UNIVERSITY OF CALIFORNIA
RIVERSIDE

Towards Deployable On-Device Machine Learning Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

Zexin Li

June 2026

Dissertation Committee:
Dr. Cong Liu, Chairperson
Dr. Hyoseung Kim
Dr. Jun Sheng

The Dissertation of Zexin Li is approved:

Committee Chairperson

University of California, Riverside

ACKNOWLEDGMENTS

When I began writing the acknowledgment section of my dissertation, I realized that this marked the closing chapter of a long and transformative journey. Looking back, these years have been filled with challenges, moments of doubt, unexpected breakthroughs, and more learning than I ever anticipated. None of this would have been possible without the support, guidance, and kindness of the people around me. To everyone who has been part of this journey, whether mentioned here or not, please accept my deepest gratitude.

First and foremost, I am deeply grateful to my advisor, Dr. Cong Liu, whose unwavering support, patience, and guidance have shaped not only this dissertation but also my growth as a researcher. From the very beginning, you saw potential in me that I often failed to see in myself. Your advice, sometimes gentle, sometimes direct, continually pushed me to think deeper, question assumptions, and aim for higher standards. I am especially thankful for the countless discussions, late-night message exchanges before deadlines, and the generous time you devoted to improving my work. Your mentorship has profoundly influenced my academic path and will continue to guide me in the years ahead.

I would also like to express my sincere appreciation to the members of my dissertation committee, Dr. Hyoseung Kim and Dr. Jun Sheng. Their insightful comments, thoughtful critiques, and encouragement significantly enhanced the quality of this dissertation. I am grateful not only for their academic guidance but also for their advice on navigating the research community and shaping my professional aspirations.

I am further indebted to my many wonderful collaborators, whose ideas, discussions, and contributions have shaped this dissertation: Dr. Yinglun Zhu, Dr. Yuqun Zhang, Dr.

Andrea Soltoggio, Dr. Lothar Thiele, Dr. Haizhou Li, Dr. Xiaoxi He, Dr. Robby T. Tan, Dr. Guoyuan Wu, Dr. Wei Yang, Dr. Shuhong Ding, Mr. Bangjie Yin, Mr. Taiping Yao, Mr. Jiancheng Zhang, Dr. Junfeng Guo, Mr. Ziliang Zhang, Dr. Yiming Chen, Dr. Yufei Li, Dr. Simin Chen, Dr. Xiaoxue Gao, Dr. Soroush Bateni, Mr. Tao Ren, and Mr. Aritra Samanta. Thank you all for your support, guidance, and contributions to this work.

I am also deeply grateful to my friends both inside and outside academia. Your companionship, humor, and emotional support helped me stay grounded, especially during periods of stress and uncertainty. Thank you for reminding me that life is larger than research and that balance is essential.

To my family, my deepest gratitude. Your unconditional love, trust, and support have carried me through every stage of this journey. Even from afar, you provided the strength I needed to keep going. You celebrated my small victories, encouraged me during setbacks, and gave me the courage to continue pursuing my goals. I hope this milestone makes you proud.

As I close this chapter, I carry with me not only the knowledge gained but also the friendships formed and the lessons learned. Thank you all for being part of this journey.

To my parents for all the support.

ABSTRACT OF THE DISSERTATION

Towards Deployable On-Device Machine Learning Systems

by

Zexin Li

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, June 2026
Dr. Cong Liu, Chairperson

Safety-critical cyber-physical systems (CPS), such as autonomous vehicles and robots, increasingly rely on deep neural networks. While these systems require high prediction accuracy, they must simultaneously satisfy strict end-to-end timing constraints under tight memory and energy budgets. The challenge extends beyond static inference: realistic deployments face non-stationary data and fluctuating power conditions, necessitating online adaptation systems. However, existing machine learning methods largely overlook the timing, memory, and accelerator-contention constraints inherent to embedded CPS, resulting in unpredictable latencies and frequent out-of-memory failures.

This dissertation addresses the gap between algorithmic machine learning and real-time systems by developing a predictable and self-adaptive On-Device machine learning stack. We demonstrate that predictable learning is a distinct systems problem requiring cross-layer coordination rather than isolated algorithmic adjustments.

Specifically, the dissertation develops three tightly coupled thrusts. First, to make modern models *runnable On-Device under stringent memory constraints*, it exploits

multi-input/multi-output structure, weight sharing, and training–inference co-design so that multi-task perception, reinforcement learning, and continual learning workloads fit within small-device memory budgets while remaining schedulable as modular units (MIMONet, MixTraining, R^3, Orion). Second, to *reduce latency and deadline misses in real-time learning pipelines*, it shapes end-to-end deadlines across perception–control DAGs, enforces intermediate timing contracts, and designs runtimes that safely overlap evaluation, inference, and in-the-loop training on shared accelerators (RED, AOCL). Third, to enable *deployable learning ecosystems in uncertain, resource-constrained environments*, it introduces environment- and energy-aware controllers that adapt sampling, buffering, and update cadence online as workloads, data distributions, and power budgets evolve, sustaining timing and accuracy under real-world drift (Genie, EOCL).

Ultimately, this work provides the systems foundation required to move beyond "deploy then freeze" edge AI, enabling continuous, resource-aware, and predictable adaptation for mission-critical robotics.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Safety-critical cyber-physical systems (CPS), including autonomous vehicles, mobile robots, assistive platforms, and immersive extended-reality devices, increasingly rely on deep neural networks (DNNs) for perception, decision making, and closed-loop control [183, 256, 187, 364]. These systems must achieve high prediction accuracy while simultaneously satisfying strict end-to-end timing constraints under tight memory and energy budgets on heterogeneous embedded platforms that combine CPUs, GPUs, and dedicated accelerators [215, 214]. A perception or control decision that arrives late is functionally incorrect, and a model that exceeds the available memory simply cannot run; predictability is therefore as fundamental to mission success as algorithmic accuracy. The challenge is sharpened by the fact that modern deployments are no longer static: data distributions drift as the environment evolves, workload mixes shift as new tasks are added, and energy supplies fluctuate over long missions [160, 52].

Within this setting, three intertwined layers of problems prevent today's edge AI pipelines from scaling to deployable, mission-critical robotics. First, modern multi-task models must *fit On-Device under stringent memory budgets*, including the additional state introduced by training, replay, and optimizer buffers when the same device must also adapt online. Second, the resulting pipelines must *reduce latency and deadline misses in real-time learning*, coordinating perception, control, evaluation, and update stages that share non-preemptive accelerators. Third, these systems must form *deployable On-Device learning ecosystems* that remain robust in uncertain, resource-constrained environments where workload bursts, sensor noise, and energy availability all change online.

A substantial body of prior work informs each of these layers. Model-compression techniques such as variational information bottleneck pruning [214] and cross-model weight sharing [208, 364] have demonstrated significant savings for multi-task DNNs. Real-time scheduling research has produced deadline-driven runtimes and DAG-aware analyses for robotic pipelines [215]. Online continual learning (OCL) and reinforcement-learning algorithms have developed strong tools for adaptation under distribution shift [52, 10, 234, 51, 194, 222, 256, 131], and continual-learning frameworks such as Avalanche [46, 233], replay-based baselines [294, 289], and server-side schedulers such as Ekya [28] and RECL [188] have shown that concurrent training and inference can be coordinated at scale. Edge-side energy-aware approaches based on dynamic voltage and frequency scaling [322] further demonstrate that energy is a first-class resource for long-mission deployments.

