Real-Time Continual Learning on Embedded GPUs

Zexin Li[1,2,†] Dongjoo Seo[3] Nikil Dutt[3] Jun Sheng[1] Cong Liu[1]
[1]University of California, Riverside [2]Nanyang Technological University
[3]University of California, Irvine
{zli536,congl}@ucr.edu, zexin.li@ntu.edu.sg, junsheng@engr.ucr.edu, {dseo3,dutt}@uci.edu

Abstract*—Online continual learning (OCL) is an enabling technology for safety-critical autonomous embedded systems whose deployment environments shift faster than fleet-trained models or cloud round-trips can correct. Perception must meet hard sub-second deadlines on a single embedded GPU while training shares the same device; existing OCL runtimes either serialize the two phases or alternate them at coarse cadence, leaving the GPU underutilized and offering no guarantee on inference timing. We present *AOCL*, a self-adaptive OCL runtime for concurrent training and inference on a single embedded GPU, designed to be timing-scoped (deadline-aware safeguard with bounded recovery), practical (drop-in Avalanche plugin, one default hyperparameter setting across all evaluated cells, no offline workload model), and generalizable (validated across two Jetson platforms, four backbones from ResNet-20 to ResNet-101 and ViT-Tiny, four OCL algorithms, and five benchmarks). *AOCL* contributes a Unified Adaptation Metric (UAM) that couples streaming-accuracy drift to training-batch granularity and a lightweight feedback controller that co-schedules training and inference at runtime. Analytical properties of the controller bound the per-axis settling envelope at its operating point and the platform-level per-window worst-case latency at the boundary configuration, supporting deployment in real-time execution. Implemented on top of the open-source Avalanche continual-learning framework and deployed on Jetson Orin Nano and AGX Orin, *AOCL* achieves up to 11.6× higher training throughput (average 5.15×) over the strongest serialized and alternating OCL baselines, while sustaining the lowest or tied-for-lowest deadline-miss rate in 13 out of 15 deadline miss rate measurements. Two realistic case studies: a high-fidelity autonomous-driving simulator and a physical soft-robotic prototype, demonstrate sustained practicality under on-device adaptation.

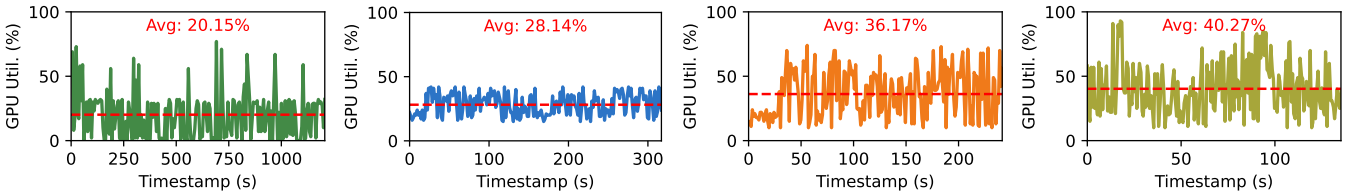
I. Introduction

Autonomous embedded systems are increasingly deployed in environments whose operating distribution is not fully known at training time: off-road and agricultural robots traversing terrain absent from any fleet log, inspection drones surveying industrial sites whose appearance evolves between visits, and post-disaster rescue unmanned ground vehicles/unmanned aerial vehicles (UGVs/UAVs) entering scenes that no curated dataset captures. In each of these settings, a single autonomous platform must adapt its perception model to patterns outside its training distribution while inference continues to drive safe actuation, and intermittent or absent connectivity rules out a round-trip to a remote retraining cluster. Online continual learning (OCL) has emerged as a lightweight, on-device alternative to offline retraining for exactly this regime, and has been demonstrated for streaming object recognition [1], person re-identification [2], and robotic manipulation [3]. The defining requirement is twofold: *inference must meet hard sub-second deadlines* so that perception keeps pace with control, and *adaptation must keep pace with distribution shift* so that the model the controller sees does not grow stale, both subject to the platform's hard compute, memory, and energy budgets on a single embedded GPU. OCL algorithms themselves [4–7] have demonstrated strong properties such as robustness to non-stationary data and mitigation of catastrophic forgetting, but they typically assume strict sequential phases of evaluation and training, leaving the concurrency and timing problem, specifically, how to share a single non-preemptive embedded GPU between latency-critical inference and throughput-hungry training, entirely to the system layer.

This mismatch between algorithmic assumptions and system-level timing requirements turns OCL on a single embedded GPU from a software-engineering nuisance into a safety barrier. Training and inference share the same GPU and submit non-preemptive kernels, so they interfere at a granularity neither layer manages. Current on-device OCL runtimes either *serialize* the phases, avoiding interference but leaving the GPU idle (Fig. 1a), or *alternate* them at a coarse fixed cadence, recovering some throughput but still wasting parallelism (Fig. 1b). These wasted cycles reflect a runtime limit, not a hardware one: the GPU has enough headroom to run training and inference concurrently if interference is managed more finely (Fig. 1c-d). Cloud-derived OCL frameworks such as Ekya [8], RECL [9], and AdaInf [10] pursue this idea, but assume multi-GPU server isolation and operate at tens-of-seconds-to-minute granularity, far beyond sub-second embedded inference deadlines. They also do not characterize schedulability under concurrent training, leaving unanswered a key safety question: how much can concurrent OCL training delay inference, and under what conditions does inference remain schedulable? This system-level gap cannot be closed by OCL algorithmic improvements alone.

Our Approach. To close this gap, we propose *AOCL* (Adaptive Online Continual Learning), a self-adaptive OCL runtime tailored for embedded platforms. Its technical core is a Unified Adaptation Metric (UAM) feeding a finite-state bang-bang controller that gates non-preemptive single-GPU co-execution of training and evaluation. *AOCL* comprises two coordinated components. First, the Unified Adaptation Metric (UAM) is a tunable metric that jointly balances streaming accuracy and end-to-end adaptation latency (normalized to a configurable

[†]This work was completed at the University of California, Riverside.



(a) Avalanche (sequential) (b) Default Alternation (c) $AOCL_{basic}$ (d) *AOCL*

Fig. 1: **Parallelism Opportunities:** GPU utilization of Avalanche, Default Alternation, $AOCL_{basic}$ and *AOCL* when deploying ResNet-20 on NVIDIA Jetson AGX Orin under CoRe50 online continual learning workloads. Average GPU utilization is marked by a horizontal line and highlighted in text.

latency budget); its change over time provides a unified feedback signal for both learning behavior and runtime control. Second, a Dynamic Adaptive Scheduler monitors UAM and continuously adjusts (i) the evaluation–retraining alternation policy and (ii) system/runtime knobs such as batch size and alternation interval. The scheduler runs as a lightweight controller decoupled from the learner, exchanging metrics and commands through a lock-free shared-memory interface and applying updates at iteration boundaries via signal-driven callbacks. This fine-grained time-slicing and contention-aware coordination enables safe overlap of inference and learning on a single GPU.

Beyond the empirical evidence, we establish three analytical properties of the *AOCL* controller in Sec. V, complementing the runtime design with predictability guarantees. These show, respectively, a logarithmic per-axis one-sided sweep-time bound under constant-sign runs that yields a predictable settling envelope (Prop. 1), per-task inference feasibility and queue backlog-freeness at the controller's operating point so steady-state operation is deadline-safe by construction (Prop. 2), and a closed-form per-window worst-case latency bound at the boundary configuration that supplies the single platform-level WCET needed for system-level budgeting (Cor. 1).

Why a lightweight scalar controller? Our design choice reflects the operational reality of on-device OCL. One could in principle solve this control problem with model-predictive control, reinforcement-learning-based scheduling, or offline-profiled lookup tables. Each requires either (a) a workload forward model that does not exist for OCL where the data distribution is unknown a priori, (b) offline traces that cannot be collected for deployments where the operating environment is novel, or (c) on-device training of a separate controller that itself competes for the GPU we are trying to schedule. We therefore adopt a one-scalar, sign-driven rule because it is the simplest design that jointly satisfies all three on-device OCL constraints, *generalizable* (easy parameter choosing across benchmarks, backbones, and platforms), *practical* (one-step-ahead measurements only, no forward model, offline traces, and low overhead), and *scalable/auditable* (its finite-state, bounded-output structure yields the predictability by construction and extends to extra knobs via added axes).

Evaluation. *AOCL* is seamlessly integrated into the widely-used open-source Avalanche [11, 12] framework (2100 GitHub stars) and deployed on Nano and Orin devices widely adopted hardware platform in robotic applications [13–18]. We will open-source this deployable toolchain that makes state-of-the-art OCL algorithms practical for embedded platforms. We extensively evaluated the framework across multiple standard OCL benchmarks and representative embedded hardware platforms, including NVIDIA Jetson Orin Nano and AGX. Evaluation across five benchmarks and four state-of-the-art OCL algorithms shows that *AOCL* achieves up to 11.6× training throughput (average 5.15×) over state-of-the-art baselines while sustaining the lowest or tied-for-lowest deadline miss rate at 13 out of 15 DMR measurements. We further validate *AOCL* through two case studies: an autonomous driving simulator [19] using real-world video input and a Jetson-powered soft robotic system prototype with onboard data collection, both showing consistent speedups and robust accuracy.

This paper makes the following key contributions:

- **Practical and generalizable on-device OCL.** We present, to our knowledge, the first OCL runtime that operates natively on a single embedded GPU without per-deployment retuning. The UAM-driven controller uses one default hyperparameter setting across five benchmarks, four OCL algorithms, four backbones spanning ResNet-20 to ResNet-101 and ViT-Tiny, and two Jetson platforms. It deploys as a non-intrusive plugin on top of the open-source Avalanche framework (2,100 GitHub stars).
- **Adaptive single-GPU runtime with safety properties.** *AOCL* co-schedules training and inference on a single non-preemptive embedded GPU through a finite-state, sign-driven controller with bounded outputs. A deadline-aware safeguard provides a single-tick fallback to a known-feasible configuration; analytical properties characterize the controller's settling behavior and the platform-level per-window worst-case latency at the boundary configuration $B = B_{max}$, supporting deployment inside real-time execution.
- **Empirical validation across diverse settings.** Across five benchmarks, four backbones, and four OCL algorithms on two Jetson platforms, *AOCL* achieves up to $11.6\times$ higher training throughput (average $5.15\times$) over state-of-the-art OCL baselines while sustaining the lowest or tied-for-lowest deadline-miss rate at 13 out of 15 DMR measurements and keeping streaming accuracy within an average of 2.3 percentage points of the sequential Avalanche baseline. Practicality is demonstrated through two realistic robotic

case studies: a high-fidelity autonomous-driving simulator and a Jetson-powered soft-robotic prototype.

