Memory-Latency-Accuracy Trade-Offs for Continual Learning on a RISC-V Extreme-Edge Node

Memory-Latency-Accuracy Trade-offs for Continual Learning on a RISC-V Extreme-Edge Node

Leonardo Ravaglia[†], Manuele Rusci[†], Alessandro Capotondi[‡], Francesco Conti[†],
Lorenzo Pellegrini[*], Vincenzo Lomonaco[*], Davide Maltoni[*], Luca Benini[†§]
[*]DISI, University of Bologna, Italy [†]DEI, University of Bologna, Italy
[‡]FIM, University of Modena and Reggio Emilia, Italy [§]IIS, ETH Zurich, Switzerland

Abstract*—AI-powered edge devices currently lack the ability to adapt their embedded inference models to the ever-changing environment. To tackle this issue, Continual Learning (CL) strategies aim at incrementally improving the decision capabilities based on newly acquired data. In this work, after quantifying memory and computational requirements of CL algorithms, we define a novel HW/SW *extreme-edge* platform featuring a low power RISC-V octa-core cluster tailored for on-demand incremental learning over locally sensed data. The presented multi-core HW/SW architecture achieves a peak performance of 2.21 and 1.70 MAC/cycle, respectively, when running forward and backward steps of the gradient descent. We report the trade-off between memory footprint, latency, and accuracy for learning a new class with *Latent Replay* CL when targeting an image classification task on the CORe50 dataset. For a CL setting that retrains all the layers, taking 5h to learn a new class and achieving up to 77.3% of precision, a more efficient solution retrains only part of the network, reaching an accuracy of 72.5% with a memory requirement of 300 MB and a computation latency of 1.5 hours. On the other side, retraining only the last layer results in the fastest (867 ms) and less memory hungry (20 MB) solution but scoring 58% on the CORe50 dataset. Thanks to the parallelism of the low-power cluster engine, our HW/SW platform results 25× faster than typical MCU device, on which CL is still impractical, and demonstrates an 11× gain in terms of energy consumption with respect to mobile-class solutions.

***Index Terms*—continual learning, extreme edge, deep learning, parallel programming, online learning, federated learning.**

I. Introduction

Novel sensors and smart devices are equipped with *ultra-low-power* digital processing platforms that process raw data locally and extract high-level information by applying intelligent algorithms, such as Deep Neural Networks (DNNs). While the DNN inference capability has been already demonstrated on extreme-edge devices [1]–[3], the training of DNN models still relies on GPU-based machines. Once trained, the inference models are deployed on edge platforms tailored for prediction-only tasks. However, these inference models cannot adapt to the present environment, which may differ significantly from the training data statistics.

A way out of this rigid *train-on-cloud – deploy-at-edge* model has been proposed in the form of Continual Learning (CL) algorithms. CL algorithms aim at adapting a network model to a new class of sensor stimuli. This process is extremely challenging because of the *catastrophic forgetting* [4]: learning new tasks or instances by fine-tuning only on new data leads to degraded recognition capabilities over initial data. Safeguarding previous knowledge is, therefore, a significant concern. Regularization techniques and rehearsal-free methodologies, i.e., ones not reusing the initial training data, can tackle this problem; alternatively, rehearsal-based techniques safeguard previous knowledge with incremental training over a combination of previous and new data. Among these latter techniques, Pellegrini et al. [5] presented a novel CL approach to effectively learn new object classes based on new samples and compressed old data, namely the Latent Replays (LRs).

In this paper, we present a Hardware/Software platform design to run for the first time Continual Learning (CL) algorithms at the *extreme-edge* on a MicroController Unit (MCU)-class architecture. In contrast with current *ultra-low-power* MCUs that are mostly tailored for DNN inference, we propose a system architecture that can run incremental learning tasks on-demand by turning on a RISC-V-based 8-core cluster subsystem.