These prior efforts, however, fall short of the cross-layer requirements of On-Device learning. Compression and multi-task architectures are usually evaluated for inference only

and assume that timing is somebody else's problem [214]. Real-time schedulers typically treat DNNs as monolithic, fixed-shape black boxes and break down when workload structure, fan-out, or precedence constraints change due to environmental dynamics [215]. Cloud-derived OCL pipelines such as Ekya [28], RECL [188], and AdaInf [324] assume multi-GPU server isolation and coordinate at tens-of-seconds-to-minute granularity, which is incompatible with sub-second embedded inference deadlines and small On-Device memory [202, 294]. RL- and OCL-side methods leave compute, memory, energy, and accelerator-contention constraints to the system layer, with the result that configurations that appear effective on server hardware become unstable on embedded devices, exhibiting long update latencies, severe inference interference, and frequent out-of-memory failures [117, 245]. Finally, energy-oblivious schedulers cannot sustain long-duration autonomy under harvested or battery power [322].

This dissertation addresses these three layers through a coordinated set of models, runtime, and ecosystem contributions. At the first layer, *memory-aware model and systems co-design* attacks the On-Device memory wall by combining multi-input/multi-output network structure, cross-task weight sharing, and training–inference reorganization so that multi-task perception, deep reinforcement learning, and online continual learning workloads fit within small-device memory budgets while remaining modular and schedulable. At the second layer, *real-time scheduling and runtimes for On-Device learning* treat neural networks as first-class scheduling entities: end-to-end deadlines are shaped across asynchronous perception–control DAGs, intermediate timing contracts are enforced, and runtime controllers safely overlap evaluation, inference, and in-the-loop training on shared, non-preemptive embedded

accelerators. At the third layer, *deployable On-Device learning in the wild* closes the loop with the environment: distributed edge caching exploits spatial locality across connected fleets, and energy-aware online controllers jointly tune DVFS and learning hyperparameters under non-stationary workloads, sustaining timing and accuracy under data and power drift. Together, these layers form a unified, pipeline- and learning-centric co-design stack for predictable On-Device learning.

1.1 Thesis Statement

The thesis statement of this dissertation is as follows: predictable, energy-aware, and self-adaptive On-Device learning for safety-critical cyber-physical systems is a distinct systems problem that requires cross-layer coordination among model structure, runtime scheduling, and resource management. By co-designing multi-task model architectures with memory- and deadline-aware schedulers and by closing the loop with environment- and energy-aware controllers, On-Device learning systems can sustain real-time inference and online adaptation within tight memory and energy budgets across diverse embedded platforms and changing environments.

This dissertation makes the following contributions:

- **Fitting modern models On-Device under stringent memory budgets.** We develop a family of memory-efficient model and training designs that make modern multi-task perception, deep reinforcement learning, and online continual learning workloads runnable on small embedded platforms. MIMONet introduces a multi-input multi-output DNN framework that combines variational information-bottleneck

pruning for residual networks with cross-model weight sharing, reducing runtime memory by up to 80.7% on Jetson-class devices [214]; MixTraining consolidates self-supervised and supervised passes into a unified pipeline that Pareto-dominates SSL+SL in accuracy and training compute [219]; R^3 co-optimizes batch size and replay-buffer memory to deliver timing-predictable On-Device deep reinforcement learning across Classic Control, Atari, and DonkeyCar workloads [217]; and Orion provides a self-adaptive memory-management ecosystem that uses a unified health-score indicator to autonomously rebalance batch size, replay-buffer memory, and optimization plugins, eliminating out-of-memory failures while preserving plasticity and stability for On-Device OCL.

- **Reducing latency and deadline misses in real-time learning.** We design runtime systems that reduce latency and deadline misses for learning-enabled perception–control pipelines on shared, non-preemptive embedded accelerators. RED is the first real-time scheduler that supports weight-shared multi-task DNNs, intermediate-deadline assignment, and on-demand synchronization across asynchronous perception–control DAGs, reducing deadline-miss rates by 40.5% on average over EDF [215]. AOCL is an OCL runtime that drives a finite-state, sign-based controller from a unified adaptation metric to safely overlap training, evaluation, and inference on a single embedded GPU, with analytical guarantees on settling time, feasibility, and transient deadline-miss bounds.

- **Deployable On-Device learning ecosystems in uncertain, resource-constrained environments.** We build deployment-time ecosystems that keep On-Device learning

reliable under environmental drift and fluctuating power budgets. Genie is a distributed ROS-based caching layer that maintains a 3D object map across edge servers, cutting perception latency by up to 95% for Autoware-based connected autonomous robots [187]. EOCL is an energy-aware model-based optimization runtime with a recency-aware Gaussian-process surrogate that jointly tunes CPU/GPU DVFS and batch size online, reducing energy by up to 70% versus state-of-the-art OCL schedulers under sudden environmental change while maintaining streaming accuracy.

1.2 Fitting Modern Models On-Device Under Stringent Memory Budgets

The first thrust of this dissertation focuses on making modern multi-task and learning-enabled models runnable On-Device under stringent memory budgets, both for inference and, increasingly, for On-Device updates. Embedded platforms such as the NVIDIA Jetson Nano, AGX Xavier, and AGX Orin have only a few gigabytes of shared CPU–GPU memory [215, 214], yet a single multi-task perception model, together with replay buffers, optimizer state, and intermediate activations introduced by training, can easily exceed these budgets. Memory pressure manifests not only as out-of-memory failures, but also as latency jitter and reduced schedulability when activations and gradients evict caches and contend with inference traffic.

MIMONet (Ch. 2) attacks this problem at the model level by restructuring multi-task workloads into a single weight-shared network with multiple inputs and outputs. It extends the variational information bottleneck to residual architectures to reduce intra-model

redundancy, then adapts cross-model weight zipping to remove inter-model redundancy across branches that operate on heterogeneous input modalities. The resulting models are not only smaller, but also expose a partitionable structure that downstream schedulers can exploit. MixTraining (Ch. 3) addresses the same memory and compute pressure on the *training* side by inserting a unified mixtraining phase between self-supervised and supervised learning, consolidating what would otherwise be separate forward and backward passes through a shared backbone and smoothing the optimization transition between objectives [219].

R^3 (Ch. 4) and Orion (Ch. 5) extend the memory-aware perspective to online learning workloads where memory pressure is jointly driven by batch size, replay buffers, optimizer state, and the inference path. R^3 introduces a deadline-driven feedback loop that selects batch size dynamically, paired with task-specific memory reductions that enable larger replay buffers without out-of-memory failures, enabling On-Device deep reinforcement learning across Classic Control, Atari, and a DonkeyCar simulator [217]. Orion generalizes this insight to online continual learning: a unified runtime health score reallocates memory across batch execution, replay storage, and optimization plugins in response to time-varying pressure, autonomously balancing training latency, plasticity, and stability under fixed device memory budgets.

1.3 Reducing Latency and Deadline Misses in Real-Time Learning

The second thrust of this dissertation develops the runtime layer required to turn memory-efficient learning models into deployable real-time systems. Embedded robotic

pipelines are inherently DAG-structured: perception, control, and learning stages depend on one another, and end-to-end latency must satisfy hard deadlines despite environment-induced changes in workload structure [215]. At the same time, the underlying accelerators are non-preemptive and shared by every stage, so co-executing training and inference without coordination quickly translates into deadline misses, accelerator contention, and unstable adaptation [28, 188, 324].

RED (Ch. 6) provides a systematic real-time scheduling framework that handles dynamically changing workload structures, supports weight-shared multi-task DNNs such as MIMONet, and absorbs asynchronous inference patterns through an intermediate-deadline assignment policy and on-demand synchronization. On four Jetson platforms RED reduces deadline-miss rates by 40.5% and response times by 24.7% on average over EDF, while integrating seamlessly with ROS 2 and existing NVIDIA IoT stacks [215, 187].

AOCL (Ch. 7) extends real-time scheduling to single-embedded-GPU On-Device OCL, where training and inference must share one non-preemptive accelerator. A Unified Adaptation Metric (UAM) drives a finite-state, sign-based bang-bang controller that safely overlaps evaluation and update, while a dynamic adaptive scheduler tunes alternation policy, alternation interval, and batch size at iteration boundaries. AOCL is implemented as a non-intrusive plugin on top of Avalanche [46, 233] and is accompanied by analytical guarantees on per-axis settling time, steady-state inference feasibility, and a transient deadline-miss bound during the controller's safety fallback, providing predictability properties needed for safety-critical deployment.

