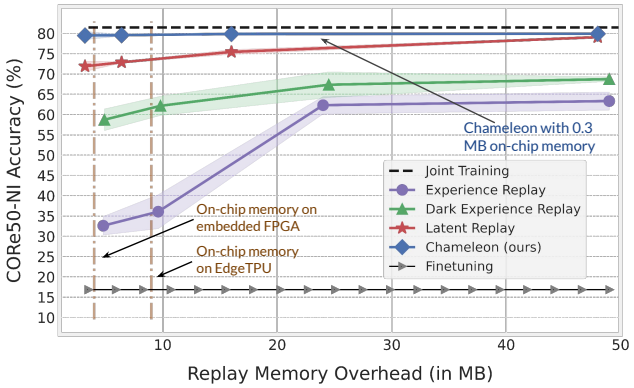

Fig. 2. Accuracy comparison of different continual learning methods with varying memory budgets on CORe50-NI [2] scenario. *(best viewed in color)*

For real-time applications, edge devices should ideally learn from a small number of data instances at a time in a single pass. The tight memory constraints on most edge devices do not allow the deployed model to loop over the entire dataset for multiple epochs like traditional machine learning. Moreover, the model should acquire knowledge from the incoming data stream without gradually losing the past information.

Online **Continual Learning** (CL) (also known as lifelong learning) [4], [5] studies the problem of learning from an online non-independent and identically distributed (non-iid) data stream while dealing with the issue of **catastrophic forgetting** (CF) [6], i.e., forgetting old information in the presence of more recent information. Replay-based CL methods [7] are the most popular and effective at mitigating CF. They store a subset of incoming data samples in an external buffer and periodically train (replay) the network with these previously seen instances.

Figure 2 shows the memory requirements for various methods incrementally learning images from the CORe50 dataset [2] for different backgrounds and lighting conditions. As can be seen, naive single-epoch (finetuning) training becomes impractical in practice with $\sim$15% accuracy. Such a method fails to adapt to varying domains due to catastrophic forgetting.

On the other hand, memory requirements for replay-based methods pose an additional burden on already constrained on-chip storage available on edge devices [8]. For instance,

Google's EdgeTPU [9] has only 8 MB of on-chip SRAM. The state-of-the-art replay-based methods such as Experience Replay (ER) [10] and Dark Experience Replay (DER) [11] exhibit dismal accuracy ($< 65\%$) under limited memory budgets. They perform better with increasing replay buffer sizes, but such large memory requirements become infeasible on edge devices.

Moreover, the memory footprint of the model parameters and activations leaves only a fraction of on-device memory for replay. This, in turn, compels the system to store/retrieve replay samples from the off-chip DRAM. However, the off-chip DRAM accesses [12] are extremely power-hungry and incur huge latency costs. Consequently, methods such as Latent Replay [13] with slightly better accuracy under limited memory budgets incurs up to 7x more latency and energy when deployed on an embedded FPGA accelerator compared to our proposed method, as discussed in Section IV-C. This naturally drives the need for an optimal strategy to reduce the DRAM accesses and effectively manage the replay buffer.

In this work, we propose **Chameleon** for low-latency, energy-efficient continual learning to achieve two major objectives simultaneously: (i) overcoming the daunting issue of *catastrophic forgetting* (ii) within the *limited on-chip memory* available on embedded devices. Figure 2 shows the relative advantage of *Chameleon*, achieving roughly 79.5% accuracy with only 0.3 MB of on-chip memory outperforming other state-of-the-art CL methods on the CORe50 dataset.

We formulate the continual learning problem on edge devices from a user-centric view. Current CL approaches are incognizant of the actual user preferences on edge devices. A general machine learning pipeline trains a global model to be accurate over a diverse range of classes to facilitate a wide range of users. However, any individual user rarely accesses all the classes at the same time, as studied by prior works [14].

We first introduce a dual-memory replay mechanism incorporating a short-term (ST) and a long-term (LT) buffer. The novel replay framework allows us to take advantage of the memory hierarchy on most hardware, comprising a small on-chip memory with low energy cost and a large off-chip memory with very high energy cost. To enable user-aware model personalization, we maintain user-preferred class instances in the on-chip memory while focusing on a holistic snapshot of the entire class distribution in the off-chip storage. Intuitively, to maximize the performance corresponding to user-preferred classes, samples of user interest should be prioritized and replayed more frequently. Hence, such instances should be stored on-chip to be accessed regularly without affecting the latency and energy cost.