The contributions of this paper are:

- We evaluate the computational and memory requirements of a CL algorithm with latent-replays on the CORe50 NICv2-391 benchmark [6] with a MobileNetV1 model.
- We define a HW/SW architecture for CL at the *extreme-edge* based on PULP [7], [8], as well as a tensor tiling strategy necessary to fit within the limited memory.
- We benchmark forward and backward steps for CL on PULP, and evaluate their performance and efficiency together with the energy-accuracy trade-offs of a MobileNetV1 network that learns over the CORe50 dataset.

The presented HW/SW design delivers an average performance of 1.84 MAC/cycle during the learning task, thanks to an almost ideal speed-up of 7.79× gained by the parallelization over 8-cores with respect to a single-core implementation. Employing our platform, we demonstrate Continual Learning over the CORe50 dataset by coupling the processing engine with external memories for low-bandwidth operations. The learning of a new class can be achieved in 1.5 h by retraining only part of the network parameters with a memory need of 70 MB for RW operations and 200 MB to store old Latent Replay data permanently. This CL setting leads to an accuracy of 72.5%, which is only 5% lower than retraining all the layers, but on it is 3.2× faster. Moreover, the proposed solution demonstrates to be 25× faster than typical low-power MCUs, and, given the estimated power cost of ≈70 mW, 11× more energy-efficient when compared to mobile-class solutions.

II. Related Works

A. Continual Learning

The starting point of CL is *Transfer Learning* [9] that applies previously learned knowledge, i.e., the feature-extraction capability, on a different - but related - task. *Incremental Learning* [10] algorithms, instead, learn on new data to extend and enhance performances of a pre-trained model. On the other hand, incrementally training deep networks leads to catastrophic forgetting, in particular when learning over a stream of non-stationary data [11], [12]. To tackle the forgetting problem [4], a model can be fully retrained on a newly enriched dataset, including either old and new data. However, this latter approach results impractical on constrained embedded platforms, due to the severe limitations in terms of memory capability, that prevents the storage of a full dataset.

To face the forgetting challenge and keep the computational and memory load contained, novel CL algorithms learn over new data [6], [13]–[15]. Among them, EWC [13] employs regularization terms inside the loss function to avoid catastrophic forgetting of previously learned samples. iCaRL [15] is a class-incremental learner that makes use of a *nearest-exemplar* algorithm for classification, so to exploit the similarity between classifiers to efficiently transfer previously learned knowledge. These algorithms differentiate on the update rules of the learning protocols and require relatively large batches for learning new classes but not yet solve the forgetting issues [16]. The AR1 algorithm [6] partially solves the issues by retraining only a subset of the network layers based on the new data, to deal with memory and computational limitations of low-end devices, and making use of Batch Re-Normalization in place of Batch Normalization layers. Unfortunately, this approach may lead to a steep accuracy reduction (down to -20% with respect to cumulative training on CORe50 NICv2391 [6]). To deal with such degraded performance, Rehearsal-based techniques [17], [18] keep in memory some representative patterns from past experiences. During the incremental learning phase, the new frames are then interleaved with the past ones. In this context, a recent technique makes use of *Latent Replays (LRs)* [5], which are the activation feature vectors obtained by feeding the network with a subset of the training data. LRs are stored in memory, demanding a lower memory footprint w.r.t. RGB inputs. This approach achieves state-of-the-art accuracy on the CORe50 NICv2-391 CL benchmark, recovering up to 15% with respect to a Rehearsal-Free approach [5]. Due to the promising performance, we benchmark this latter CL algorithm on the proposed HW/SW extreme-edge platform.

B. Training at the Extreme Edge

Edge and extreme-edge devices have constrained memory and computing resources that make the on-device training challenging. Federated Learning addresses this limitation by distributing the training tasks on multiple devices [19], [20]. The work presented by Xu et al. [21] studied the trade-offs between computation latency for a training step and devices frequency over six mobile-class processors while aggregating the results on the server side. Some devices feature a higher frequency and power consumption with respect to others (up to 64% higher) to balance the latency time among the different devices. Yet, the most efficient mobile-class platforms still present a power consumption in the range of few Watts. In contrast to them, we present an HW platform that allows training with a power budget below 100 mW.