II. Background: Online Continual Learning

Online Continual Learning (OCL) updates models incrementally as new data arrives, without retraining from scratch [4, 5, 7, 20]. Unlike offline deep learning on fixed datasets, OCL operates under non-stationary inputs caused by environment shifts, sensor changes, or evolving user behavior [19]: such drift degrades accuracy if the model is not adapted, while incremental updates risk catastrophic forgetting, which replay-based methods such as Experience Replay (ER) [4] mitigate by revisiting stored samples. Continual adaptation is essential in robotics, autonomous vehicles, and smart sensors [21–23], where conditions change unpredictably and embedded platforms impose tight memory, compute, and power budgets that make offline retraining impractical. OCL is instead a lightweight on-device alternative, demonstrated for object recognition [1], person re-identification [2], and manipulation [3].

Streaming Accuracy (SA). SA measures model performance continuously over a stream of inputs. Given sequential samples $\{(x_i, y_i)\}_{i=1}^{T}$,

$$SA(T) = \frac{1}{T}\sum_{i=1}^{T}\mathbb{I}(\hat{y}_i = y_i), \tag{1}$$

where $\hat{y}_i$ is the predicted label, and $\mathbb{I}(\cdot)$ is the indicator function. This metric reflects how well an OCL system maintains accuracy under evolving data [4, 5].

Training Throughput (QPS). Training throughput quantifies how many training samples are processed per second (*queries per second*, QPS).

$$QPS = \frac{\text{\# training samples processed}}{\text{elapsed time (s)}} \tag{2}$$

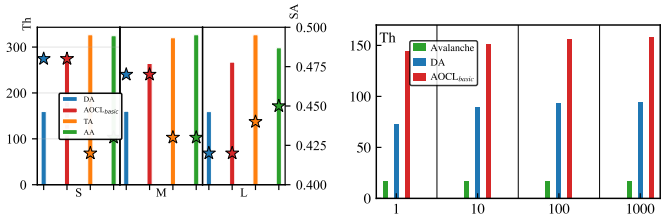
Higher QPS enables the model to incorporate new data faster, reaching a given accuracy level in less time. It is a critical factor for on-device OCL where online adaptation can directly impact system performance and safety. In embedded contexts, QPS complements SA by capturing the system's ability to balance learning effectiveness with computational responsiveness under constrained resources.

III. Motivational Case Studies

In this section, we conduct several motivational case studies to reveal general challenges and optimization opportunities of online continual learning on embedded hardware.

A. Parallelism Opportunities in OCL Systems

We investigate parallelism potential in on-device OCL through a motivational case study on an NVIDIA Jetson AGX Orin, using Experience Replay (ER) [4] across three benchmarks of increasing scale: SplitCIFAR10 (small), CoRe50-NC (medium), and CoRe50-NIC (large). We evaluate three execution modes: (1) Sequential OCL (Avalanche [11]), (2) Default Alternation (DA): a fixed alternation interval, round-robin between evaluation and updates, and (3) Fully concurrent training and evaluation ($AOCL_{\text{basic}}$). Metrics include training throughput (QPS) and streaming accuracy.



(a) Alternation Method (b) Alternation Interval

Fig. 2: Impacts of alternation in OCL systems.

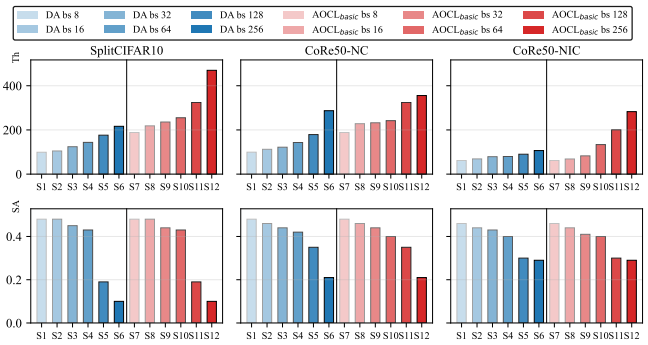


Fig. 3: Impacts of batch size in OCL systems.

Fig. 1 contrasts GPU utilization under four execution modes. The sequential baseline Avalanche shows bursty training spikes separated by long low-utilization inference phases (mean 29%). Default Alternation interleaves the two to reduce blocking, but scheduler overhead suppresses peaks and leaves the mean roughly unchanged (28%). $AOCL_{basic}$ instead overlaps them, filling idle windows and lifting the mean to about 36% with smoother traces, and the full *AOCL* adds self-adaptive coordination that raises it to about 40% while stabilizing utilization. Since higher and steadier utilization at a fixed power budget translates directly into throughput, this case study indicates that concurrent execution of inference and training is a real parallelism opportunity on embedded devices.

B. Challenges in Dynamic OCL Systems

We study two complementary control knobs for on-device OCL: *how* to alternate training and inference, and *how much* work to batch per step. Across three benchmarks (S: SplitCIFAR10, M: CoRe50-NC, L: CoRe50-NIC), we quantify their effects on training throughput (QPS) and streaming accuracy (SA), and use these observations to motivate lightweight, dynamic mechanisms in our ecosystem.

Alternation methods (Fig. 2(a)). We compare four alternation policies: DA and $AOCL_{basic}$ are application-agnostic concurrent policies, whereas TA (throughput-oriented alternation) and AA (accuracy-oriented alternation) add application-aware approximations atop $AOCL_{basic}$ to reduce redundant computation and steer the objective. With a fixed 0.1 s alternation interval, $AOCL_{basic}$ improves QPS over DA by ≈65–69% on

all datasets, and TA/AA push gains to ≈87–105% (e.g., on CoRe50-NC: +100% and +104%; on CoRe50-NIC: +105% and +87%). Accuracy trends align with intent: TA trades a small SA drop on S/M (-0.06/-0.04) but *improves* SA on L (+0.02); AA similarly raises SA on L (+0.03) with modest reductions on S/M. These results show that application-aware alternation not only delivers the highest throughput but can also improve accuracy in large-scale workloads, motivating the need for *dynamic alternation methods* that can switch between policies at runtime.

Alternation interval (Fig. 2(b)). We sweep four alternation intervals (1, 10, 100, 1000ms) for DA and $AOCL_{basic}$, using Avalanche as the sequential reference on a large OCL benchmark CoRe50-NIC. Streaming accuracy is essentially unchanged across alternation intervals (variation ¡0.1%), so we focus on QPS. For DA, longer alternation intervals reliably help by reducing context switches. And on this benchmark, $AOCL_{basic}$ demonstrates similar patterns. However, based on additional measurements, the optimal of $AOCL_{basic}$ alternation interval depends on workload scale: small workloads peak at 1–10ms (SplitCIFAR-10: 1.65×–1.64×); medium workloads peak at 100–1000ms (CORe50-NC: 2.01×–2.00×); and large streams continue to benefit up to 1000ms (CORe50-NIC: 9.17× to 9.32×). These scale-dependent optima motivate a *dynamic alternation interval controller* that adjusts T online based on load and progress.

Batch size (Fig. 3). We evaluate six batch sizes (8–256) under DA (S1-S6) and $AOCL_{basic}$ (S7-S12). Larger batches reliably lift QPS (e.g., on SplitCIFAR10, $AOCL_{basic}$ rises from 218 QPS at bs16 to 470 QPS at bs256; on CoRe50-NIC from 69 QPS to 283 QPS), but reduce SA, gently up to bs64 and sharply at bs128–256. In short, batching offers substantial throughput headroom (i), but large batches risk under-adaptation and accuracy loss (ii), motivating a *dynamic batching* policy that expands when throughput is the bottleneck and contracts when accuracy drifts.

These observations motivate the design of our proposed self-adaptive OCL framework, which extends beyond naive parallelism to dynamically optimize training–evaluation concurrency for diverse embedded workloads.

C. Embedded Challenges for Practical OCL Systems

Deploying OCL on autonomous robots imposes three embedded constraints that fixed alternation or full parallelism cannot jointly satisfy: *(i) tight compute/memory/power budgets*, Jetson-class SoCs offer only a few cores, a modest GPU, 4–32 GB shared memory, and 5–30 W envelopes that bound batch sizes and replay buffers; *(ii) deterministic sub-second latency*, closed-loop perception and control demand predictable end-to-end delays that static schedulers can hardly provide predictability under bursty inputs (no existing embedded OCL framework characterizes whether inference latency remains bounded under concurrent training; we formalize this question in Sec. V); and *(iii) dynamic workload mix*, input rate, sensor modality, and learning progress all shift online, requiring runtime resource management. The next section introduces

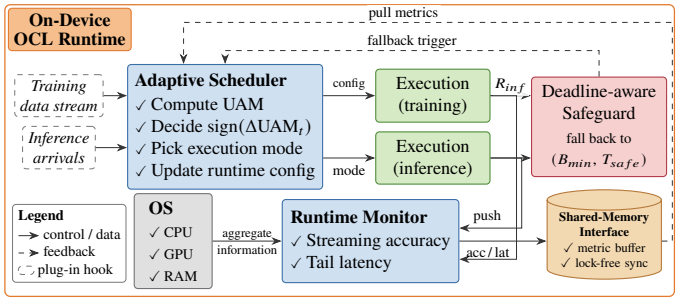


Fig. 4: High-level design of the *AOCL* toolchain.

lightweight, application-aware scheduling that dynamically balances training and evaluation in response to both model progress and device-level signals.