For this purpose, we develop two sampling strategies that carefully balance the trade-off between the uncertain and representative instances from the incoming data stream. For the short-term store, we introduce a sampling method based on user affinity and uncertainty to quickly collect the most difficult-to-learn samples, most of which belong to user-preferred classes. Finally, for the long-term store, we propose a novel class-prototype-based sampling scheme leveraging inter-class diversity to retain the most meaningful information.

In summary, the main contributions of the work include:

- We propose *Chameleon*, a novel, accurate, low-latency, user-preference aware continual learning approach for resource-constrained edge devices.
- We propose dual-memory replay mechanism incorporating on-chip short-term and off-chip long-term buffer to overcome catastrophic forgetting with limited memory.
- We propose a novel user-aware sampling strategy and a class-prototype based acquisition scheme to carefully select the most salient samples from the incoming stream.
- Extensive experiments validate the effectiveness of Chameleon in terms of both training accuracy and hardware efficiency on the CORe50 and OpenLORIS datasets.

II. Related Work

There are **three major continual learning techniques**: *regularization* techniques such as online Elastic Weight Consolidation (EWC++) [15] and Learning Without Forgetting (LwF) [16] to constrain weight updates, *architecture-based* techniques for dynamically growing network space, and *replay-based* techniques for storing a representation of previously learned knowledge with the new data, with a majority of them restricted to cloud-centred training.

Replay-based (also known as rehearsal) methods offer the most effective approach for continual learning on edge devices. Early works such as ER [10] interleave new incoming data with old, already seen data stored in a buffer to mitigate catastrophic forgetting during continual learning. Gradient based Sample Selection (GSS) [17] introduces a gradient direction based sample selection strategy to optimally select the most relevant replay instances. Other works such as DER [11] and PRE-DFKD [18] leverage Knowledge Distillation [19], combining replay with the distillation loss and achieving superior performance over various other baselines. More recently, Latent Replay [13] randomly stores feature maps from the middle (latent) layers of the network in place of raw input images. This helps them store more samples in the replay buffer within the same memory budget. However, these works are agnostic of the underlying memory hierarchy of the device, consuming up to 50 MB or more of extra memory for replay. As discussed before, such a large memory requirement cannot be satisfied with limited on-chip storage on edge devices. To address this shortcoming, building on top of Latent Replay, we introduce dual-replay buffers, varying in two ways: *capacity* and *access frequency*. The short-term memory is 10-15 times smaller than the long-term memory and updated more frequently. We allow the seamless acquisition of new information by replacing short-term samples with new ones from the stream at every iteration. Likewise, as the long-term samples are used to retain cumulative information of all classes, they are updated less frequently.

Other recent methods which do not fall under these three techniques include **SLDA** [20], an online non-parametric classifier that can dynamically adapt to varying domain shifts. However, the pseudo-inverse matrix operation in SLDA incurs high computational costs, making it unsuitable for edge devices.

III. Chameleon Framework

A. Problem Formulation

We consider the supervised image classification task with an online non-i.i.d stream of incoming samples. We define a data stream of unknown domains $D = D_1, \ldots, D_d$ over d different domains. Further, $\bigcap D_d = \phi$, i.e., any two input domains D_{d-1} and D_d are different. Let $h_\psi : X \rightarrow Y$ be the deep neural network (DNN) model with parameters ψ. At each time step t, the network h_ψ receives a mini-batch of samples $B_t \in (X_t, Y_t)$, where X_t, Y_t is the input-label pair of the incoming instance. Formally, at any given time stamp t_c, the goal is to correctly classify all the previously encountered samples in addition to the instances belonging to the current mini-batch, B_{t_c} with loss function ℓ:

$$\min_{\psi} \sum_{t=1}^{t_c} \mathbb{E}_{(X_t,Y_t)\sim D_t} \left[\ell(h_\psi(X_t; \psi, Y_t)\right] \tag{1}$$

A naive machine learning algorithm would not be able to remember previously learned information without any access to old data, resulting in catastrophic forgetting. For this purpose, we adopt a replay-based strategy to maintain previously seen samples in a buffer to rehearse them with the incoming data stream. Therefore, given the data stream D, the goal of our continual learning framework is to maintain two replay buffers $\mathcal{M}_s$ (on-chip) and $\mathcal{M}_l$ (off-chip) such that we can maximise the accuracy corresponding to classes of interest within the memory and compute capability of the device.