From a platform-perspective, processing devices tailored for machine learning tasks feature a power consumption varying between a few mW up to hundreds of Watts [22]. We focus on flexible SW-programmable and low power devices distinguishing between 1-10 W (*edge* device) or <1 W (*extreme-edge* device) respectively. Among the edge devices, we report the TPUEdge [23], which is developed mainly for embedded inference applications, and the Qualcomm Snapdragon 845 that features a quad-core CPU, 4 Kryo 280 Gold coupled with an embedded GPU, and a dedicated DSP, namely the *AI engine*. This latter device features a power consumption up to 4.5 W, which is above the requirement for an extreme-edge device that we target in this paper.

Moving to the *ultra-low-power* spectrum, the *extreme-edge* single-core MCUs, e.g., the STM32 MCU series [24], feature low power consumption and compliant with the requirements of battery-powered sensors. However, this comes at the cost of the modest computational capacity, e.g. an STM32L4 device includes a single CPU running up to 48 MHz. To address this limitation, MrWolf [8] features an MCU-based architecture accelerated by a multi-core cluster. In particular, MrWolf has a power budget as little as 150 mW when running the 8-core cluster at 450 MHz and also supports floating-point (FP32) arithmetic. Relying on the design principle of this latter systems, we present a HW/SW design to enable CL at the *extreme-edge*.

III. CL with Latent Replays

CL aims at fine-tuning deep networks based on new data samples, without retraining on the entire dataset. Newly collected data used by CL can include new instances of previously seen classes, or instances of new classes, or both. This section sketches *i)* the computational model of the CL algorithm presented by Pellegrini et al. [5] when targeting an extreme-edge platform and *ii)* compute its memory requirements.

Computational Model. To bring CL with Latent Replay on a smart sensing platform, we model the incremental learning task to operate on a set of new data coming from a sensor, which is interfaced with an embedded digital processing engine. In the following, we focus on image classification as a representative problem for CL.

Fig. 1 illustrates the learning process that takes place over a set of data samples, including both N_I new images and N_{LR} old LR vectors, which are the feature maps of the Latent Replay (LR-th) layer, generated during previous trainings.

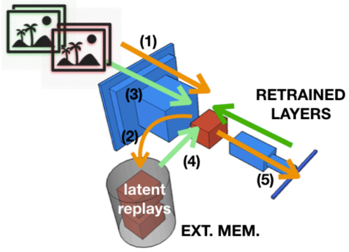

Fig. 1. **Latent Replay computational model.** New images are elaborated up to the LR layer (1) and stored in the external memory (2). Afterwards, a mix of new images (3) and LRs (4) are used to retrain the final layers (5).

At runtime, a set of N_I new class data are sampled and converted into LR vectors. For this purpose, the network is fed with the new data up to the Latent Replay layer. The new activations are then mixed with the old LR tensors, and afterwards, the learning algorithm updates the parameters of the remaining layers. Hence, the dataset for a training step of CL is composed of old and new LR vectors. Typically, $N_I/N_{LR} = 1/5$ [5]. The batch gradient descent, which consists of forward and backward passes from the Latent Replay layer, is iterated for several epochs to converge to the best solution. Furthermore, to face the catastrophic forgetting issue, we employ the AR1 training algorithm [6]. Within the parameter update rule, AR1 applies a per-parameter scaling factor on the computed gradient, expressed by an approximation of the Fisher matrix. The Fisher matrix is defined as the integral of the loss function over the weights. The intuition behind this term is to keep the most meaningful parameters unchanged.