1.4 Deployable On-Device Learning Ecosystems in Uncertain, Resource-Constrained Environments

The third thrust addresses what is required to turn predictable On-Device learning into a deployable ecosystem for autonomous robots operating in uncertain, resource-constrained environments. Two complementary dimensions emerge: how to exploit the surrounding infrastructure to alleviate On-Device pressure, and how to remain robust to environmental and energy variability that no static configuration can anticipate.

Genie (Ch. 8) takes a fleet-level view of this problem. In collaboration with Fujitsu, we develop a distributed ROS-based caching layer that intercepts ROS communications transparently, builds a shared 3D object map of an edge server's surroundings, and allows neighboring Genies to enrich one another's caches. Deployed on Jetson TX2 and AGX Xavier nodes with Autoware [187], Genie reduces vision-detection latency by 82% on average (up to 95%), with 31% reusable objects on average and up to 67% under multi-vehicle collaboration. This shows that locality- and structure-aware caching at the edge is a practical way to absorb the latency and SWaP constraints of mission-critical autonomous platforms [246].

EOCL (Ch. 9) closes the loop by making On-Device OCL itself energy- and environment-aware. Embedded SoCs expose a high-dimensional reconfiguration space spanning CPU/GPU dynamic voltage and frequency scaling and learning hyperparameters such as batch size; the energy-optimal configuration is batch-size-dependent and shifts whenever the environment changes, leaving static or energy-oblivious schedulers [46, 28, 188] with substantial untapped headroom. EOCL formalizes deployment as a constrained online optimization problem and introduces a recency-aware Gaussian-process surrogate with

Expected Improvement acquisition that selectively forgets stale observations after drift. Integrated with Avalanche, EOCL cuts energy by up to 26% versus the strongest online schedulers and by up to 70% versus RECL [188], while keeping streaming accuracy within roughly two percentage points and sustaining the lowest control-error rate under sudden environmental change.

1.5 Dissertation Organization

The rest of this dissertation is organized as follows:

- Chapter 2 presents **MIMONet**, a multi-input multi-output On-Device deep learning framework that combines an extended variational information bottleneck for residual networks with cross-model weight zipping to deliver memory-, latency-, and energy-efficient multi-task perception on embedded platforms.

- Chapter 3 presents **MixTraining**, a training framework that interposes a unified mixtraining phase between self-supervised and supervised learning, consolidating shared computation and smoothing the objective transition to yield Pareto improvements in accuracy and training time.

- Chapter 4 presents $\mathbf{R^3}$, an On-Device real-time deep reinforcement learning system that co-optimizes batch size, replay-buffer memory, and runtime coordination to deliver timing-predictable On-Device DRL training across robotic benchmarks.

- Chapter 5 presents **Orion**, an adaptive memory-management ecosystem for On-Device online continual learning that uses a unified health-score indicator to autonomously

balance training latency, plasticity, and stability under stringent device memory budgets.

- Chapter 6 presents **RED**, a systematic real-time scheduling framework for weight-shared multi-task DNN workloads under robotic environmental dynamics, with intermediate-deadline assignment and on-demand synchronization for asynchronous inference DAGs.

- Chapter 7 presents **AOCL**, a real-time continual-learning runtime that safely overlaps training and inference on a single non-preemptive embedded GPU via a Unified Adaptation Metric and a finite-state controller with analytical timing guarantees.

- Chapter 8 presents **Genie**, a distributed ROS-based caching layer for connected autonomous robots that builds a shared 3D object map across edge servers and reduces perception latency under SWaP-constrained deployment.

- Chapter 9 presents **EOCL**, an energy-aware On-Device OCL runtime that uses a recency-aware model-based optimizer over CPU/GPU DVFS and batch size to minimize energy under latency and accuracy constraints across non-stationary environments.

- Chapter 10 concludes the dissertation and outlines directions for predictable, energy-aware, and self-adaptive On-Device learning beyond "deploy then freeze" edge AI.

Part I

Fitting Modern Models On-Device Under Stringent Memory Budgets

Chapter 2

MIMONet: Multi-Input Multi-Output On-Device Deep Learning

2.1 INTRODUCTION

Single-input single-output (SISO) deep neural networks (DNNs) have demonstrated impressive performance in various robotics applications [81, 251, 372, 336]. However, multi-input single-output (MISO) DNNs have emerged as a promising alternative, as they have been shown to surpass SISO DNNs both theoretically and empirically [352, 304, 166]. Developing MISO DNNs is a crucial step towards creating multi-input multi-output (MIMO) DNNs for intelligent embedded systems, especially in the field of robotics.

According to Stanford's "Artificial Intelligence and Life in 2030" report [335], AI is expected to impact various fields, including home services [379], healthcare [239, 238, 237, 236], and transportation [240, 241, 69, 402]. To build such robots that thrive in these areas that inherently require MIMO systems, they must process real-time inputs like spoken language and facial expressions, providing personalized feedback based on factors like gender and emotion [334]. Interactive companion robots, like the Vector Robot [83], are expected to receive camera and microphone inputs and use this information to predict user preferences and respond appropriately. In On-Device learning scenarios with strict hardware constraints and performance requirements, deploying MIMO DNNs can be advantageous. By reducing the need to deploy MISO DNN instances on the device, MIMO DNNs can significantly lower resource demands. This paper presents MIMONet, a novel On-Device MIMO DNN framework that achieves both high accuracy and On-Device efficiency in terms of critical performance metrics, including latency, energy, and memory usage. The MIMO approach inherently offers speed improvements by requiring only one forward pass for multiple tasks (instead of multiple passes), reducing computational redundancy.

We first constructed a baseline MIMO DNN model and observed that it requires high memory consumption and struggles to meet real-time constraints due to computational demand. To minimize resource demands, MIMONet builds on an existing DNN model compression technique, VIB [74], which is effective for SISO DNN models. However, extending VIB to MIMO DNN models posed two new challenges. First, VIB exclusively supports the compression of feed-forward networks [23], which makes it unsuitable for more advanced networks, such as ResNet [145]. To address this limitation, MIMONet extends and develops a

modified compression method for ResNet with residual blocks. Second, VIB was not designed to handle MIMO scenarios and did not account for the unique characteristics of MIMO models. VIB focuses on reducing intra-model redundancy but is unable to address common inter-model redundancy in MIMO models. To address this issue, MIMONet develops a new deep-compression method that reduces both inter- and intra-model redundancy, resulting in deeper lossless compression for MIMO scenarios.

We conducted a two-fold evaluation of MIMONet. First, we compared its accuracy against state-of-the-art SISO, MISO, and MIMO models using the RAVDESS dataset [230], showing that MIMONet outperforms them in most scenarios. In the second set, we conducted a comprehensive evaluation of MIMONet's components' impacts on On-Device efficiency, including memory, latency, and energy. We deployed MIMONet on three widely used embedded system platforms: NVIDIA Jetson Nano Orin, NVIDIA AGX Xavier, and NVIDIA AGX Orin, which are most recently used platforms for various robotics applications [81, 251, 372, 126] such as Duckiebot [269], SparkFun Jetbot [271], and Waveshare Jetbot [273]. In addition, we evaluated MIMONet on a PC machine for a more comprehensive assessment. In summary, experimental results demonstrate that MIMONet can achieve:

- **Reduced Memory Usage:** MIMONet significantly reduces runtime memory usage by 80.7% compared to the baseline MISO models.
- **Improved Inference Speed:** MIMONet exhibits speed-ups of 1.98x, 2.29x, and 1.23x compared to the baseline MISO model when tested on Nano, AGX, and Orin, respectively.