B. Method Overview

As shown in Figure 3, the method consists of a DNN coupled with two-stage storage: ($\mathcal{M}_s$, $\mathcal{M}_l$) mimicking the on-chip/off-chip memory subsystem on the underlying hardware. The DNN model can be defined as $g(f(\cdot))$ consisting of two nested functions: $f(\,\cdot\,)$ is fixed and extracts higher-dimensional feature representations (latent activations) of the inputs while $g(\,\cdot\,)$ undergoes model training and maps latent activations to their categorical classes. The two functions are parameterized by: θ and ϕ, respectively. Our training algorithm performs only single step forward and backward update for each data sample keeping in mind the resource-constrained environment.

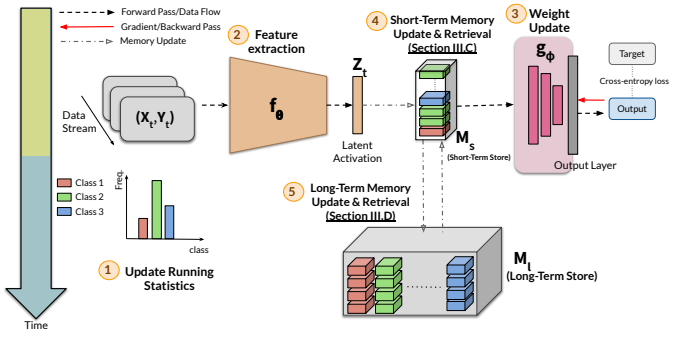

Fig. 3. Schematic illustration of the proposed Chameleon framework for continual model personalization

As illustrated in Figure 3 and algorithm 1, given an incoming batch of data $B_t = (X_t, Y_t)$, where time step t tends to T, our framework processes samples in a *five-step* process. The first step consist of user-preference estimation ①. We estimate user preferences on-device by tracking the frequency of samples n_c *(Line 3)* corresponding to each class. Specifically, we identify the most frequently occurring k classes as *user-preferred classes*. Since our learning scenario is agnostic of the total number of samples to be seen across the stream, the tracking mechanism re-calibrates the top k (= 5) classes of interest captured in a predefined learning window. The re-calibration process ensures that the tracking mechanism can dynamically adapt to the changing user inclinations. The second step *(Line 4)* extracts the intermediate feature map for the incoming batch of data ②.

In the next step, we train the network parameters g_ϕ with the complete short-term memory $\mathcal{M}_s$ for each new data sample ③. Furthermore, we periodically select a subset of elements *(Line 5-6)* from the long-term store $\mathcal{M}_l$ and train the network with the selected data instances. Concurrently, we update the short-term memory *(Line 10-12)* using our proposed user-aware uncertainty-guided sampling method in the following step ④. In the last and final step, we leverage latent feature maps stored in the short-term store to update long-term memory *(Line 14-17)* based on our class-prototype-based sampling scheme ⑤. We explain our memory update and retrieval strategies in more detail next.

Algorithm 1 Chameleon: Online Training Algorithm

Input: Data Stream $\{D_t\}_{t=1}^{T}$, neural network parameters f_θ, g_ϕ, learning rate η, short-term memory $\mathcal{M}_s$, long-term memory $\mathcal{M}_l$, long-term memory access rate h