Memory Requirements. Consider a network model featuring N stacked layers. Besides the data for the learning process, a total of $N_w = \sum_{i=0}^{N-1} N_w(i)$ network parameters have to be stored in memory, where $N_w(i)$ represents the number of parameters at the i-th layer. Moreover, we account:

- N_a is the total amount of intermediate features computed during the forward pass, which have to be preserved in memory for gradient computation;
- N_g is the number of gradient's components of the network's parameters to be retrained.
- N_{Fi} is the number of parameters in the Fisher matrix, which is equal to the number of parameters to update.
- N_{fw} is the required memory footprint for temporarily storing activation feature maps during the forward pass. Its size is typically negligible with respect to other terms.

IV. HW/SW PLATFORM FOR CL

In this section, we present our HW/SW platform design tailored for CL workloads. We evaluate the accuracy-latency-memory trade-offs based on the described system architecture.

A. HW Platform for edge learning

The proposed CL platform is based on the design principles of the PULP platform [25], which combines parallel programming for high-performance and ultra-low-power features. The system architecture, which is depicted in Fig. 2, is based on an MCU platform accelerated by a multi-core cluster of RISC-V cores. The MCU side features a single RISC-V core, namely the Fabric-Controller (FC), and a large set of peripherals. Besides the FC core, which is equipped with a private L1 data memory, the MCU-side of the platform includes a relatively large (256-1024 KB) on-chip L2 memory. The cluster features 8 processing elements (PE), each one including a RISC-V CPU equipped with a private Floating-Point Unit (FPU). To avoid memory coherency overheads and increasing area efficiency, the cores share a 64-256 KB L1 data scratchpad memory. Additionally, a multi-channel DMA engine handles data transfers between in-cluster and off-cluster memories.

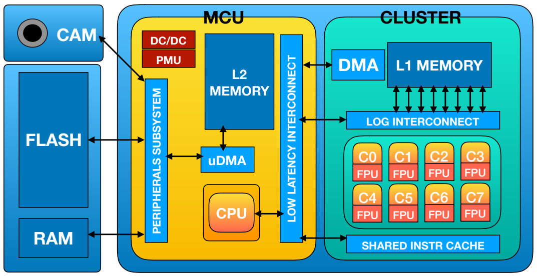

Fig. 2. **The proposed platform architecture for *extreme-edge* CL.** The architecture consists of a cluster of 8 RISC-V tightly coupled cores featuring private FPUs and two shared L1 and L2 scratchpad memories.

The L2 memory is used as a temporary buffer for the peripheral data: a μDMA unit autonomously handles the data transfers between the L2 memory and the external peripherals. Among them, the system can be interfaced with a large L3 off-chip memory, through a parallel interface, and an external FLASH for data storage through Quad-SPI, e.g., a SD FLASH memory. Both external memories feature low power consumption when in idle state.

To efficiently implement power saving mechanisms, the FC core can switch-on the cluster on-demand at runtime, by controlling the internal cluster DC-DC regulator. Once powered-on, the FC core dispatches tasks on the 8-cores cluster, relying both on data- or task- parallelism for efficient and fast computation. On the other side, the system pays an energy overhead when activates the cluster computation on-demand. Likewise, external memories can be power-gated if not running the learning tasks.

B. SW stack for Continual Learning

In this section, we detail the mapping of the CL data flows on the architecture defined in Fig. 2. Because of the high memory requirements and their constant nature, Latent Replays (LRs) are kept in external FLASH memory. During the learning phase, LRs are loaded into the L2 memory to feed the network model. On the other side, an external L3 DRAM memory is used to store network parameters, gradients, and intermediate activations values. Data transfers from L3 RAM to L2 occurs in the background of the computation thanks to the pipelined μDMA operation.

CL updates the model's parameters based on the back-propagation algorithm, which consists of the basic operations depicted in Fig. 3. During the *Forward pass* of a network,

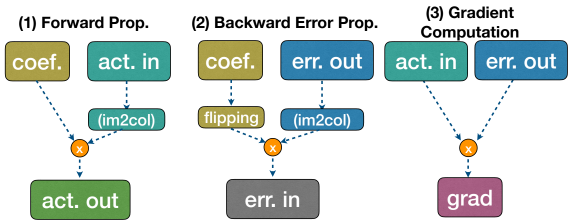

Fig. 3. **CL Computational Phases Graphs.** (1) Forward step; (2) Error back-propagation; (3) Gradient Descent computation.