IV. Methodology

This section presents the design and implementation of our on-device OCL toolchain for resource-constrained embedded platforms, which dynamically orchestrates concurrent training and evaluation in response to online streaming-accuracy and latency metrics, delivering application-system co-optimization in dynamic, latency-sensitive environments.

A. System Overview

Fig. 4 illustrates the high-level design of our OCL toolchain, which integrates two concurrent processes: *training*, which incrementally updates the model from incoming data streams, and *evaluation*, which continuously monitors streaming accuracy. At its core is a lightweight adaptive scheduler that coordinates the two through a shared-memory interface for low-latency communication. Guided by continuously monitored runtime metrics, the scheduler adjusts key parameters such as batch size and alternation interval, and alternates between latency-oriented and accuracy-oriented goals, balancing timely adaptation against sustained inference quality.

B. Execution Model and Alternation Granularity

Since *AOCL* shares one non-preemptive GPU, we first state the execution model. Three activities run concurrently: (i) τ_{inf}, a sensor-driven reactive loop released at the sensor period P_{inf} with relative deadline $D_{inf} = P_{inf}$; (ii) τ_{train}, which executes one training step per alternation interval T; and (iii) the *controller*, a separate process ticking every $\Delta = T_q = 100$ ms, with T an integer multiple of T_q. The "while system is running" loop of Alg. 1 is this controller loop, *not* the inference loop. *AOCL* never preempts a GPU kernel: a training step is a chain of many short kernels and non-preemption holds only *within* a kernel, so the driver interleaves the two processes at kernel granularity without splitting any kernel. The blocking term in our analysis is therefore C_{block}, the longest single training kernel, not C_{train}, a full step. Controller outputs $\langle B, T, \text{mode} \rangle$ propagate via signal-driven callbacks and take effect at the worker's next *training-iteration boundary*, never mid-kernel. CUDA event and `nsys` traces confirm the two processes do not overlap, empirically supporting assumption H1 in Sec. V.

C. Unified Adaptation Metric

We propose a Unified Adaptation Metric (UAM). This metric explicitly balances streaming accuracy (SA) and end-to-end latency (L) to suit varying user preferences and constraints. Formally, UAM at time step t is defined as:

$$\text{UAM}_t = \omega \cdot SA_t - (1-\omega) \cdot \hat{L}_t, \tag{3}$$

where $\omega \in [0,1]$ is a user-defined parameter reflecting the relative importance between accuracy and latency. Here, $\hat{L}_t$ denotes latency normalized by a latency budget L_{budget} as $\hat{L}_t = L_t / L_{budget}$, ensuring comparability. Higher ω values prioritize accuracy improvements, whereas lower values prioritize latency reduction. In UAM, L_t denotes end-to-end latency for a complete adaptation window that encompasses both (i) the evaluation pass over the arriving samples and (ii) one retraining step (including replay) whose weight update is committed before the next window begins. We measure L_t using a monotonic clock from the instant the first sample of the window is enqueued to the instant the update is applied, and the next evaluation cycle is eligible to start. Latency budget. L_{budget} is a user-defined target for this end-to-end latency; unless specified, we set L_{budget} to the idle end-to-end latency of the unmodified Avalanche pipeline on the same hardware/workload, providing a normalized baseline for UAM. The linear UAM is a computationally efficient choice, avoiding non-linear scheduling overhead. Our evaluation confirms the chosen hyperparameters generalize well across tasks.

Why this scalarization, and what ω does. An OCL deployment exposes exactly two qualities that are observable online without an offline model: streaming accuracy and end-to-end adaptation latency. UAM scalarizes them into one signal so that $\text{sign}(\Delta\text{UAM})$ answers a single question, "did the last $\langle B, T\rangle$ move help?", at microsecond cost. We choose a weighted *linear* form over the two natural alternatives for concrete reasons. A multiplicative (product) form couples the two terms, so the same relative accuracy loss is penalized differently depending on the current latency, which makes ω platform-dependent and defeats the single-default-setting goal. A lexicographic form makes the controller blind to the secondary objective whenever the primary one is not saturated, which in practice pins the system at one extreme of the trade-off. The linear form keeps the two gradients decoupled and bounded, which is what allows one default ω to transfer across benchmarks and platforms.

Regarding the role of ω itself: ω does not affect controller convergence or stability, because the update rule consumes only $\text{sign}(\Delta\text{UAM})$ and never its magnitude; γ and η likewise set only the sweep rate along each axis, as the logarithmic bounds of Prop. 1 make explicit. What ω selects is the *operating point* on the accuracy–throughput trade-off: raising ω moves the controller toward configurations that favor streaming accuracy at the cost of training throughput, and lowering it does the reverse, with $\omega = 0$ and $\omega = 1$ degenerating to latency-only and accuracy-only control (Sec. IV-G).The mid-range default we use throughout is a balanced operating point chosen from that sweep, and we claim only that the default transfers well, not that behavior is ω-invariant. The incremental change in UAM is computed as:

$$\Delta\text{UAM}_t = \text{UAM}_t - \text{UAM}_{t-1}, \tag{4}$$

serving as the core adaptation signal to guide scheduling decisions. A positive ΔUAM_t indicates improvement, motivating the scheduler to continue or intensify the current configuration. Conversely, non-positive values trigger compensatory adjustments to configurations. For the initial step (t = 0), we set $\Delta\text{UAM}_0 = 0$ by default since no prior state for comparison.

D. Design Rationale: Why a Lightweight Scalar Control Signal?

We design UAM as a one-scalar feedback signal driving a sign-only update rule for three reasons:

- ***Generalizable***: the hyperparameters can be easily chosen across benchmarks, backbones, and platforms without retuning.
- ***Practical***: the controller needs only one-step-ahead measurements already collected for monitoring, no offline traces, forward workload model, or on-device training, matching OCL deployments where task boundaries are unknown a priori and any controller compute competes with the managed GPU.
- ***Scalable and auditable***: a finite-state, sign-driven rule admits the predictability properties of Sec. V by construction and extends to additional knobs (e.g., DVFS, multi-stream placement) by adding axes to the discrete state.

Why only the sign of ΔUAM, and not its magnitude? Discarding the magnitude is deliberate: at a 100 ms window the magnitude of ΔUAM is dominated by measurement noise, turning it into a proportional gain would require the per-platform tuning *AOCL* is designed to avoid, and consuming only the sign keeps the reachable state space finite, which is what makes the properties of Sec. V provable. Severity is nonetheless handled, on two separate paths. Timing-side severity bypasses ΔUAM entirely: a predicted deadline or tail-latency violation triggers a single-tick jump to (B_{min}, T_{safe}) (lines 4–8 of Alg. 1). Accuracy-side severity appears as a run of constant-sign ticks, bounded logarithmically by Prop. 1 and hence corrected well under a second at $T_q = 100$ ms, while small fluctuations are absorbed by the 1- or 2-cycle behavior near a local optimum (Sec. V-B). Severity thus determines *how many* ticks the controller moves and which path fires, not the size of one step.

E. Dynamic Configuration

Batch size (B) and alternation interval (T) are dynamically (re)-configured based on ΔUAM. Adjustments are computed as follows:

Batch Size Update:

$$B_{new} = \text{clip}\big(\text{round}\big(B_{old} \cdot (1 + \gamma \cdot \text{sign}(\Delta\text{UAM}_t))\big), \\ B_{min},\, B_{max}\big), \tag{5}$$

where γ is a positive hyperparameter to control adjustment granularity, and B_{min}, B_{max} are upper/lower bounds determined via offline profiling to stabilize application performance and prevent memory overflow.

Alternation Interval Update:

$$T^{new} = \text{clip}\Big(T_q \cdot \text{round}\Big(\frac{T^{old} \cdot (1+\eta \cdot \text{sign}(\Delta\text{UAM}_t))}{T_q} \Big), \; T_{min}, T_{max} \Big), \tag{6}$$

where η is a positive hyperparameter to govern the alternation interval adaptation granularity, and T_{min}, T_{max} explicitly bound the interval to avoid excessive context switching and starvation, respectively.

Temporal quantization. The adaptive controller observes runtime metrics and re-computes $\langle B, T \rangle$ only at discrete tick boundaries spaced $\Delta = T_q = 100$ms apart. Because the controller cannot meaningfully react faster than its own tick and because the metric collection + scheduler computation + SIGUSR1 propagation pipeline requires approximately one tick to settle to a new $\langle B, T \rangle$ before the next reconfiguration is meaningful, we treat T_q as an irreducible scheduling unit and require T to be an integer multiple of T_q. Eq. (6) above therefore rounds the multiplicative update to the nearest T_q multiple before clipping. Together with the clipping to $[T_{min}, T_{max}]$, the reachable set of T is the finite grid $T_{disc} := \{ k \cdot T_q : k \in \mathbb{Z}_{>0} \} \cap [T_{min}, T_{max}]$ of cardinality $N_T = \lfloor (T_{max} - T_{min})/T_q \rfloor + 1$ (e.g., $N_T = 91$ for $T_{min} = 1$s, $T_{max} = 10$s, $T_q = 100$ms). Combined with the integer-valued B produced by Eq. (5) and the binary mode component $\in \{\text{TA}, \text{AA}\}$, the controller state space is finite.

F. Dynamic Alternation

We build two adaptive alternation policies *on top of* *AOCL*$_{basic}$: a *Throughput-oriented Alternation* (TA) that targets higher training throughput (QPS), and an *Accuracy-oriented Alternation* (AA) that targets higher streaming accuracy (SA). Both retain fully parallel execution and temporarily gate evaluation based on simple feedback signals.

Throughput-oriented Alternation (TA). TA prioritizes rapid adaptation by allocating an initial fraction of each task exclusively to training. Let E_{curr} be the current experience index and E_{total} the total number of experiences; with priority fraction $\alpha \in [0, 1]$, TA runs *training-only* if

$$\frac{E_{curr}}{E_{total}} \leq \alpha, \tag{7}$$

then reverts to the standard *AOCL*$_{basic}$ alternation. This uninterrupted phase maximizes early-stage plasticity and device utilization. Unless otherwise noted, we set $\alpha = 0.1$.