1: **for** $t \leftarrow 1$ to T **do**
2: **for** batch $B_t = (X_t, Y_t)$ **do**
3: update running stats n_c for each class c
4: $Z_t = f_\theta(X_t)$ ▷ feature extraction
5: sample $\hat{m}_l$ from $\mathcal{M}_l$ every h cycles
6: $\hat{Z}_t \leftarrow Z_t \cup \mathcal{M}_s \cup \hat{m}_l$
7: $\phi \leftarrow \phi - \eta \nabla g_\phi(\hat{Z}_t)$ ▷ weight update
8: randomly sample m_s from $\mathcal{M}_s$
9: select an element b_t from B_t with Eq. 4
10: $\mathcal{M}_s \leftarrow replace(m_s, b_t)$ in $\mathcal{M}_s$
11: **end for**
12: select m_s^c from $\mathcal{M}_s$ with Eq. 6 every h cycles
13: randomly sample m_l^c from $\mathcal{M}_l$
14: $\mathcal{M}_l \leftarrow replace(m_l^c, m_s^c)$ in $\mathcal{M}_l$
15: **end for**

C. Short-Term Memory Update & Retrieval

The objective of the short-term memory is to mitigate catastrophic forgetting and build personalized models. For this purpose, we leverage two major characteristics of the incoming samples: user affinity and uncertainty as illustrated in Figure 4. First, if the samples belong to different classes in the incoming batch, then instances belonging to the user-preferred classes will be given a priority. Secondly, if all or most samples belong to the same set of classes, the ones that are either more difficult to predict or newly discovered by the model will have a higher chance of getting stored in the short-term memory.

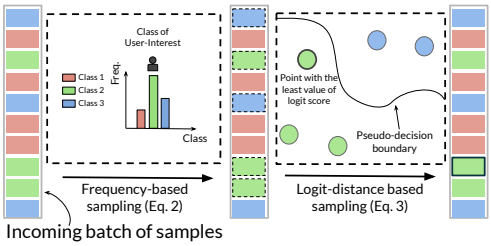

Fig. 4. Overview of user-aware uncertainty sampling

1) Condition 1: In particular, for user-centric sampling, we employ an allocation factor Δ_k based on the running statistics of the data instances as:

$$\Delta_k = \frac{{n_k}^\rho}{(n_k + n_{N-k})^\rho} \tag{2}$$

where N is the total number of encountered classes, k refers to the total number of classes of interest, ρ is a hyper-parameter controlling the allocation factor and n_k is the average running frequency of classes of user-interest estimated during learning window l.

This criteria Δ upscales the ratio of the average number of samples belonging to preferred classes w.r.t total class instances. For $\rho = 0$, all classes are equally favorable, and for $\rho = 1$, classes are assigned in proportion to their running frequencies. In our experiments, ρ is chosen between 0 and 1 such that the data acquisition favors the classes of user-interest and suppresses the interference caused by non-preferred class instances. We update the value of the allocation factor after each learning window ($\sim$1500 images).

2) Condition 2: As we are learning in a streaming fashion, not all samples would equally affect the model performance. If some samples are more challenging to learn or previously unseen by the model, it is essential to store and rehearse the network with such instances to improve the overall robustness. Generally, samples that are closer to the decision-boundary fall under this purview. However, directly computing decision boundaries for each sample is intractable and computationally expensive. Instead, we leverage logit-values, i.e., the final output response of the model as a measure of their pseudo-distance from the decision boundary. To meet this objective, we assign a score U_i for each element x_i in the incoming mini-batch B_t:

$$U_i = \sum_{c=1}^{N} |o(x_i)_c \mathbf{y}_c| \tag{3}$$

where $o(x_i)$ refers to the network logit scores for each element x_i, $\mathbf{y}$ refers to the one-hot encoded output of the label y_i and N refers to the total number of classes. A low logit score U_i indicates that the network is uncertain about these samples and hence must be replayed again. In other words, samples with high U_i^{-1} value should be retained.

We define a discrete non-uniform probability distribution to jointly select an element from the incoming mini-batch B_t based on the above two conditions and randomly replace it with a sample currently stored in the short-term store, where

$$p_i \propto \alpha \cdot \frac{\Delta_i}{\sum_{y_i \in k} \Delta_k + \sum_{y_i \in N-k} (1 - \Delta_k)} + \beta \cdot U_i^{-1} \tag{4}$$

The weighted combination of Δ_k and U_i^{-1} (via hyperparameters α and β) helps us cater to instances belonging to user-interest more effectively and handle their relative complexities in terms of model uncertainty.