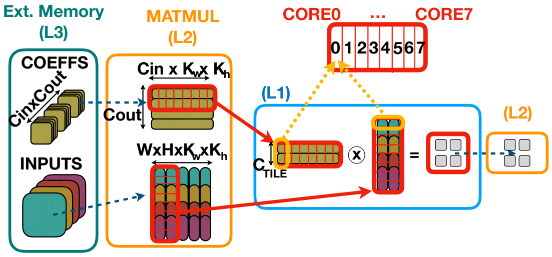

Fig. 4. **GEMM dataflow and parallelization scheme.** GEMMs use tiling to benefit from L2 and L1 data locality, and on GEMMs we use data-parallelism (SPMD) to distribute the work over the 8 cores of the cluster.

the computational layers apply kernel convolutions over the activation values (*act_in*). Each convolution is reshaped as a 32-bit Floating-Point (FP32) generic matrix multiplication (GEMM), performed on the Cluster, by means of an *im2col* transformation applied on the input activation tensor (*act_in*); the *im2col* tensor is stored in a shared L2 buffer. The results of the forward kernels, i.e., the intermediate activation tensors, are then stored back in L3 memory for the update step. During the *Backward Pass*, which runs up to the *LR*-th layer, the activations gradient (*err_out*) is propagated to the next layer, after applying a convolution (GEMM) operation with the flipped (*coeff*) vector (graph (2) in Fig. 3). Likewise, the computation of the parameters gradient (*grad*) is computed using a GEMM between *act_in*, computed during the forward pass, and the *err_out* tensors.

Fig. 4 represents the memory management and the parallelization scheme to implement the computational graph of Fig. 3 in the target platform. Data (i.e., gradients, coefficients, activation values, or input data), which resides on the L3 memory, is loaded into the on-chip L2 and L1 memories for layer-wise processing. Given the on-chip memory limitations, for some layers, tensor operands have to be sliced, and computation is performed on sub-tensors. This tensor slicing is called *tiling* and previous works already presented a lightweight strategy, hence implementing a software-cache mechanism for DNN deployment [26]. In Fig. 4 we visualize the tiling of the coefficients along the output-feature dimension C_{out}: a subset of C_{TILE} filters, hence a total of $C_{TILE} \times C_{in} \times K_w \times K_h$ parameters, is transferred to the cluster L1 memory for the computation. Inside the cluster, the FP32 GEMM is *parallelized* over 8 cores, both for forward and backward passes, according to a data-parallelism paradigm. Such scheme holds for the types of computations reported in Fig.3. The operand *INPUTS* of Fig. 4 refers either to the *act_in* or the *err_out* tensors. During forward propagation and gradient computation, the parallelization operates over the input-feature dimension C_{in}, whereas during backward error propagation the workload is parallelized over the C_{out} dimension of the activations tensor.

V. Experimental Results

In this section, we evaluate our HW/SW design over a *Latent Replay* CL scenario, and we compare our solution against other proposed CL algorithm implementations.

A. Experimental Setup

CL Task and Dataset. To benchmark our HW/SW design, we consider a CL task consisting of learning a new object class from the CORe50 dataset [6]. This dataset includes 160k 128x128 training images of 50 objects. To make the benchmark close to a real application, new objects are discovered sporadically over time. Thus, a three-way protocol was introduced (denoted as NICv2), where the first insertion of the samples of a new class is balanced over the training batches (see Fig. 3 in [6]). In particular, in NICv2-391, each of the 390 incremental batches includes only one training session (300 images) of a single class.