Accuracy-oriented Alternation (AA). AA postpones evaluation until the model achieves a basic competence threshold. Let SA_t denote streaming accuracy at time t and $\delta_{acc} \in (0, 1)$ an accuracy threshold; AA executes *training-only* while

$$SA_t \leq \delta_{acc}, \tag{8}$$

and switches back to *AOCL*$_{basic}$ once $SA_t > \delta_{acc}$. To avoid starvation, we bound the number of checks by N_{max}. We use $\delta_{acc} = 0.1$ in our experiments.

To further enhance performance adaptivity, we introduce a dynamic alternation strategy to ensure responsiveness to changing operational conditions and balances accuracy-throughput trade-offs dynamically. The adaptive logic leverages computed ΔUAM_t as follows:

- If $\Delta\text{UAM}_t > 0$: indicating improving performance, the scheduler adopts the Throughput-oriented Alternation (TA) strategy, prioritizing training for fast model adaptation.
- If $\Delta\text{UAM}_t \leq 0$: indicating degraded joint performance, the scheduler switches to Accuracy-oriented Alternation (AA), suspending evaluation until streaming accuracy achieves/recovers to a threshold δ_{acc}.

G. Generalization to Different Scenarios

The adaptive scheduler gracefully generalizes to different scenarios under specific hyperparameter configurations, demonstrating flexibility and robustness:

- Generalize to latency-oriented scenarios: Occurs if set $\omega = 0$, where Eq. 3 is only influenced by latency.
- Generalize to accuracy-oriented scenarios: Occurs if set $\omega = 1$, where Eq. 3 is only influenced by accuracy.
- Generalize to strict continuous inference requirement scenarios: Occurs if no adaptation ($\gamma = \eta = 0$) is allowed. This configuration continuously maintains training and evaluation concurrently without adjustment.

When should one deploy *AOCL*$_{basic}$? We report *AOCL*$_{basic}$ stand-alone because it is both the concurrency-only anchor that separates the benefit of concurrent execution from that of *adapting* it in the ablation of Sec. VI-E, and a legitimate deployment choice, namely the strict-continuous-inference configuration ($\gamma = \eta = 0$) preferable when the workload is stationary enough that adaptation does not repay its reconfiguration cost or when the deployment forbids runtime reconfiguration of the learner.

Operating without ground-truth labels. UAM's accuracy term assumes supervised streaming labels, as does the OCL literature it builds on, but timing safety never depends on them (the safeguard uses measured latencies only), $\omega = 0$ yields a fully label-free controller, and soft labels from a server-side "golden" model can substitute for ground truth without changing the controller [8], which consumes only $\text{sign}(\Delta\text{UAM})$.

H. Algorithm Details

Algorithm 1 formally describes our adaptive scheduler. After initializing parameters and the initial UAM (line 1), each tick of the control loop collects streaming accuracy and latency from shared memory (lines 2–3). The deadline-aware safeguard (lines 4–8) then checks whether the worst-case inference response time $R_{\text{inf}}(B)$, a table lookup indexed by the current B, exceeds D_{inf} or the measured tail latency exceeds $L_{\text{tail}}^{\max}$; if so it falls back to the safe configuration $(B_{\min}, T_{\text{safe}})$, switches to AA, and skips the remainder of the iteration. Otherwise the

Algorithm 1 Dynamic Adaptive Scheduler

Require: Initial batch size B, alternation interval T, hyperparameters γ, η, ω, upper bound $B_{\max}$, latency budget L_{budget}, accuracy threshold δ_{acc}, inference deadline D_{inf}, tail-latency threshold $L_{\text{tail}}^{\max}$, safe configuration $(B_{\min}, T_{\text{safe}})$

Ensure: Continuous adaptation of B and T to balance streaming accuracy and throughput, while not violating the inference deadline

1: Compute initial UAM_0 using Eq. (3)
2: **while** system is running **do**
3: Collect current metrics SA_t and L_t
4: **if** $R_{\text{inf}}(B) > D_{\text{inf}}$ **or** $L_t^{p99} > L_{\text{tail}}^{\max}$ **or** $L_t^{p99.9} > L_{\text{tail}}^{\max}$ **then**
5: $B \leftarrow B_{\min}$, $T \leftarrow T_{\text{safe}}$
6: Switch to Accuracy-oriented Alternation (AA)
7: **continue**
8: **end if** ▷ Deadline-aware safeguards
9: Compute current UAM_t using Eq. (3)
10: Compute incremental ΔUAM_t using Eq. (4)
11: **if** $\Delta\text{UAM}_t > 0$ **then**
12: Updated B via Eq. (5) ▷ Increase batch size
13: Updated T via Eq. (6) ▷ Increase alternation interval
14: Switch to Throughput-oriented Alternation (TA) based on Eq. (7)
15: **else**
16: Updated B via Eq. (5) ▷ Decrease batch size
17: Updated T via Eq. (6) ▷ Decrease alternation interval
18: Switch to Accuracy-oriented Alternation (AA) based on Eq. (8)
19: **end if**
20: Asynchronously propagate updates to training and evaluation processes
21: **end while**

scheduler computes UAM_t and ΔUAM_t (lines 9–10), and either increases $\langle B, T\rangle$ and switches to TA when $\Delta\text{UAM}_t > 0$ (lines 11–14) or decreases both and selects AA otherwise (lines 15–19). Updates are propagated asynchronously to the training and evaluation processes (line 20) before the loop repeats (line 21).

V. Timeliness Analysis

We provide predictability properties to support deploying *AOCL* inside real-time execution and formalize the system's execution flow and bound its worst-case latency for a single adaptation window.

A. Execution Operations in One Adaptation Window

We enumerate the operation costs contributing to the end-to-end latency of one adaptation window: inference $C_{inf}(B)$ at batch size B, training $C_{train}(B)$ at batch size B, double-buffer swap C_{swap}, UAM computation C_{uam}, and reconfiguration $C_{reconfig}$. Each C_* denotes the worst-case execution time of the corresponding operation: $C_{inf}(B)$ and $C_{train}(B)$ are the maxima of the per-kernel worst-case execution times over the inference and training kernel chains at batch size B, while $C_{swap}, C_{uam}, C_{reconfig}$ are constants independent of B. We assume H1: *within a single CUDA stream, kernels execute sequentially and non-preemptively*. Real-time self-adaptive systems should be predictable and avoid unbounded oscillations. Our adaptive controller has a bang-bang structure driven by $\text{sign}(\Delta\text{UAM})$ with strictly bounded outputs and under a finite state space.

B. Time-to-Recurrent-Regime Bound for the Discrete Controller

Remark (Bounded state space and ISS). Let state $s^{(k)} := (B^{(k)}, T^{(k)}, m^{(k)})$ with $s^{(k)} \in \mathcal{S} := B_{disc} \times T_{disc} \times \{\text{TA}, \text{AA}\}$, where $B_{disc} := [B_{min}, B_{max}] \cap \mathbb{Z}$ and T_{disc} is the finite grid of Sec. IV-E. By construction $|\mathcal{S}| = N_B \cdot N_T \cdot 2$ with $N_B = B_{max} - B_{min} + 1$ and $N_T = \lfloor (T_{max} - T_{min})/T_q \rfloor + 1$. Pigeonhole on $|\mathcal{S}| + 1$ states immediately gives a worst-case state-recurrence (state-revisit, not cycle-entry, since the controller is non-autonomous in $\text{sign}(\Delta\text{UAM}_t)$) interval of at most $|\mathcal{S}|$ ticks, and the bang-bang controller with clipping enjoys a discrete-time analogue of input-to-state stability (ISS) only in the trivial sense that the clipped state trajectory remains in the compact set $\mathcal{S}$ by construction, for any bounded ΔUAM disturbance sequence. Both statements follow directly from the system construction; the substantive per-axis one-sided sweep-time claim is the content of Prop. 1 below.

Empirically, the controller is observed at runtime to settle into one of two natural regimes; we report these as observed runtime behavior: (a) a *1- or 2-cycle near a local optimum of* ΔUAM, the controller settles on a single configuration or alternates between two adjacent configurations as ΔUAM oscillates around zero; and (b) a *recurrent visit pattern of length* $\ell \in [2, |\mathcal{S}|]$, the controller traverses a small set of configurations driven by persistent ΔUAM alternation.

Proposition 1 (State-Recurrence and One-Sided Sweep Bounds)**.** *Under the assumptions of the bounded-state-space remark, the trajectory revisits some state in* $\mathcal{S}$ *within at most*

$$K_{transient} \le |\mathcal{S}| = N_B \cdot N_T \cdot 2 \tag{9}$$

ticks, equivalently $K_{transient} \cdot \Delta$ *wall-clock time. In the one-sided sweep regime, where* $\text{sign}(\Delta\text{UAM}_t)$ *is constant for a run of consecutive ticks starting from a boundary configuration (the regime forced by the deadline-aware safeguard branch in Alg. 1), the number of ticks the controller needs to drive* B *(resp.* T*) from one boundary to the opposite boundary is bounded by*

$$K_B^{transient} \le \left\lceil \log_{1+\gamma}\left(B_{max}/B_{min}\right) \right\rceil + 1 \quad \textit{ticks sweep time}, \tag{10}$$

$$K_T^{transient} \le \left\lceil \log_{1+\eta}\left(T_{max}/T_{min}\right) \right\rceil + 1 \quad \textit{ticks sweep time}, \tag{11}$$

Eqs. (10)–(11) bound the number of controller ticks *needed for a monotonic sweep along each axis, not the values of* B *or* T *themselves. The additive* $+1$ *in each bound conservatively absorbs the per-step rounding error of magnitude at most* $\frac{1}{2}$ *(in* B*-units) or* $\frac{1}{2}T_q$ *(in* T*-units).*