D. Long-Term Memory Update & Retrieval

The goal of the long-term memory is to promote prolonged and effective retention and improve overall model generalizability. To maintain a well-rounded representation of the past and present incoming samples, preserving diverse and representative samples from the stream is vital. For this purpose, we introduce a class-balanced sampling technique storing an equal number of samples for each class in the long-term store. Inspired by [21], we propose to select contrastive samples based on their distance from the class prototypes in the latent space representation. We first define class prototypes for each class:

$$P_c = \frac{1}{L_c} \sum_{l=1}^{L_c} Z_l \tag{5}$$

where Z_l corresponds to the latent activation maps corresponding to class c currently stored in the long-term memory and L_c is the total number of such class instances in the long-term memory. Here, the class prototypes roughly approximate the center of mass in the latent space.

We hypothesize that if any two samples belong to the same class (similarity) and their model predictions vary drastically (diversity), then such samples should be more informative about their respective class distribution in the latent space. To select such samples, we formulate a computationally inexpensive measure using Kullback-Leibler divergence (KL-divergence) criteria to approximate the distance between a datapoint belonging to that class and the respective class prototype.

For each sample st_j belonging to class c already stored in the short-term store, we calculate:

$$S_j = \tanh(KL(p(y|st_j) \parallel p(y|P_c))) \tag{6}$$

where $p(y|st_j))$ refers to the softmax probabilities for each element st_j and $p(y|P_c))$ indicates the softmax probabilities for class-prototypes P_c. A high S_j score indicates that the data point disagrees in softmax probabilities against other points of the same class already present in the long-term store. Thus, we greedily select the sample with the *maximum* value of S_j score and randomly replace it with other samples of the same class already present in the long-term memory.

IV. Evaluation

A. Experimental Methodology

Datasets. To evaluate our experiments in online continual streaming learning setup, we rely on two robotics-based large-scale CL benchmarks: **CORe50** [2] and **OpenLORIS-Object** [22] datasets. The CORe50 dataset comprises 164,866 temporally-correlated video frames and a total of 50 classes, collected under 11 different domains, where each domain is characterized by distinct backgrounds and lighting, as shown in Figure 1. The OpenLORIS-Object dataset is based on a sequence of 12 different domains such as illumination, clutter, and occlusion. Each domain consists of $\sim$14k training samples

and ∼2k test images of the same 69 classes. We evaluate the two datasets in the *Domain Incremental Learning (Domain-IL)* setting, where the task is to identify objects under different domains from the training set incrementally.

Base Models & Hyperparameters. Considering the limited computational resources available on the device, we pick a hardware-friendly CNN, **MobileNetV1** [23], pre-trained on the ImageNet dataset for all our experiments. We use a small batch size of 10, and a learning rate of 0.001 with the SGD optimizer [3]. Each sample passes through the model only once. We update short-term memory for every incoming batch while long-term memory for every ten batches to maintain the on-chip/off-chip memory access trade-off. The framework sweeps through the complete short-term memory for each new sample while following an iterative mini-batch concatenation scheme for the long-term store. We experiment with the last few layers as the latent layer to keep the training overhead minimal. Based on the overall model accuracy, we choose layer 21 (out of 27 layers) of the MobileNetV1 as the latent layer.

Metrics. Following [11], we report final model accuracy, Acc_{all} averaged over all classes at the end of model training over all domains. Acc_{all} helps us quantify the generalizability of the model. Additionally, we quantify the memory overhead for each continual learning method. For the same number of replay samples, different methods correspond to different memory overhead. Further, we report latency and energy consumption on various edge devices to validate the efficacy of these methods on resource-constrained devices.

Baselines. For **lower-bound** on the average model accuracy, we finetune the training part of the network with a single epoch over the entire dataset without any replay buffer. For **upper-bound** on the average model accuracy, the model is jointly trained in a traditional machine learning paradigm for 4 epochs. We compare Chameleon against several state-of-the-art CL methods. We include two *regularization-based* methods: **EWC++** [15] and **LwF** [16]. We also compare our method against **SLDA** [20], and adapt it for our object-level classification task. Finally, we compare our method against four single *replay* buffer based methods: **GSS** [17], **ER** [10], **DER** [11], and **Latent Replay** [13] with varying replay buffer sizes: 100, 200, 500, and 1500 samples to evaluate the memory-efficiency and scalability of different methods. For our method, we keep the size of short-term memory ($\mathcal{M}_s$) constant at 10 samples while varying the size of long-term memory ($\mathcal{M}_l$).