- **Enhanced Energy Efficiency:** The energy savings offered by MIMONet are 2.01x, 8.64x, and 2.71x compared to the baseline MISO model when tested on Nano, AGX, and Orin, respectively.

The major contributions of this paper can be summarized as follows:

- **Innovative MIMO Framework for Robotics:** We introduce MIMONet, one of the first frameworks to utilize a unified neural network for processing multiple inputs and outputs simultaneously, addressing complex inference tasks in robotics.

- **New Deep-Compression Method:** MIMONet implements a novel deep-compression method that enhances accuracy and On-Device efficiency, specifically tailored for MIMO applications in robotics.

- **Extensive Experimental Results:** Our experiments on three embedded platforms and a PC demonstrate MIMONet 's high accuracy and superior On-Device efficiency in terms of latency, energy, and memory usage, making it suitable for robots under stringent hardware constraints.

- **Real-World Case Study Validation:** We conducted a realistic case study using the TurtleBot3 robot, showing MIMONet 's practical applicability and superiority over state-of-the-art SISO and MISO models in face and audio recognition within cluttered environments. This further validates its effectiveness for interactive robotic systems.

2.2 BACKGROUND and RELATED WORK

2.2.1 MIMO Architecture

Classical deep neural networks typically follow a SISO format, designed to process one input and produce one output at a time. However, as deep learning has evolved, certain networks have been adapted to handle multiple inputs from varied domains, enhancing accuracy. This approach, often referred to as multimodal learning [392] or multi-view learning [403], leads to what is termed MISO networks [352, 304, 166, 152]. Building on this, models with advanced accuracies have progressed to MIMO configurations [141], which are capable of generating several outputs in one go. This MIMO technology is particularly beneficial in fields like autonomous driving [228, 168] and robotic navigation [49], although it does come with substantial demands on memory and computational resources. Few concurrent works, like BEVFusion [228], try to alleviate these demands by optimizing data processing from sensors to mitigate performance issues. Different from these works, our study aims to tackle these challenges by focusing on model compression to facilitate practical deployment in real-world scenarios.

2.2.2 Model Compression for On-Device Scenarios

Modern DNNs, like the ViT-Huge [360] and GPT-3 [40], although achieving state-of-the-art accuracy, strain memory and computation resources for resource-constrained devices during inferences. A practical solution to mitigate such an issue is model compression, which aims to reduce model parameter count without affecting accuracy. They can involve removing redundant parameters [74, 156, 91, 140, 227] or minimizing redundant precision

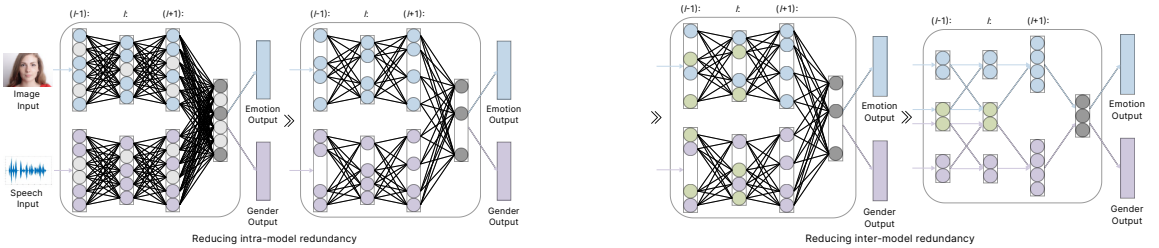


Figure 2.1: Overview of MIMONet. In the left part, gray circles represent neurons inducing intra-model redundancy. In the right part, green circles denote sharable neurons inducing inter-model redundancy. Best viewed in color.

[70]. Two subclasses of model compression for multi-branch models consist of single-model pruning and cross-model compression. The former compresses a single network by eliminating unnecessary operations and parameters, while the latter minimizes inter-model redundancy by identifying and removing shared parameters and operations across models [74, 156, 91, 140, 257, 80, 153, 67]. Notably, the Multi-Task Zipping (MTZ) approach has pioneered cross-model compression by automating the merging of pre-trained DNNs [156]. However, MIMONet leverages the Variational Information Bottleneck (VIB) method to reduce intra-model redundancy, extending it for application to the ResNet architecture. Furthermore, it adapts modified MTZ to reduce inter-model redundancy, which further decrease the total parameters in the proposed MIMO model.

2.3 METHODOLOGY

2.3.1 Overview of MIMONet

Fig. 2.1 illustrates an overview of the On-Device MIMO inference framework. First, inspired by the famous multi-branch architecture ResNext [368], we adopt a MIMO deep neural network with multiple inputs backbone and multiple outputs classifiers as our baseline

model (as shown in the first block). One forward prediction of this network requires multiple inputs (image and speech) and generates multiple prediction outputs (emotion and gender) simultaneously. To align and fuse feature information from the input image and speech, we use ResNet as the backbone to extract features from both modalities. The feature maps from both are then concatenated to create a unified representation. This fused feature map serves as the input for the subsequent prediction layers.

However, directly deploying such baseline MIMO models to embedded devices remains challenging. Specifically, embedded devices including most robots have *limited memory* and require *real-time performance* for many real-world scenarios. For instance, handling streaming data (e.g. high-speed cameras sampling 60 times per second) and ensuring robustness in safe-critical scenarios (e.g., autonomous driving robots [269, 407, 196, 329, 330]). In addition, *energy savings* are particularly needed for embedded devices because of limited battery capacity and heat dissipation.

To address these challenges, employing model compression is an intuitive and effective direction. First, we adopt the idea of VIB [74] to perform channel-wise hard masking (the masked channel generates zero output, which can be pruned, while the unmasked channel's output remains the same) and follow pruning. Then, the pruned MIMO model can maintain the original structure in the forward direction. However, VIB is designed for the SISO model and thus reduces the intra-model redundancy. However, it does not consider the characteristics of the MIMO framework, for instance, the MIMO framework is naturally multi-branching and exhibits inter-model redundancy. Therefore, aiming to reduce such redundancy intuitively can boost model compression performance. Furthermore,

inspired by the idea from one cross-model compression work MTZ [156], we perform weights merging between multiple independent branches to improve model compression efficacy further. In summary, in MIMONet we focus on both intra-model redundancy and inter-model redundancy, and thus can theoretically further improve the compression effectiveness and achieve efficient On-Device deployment.

2.3.2 Reduce Intra-model Redundancy

We aim to reduce intra-model redundancy by performing single-model compression. Specifically, we adopt and extend the idea from VIB [74] that reduces the redundant mutual information between adjacent layers within the model. The vanilla VIB only applies to neural networks such as VGG [325]. However, when dealing with neural networks that possess a non-linear architecture, such as ResNet [145], which forms the foundation of our MIMO model, the direct application of the conventional VIB approach is not feasible.

To address this, the design incorporates an "information bottleneck", a learnable mask, into each layer of the network that possesses learnable parameters (e.g., convolutional layers). This bottleneck selectively filters information flow between layers, allowing only the most relevant features to pass through, thus reducing redundancy. As shown in the left part of Fig. 2.1, neurons with gray color are masked. Furthermore, combining the characteristics of ResNet, we designed the following adaption scheme, detailed in Fig. 2.2: (1) Module-level design: we add the information bottleneck layer as in the original VIB design for the layers not in the residual block. However, we keep the input/output channel number unchanged for each residual block, i.e., we do not insert an information bottleneck layer between two residual blocks. (2) Block-level design: After the second information bottleneck layer, we

insert a *channel-recover* operation. This operation restores the channel to the pre-pruning mask based on the pruning mask by filling pruned channels with zeros, which ensures that the input channel is matched during the concatenation operation between the main branch and the bypass. Inspired by the design idea of ResNet [145], we intuitively assume the major information throughout the network is concentrated in the bypass (consisting of direct input or input after downsampling). Therefore, we do not set the mask in the bypass section.