We rely on the experimental settings of a previous study [5], where the authors make use of a MobileNetV1 model with input resolution of 128x128 and width multiplier 1. In this section, we indicate the individual layers of the networks with the naming convention used by [5] and also reported in Fig. 5. The model is initially trained to distinguish only 10 of the 50 classes of the dataset. Each incremental learning step applies the gradient descent algorithm over a single batch composed of 1500 LRs vectors (30 LRs for every class) and 300 new images. This learning step iterates for 8 epochs.

Evaluation Metrics. The computation latency is measured through a cycle-accurate simulator of the proposed architecture. The base version of the simulator, which we extended for this work, is validated against RTL for the open-source PULP platform and MrWolf, that is our reference SoC. The selected running frequency is 150 MHz. The reported accuracy represents the overall precision reached by the network after complete learning on the CORe50 NICv2-391 benchmark.

B. Memory Evaluation

Fig. 6 shows the memory footprint needed (A) to permanently store the 1500 LRs vectors, i.e. the ROM memory requirement, and (B) to store the new data samples, the network parameters and gradients, the intermediate features maps for the backward pass and the approximated Fisher matrix components. All these values result in a memory footprint of few MBs and change over time hence they are stored on the RAM. Note that the memory requirements vary depending on on the chosen *LR* layer: retraining only the last layer (indicated as mid_fc7) requires the storage of smaller LR vectors and less gradients and activation features maps than selecting any other middle layer as the LR layer (e.g. $conv5\_4/dw$). This also explains the higher memory demands

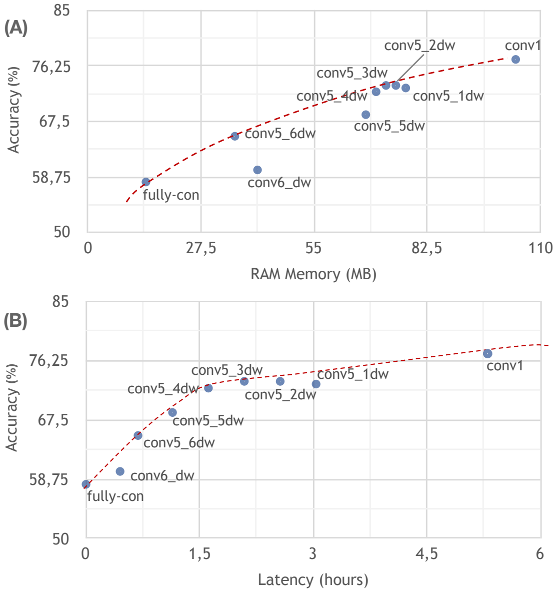

Fig. 5. **Latency-Memory-Accuracy trade-off for different LR cuts.** The red dashed lines represent the two Pareto sets. The red dot (*conv1*, the first layer) assumes that we retrain the whole network.

when choosing a LR layer closer to the first one. Concerning the non-volatile FLASH memory (Fig. 6(A)), the requirement ranges from few MB (6 MB if the last layer is the LR layer) to up to 300 MB, when retraining the full network based on the image samples (the only CL setting not featuring LRs). A typical-size SPI FLASH memory is therefore sufficient in all cases. Also note that LR arrays are accessed sporadically only to perform retraining. Hence, the external FLASH memory can idle, or even be power-gated, for the rest of the time.

Fig. 6(B), instead, plots the RAM requirements, which also includes the 300 LRs related to the new images that require >60% of the overall memory space. The memory breakdown shows that individual memory terms vary depending on the LR layer selection. Only the size of network parameters is constant. Fig. 5(A) combines the RAM memory requirements with the network accuracy, which were demonstrated to be state-of-the-art on the CORe50 dataset [5]. If setting the LR layer as one of the last layers, the accuracy drop increases because of the higher number of frozen parameters. If imposing a constraint of 32 MB for the external DRAM memory, the only match is the retraining of only the last layer (*mid_fc7*). But in this case, accuracy is limited to 58%, which is nearly 20% lower than the baseline (retraining all the layers). Conversely, to reach a higher precision of 72.2% (*conv5_4/dw*), 70 MB of external RAM is needed, which is achieved using a multi-bank DRAM memory.