B. Accuracy versus Memory Trade-off

Table I compares Chameleon with state-of-the-art CL methods across varying replay buffer sizes averaged across ten runs. Chameleon achieves the best accuracy on both OpenLORIS and CORe50 datasets (closer to the upper bound) with only 0.3 MB of on-chip memory, demonstrating the effectiveness of our memory update and retrieval strategies. Specifically, the short-term store dynamically switches between different classes of interest after each learning window. At the same time, the long-term store maintains the most salient class instances from previous domains uniformly, ensuring better results over other baseline methods. With the increasing size of the long-term store, we observe marginal improvement in the model accuracy since the performance is already at par with the upper bound even at small buffer size.

TABLE I
COMPARISON OF CHAMELEON AND OTHER BASELINES ON THE OPENLORIS AND CORE50 DATASET. WE REPORT THE MEAN AND STANDARD DEVIATION ACROSS TEN RUNS.

Method	Replay Buffer Size (# of Samples)	Memory Overhead (MB)	OpenLORIS $\text{Acc}_{all}(\%)$	CORe50 $\text{Acc}_{all}(\%)$
JOINT	–	–	97.14 ± 0.24	81.48 ± 0.86
Finetuning			65.97 ± 10.18	16.86 ± 1.61
EWC++ [15]		13.0	61.89 ± 4.27	23.22 ± 2.19
LWF [16]	–	12.5	72.57 ± 1.21	27.91 ± 3.67
SLDA [20]		1.2	90.17 ± 0.44	77.20 ± 0.23
GSS [17]	100	48.8	91.20 ± 0.32	43.51 ± 3.85
	200	53.6	92.00 ± 1.04	47.47 ± 1.42
	500	68.0	91.99 ± 0.63	48.57 ± 1.96
	1500	204.0	95.50 ± 0.38	53.19 ± 3.45
ER [10]	100	4.8	90.45 ± 0.58	32.61 ± 2.25
	200	9.6	90.68 ± 0.84	36.07 ± 4.15
	500	24.0	93.72 ± 0.19	62.31 ± 2.12
	1500	72.0	95.50 ± 0.38	63.33 ± 2.20
DER [11]	100	4.9	90.33 ± 0.35	58.72 ± 2.64
	200	9.8	92.12 ± 0.31	62.15 ± 2.34
	500	24.5	94.37 ± 0.41	67.35 ± 3.14
	1500	73.5	95.50 ± 0.38	68.73 ± 0.53
Latent Replay [13]	100	3.2	90.57 ± 0.46	71.89 ± 1.23
	200	6.4	92.32 ± 0.19	72.87 ± 0.14
	500	16.0	94.89 ± 0.54	75.43 ± 0.56
	1500	48.0	95.50 ± 0.38	79.07 ± 0.16
Chameleon (ours)	$\mathcal{M}_s$=10, $\mathcal{M}_l$=100	$\mathcal{M}_s$=0.3, $\mathcal{M}_l$=3.2	$\mathbf{96.10 \pm 0.12}$	$\mathbf{79.48 \pm 0.99}$
($\mathcal{M}_s$ = Short-term memory	$\mathcal{M}_s$=10, $\mathcal{M}_l$=200	$\mathcal{M}_s$=0.3, $\mathcal{M}_l$=6.4	$\mathbf{96.43 \pm 0.47}$	$\mathbf{79.56 \pm 0.17}$
$\mathcal{M}_l$ = Long-term memory)	$\mathcal{M}_s$=10, $\mathcal{M}_l$=500	$\mathcal{M}_s$=0.3, $\mathcal{M}_l$=16	$\mathbf{96.70 \pm 0.62}$	$\mathbf{79.86 \pm 0.31}$
	$\mathcal{M}_s$=10, $\mathcal{M}_l$=1500	$\mathcal{M}_s$=0.3, $\mathcal{M}_l$=48.0	$\mathbf{97.10 \pm 0.24}$	$\mathbf{79.92 \pm 0.12}$

TABLE II
PERFORMANCE COMPARISON OF CHAMELEON ON EDGE DEVICES.