The surprising result of this design is that, according to top-5 layer-wise compression statistics upon application to the baseline MIMO model, the image branch is (100.0%, 98.4%, 97.9%, 96.1%, 87.1%), and the speech branch is (100.0%, 100.0%, 99.4%, 92.2%, 86.7%). The information bottleneck within the residual block naturally led to all output channels except the bypass from preceding layers being masked when a high compression rate was achieved, suggesting that the primary information indeed flows through the bypass. This outcome serves as empirical validation for the assumption underlying the adapted VIB approach for ResNet. The effectiveness of this method is further supported by the observed layer-wise compression rates, indicating the potential for significant model size reduction without compromising the essential information flow through the network.

For illustration, we show an example of a basic block of ResNet in Fig. 2.2. Surprisingly, as shown in Tab. 2.1, we find that with our design, certain information bottlenecks within the residual block even naturally masked all the output channels (100.0% layer-wise compression rate) from the previous layers when a high compression rate was achieved. This finding verifies our assumption that the main information of ResNet is concentrated in bypass, thus validating the effectiveness of our method.

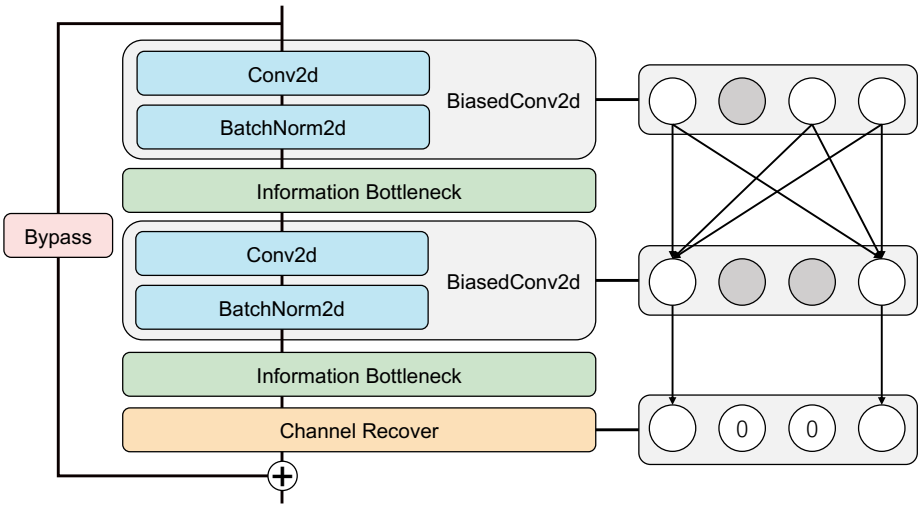


Figure 2.2: Design for compression of the residual block for ResNet [145]. The left side shows the structure of the residual block. The right side shows channel-level pruning and recovery. White and gray circles exhibit kept pruned channels. The pruned channels are filled with zeros before the feature map summing. Best viewed in color.

Furthermore, we integrate performance optimization in the residual block to collapse each convolutional (Conv2d) layer and the subsequent batch normalization (BatchNorm2d) layer into a biased convolutional (BiasedConv2d) layer, as shown in the blue box in Fig. 2.2. Since the On-Device deployed models mainly only require forward inference, i.e., all parameters are fixed. Therefore, we could replace the BatchNormalization (BN) layer [170] with multiplication and addition. Specifically, the output of the BN layer applied on the i-th channel of layer l is:

$$BN(y_{l,i}) = \gamma_{l,i} \cdot \frac{y_{l,i} - \mu_{l,i}}{\sqrt{\sigma_{l,i}^2 + \epsilon}} + \beta_{l,i} \tag{2.1}$$

where $y_{l,i}$ is the pre-activation output of the convolution layer, $\gamma_{l,i}$ and $\beta_{l,i}$ are the two learnable parameters (scaling and shifting) for the BN layer, $\mu_{l,i}$ and $\sigma_{l,i}$ are the pre-calculated mean and standard deviation. The effect of the BN layer can be replaced by multiplying the incoming weight of convolutional layer $\mathbf{w}_{l,i}$ by a scalar $\frac{\gamma_{l,i}}{\sigma_{l,i}}$ and adding $\beta_{l,i} - \frac{\gamma_{l,i} \cdot \mu_{l,i}}{\sigma_{l,i}}$ to the bias $b_{l,i}$.

Table 2.1: An example of top-5 layer-wise compression statistics upon applying to baseline MIMO model.

Compression Rate Index	0	1	2	3	4
Image Compression Rate	100.0	98.4	97.9	96.1	87.1
Speech Compression Rate	100.0	100.0	99.4	92.2	86.7

2.3.3 Reduce Inter-model Redundancy

After reducing intra-model redundancy, we explore the possibility of decreasing the total number of parameters by reducing the inter-model redundancy through cross-model compression techniques. In a MIMO model, it is not obvious that sharable knowledge exists between the network's branched parts, as they are processing inputs from an entirely different domain, e.g., one for audio and the other for video input. However, as shown in [154], sharable knowledge still exists within models designed for different input domains, as long as their outputs are fused and serve the same output tasks. Specifically, we have observed a significant amount of sharable knowledge can be found within the later layers, as features are becoming more abstract at this stage, and these abstract features are used for the same tasks.

To this end, we extend the MTZ [156, 154] framework to merge model weights from multiple independent branches. MTZ is a framework that automatically and adaptively merges deep neural networks for cross-model compression via neuron sharing. t the core of MTZ is the neural similarity metric, derived from a second-order analysis of the model

parameters. It measures the similarity in function between two neurons' incoming weights, which is given by:

$$d[\tilde{\mathbf{w}}_{l,i}^{A}, \tilde{\mathbf{w}}_{l,j}^{B}] = \frac{1}{2}(\tilde{\mathbf{w}}_{l,i}^{A} - \tilde{\mathbf{w}}_{l,j}^{B})^{\top} \cdot \left((\tilde{\mathbf{H}}_{l,i}^{A})^{-1} + (\tilde{\mathbf{H}}_{l,j}^{B})^{-1}\right)^{-1} \cdot (\tilde{\mathbf{w}}_{l,i}^{A} - \tilde{\mathbf{w}}_{l,j}^{B}) \tag{2.2}$$

where $\tilde{\mathbf{w}}_{l,i}^{A}, \tilde{\mathbf{w}}_{l,j}^{B} \in \mathbb{R}^{\tilde{N}_{l-1}}$ are the incoming weights, and $\tilde{\mathbf{H}}_{l,i}^{A}, \tilde{\mathbf{H}}_{l,j}^{B}$ are the associated layer-wise Hessians [156, 154].

Motivated by this principle, we treat each branch as independent computing subgraphs in our MIMO model. As shown in the right part of Fig. 2.1, neurons with green color are shared between different branches. The branches are aligned from the very first layer such that they can be merged and optimized by MTZ layer after layer. This design can also deal with branched layers with different widths, as long as they share the same underlying structure, such as convolutional layers or residual blocks. As proved by experiments, reducing inter-model redundancy effectively reduces the baseline MIMO model size for more efficient On-Device deployment.

2.3.4 Integrate with Quantization Techniques

To further enhance the On-Device deployment efficiency of our MIMO inference framework, integrating quantization techniques presents a promising avenue. Quantization involves converting a model's floating-point weights and activations to lower precision formats, such as fixed-point integers, which significantly reduces the model size. This process is crucial for embedded devices with stringent memory and computational resource constraints.