Proof. Eq. 9 follows from the same pigeonhole argument as the bounded-state-space remark (Sec. V-B): among the first $|\mathcal{S}| + 1$ states the trajectory must repeat. For Eq. (10), under sustained $\text{sign}(\Delta\text{UAM}_t) = +1$ from $B^{(0)} = B_{min}$, Eq. (5) yields $B^{(k+1)} \ge \text{round}\big((1+\gamma)B^{(k)}\big) \ge (1+\gamma)B^{(k)} - \frac{1}{2}$ before clipping. Iterating gives $B^{(k)} \ge B_{min}(1+\gamma)^k - \frac{1}{2}\sum_{i=0}^{k-1}(1+\gamma)^i$, so $B^{(k)} \ge B_{max}$ whenever the

real-valued threshold $\log_{1+\gamma}(B_{max}/B_{min})$ is exceeded by the integer index k; since k is an integer, the smallest such k is $\lceil \log_{1+\gamma}(B_{max}/B_{min}) \rceil$, and the additive $+1$ absorbs the per-step rounding error of magnitude at most $\frac{1}{2}$, giving $K_B^{transient} \leq \lceil \log_{1+\gamma}(B_{max}/B_{min}) \rceil + 1$ as in Eq. (10); the clip then activates and B stays at B_{max} thereafter. The symmetric -1 case from B_{max} yields the same $\lceil \log_{1+\gamma}(B_{max}/B_{min}) \rceil + 1$ bound. Eq. (11) follows by an identical argument for T with η in place of γ and T_q as the rounding granularity instead of 1, yielding $K_T^{transient} \leq \lceil \log_{1+\eta}(T_{max}/T_{min}) \rceil + 1$. □

C. Feasibility and Backlog-Freeness for Continuous Inference

As defined in Sec. IV-B, τ_{inf} is the hard-deadline task and τ_{train} is the best-effort co-runner with no hard deadline, sharing the non-preemptive GPU resource. Per-task boundary feasibility for τ_{inf} is then a single-task response-time check under blocking from the longest in-flight τ_{train} kernel. We use $D_{inf} = P_{inf}$ as a representative inference deadline.

- **Inference Task (τ_{inf}):** Arrives periodically with period P_{inf} (the sensor cadence; e.g., a typical $P_{inf} = 33.3$ ms for a 30 FPS camera). It has a worst-case execution time $C_{inf}(B)$ and a relative deadline $D_{inf} \leq P_{inf}$. To ensure no dropped inputs, it requires $D_{inf} = P_{inf}$.
- **Training Task (τ_{train}):** Arrives periodically with the alternation interval T. It has a worst-case execution time $C_{train}(B)$ but no hard real-time deadline.

Under this non-preemptive GPU sharing model, the worst-case response time for inference is:

$$R_{inf} = C_{inf}(B) + C_{block}(B) \tag{12}$$

where $C_{block}(B) := \max_{k \in \text{kernels}(\tau_{train}, B)} \text{duration}(k)$ is the worst-case blocking from the single longest non-preemptible training kernel. Strict continuous inference is ensured if $R_{inf} \leq D_{inf}$. The fundamental feasibility requirement is that the minimum-batch configuration meets the deadline:

$$C_{inf}(B_{min}) + C_{block}(B_{min}) \leq D_{inf} \tag{13}$$

The training task τ_{train} executes once per alternation interval T and shares the GPU with τ_{inf}, which arrives every P_{inf}. Because both B and T are controller-actuated (Eqs. 5 and 6), we state utilization as a function of both:

$$U(B,T) := \frac{C_{inf}(B)}{P_{inf}} + \frac{C_{train}(B)}{T} \leq 1. \tag{14}$$

Eq. 14 is a *necessary feasibility condition*: it ensures τ_{train} does not unboundedly accumulate backlog at the operating (B,T). For each $(B, \text{algorithm}, \text{platform})$ cell, the minimum feasible alternation interval is

$$T_{\text{feas}}(B) = \frac{C_{train}(B)}{1 - C_{inf}(B)/P_{inf}}, \tag{15}$$

so the cell is reachable by the controller if and only if $T_{\text{feas}}(B) \leq T_{max}$ and $C_{inf}(B)/P_{inf} < 1$.

Proposition 2 (Boundary Feasibility and Backlog-Freeness)**.** *Under the task model of Sec. V, the following two results hold:*

(a) ***(Boundary feasibility.)*** *If Eq. (13) holds, then at the boundary configuration $B = B_{min}$ the worst-case response time of τ_{inf} satisfies $R_{inf}(B_{min}) \leq D_{inf}$. The controller guarantees deadline satisfaction at B_{min} and retreats toward B_{min} whenever $R_{inf}(B) > D_{inf}$ at larger B.*

(b) ***(Backlog-freeness.)*** *If the utilization condition Eq. (14) holds at operating point (B,T), then the best-effort task τ_{train} does not accumulate unbounded backlog at that operating point.*

Proof. (a) Eq. (13) gives $R_{inf}(B_{min}) = C_{inf}(B_{min}) + C_{block}(B_{min}) \leq D_{inf}$, so τ_{inf} meets its deadline at the smallest reachable batch B_{min}. At larger B, $R_{inf}(B)$ increases monotonically in B (Sec. V-A) and may exceed D_{inf}; Eq. (13) therefore does not by itself guarantee deadline satisfaction at $B > B_{min}$. Instead, the controller monitors $R_{inf}(B)$ at the operating point and retreats toward B_{min} whenever $R_{inf}(B) > D_{inf}$; the retreat is always feasible because Eq. (13) guarantees the destination B_{min} satisfies the deadline. (b) Eq. (14) bounds the combined GPU fraction consumed by τ_{inf} and τ_{train} at (B,T) by one, so τ_{train} does not accumulate unbounded backlog. The blocking term $C_{block}(B)$ in $R_{inf}(B)$ is therefore well-defined (bounded by the longest in-flight τ_{train} kernel rather than an unbounded queue). □

Although *AOCL*'s analysis targets the single embedded GPU regime that dominates edge deployments, the controller skeleton extends naturally to multi-GPU embedded platforms by composing it with workload-partitioning schemes such as pipeline parallelism [24] or introspective per-job placement [25], and with microsecond-scale preemptive GPU schedulers such as REEF [26]. We will leave multi-GPU extension as future work. While Prop. 2 establishes per-task feasibility and backlog-freeness at the operating point, system-level budgeting additionally requires a single platform-level worst-case window latency. Cor. 1 supplies this bound by collapsing the per-window WCET sum to its boundary value at $B = B_{max}$ under monotonicity in B.

Corollary 1 (Adaptation Window Latency Bound)**.** *For any reachable controller configuration with $B \in [B_{min}, B_{max}]$, assuming $C_{inf}(B)$, $C_{train}(B)$, $C_{block}(B)$ are non-decreasing in B, the per-window worst-case wall-clock latency satisfies*

$$\begin{aligned} L_{window}^{WC} \leq\ & C_{inf}(B_{max}) + C_{train}(B_{max}) \\ & + C_{swap} + C_{uam} + C_{reconfig}, \end{aligned} \tag{16}$$

and this bound is attained at $B = B_{max}$. Under H1 the per-window wall time is bounded by the per-task WCET sum; monotonicity in B makes the data-path summands C_{inf}, C_{train} maximal at $B = B_{max}$, while $C_{swap}, C_{uam}, C_{reconfig}$ are B-independent controller-side constants declared in Sec. V-A, yielding the closed-form bound in Eq. (16).

Scope of the guarantee. *AOCL* is not an unconditional hard real-time system: under H1 and measured worst-case execution times we prove per-task feasibility of τ_{inf} at the

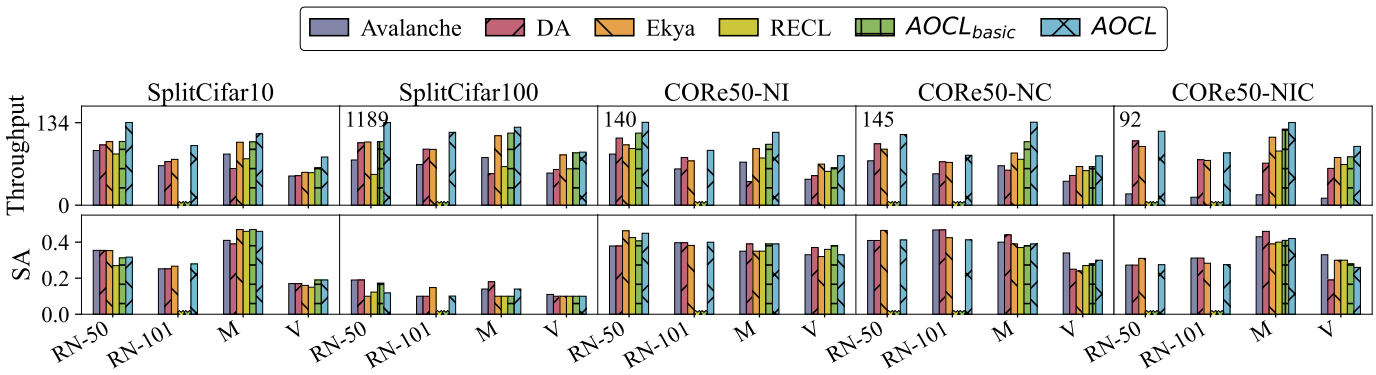


Fig. 5: Overall effectiveness of *AOCL* on different deep learning models on the ER algorithm across benchmarks. The top row reports training throughput (QPS); the bottom row reports streaming accuracy (SA). Each cell shows four backbones: RN-50 (ResNet-50), RN-101 (ResNet-101), M (MobileNetV1), and V (ViT-Tiny). Crosses ($\times$) at $y = 0$ mark cells where the baseline failed due to out-of-memory (OOM) errors.

TABLE I: Existing baselines and their supported system features on on-device online continual learning systems. ●, ◐, ◐ represent support, partial support, and no support.

Baseline	Concurrent Exc.	Strict Continuous Inf.	Multi-instance Exc.
Avalanche [11]	○	○	○
DA	●	○	○
Ekya [8]	●	◐	◐
RECL [9]	●	◐	◐
AOCL$_{basic}$	●	◐	●
AOCL	●	●	●

boundary configuration B_{min} (Eq. 13) and backlog-freeness at the operating point, while at larger batch sizes the deadline is enforced by the runtime's monitored retreat to (B_{min}, T_{safe}) rather than by an a priori zero-miss proof.