Method	Memory Overhead (MB)	Acc_{all} (%)	Jetson Nano		ZC102 FPGA		EdgeTPU
			Latency (ms)	Energy (J)	Latency (ms)	Energy (J)	Latency (ms)
Latent Replay	48.0	79.07 ± 0.16	115	1.14	2788	8.62	-
SLDA	1.2	77.20 ± 0.23	69	0.68	-	-	554
Chameleon	$\mathcal{M}_s$=0.3, $\mathcal{M}_l$=3.2	79.48 ± 0.99	**33**	**0.31**	**413**	**1.22**	**47**

Overall, all methods perform better on the OpenLORIS dataset than the CORe50 dataset, primarily due to more training samples and smoother transitions between consecutive domains in the OpenLORIS dataset. On both datasets, the dismal accuracy of EWC++ and LwF is primarily attributed to gradient explosion on previous domains, leading to catastrophic forgetting. Interestingly, SLDA achieves comparable average model accuracy with limited memory utilization.

For replay-based methods, GSS seems less effective due to the sub-optimal selection strategy on previous domains, despite having a much larger memory footprint. GSS stores gradient direction vectors for each element in the buffer, resulting in up to 10x more memory overhead for the same number of replay samples. Methods like ER and DER fail to perform under tight memory constraints ($\leq$ 200 replay samples), as also shown in Figure 1. Both methods store original input images and output responses for each element in the buffer, incurring huge space overhead. In comparison, both Latent Replay and Chameleon store only intermediate activations in the replay buffer, leading to substantial memory savings. However, Chameleon demonstrates a much better trade-off, outperforming Latent Replay by up to ∼7-8% in accuracy, especially across smaller memory budgets. Note that Chameleon also has significantly lower processing latency and energy needs in comparison to both Latent Replay and SLDA, as we see in Section IV-C, showing a clear advantage for online, on-device continual learning.

C. Latency and Energy on Edge devices

As Latent Replay and SLDA are the closest to Chameleon in terms of accuracy, we further investigate these three methods for latency and energy on edge devices.

Jetson Nano GPU Results. Table II shows latency and energy per image for Chameleon, SLDA, and Latent Replay. We observe up to $2.1\times$ and $3.5\times$ speedup for Chameleon compared to SLDA and Latent Replay, respectively. Similar to Chameleon, SLDA also processes the input in a single pass; however, it still incurs huge latency and energy costs as it requires a pseudo-matrix inverse operation for each image. Note that while we could not take advantage of the on-chip L2 Cache on the GPU, our proposed framework still outperforms the other two baselines, establishing the effectiveness of our sample selection strategies.

EdgeTPU Results. We evaluate Chameleon and SLDA on custom TPU-like edge accelerator. We utilize uSystolic-Sim [24], a cycle-accurate simulator to deploy our proposed algorithm with (64,64) PE array, 8 MB of on-chip memory, and 400 MHz clock frequency. We leverage Block Floating Point (BFP) datatype to compute the forward and backward pass. SLDA requires a matrix pseudo-inverse operation for updating the covariance matrix in the final output computation. As shown in Table II, our method achieves roughly $11.7\times$ speedup over SLDA in terms of latency per input image. This is due to $O(N^3)$ complexity of the matrix inverse operation.

FPGA Results. We implement a CNN training accelerator from scratch using Xilinx Vitis software on the Zynq UltraScale+ MPSoC ZCU102 evaluation kit. The exported RTL is synthesized and implemented using Vivado 2021.2 and reaches 150 MHz clock frequency. Additionally, 16-bit floating-point numbers are used for the accelerator to estimate the compute and communication cost of continual learning correctly. Table III shows FPGA resource usage of Chameleon. As SLDA requires complex processing, it is challenging to implement on FPGA. Therefore, we implement only Chameleon and Latent Replay on FPGA to comprehensively analyze the relative advantage of our hierarchical replay buffers in comparison to Latent Replay with a single replay buffer.