Our proposed method, which already incorporates strategies for intra-model and inter-model redundancy reduction, can naturally extend to include quantization. In the quantization process, the weights and activations of the pruned and merged MIMO model are mapped from floating-point to fixed-point representation. This mapping can be achieved through various quantization schemes, such as uniform or non-uniform quantization, with or without re-training (quantization-aware training or post-training quantization). For instance, post-training quantization could be directly applied to the compressed model to reduce its precision without the need for further training quickly. Specifically, for a parameter w, the quantization to 8-bit (int8) and 4-bit (int4) integers can be expressed as:

$$w_{\text{int8}} = \text{round}\left(\frac{w - \min(W)}{\max(W) - \min(W)} \times 255\right) - 128$$

$$w_{\text{int4}} = \text{round}\left(\frac{w - \min(W)}{\max(W) - \min(W)} \times 15\right) - 8$$

, where W represents the set of all parameters in the model that are subject to quantization, $\min(W)$ and $\max(W)$ denote the minimum and maximum model parameters, respectively. This process introduces a trade-off between efficiency and accuracy. The integration with quantization techniques not only further reduces the memory footprint and computational requirements of the MIMO model but also leverages the hardware accelerators often found in embedded devices, which are optimized for low-precision arithmetic operations. Consequently, this integration can lead to significant improvements in inference speed and energy efficiency,

Figure 2.3: Data examples of RAVDESS dataset [230].

making our MIMO inference framework even more suitable for real-time applications on resource-constrained devices.

2.4 EVALUATION

2.4.1 Experimental Setup

We set up four baselines for comparison with MIMONet: two SISO models and two MISO models, both using ResNet-18 as the backbone. In MISO models, ResNet-18 extracts features from each input (image and audio), which are then concatenated and passed through three fully connected layers for prediction. MIMONet differs by having multiple predictors after the fully connected layers, enabling it to generate multiple predictions in a single forward pass. We also apply approximations like half-precision (FP16) and post-training quantization (INT8 and INT4).

We leverage the RAVDESS dataset [230], with 7356 multimodal clips from 24 actors, to evaluate our method's performance in multi-input scenarios, specifically for concurrent emotion and gender predictions. Some examples are shown in Fig. 2.3. We use 80% of the data (19 actors) for training and 20% (the other 5 actors) for testing. All experiments are run three times, with averaged results reported. Such tasks are pivotal for developing robots designed for human-robot interactions, applicable in healthcare for patient monitoring by

Table 2.2: Experimental results of MIMONet. The best results are highlighted in bold. The second-best results are underlined.

Method	**Acc./emo**	**Acc./gen**	**Par.**($\times 10^6$)	**Mem.**(MB)	**Latency**(ms)			**Energy**(mJ)		
					Nano	**AGX**	**Orin**	**Nano**	**AGX**	**Orin**
SISO/img-emo	62.45	N/A	11.82	1003.49	6.24	7.46	2.42	49.30	298.40	134.56
SISO/img-gen	N/A	77.40	11.82	1000.12	6.04	7.50	2.38	47.60	298.02	131.76
SISO/aud-emo	61.81	N/A	11.82	983.49	3.04	4.30	1.92	24.20	171.90	71.62
SISO/aud-gen	N/A	**88.24**	11.82	956.92	2.98	4.16	1.88	23.62	166.42	70.12
MISO/emo	73.04	N/A	25.51	1362.10	10.08	8.30	3.36	80.24	348.60	161.28
MISO/gen	N/A	77.43	25.51	1342.87	9.78	8.22	3.30	77.76	343.58	158.60
MIMO	74.26	83.99	25.51	1364.96	5.26	4.15	2.95	39.98	175.29	81.90
MIMONet (FP32)	76.28	83.17	0.92	910.11	5.21	3.96	2.84	40.88	45.93	66.17
MIMONet (FP16)	76.29	83.16	0.92	688.15	5.15	3.72	2.81	40.42	43.15	61.54
MIMONet (INT8)	**76.40**	84.48	0.92	577.17	5.11	3.66	2.74	39.96	41.34	59.98
MIMONet (INT4)	75.72	87.01	**0.92**	**521.68**	**5.02**	**3.61**	**2.70**	**39.26**	**40.07**	**59.01**

interpreting emotions, in customer service for personalized assistance, and in education to tailor teaching strategies based on emotional and demographic insights.

2.4.2 Implementation Details

For the SISO, MISO, and MIMO models without pruning, we set the learning rate to 1e-4 for the Adam optimizer [192] and train 2000 epochs to ensure training stability and convergence. All models in this paper are trained from scratch without using extra data for fair comparisons. We adjust the learning hyperparameters of VIB to balance weighted loss values into the same order of magnitude, specifically, we select the Adam optimizer with a weight decay rate of 5e-5 and set the learning rate to 1e-3, IB learning rate to 1e-3, and KL parameter to 1e-7. We set merge degree to 0.9 as default in [156]. All of our implementations are based on native PyTorch [287]. Quantization is implemented by referring to native PyTorch dynamic post-training quantization. System performance of NVIDIA devices is measured by tegrastats profiler [272]. We train all DNNs on a server with an Intel E5-2650 CPU and a GeForce RTX 2080 Ti GPU. Inference experiments are conducted on embedded

devices based on NVIDIA Jetson Developer Kits (Nano, AGX Xavier, and AGX Orin) [1] , which are widely used as the mainboard of various commercial robots, e.g., Duckiebot [269], SparkFun Jetbot [271], Waveshare Jetbot [273].

2.4.3 Metrics

We study the following five metrics to evaluate the accuracy and On-Device efficiency of MIMONet. 1) Accuracy: the ratio of correctly identified samples in the test set to the total number of samples in the test set. Note that the accuracies of the two tasks (emotion and gender) are evaluated separately. 2) Parameter number: number of learnable parameters. 3) Memory: required runtime memory for models. 4) Latency: average wall-clock time required to perform one forward prediction. 5) Energy: We measure system-wide energy consumption. For a fair comparison, we calculate the average latency and energy consumption for 10,000 forward passes. We aim to improve On-Device efficiency (reduce parameter number, latency, memory, and energy) as much as possible while maintaining the model's effectiveness (accuracy).

2.4.4 Effectiveness

Table 2.2 provides a comparative analysis of the MIMONet's performance against its SISO and MISO counterparts on the RAVDESS dataset. Observably, MIMONet and its variants with different precision (FP32, FP16, INT8, and INT4) consistently outperform SISO and MISO configurations across several metrics:

[1]NVIDIA embedded devices equip an integrated GPU with different data processing costs from an independent GPU on an x86 architecture PC or server. To ensure fair comparisons, all inferences experiments are run on an integrated GPU with maximal power mode.

Table 2.3: Ablation study results of accuracy and On-Device efficiency. The best results are in bold.

Method	**Acc./emo**	**Acc./gen**	**Par.**($\times 10^6$)	**Mem.**(MB)	**Latency**(ms)			**Energy**(mJ)		
					Nano	**AGX**	**Orin**	**Nano**	**AGX**	**Orin**
MIMONet	**76.40**	**84.48**	**0.92**	**577.17**	**5.11**	**3.66**	**2.74**	**39.96**	**41.34**	**59.98**
w/o VIB	74.26	83.99	25.51	1364.96	5.26	4.15	2.95	39.98	175.29	81.90
w/o MTZ	75.63	82.70	1.07	930.09	5.15	3.98	2.80	40.28	44.23	62.34
w/o PTQ	76.28	83.17	0.92	910.11	5.32	3.96	2.84	40.88	45.93	66.17
MIMONet	**76.40**	**84.48**	**0.92**	**577.17**	**5.11**	**3.66**	**2.74**	**39.96**	**41.34**	**59.98**
w/ RandPruning	12.66	51.22	0.92	579.22	5.15	3.71	2.77	40.43	42.23	60.12

- **Accuracy (Acc.:** MIMONet models demonstrate superior performance, with the INT8 variant achieving the highest emotion recognition accuracy (Acc./emo) at 76.40%, surpassing SISO and MISO models by 3.36% and a relative gain of 4.6%. The best gender recognition accuracy (Acc./gen) is 87.01%, competitive with state-of-the-art results. Although SISO models excel in gender recognition using vocal features alone, adding image inputs can introduce variability, as facial cues are not always distinct. Nevertheless, MIMONet maintains excellent recognition ability even after compression due to ResNet's efficient feature extraction and fusion of image and audio inputs. Compression techniques like INT8 further enhance efficiency by reducing resource requirements while preserving high accuracy, showcasing MIMONet 's adaptability in processing multi-modal data.