C. Latency-Accuracy Trade-off

Fig. 7 reports the latency measured on our HW/SW platform over various computation kernels, either for single-core or 8-core execution. The performance is expressed as MAC/cycle in case of forward and backward passes for three representative layers of our benchmark network, namely Pointwise, Depthwise, and Fully Connected layers. Peak values of performance are achieved in Pointwise layers, reaching 2.21 MAC/cycle in forward and 1.70 MAC/cycle in backward computations. The performance measured for the forward pass results higher than backward because GEMM operates on larger vectors, hence the computation density is higher. The speed-up achieved by the 8-cores implementation against single-core reaches 7.79× on average, close to the theoretical limit of 8×.

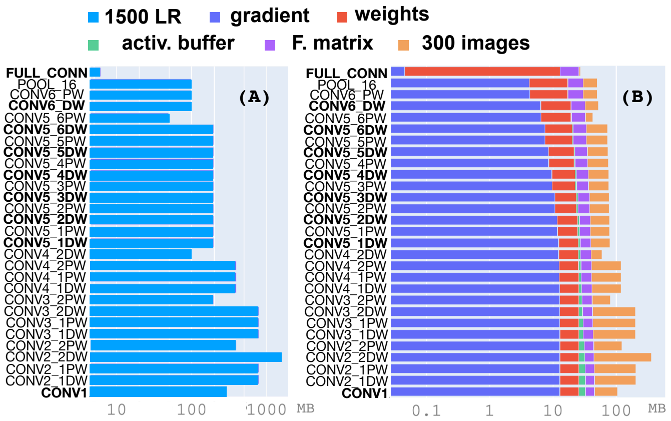

Fig. 6. **FLASH and RAM Memory Footprint for different LR cuts.**

Fig. 5(B) reports the accuracy-latency trade-off on the learning task, with a cluster clock frequency of 150 MHz. The latency refers to the total time to learn a new class, which depends on the chosen Latent Replay layer. To reduce the memory access time overhead, we consider a tiling mechanism between L1 and L2 memories, realized using the DMA engine to minimize latency overhead (below 5% [26] with respect to the execution latency when on all data reside in L1). The L3 to L2 overhead instead is almost negligible because data transfers occur concurrently to the computation on large sub-tensors. Retraining the whole network with 1500 replays results in an accuracy of 77.3%, and the overall latency for 1-class learning is of 318 min. A faster solution is obtained by choosing LR=*conv5_4*. In this setting, the learning latency for a new class is 98 minutes and the reached overall accuracy is 72.2%. If we reduce the number of network's layers to retrain, the accuracy drops more substantially, as shown in Fig. 5. A *ultra-fast* solution retrains only the Fully Connected layer, leading to a latency of 867 ms to learn a single class, but with a 58% accuracy which sets a lower bound for the CL algorithm. Our proposed platform opens up the possibility to perform CL on extreme-edge nodes, which is not practical on current-generation commercial MCUs. For example, compared with a state-of-the-art low-power STM32L476 running at 48 MHz, our platform runs the learning task 25× faster.

D. Energy Estimation

We compare the energy consumption based on the CL mobile application presented in [5], which operates over mini-batches of 500 replays and 100 new images, concluding the convenience and feasibility of CL at the extreme edge. The mobile-class device used for the reference design by Pelle-

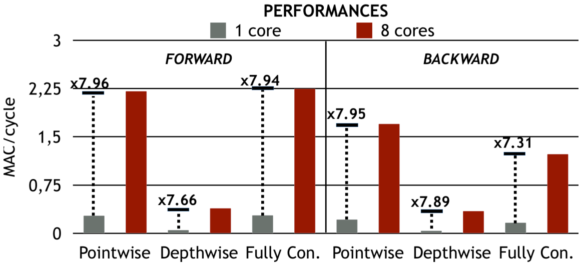

Fig. 7. **Measured Forward and Backward Throughput and parallelization efficiency.**

grini et al. is a OnePlus6 featuring a Qualcomm Snapdragon 845 processor. To estimate the energy consumption of the proposed HW/SW platform, we refer to power measurements from the silicon prototype of MrWolf [8], which operates at 9 MMAC/s/mW and features an average power of 70 mW when running at 150 MHz.