TABLE III
RESOURCE UTILIZATION FOR CHAMELEON

	DSP	BRAM	LUTs
Available	2520	656	233707
Utilized	1164	632	169428
Percentage (%)	46.19	96.34	72.50

Table II shows roughly $6.75\times$ latency and $7\times$ energy efficiency of Chameleon against Latent Replay when trained with a batch size of one and ten replay elements per incoming input. Latent Replay maintains a unified replay buffer too large to be stored in the on-chip memory. In contrast, we effectively leverage on-chip memory due to our proposed hierarchical buffer and sampling policies, leading to superior performance. The cost of compute and data movement for weights remains the same for both methods. However, Latent Replay spends 44% of overall latency in data movement of latent activations (load and store ten latent activations for each new sample) from the off-chip buffer. Hence, we achieve significant improvement in latency and energy due to the smaller on-chip memory footprint and reduced off-chip memory accesses.

V. CONCLUSION

We propose *Chameleon*, a resource-efficient personalized continual learning framework. Chameleon maintains decoupled replay buffers - short-term and long-term stores complementary to the on-chip and off-chip memory interface. The framework introduces two novel sampling strategies taking into account user preferences and uncertainty-diversity-induced measures to rehearse previously seen samples from memory at different rates. Our evaluations show Chameleon accomplishes over $7-8\%$ better accuracy with $16\times$ less memory, $6.75\times$ less latency, and $7\times$ less energy compared to the state-of-the-art methods.

VI. ACKNOWLEDGMENT

This work is partially supported by the National Research Foundation, Singapore under its Competitive Research Programme Award NRF-CRP23-2019-0003 and Singapore Ministry of Education Academic Research Fund T1 251RES1905. We thank Dhananjaya Wijerathne (NUS) and Zhaoying Li (NUS) for their valuable comments.

REFERENCES

[1] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *J. Stat. Plan. Inference*, 2000.
[2] V. Lomonaco *et al.*, "Core50: a new dataset and benchmark for continuous object recognition," in *CoRL*, 2017.
[3] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv*, 2016.
[4] Z. Mai *et al.*, "Online continual learning in image classification: An empirical survey," *Neurocomputing*, 2022.
[5] G. I. Parisi *et al.*, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019.
[6] I. J. Goodfellow *et al.*, "An empirical investigation of catastrophic forgeting in gradient based neural networks," in *ICLR*, 2014.
[7] T. L. Hayes *et al.*, "Replay in Deep Learning: Current Approaches and Missing Biological Elements," *Neural Computation*, 2021.
[8] Y. Zhang *et al.*, "Power-performance characterization of tinyml systems," in *ICCD*, 2022.
[9] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.
[10] A. Chaudhry *et al.*, "Continual learning with tiny episodic memories," *ArXiv*, 2019.
[11] P. Buzzega *et al.*, "Dark experience for general continual learning: a strong, simple baseline," in *NeurIPS*, 2020.
[12] M. Horowitz, "Energy table for 45nm process," in *Stanford VLSI wiki*.
[13] L. Pellegrini *et al.*, "Latent replay for real-time continual learning," in *IROS*, 2020.
[14] T. Shibuya *et al.*, "Captor: A class adaptive filter pruning framework for convolutional neural networks in mobile applications," *ASP-DAC*, 2019.
[15] A. Chaudhry *et al.*, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," *Lect. Notes Comput. Sci.*, 2018.
[16] Z. Li *et al.*, "Learning without forgetting," *IEEE TPAMI*, 2018.
[17] R. Aljundi *et al.*, "Gradient based sample selection for online continual learning," in *NeurIPS*, 2019.
[18] K. Binici *et al.*, "Robust and resource-efficient data-free knowledge distillation by generative pseudo replay," *AAAI*, 2022.
[19] G. Hinton *et al.*, "Distilling the knowledge in a neural network," 2015.
[20] T. L. Hayes *et al.*, "Lifelong machine learning with deep streaming linear discriminant analysis," *CVPRW*, 2020.
[21] K. Margatina *et al.*, "Active learning by acquiring contrastive examples," in *EMNLP*, 2021.
[22] Q. She *et al.*, "OpenLORIS-Object: A robotic vision dataset and benchmark for lifelong deep learning," in *ICRA*, 2020.
[23] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, 2017.
[24] D. Wu *et al.*, "uSystolic: Byte-Crawling Unary Systolic Array," in *HPCA*, 2022.