- **Parameter number (Par.):** MIMONet largely reduce parameter number via reducing intra- and inter-task redundancies. It can reach a 96.4% compression rate while achieving very competitive accuracy performance.

- **Memory Usage (Mem.):** MIMONet demonstrates significant memory efficiency for memory-constrained robotic scenarios. For instance, while MISO/emo and MISO/gen configurations require about 1362.10 and 1342.87 MB of memory respectively, the MIMONet variants operate with markedly less, with the MIMO (INT4) requiring only 521.68 MB.

Note that MIMO can operate both tasks simultaneously, unlike the two separate MISO models, which in total consume 2704.97 MB[2], representing approximately up to **80.7%** reductions[3] in memory requirements compared to MISO models.

- **Latency (Latency/FP):** MIMONet shows better latency performance on all three embedded testbeds. Note that MIMONet could generate results of two tasks in one forward pass, while SISO or MISO requires two forward passes. Specifically, MIMONet exhibits speedup compared to MISO models up to 1.98x, 2.29x, and 1.23x on Nano, AGX, and Orin, respectively.

- **Energy (Energy/FP):** MIMONet also demonstrates superior efficiency on all three embedded testbeds. Specifically, MIMONet exhibits energy saving compared to MISO models up to 2.01x, 8.64x, and 2.71x on Nano, AGX, and Orin, respectively.

Effective and Efficient On-Device deployment: our experimental results demonstrate that MIMONet can address all challenges in Sec. 2.3.1 (limited memory, real-time constraints, and energy constraints) while not sacrificing and often improving accuracy performance.

---

[2]The 2704.97 MB memory usage for two MISO models may double-count shared PyTorch framework memory. In practice, models can share some memory, reducing the total. However, MIMONet still provides substantial memory savings.

[3]Although the parameter count of MIMONet is reduced by up to 96.4%, memory consumption is not reduced by as much. This is because the PyTorch framework reserves some internal memory and introduces memory overhead.

2.4.5 Ablation Study

We exhibit the ablation study results in Table 2.3. We interpret the results as follows:

- **Efficacy of Reducing Intra-Task Redundancy (VIB):** Removing VIB increases memory usage from 577.17 MB to 1364.96 MB and parameters from 0.92 million to 25.51 million, highlighting its effectiveness in reducing intra-task redundancy. Unlike random pruning, which largely degrades accuracy, VIB retains relevant information and reduces noise, making it crucial and particularly work for managing MIMO tasks efficiently.
- **Efficacy of Reducing Inter-Task Redundancy (MTZ):** Omitting MTZ reduces accuracy in emotion and gender recognition, increases memory usage, and raises latency slightly, demonstrating its importance in minimizing inter-task redundancy and maintaining accuracy.
- **Efficacy of Post-Training Quantization (PTQ):** PTQ significantly reduces the model footprint through low-precision data structures, with minimal impact on accuracy. It decreases latency and energy consumption, enhancing post-training efficiency without major accuracy loss.

Component Efficacy: The ablation study confirms the critical role of each MIMONet component in ensuring optimal On-Device performance.

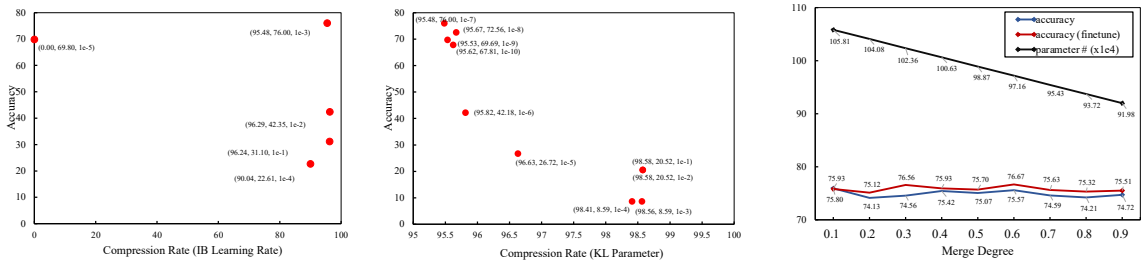


Figure 2.4: Parameter study results demonstrating the effect of hyperparameters in our algorithm. The first two figures show the relationships between compression rate and accuracy for the intra-model compression, and the third figure shows the parameter study for the inter-model compression. We visualize emotion task accuracy in this parameter study.

2.4.6 Parameter Study

We performed a series of parameter studies to assess the impact of different hyperparameters in MIMONet. As shown in Fig. 2.4, we present scatterplots of accuracy and compression rate for different hyperparameters in MIMONet. We found that the IB learning rate parameter requires careful selection, with 1e-3 being optimal for this study. For the KL parameter, a lower value setting (less than or equal to 1e-7) produced models with high accuracy but less compression rate, while higher values resulted in higher compression rates but very low accuracy. Therefore, a value of 1e-7 was optimal for the KL parameter in this study. Additionally, we investigated the impact of the merge degree hyperparameter on MIMONet, and found that it is insensitive to hyperparameters, with nearly lossless compression in all cases. The learnable parameter number linearly decreases with an increase in the merge degree, and model accuracy ranges from 74.21% to 75.80%, with model accuracy after finetuning ranging from 75.12% to 76.67%.

2.4.7 Case study on TurtleBot3

We conduct a case study using the TurtleBot3 robot to evaluate the real-world applicability of MIMONet. Specifically, we replace the TurtleBot3's main control device with a Jetson Nano and install an RGB camera, USB audio card, and speakers. Subsequently, a native speaker was invited to converse with the robot using happy tones while facial videos were captured. The robot will make sounds and specific movements based on the predictions. We evaluate three models (MISO, vanilla MIMO, and MIMONet) and find that MISO generated wrong results and exhibited the longest response time due to two-pass inferences. In contrast, vanilla MIMO and MIMONet both yield accurate results, while MIMONet yields significantly shorter response times (3.00x speedup than vanilla MIMO on average), which aligns with our evaluation (Sec. 2.4.4 and Sec. 2.4.5).

2.5 Discussion on Future Directions

Recent research on model compression has primarily focused on simpler models, often neglecting complex architectures like ResNet [145], Transformer [344], and complex graph-based models [193, 134, 345, 5, 106, 87, 88, 89]. Neural architecture search offers promising methods for creating efficient designs suitable for rapid inference [398]. Our goal is to adapt MIMONet for these advanced architectures, optimizing them for On-Device use by addressing latency, energy, and memory limitations in models [352, 304, 107, 139]. While our approach improves performance on GPU-based devices, integrating FPGAs [42, 399], autonomous vechicles [90], and emerging VR technologies [327] into AI-optimized embedded systems highlights a shift towards software-hardware co-design [389] for heterogeneous

computing, paving the way for broader On-Device model deployment. In complex and diverse multi-task models, compression gains might diminish with less overlap between complex multiple tasks. We leave this to future work, where we plan to explore adaptive compression strategies tailored for complex multi-task scenarios.

2.6 CONCLUSION

This paper introduces MIMONet, one of the first On-Device Multi-Input Multi-Output (MIMO) Deep Neural Network (DNN) frameworks targeting robotic applications. MIMONet achieves high accuracy and On-Device efficiency in terms of critical performance metrics, such as latency, energy, and memory usage. Extensive experiments on three widely used embedded platforms demonstrate that MIMONet can provide high accuracy while enabling superior On-Device efficiency, particularly when compared to state-of-the-art Single-Input Single-Output (SISO) and Multiple-Input Single-Output (MISO) models. Our research is an important first step toward building efficient MIMO DNN frameworks for practical On-Device learning scenarios, particularly for robots that often face hardware and performance constraints.