VI. Evaluation

In this section, we thoroughly evaluate *AOCL* across various online continual learning scenarios.

A. Experimental Setup

Setup overview. We evaluate *AOCL* on two NVIDIA Jetson platforms (AGX Orin and Orin Nano) using a wide spectrum of deep learning settings: four representative OCL algorithms (ER, GSS, GEM, AGEM), five deep learning models (ResNet-20, ResNet-50, ResNet-101, MobileNetV1, ViT-Tiny), and five continual learning benchmarks. We also evaluate *AOCL*'s practicality in two robotic case studies.

Baselines. We compare *AOCL* against four methods, contrasted with our system-level capabilities in Tab. I.

- **Avalanche [11]:** a widely used standard open-source OCL library (2,100+ GitHub stars), used as the purely sequential vanilla baseline.
- **Default Alternation (DA):** round-robin alternation of training and evaluation, capturing the basic concurrency benefit over sequential execution.
- **Ekya [8]:** originally for offline continual learning on multi-GPU servers; adapted to the embedded OCL setting.
- **RECL [9]:** originally for autonomous-driving and video-analytics continual learning on multi-GPU servers; adapted to the embedded OCL setting.

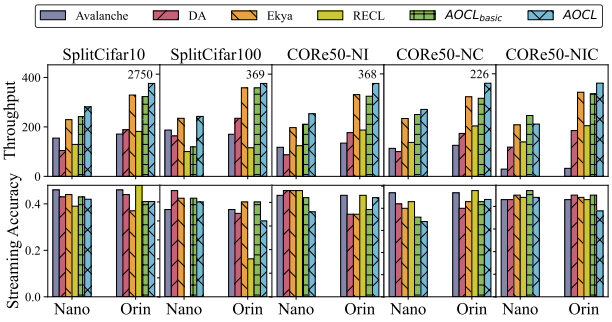


Fig. 6: Overall effectiveness of *AOCL* using the ER algorithm evaluated on five benchmarks on ResNet-20 model.

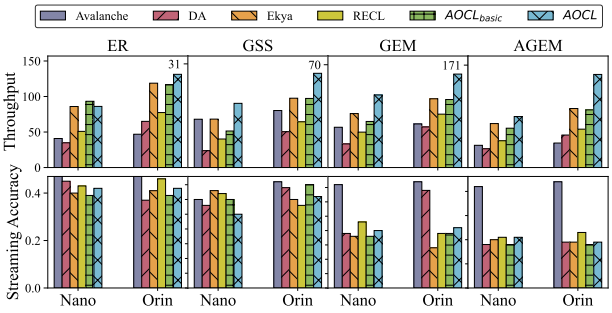


Fig. 7: Overall effectiveness of *AOCL* on four OCL algorithms on the CORe50-NC benchmark.

Neither DA, Ekya, nor RECL natively supports strict continuous inference (CI) or multi-instance execution; we add lightweight deadline-aware preemption and per-instance throttling so they can run in these settings (◐ in Tab. I). *AOCL* support all three features natively (●) via fine-grained slicing, deadline tracking, and contention-aware scheduling.

B. Overall Effectiveness

We evaluate *AOCL* on both embedded GPU platforms across (i) standard OCL benchmarks of varying scale, (ii) OCL algorithms of varying complexity, and (iii) representative deep learning architectures of varying model size.

Effectiveness Across Benchmarks. Fig. 6 reports ResNet-20 training throughput across five benchmarks. On Nano

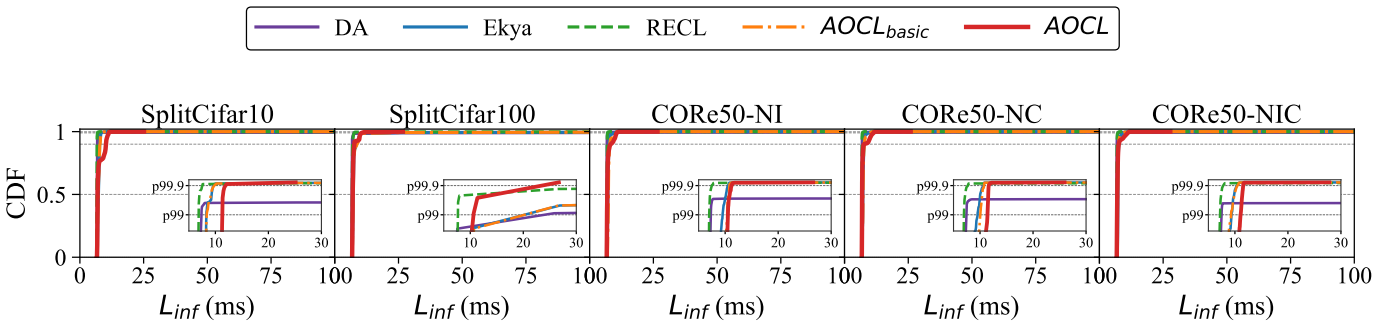


Fig. 8: Per-frame inference-latency CDFs on Jetson Orin under sustained ER training with ResNet-20 on five benchmarks. The x-axis is capped at 100 ms to expose method separation in the typical-frame regime; horizontal dashed lines mark the $p50/p90/p99/p99.9$ percentile reference levels. The lower-right inset of each panel zooms into the upper-tail $[5, 30]$ ms range with a $[0.985, 1.000]$ CDF window so the $p99/p99.9$ separation between the four concurrent methods is visually distinguishable.

(Orin), *AOCL* delivers $4.57\times$ ($5.72\times$) average throughput over Avalanche, peaking at $7.18\times$ ($11.61\times$) on CORe50-NIC (6.70 to 48.12 QPS on Nano, 7.39 to 205.65 QPS on Orin), and beats the static $AOCL_{basic}$ by 18.1% (15.4%) on average; the largest adaptivity gain is on CORe50-NI for Nano (40.6%) and CORe50-NIC for Orin (45.8%). Streaming accuracy stays competitive (e.g., 0.42/0.41 on SplitCIFAR10/Nano/Orin vs. Avalanche's 0.46), exceeding all concurrent baselines on the remaining benchmarks. RECL fails on SplitCIFAR100/Nano due to its computational overhead under the platform's resource constraints.

Effectiveness Across Algorithms. Figure 7 reports throughput across ER, GSS, GEM, and AGEM on CORe50-NC. On Nano with ResNet-20, *AOCL* delivers $1.80\times$ on GEM and $2.29\times$ on AGEM over Avalanche, with $1.76\times$ (GSS) and $1.58\times$ (GEM) over the static $AOCL_{basic}$. On Orin, it achieves $1.65\times$ on GSS and $3.80\times$ on AGEM over Avalanche, plus 12.9% on ER and 61.5% on AGEM over $AOCL_{basic}$. Breaking down by complexity: ER incurs the lowest overhead and the highest absolute throughput but smaller relative gains; gradient-based GEM/AGEM benefit most from overlap; and GSS at B_{max}=32 incurs a per-step training WCET of 628.4 ms on Orin and 930.6 ms on Nano, roughly $31\times$ and $18\times$ the ER cost, nearly $19\times$ a 30 FPS frame budget, making the adaptive batch, shrinking response essential rather than optional. Streaming accuracy remains competitive across all algorithms: *AOCL* matches or exceeds all concurrent baselines and approaches the Avalanche oracle (e.g., 0.42 vs. 0.47 on ER/Orin).

Effectiveness Across Deep Learning Models. To verify generalization beyond ResNet-20, we evaluate *AOCL* on Jetson Orin across four deep learning models spanning four deep learning models with different sizes and architectures in Figure 5. On ResNet-50, ResNet-101, MobileNetV1, and ViT-Tiny, *AOCL* achieves $1.21\times$, $1.21\times$, $1.16\times$, and $1.20\times$ averaged QPS gain over the strongest concurrent baseline, respectively, while trading off -4.2, -2.5, -2.8, and -1.4 percentage points of streaming accuracy against the best non-Avalanche method on average. The heavier ResNet-50 and ResNet-101 backbones *tighten* the per-task feasibility envelope (Eq. 13) by enlarging C_{train}; *AOCL* still retains throughput SOTA at every (benchmark, backbone) cell under the batch-size caps B_{max}=32 (ResNet-50) and B_{max}=16 (ResNet-101) imposed to honor Prop. 2. RECL and $AOCL_{basic}$ deadlock on ResNet-101 via a Python-multiprocessing socket-FD interaction unrelated to the controller, leaving *AOCL* as the only schedulable method across the full 5.5–170 MB model-size.

Overall effectiveness

AOCL consistently optimizes training throughput across benchmarks and OCL algorithms on both embedded GPU platforms while maintaining competitive streaming accuracy.

C. Empirical Tail Inference Latency Measurement

Safety-critical workloads also need bounded *tail* inference latency under sustained co-running training. We collect $\geq$ 10,000 consecutive per-frame L_{inf} samples per cell on Jetson Orin under sustained ER training with ResNet-20. Avalanche is omitted as it never co-runs training and inference.

Tab. II reports deadline-miss rates (DMR) at three tiers: 16/33 ms match 60/30 FPS sensor cadences, and 100 ms is the sub-second envelope of larger pipelines [27]. Each cell measures per-frame inference co-running with sustained ER training of ResNet-20 on AGX over $\geq$ 10,000 frames, with deadlines fixed a priori from application cadences rather than fitted to the measurements. DA's sequential alternation makes inference wait for a full training step, yielding 0.50–1.43% DMR and a $>$ 1 s CDF tail. Among concurrent methods, *AOCL* attains the lowest or tied-for-lowest DMR at 33 and 100 ms on every benchmark, stays within 0.04% of Ekya/$AOCL_{basic}$ at 16 ms on small benchmarks, and keeps DMR $< 0.5\%$, validating Prop. 2: retreating B when $R_{inf}(B) > D_{inf}$ holds inference within the deadline while preserving the gains of Sec. VI-B. Residual misses at 16 ms do not contradict Sec. V, as feasibility is checked against *measured* worst-case execution times that occasional OS or memory jitter can exceed.