Given an application scenario requesting 1 inference/sec and 1 retraining step per hour, our platform features a total energy consumption of 34.2 J per hour in the fastest CL configuration setting (LR=*mid_fc7*). This implies a battery lifetime of about 710 hours (we consider a 3100 mAh battery), which is 11× higher than a Snapdragon845-based solution (assume $P_{idle} = 0W$). To gain higher accuracy, hence retraining more layers, i.e., LR=*conv5_4* layer, the energy consumption increases to 1530 J per hour, therefore leading to 15.8 h of battery life.

VI. Conclusions

In this work, we presented a novel HW/SW architecture specifically tailored for the execution of CL algorithms. In particular we assess the memory and computational requirements of a Rehearsal-based CL algorithm with Latent Replay. Moreover, we show the accuracy-latency trade-off on the proposed *extreme-edge* system, which results to be 11× more energy-efficient than previous mobile-class processors. The achieved results motivate to further explore Continual Learning on extreme-edge devices.

References

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[2] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[3] M. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey. arxiv 2019," *arXiv preprint arXiv:1908.00080*.

[4] J. Serra, D. Surıs, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," 2018.

[5] L. Pellegrini, G. Graffieti, V. Lomonaco, and D. Maltoni, "Latent replay for real-time continual learning," *arXiv preprint arXiv:1912.01100*, 2019.

[6] V. Lomonaco, D. Maltoni, and L. Pellegrini, "Fine-grained continual learning," *arXiv preprint arXiv:1907.03799*, 2019.

[7] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "Pulp: A parallel ultra low power platform for next generation iot applications," in *27th IEEE Hot Chips Symposium, HCS 2015*. Institute of Electrical and Electronics Engineers Inc., 2016, pp. 1–39.

[8] A. Pullini, D. Rossi, I. Loi, A. Di Mauro, and L. Benini, "Mr. wolf: A 1 gflop/s energy-proportional parallel ultra low power soc for iot edge processing," in *ESSCIRC 2018-IEEE 44th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2018, pp. 274–277.

[9] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[10] V. Losinga, B. Hammera, and H. Wersingb, "Incremental on-line learning: A review and comparison of state of the art algorithms," 2018.

[11] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[12] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[13] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[14] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 6467–6476.

[15] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[16] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019.

[17] T. L. Hayes, N. D. Cahill, and C. Kanan, "Memory efficient experience replay for streaming learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9769–9776.

[18] G. Williams, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Locally weighted regression pseudo-rehearsal for online learning of vehicle dynamics," *arXiv preprint arXiv:1905.05162*, 2019.

[19] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2017.

[20] J. Kone, H. B. McMahan, D. Ramage, and P. Richtarik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016.

[21] Z. Xu, L. Li, and W. Zou, "Exploring federated learning on battery-powered devices," in *Proceedings of the ACM Turing Celebration Conference-China*, 2019, pp. 1–6.

[22] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," *arXiv preprint arXiv:1908.11348*, 2019.

[23] Google. (2019) Edge tpu. [Online]. Available: http://cloud.google.com/edge-tpu/

[24] S. Stmicroelectronics, "stm32h743 datasheet," 2018.

[25] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[26] A. Burrello, F. Conti, A. Garofalo, D. Rossi, and L. Benini, "Work-in-progress: Dory: Lightweight memory hierarchy management for deep nn inference on iot endnodes," in *2019 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2019, pp. 1–2.