The L_{inf} CDFs in Fig. 8 corroborate this: DA's $p99.9$ exceeds 845 ms, a flat segment near CDF $= 0.99$; RECL is

TABLE II: Deadline-miss rate (DMR) on Jetson Orin under ER with ResNet-20. Best results in bold.

Benchmark	Method	DMR @ 16 ms	DMR @ 33 ms	DMR @ 100 ms
SplitCIFAR10	DA	0.63%	0.62%	0.62%
	Ekya	**0.04%**	0.02%	0.02%
	RECL	0.05%	0.03%	0.03%
	$AOCL_{basic}$	**0.04%**	0.02%	0.02%
	AOCL	0.05%	**0.00%**	**0.00%**
SplitCIFAR100	DA	1.43%	0.95%	0.95%
	Ekya	1.43%	0.71%	0.71%
	RECL	**0.41%**	0.20%	0.20%
	$AOCL_{basic}$	1.43%	0.71%	0.71%
	AOCL	0.48%	**0.00%**	**0.00%**
CORe50-NI	DA	0.50%	0.50%	0.50%
	Ekya	**0.01%**	0.01%	0.01%
	RECL	0.02%	0.01%	0.01%
	$AOCL_{basic}$	**0.01%**	0.01%	0.01%
	AOCL	**0.01%**	**0.00%**	**0.00%**
CORe50-NC	DA	0.52%	0.52%	0.52%
	Ekya	**0.01%**	0.01%	0.01%
	RECL	0.04%	0.03%	0.03%
	$AOCL_{basic}$	**0.01%**	0.01%	0.01%
	AOCL	**0.01%**	**0.00%**	**0.00%**
CORe50-NIC	DA	0.64%	0.64%	0.64%
	Ekya	**0.01%**	**0.00%**	**0.00%**
	RECL	0.03%	0.02%	0.02%
	$AOCL_{basic}$	**0.01%**	**0.00%**	**0.00%**
	AOCL	**0.01%**	**0.00%**	**0.00%**

tightest at $p99$ ($\leq$ 7.66 ms) but sacrifices training throughput (Sec. VI-B); *AOCL* stays competitive (10.5–11.4 ms, below the 16 ms line) and above $p99.9$ keeps the tightest tail (11.2–23.5 ms) while other baselines reach 100 ms, explaining its dominance at 33/100 ms DMR.

Timing predictability

Across benchmarks and deadline tiers, *AOCL* sustains the lowest deadline-miss rate and the tightest latency tail among concurrent methods, showing that the throughput gains do not come at the expense of inference timeliness.

D. Practical Robotic Case Study

To demonstrate the applicability of *AOCL* in realistic embedded scenarios, we prototype on NVIDIA Jetson platforms and evaluate in both simulated (Fig.10) and physical environments (Fig.9). These studies test the system's ability to sustain training throughput and streaming accuracy under diverse perception workloads and dynamically changing conditions, using ER as the representative OCL algorithm.

Autonomous Robotic Navigation. We deploy *AOCL* on a Jetson Orin navigation platform in the Endless-Sim procedural environment [19], covering two perception tasks: instance-level classification (ILC) and semantic segmentation (SS), each under three OCL scenarios: Incremental Class (IC, novel objects/terrain), Incremental Illumination (IL, lighting changes), and Weather Condition variation (WC, rain/fog). Tab. IIIa shows *AOCL* reaches up to 1.80 QPS on ILC and 1.09 QPS on the heavier SS (both in IL), corresponding to 1.36–3.12$\times$ speed-ups over the sequential Avalanche oracle with absolute accuracy drops $\leq$ 0.02 ($\geq$ 0.97 in most ILC cells; best or second-best on SS).

Soft Robotic Case Study. We further evaluate *AOCL* on a Jetson Orin soft-robot prototype (Fig. 9) under two OCL scenarios: IC (classifying varying actuator pressure levels) and IL (illumination changes on the vision stream). In Tab. IIIb, IL is the hardest setting: all concurrent methods lose SA vs. the sequential Avalanche oracle (0.68); *AOCL* attains 0.57 SA, +1 pp over RECL and +15/+13 pp over DA/Ekya. In exchange for this 9–11 pp gap, *AOCL* delivers the highest training throughput at 2.16 QPS (IC) and 2.31 QPS (IL), a 14.9% and 8.45% gain over the strongest prior baseline (Ekya at 1.88 and 2.13 QPS). Averaged across both robotic case studies, *AOCL* stays within $\approx$2.3 pp of the sequential baseline in streaming accuracy.

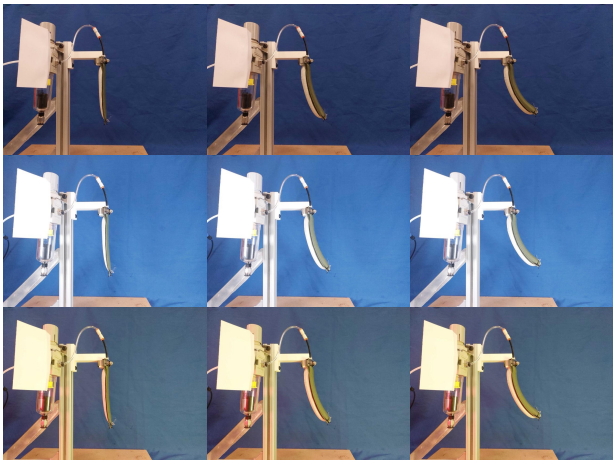


Fig. 9: OCL-driven soft robotic system prototype under dynamic actuator pressure and lighting conditions.



Fig. 10: High-fidelity navigation simulation with dynamic environmental factors (lighting, weather) for on-device OCL.

Practical usability

In both simulated/real robotic prototypes, *AOCL* maintains competitive streaming accuracy while significantly increasing training throughput, underscoring *AOCL*'s effectiveness in addressing practical constraints encountered in real-world embedded continual learning deployments.

E. Ablation Study

We ablate each component of *AOCL* using ER across five benchmarks (Tab. IV). $AOCL_{\text{basic}}$ achieves moderate throughput at moderate accuracy; TA pushes throughput at modest accuracy cost (up to 2500.00 QPS on SplitCIFAR100), while AA maximizes accuracy on three of five benchmarks at lower throughput (0.45 on SplitCIFAR10 and CORe50-NIC, 181.40 QPS on CORe50-NIC). The full adaptive *AOCL* attains the best throughput (375.94 vs. 326.80 QPS on SplitCIFAR10 over TA) while keeping accuracy within 0.03 of the per-benchmark best, so the controller both aggregates the TA/AA benefits and smooths their trade-offs across OCL paradigms. Concurrency alone accounts for the bulk of the throughput gain; the UAM-driven controller adds further improvement over $AOCL_{basic}$

TABLE III: Robotic case study results: (a) autonomous navigational simulation and (b) OCL-driven soft robots. TT indicates training throughput, and SA indicates streaming accuracy. Best values are highlighted in bold.

Baseline	Training Throughput [QPS] ↑						Streaming Accuracy ↑					
	ILC			SS			ILC			SS		
	IC	IL	WC	IC	IL	WC	IC	IL	WC	IC	IL	WC
Avalanche	2.13	0.51	0.49	0.52	0.35	0.34	**0.97**	**0.99**	**0.99**	**0.97**	**0.94**	**0.94**
DA	3.73	1.01	0.97	0.77	0.59	0.46	0.48	0.99	0.91	0.96	0.83	0.86
Ekya	6.30	1.73	1.72	0.98	0.72	0.71	0.49	0.99	0.82	0.96	0.91	0.93
RECL	2.18	1.12	1.09	0.46	0.38	0.34	0.97	0.99	0.94	0.96	0.94	0.93
$AOCL_{basic}$	**6.30**	1.77	**1.74**	**1.00**	0.79	**0.77**	0.49	0.96	0.91	0.97	0.85	0.93
AOCL	6.10	**1.80**	1.73	0.71	**1.09**	0.58	0.95	**0.99**	0.97	**0.97**	0.93	0.92

(a) Autonomous navigational simulation

Baseline	TT [QPS] ↑		SA ↑	
	IC	IL	IC	IL
Avalanche	1.73	1.07	0.06	**0.68**
DA	1.32	1.30	0.07	0.42
Ekya	1.88	2.13	0.06	0.44
RECL	1.16	1.32	0.06	0.56
$AOCL_{basic}$	2.12	2.30	**0.07**	0.59
AOCL	**2.16**	**2.31**	0.06	0.57

(b) OCL-driven soft robots

TABLE IV: Ablation study results of training throughput [QPS] and streaming accuracy. Best results are in bold.

Benchmark	$AOCL_{basic}$	TA	AA	*AOCL*
SplitCIFAR10	322.58/0.41	326.80/0.42	324.68/**0.43**	**375.94**/0.41
SplitCIFAR100	2380.95/0.25	2500.00/0.26	2500.00/**0.28**	**2500.00**/0.20
CORe50-NI	289.61/0.37	296.78/0.37	295.33/0.35	**336.16/0.42**
CORe50-NC	280.94/0.43	272.51/0.43	278.09/0.43	**334.08/0.44**
CORe50-NIC	181.95/0.45	198.68/0.44	181.40/**0.45**	**205.70**/0.38

while holding accuracy near the per-benchmark best, and the deadline-aware safeguard governs deadline-miss behavior rather than throughput, which is why its effect appears in Tab. II instead.

Component attribution

Each mechanism contributes along the axis it was designed for, concurrency for throughput, the UAM-driven controller for the accuracy–throughput balance, and the safeguard for timeliness, and the full *AOCL* aggregates them without inheriting the weakness of any single one.

F. System Overhead Analysis

We profile per-component runtime, memory, and energy on Jetson Orin across five benchmarks (Tab. V). (i) Runtime is dominated by D-Buffer (snapshot + swap), with UAM and the scheduler adding ≤ 20 ms except on CORe50-NIC; since cost scales with the number of adaptation windows rather than model size, the four steady benchmarks stay within 156–215 ms, whereas CORe50-NIC, with many reconfigurations, incurs UAM/Scheduler/D-Buffer of 144/16/1,540 ms. (ii) Energy and memory remain tight, 32.7–55.5J and a flat 1.68–1.72MB peak footprint, driven by a fixed shadow tensor and a small UAM ring buffer. Overall, the runtime layer adds $\leq$ 1.7s per stream and consumes well under 0.1% of the device's 5–15W envelope.

Low overhead

The adaptation layer is cheap enough to be always on, adding negligible energy and a flat, model-independent memory footprint, which is what makes *AOCL* practical to leave running on resource-constrained embedded platforms.

VII. Related Work

Online continual learning (OCL) provides opportunities to update models from streams while mitigating catastrophic forgetting [4, 6, 28, 29]. Algorithmic approaches include rehearsal [4, 30], gradient constraints [5–7], and regularization [31, 32]. Surveys consolidate these streaming algorithms [22, 33, 34]. Our work is orthogonal to algorithmic OCL; we target the *system* dimension, co-scheduling training and inference on-device to boost throughput. *AOCL* integrates transparently with Avalanche [11] for diverse OCL algorithms.

TABLE V: Per-component runtime overhead of *AOCL*.

Benchmark	UAM Calc	Scheduler	D-Buffer	Exec (ms)	Energy (J)	Mem (MB)
SplitCIFAR10	18	2	190	2210	33.15	1.68
SplitCIFAR100	18	2	195	2215	33.23	1.68
CORe50-NI	14	8	156	2178	32.67	1.68
CORe50-NC	17	8	176	2201	33.02	1.68
CORe50-NIC	144	16	1540	3700	55.50	1.72

Despite recent algorithmic progress, system-level optimizations for OCL remain underexplored. Prior work largely targets multi-GPU, offline-centric settings: Ekya [8] adapts offline CL (e.g., iCaRL [30]) for video analytics with heavy profiling; RECL [9] and AdaInf [10] rely on complex model selection or structural changes for multi-model CL on server-class GPUs. LifeLearner [35] optimizes hardware-level meta-CL. These designs diverge from strict embedded constraints. *AOCL* is the first concurrent OCL runtime tailored for embedded platforms, coupling algorithmic efficiency with resource-constrained system practicality.

Real-time GPU scheduling has been widely explored [36–43]. These existing works schedule *static* DNN tasks with known WCETs, which do not directly apply to OCL, where the training task's per-window cost and the model parameters themselves could drift online as new experiences arrive. *AOCL* inherits their non-preemptive single-CUDA-stream execution model and fixed-frequency WCET profiling, but adds a UAM-driven controller and per-window reconfiguration so that schedulability can still be argued in the training-inference mixed regime that those prior systems do not cover.

VIII. Conclusion

We presented *AOCL*, a self-adaptive runtime for concurrent on-device OCL. Through dynamic task alternation and reconfiguration, *AOCL* adapts to changing workloads while preserving accuracy. Case studies in simulation and a real-world robotic prototype demonstrate its effectiveness and ability to sustain responsive learning under tight resource constraints.

Acknowledgment

This research was supported by the National Science Foundation under Grants CNS Career 2230968, CPS 2230969, CNS 2300525, CNS 2343653, CNS 2312397.

References

[1] L. Pellegrini, G. Graffieti, V. Lomonaco, and D. Maltoni, "Latent replay for real-time continual learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 203–10 209.

[2] H. Ye, J. Zhao, Y. Zhan, W. Chen, L. He, and H. Zhang, "Person re-identification for robot person following with online continual learning," *IEEE Robotics and Automation Letters*, 2024.

[3] D. Lee, M. Yoo, W. K. Kim, W. Choi, and H. Woo, "Incremental learning of retrievable skills for efficient continual task adaptation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 17 286–17 312, 2024.

[4] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. Dokania, P. Torr, and M. Ranzato, "Continual learning with tiny episodic memories," in *Workshop on Multi-Task and Lifelong Reinforcement Learning*, 2019.

[5] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Gradient based sample selection for online continual learning," *Advances in neural information processing systems*, vol. 32, 2019.

[6] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," *Advances in neural information processing systems*, vol. 30, 2017.

[7] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-gem," *arXiv preprint arXiv:1812.00420*, 2018.

[8] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 119–135.

[9] M. Khani, G. Ananthanarayanan, K. Hsieh, J. Jiang, R. Netravali, Y. Shu, M. Alizadeh, and V. Bahl, "{RECL}: Responsive {Resource-Efficient} continuous learning for video analytics," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 917–932.

[10] S. S. Shubha and H. Shen, "Adainf: Data drift adaptive scheduling for accurate and slo-guaranteed multiple-model inference serving at edge servers," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 473–485.

[11] A. Carta, L. Pellegrini, A. Cossu, H. Hemati, and V. Lomonaco, "Avalanche: A pytorch library for deep continual learning," *Journal of Machine Learning Research*, vol. 24, no. 363, pp. 1–6, 2023. [Online]. Available: http://jmlr.org/papers/v24/23-0130.html

[12] V. Lomonaco, L. Pellegrini, A. Cossu, A. Carta, G. Graffieti, T. L. Hayes, M. D. Lange, M. Masana, J. Pomponi, G. van de Ven, M. Mundt, Q. She, K. Cooper, J. Forest, E. Belouadah, S. Calderara, G. I. Parisi, F. Cuzzolin, A. Tolias, S. Scardapane, L. Antiga, S. Amhad, A. Popescu, C. Kanan, J. van de Weijer, T. Tuytelaars, D. Bacciu, and D. Maltoni, "Avalanche: an end-to-end library for continual learning," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, ser. 2nd Continual Learning in Computer Vision Workshop, 2021.

[13] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[14] B. Kisačanin, "Deep learning for autonomous vehicles," in *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2017, pp. 142–142.

[15] A. Popov, P. Gebhardt, K. Chen, R. Oldja, H. Lee, S. Murray, R. Bhargava, and N. Smolyanskiy, "Nvradarnet: Real-time radar obstacle and free space detection for autonomous driving," *arXiv preprint arXiv:2209.14499*, 2022.

[16] NVIDIA, "Duckiebot (db-j)," https://get.duckietown.com/products/duckiebot-db21, 2022.

[17] ——, "Sparkfun jetbot ai kit," https://www.sparkfun.com/products/18486, 2022.

[18] ——, "Waveshare jetbot ai kit," https://www.amazon.com/Waveshare-JetBot-AI-Kit-Accessories/dp/B07V8JL4TF/, 2022.

[19] T. Hess, M. Mundt, I. Pliushch, and V. Ramesh, "A procedural world generation framework for systematic evaluation of continual learning," *arXiv preprint arXiv:2106.02585*, 2021.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] X. Nie, Z. Deng, M. He, M. Fan, and Z. Tang, "Online active continual learning for robotic lifelong object recognition," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[22] N. Vödisch, D. Cattaneo, W. Burgard, and A. Valada, "Covio: Online continual learning for visual-inertial odometry," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2464–2473.

[23] E. Hajizada, B. Swaminathan, and Y. Sandamirskaya, "Continual learning for autonomous robots: A prototype-based approach," *arXiv preprint arXiv:2404.00418*, 2024.

[24] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: Generalized pipeline parallelism for DNN training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 1–15.

[25] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 595–610.

[26] M. Han, H. Zhang, R. Chen, and H. Chen, "Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022, pp. 539–558.

[27] NVIDIA, "tegrastats utility," https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Utilities/util_tegrastats.html, 2022.

[28] A. Prabhu, H. A. Al Kader Hammoud, P. K. Dokania, P. H. Torr, S.-N. Lim, B. Ghanem, and A. Bibi, "Computationally budgeted continual learning: What does matter?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 3698–3707.

[29] W. Yu, Z. Wang, S. Niu, S. Lin, and L. Yang, "More than memory savings: Zeroth-order optimization mitigates forgetting in continual learning," in *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2026, Tucson, AZ, USA, March 6-10, 2026*. IEEE, 2026, pp. 4766–4776. [Online]. Available: https://doi.org/10.1109/WACV61042.2026.00463

[30] 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.

[31] 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.

[32] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[33] N. Vödisch, K. Petek, W. Burgard, and A. Valada, "Codeps: Online continual learning for depth estimation and panoptic segmentation," *arXiv preprint arXiv:2303.10147*, 2023.

[34] L. Castri, S. Mghames, and N. Bellotto, "From continual learning to causal discovery in robotics," in *AAAI Bridge Program on Continual Causality*. PMLR, 2023, pp. 85–91.

[35] Y. D. Kwon, J. Chauhan, H. Jia, S. I. Venieris, and C. Mascolo, "Life-learner: Hardware-aware meta continual learning system for embedded computing platforms," *arXiv preprint arXiv:2311.11420*, 2023.

[36] G. A. Elliott, B. C. Ward, and J. H. Anderson, "GPUSync: A framework for real-time GPU management," in *2013 IEEE 34th Real-Time Systems Symposium (RTSS)*. IEEE, 2013, pp. 33–44.

[37] Y. Xiang and H. Kim, "DART: A scalable and adaptive edge stream processing engine for real-time multi-DNN inference," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 392–405.

[38] W. Kang, K. Lee, J. Lee, I. Shin, and H. S. Chwa, "Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 329–341.

[39] Y. Wang, C. Liu, D. Wong, and H. Kim, "Gcaps: Gpu context-aware preemptive priority-based scheduling for real-time tasks," *arXiv preprint arXiv:2406.05221*, 2024.

[40] Z. Li, A. Samanta, Y. Li, A. Soltoggio, H. Kim, and C. Liu, "r^3: On-device real-time deep reinforcement learning for autonomous robotics," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 131–144.

[41] Z. Li, T. Ren, X. He, and C. Liu, "Red: A systematic real-time scheduling approach for robotic environmental dynamics," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 210–223.

[42] Y. Li, Z. Li, W. Yang, and C. Liu, "Rt-lm: Uncertainty-aware resource management for real-time inference of language models," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 158–171.

[43] Y. Li, Z. Li, Y. Zhu, and C. Liu, "Lemix: Unified scheduling for llm training and inference on multi-gpu systems," *arXiv preprint arXiv:2507.21276*, 2025.