Chapter 3

MixTraining: Trade-Off Between Compute and Performance

3.1 Introduction

Self-supervised learning (SSL) has emerged as a powerful paradigm for learning general representations from unlabeled data, significantly improving performance on downstream tasks [40, 143]. The standard approach combines SSL with supervised learning (SL) into a two-phase pipeline: first training a model on unlabeled data to learn general representations, then adapting it to specific tasks using labeled data. This SSL+SL pipeline has become the de facto standard across computer vision [61, 144, 143, 353], natural language processing [82, 298, 40, 279, 254], and speech recognition [164, 55], particularly excelling in data-scarce scenarios where labeled examples are limited.

However, this two-phase pipeline introduces a fundamental trade-off between computational efficiency and model performance. While the first phase of SSL improves downstream accuracy, it requires substantial additional computation that can be prohibitive in resource-constrained environments. In fact, the SSL phase often demands hundreds or thousands of GPU-days before any task-specific training begins [185, 92, 68]. This computational overhead creates a significant barrier for practitioners, particularly those who must train models from scratch due to data privacy requirements, domain-specific constraints, or the absence of suitable pretrained models.

The abrupt transition between SSL and SL phases in the standard pipeline also presents optimization challenges. The model must suddenly shift from optimizing self-supervised objectives (e.g., reconstruction loss) to supervised objectives (e.g., classification loss), potentially causing training instability and suboptimal adaptation [292, 258, 200]. This discontinuous transition may prevent the model from fully leveraging the synergies between self-supervised and supervised learning signals.

To address these limitations, we propose MixTraining, a novel training framework that introduces a dedicated mixtraining phase between the SSL and SL phases. Our key insight is that the rigid separation between SSL and SL phases in conventional pipelines is neither necessary nor optimal. The mixtraining phase jointly optimizes both self-supervised and supervised objectives, creating a smooth transition that better preserves learned representations while adapting to downstream tasks. Our design also consolidate what would be separate forward and backward passes in the standard pipeline into unified operations (see Fig. 3.3 for details), substantially reducing training time while improving model performance.

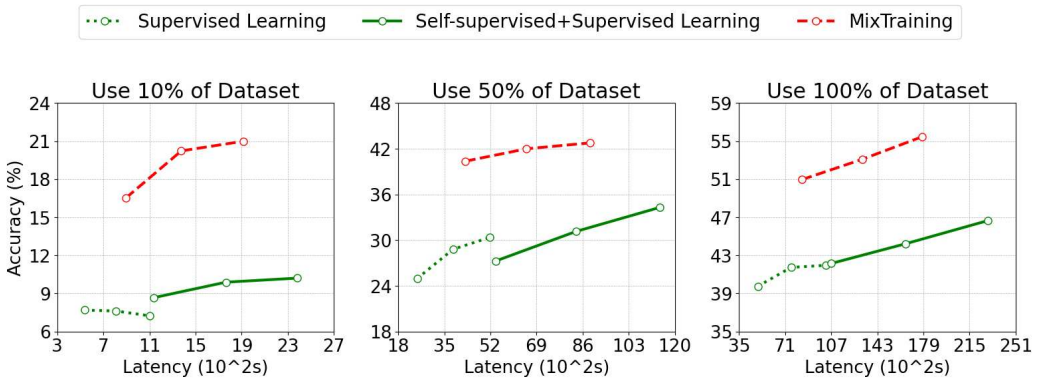


Figure 3.1: MixTraining demonstrates significant accuracy and computation gains over standard self-supervised learning and supervised learning pipeline across various data limitations levels (10%, 50%, and 100%). Experiments are conducted on the TinyImageNet dataset with the ViT-Tiny model; Each set of three points on the same line represents results obtained with different training durations (50, 75, and 100 epochs).

As shown in Fig. 3.1, MixTraining consistently achieves superior accuracy and lower training latency across varying data availability levels and training durations. For example, on the full TinyImageNet dataset [205] with reconstruction-based SSL [162, 143], MixTraining delivers an 18.89% relative improvement in accuracy (8.81% absolute) and achieves 1.29× speedup over the standard SSL+SL pipeline. The advantage becomes even more pronounced under severe data constraints: at 10% data availability, MixTraining improves accuracy by 105.58% (relative) and reduces training latency by 1.24×.

In this work, we focus on the widely used reconstruction-based self-supervised learning (SSL) paradigm (e.g., MAE) followed by supervised learning (SL) with classification losses. Within this scope, MixTraining consistently provides a better trade-off by smoothing the transition between reconstruction-based SSL and SL while reducing redundant computation. We view extending MixTraining to other SSL objectives (e.g., contrastive methods) as an important direction for future work.

Contributions. We introduce the MixTraining framework, which enhances both computational efficiency and model performance over the conventional SSL+SL pipeline. Focusing on reconstruction-based SSL and classification-based SL, our core contributions are:

- **Superior accuracy through smooth objective transition.** MixTraining introduces a mixtraining phase that creates a gradual transition between self-supervised and supervised objectives, avoiding the abrupt shift that can destabilize training in standard pipelines. This smooth transition leads to consistently higher accuracy across all evaluated settings. On TinyImageNet with ViT-Tiny, this design achieves an 18.89% relative accuracy gain over the standard SSL+SL baseline.

- **Computational efficiency via unified forward/backward passes.** By merging SSL and SL epochs into a joint optimization phase, MixTraining consolidates separate forward and backward passes over the backbone into single operations. Since backbone computations typically dominate training cost (the backbone is usually much larger than task-specific heads), this system-level optimization yields substantial speedups (e.g., 1.29× for ViT-T on TinyImageNet) while simultaneously improving accuracy.

- **Robust generalization across diverse settings.** MixTraining demonstrates consistent Pareto improvements over SSL+SL baselines across different experimental configurations, including multiple architectures (ViT, ResNet), datasets (CIFAR-10, CIFAR-100, TinyImageNet), data limitation levels (from 10% to 100%), and training epochs. The framework's modular design enables integration with existing SSL+SL pipelines with minimal modifications.

Scope and Applicability. It is important to clarify that MixTraining does not aim to replace the existing pretraining-finetuning paradigm when high-quality pretrained models are readily available. Instead, our framework targets scenarios where practitioners must train models from scratch due to data privacy constraints, domain-specific requirements where generic pretrained models prove inadequate, or the absence of suitable pretrained models. In these settings, MixTraining provides a more efficient alternative to the standard SSL+SL pipeline, achieving superior performance while reducing training latency.

Organization. The remainder of this paper is structured as follows. Section 3.2 reviews related work. Section 3.3 introduces the MixTraining framework, detailing its design intuition and operational procedure. Section 3.4 presents comprehensive empirical evaluations across multiple datasets and architectures. Section 3.5 concludes with a discussion of limitations and future directions.

3.2 Related Work

Learning Pipelines and the Compute-Performance Trade-off. Early deep learning models primarily relied on supervised learning (SL) with large labeled datasets [199, 326, 146]. More recently, self-supervised learning (SSL) has emerged as a powerful paradigm for learning transferable representations without labels, giving rise to a training pipeline that first leverages SSL for representation learning and then adapts the model to downstream tasks via supervised learning. This SSL+SL pipeline has been widely adopted across domains, including computer vision [61, 144, 143, 353], natural language processing [82, 298, 40, 279, 254], and speech recognition [164, 55]. While SSL+SL consistently improves accuracy over SL

alone, especially in low-label regimes, it incurs a substantial increase in compute due to the additional SSL phase [185, 68], leading to a compute-performance trade-off between the two approaches. Recent work has begun to examine aspects of this trade-off; for example, [225] explore strategies to reduce the compute cost of the SSL stage. In this work, we directly study on the compute-performance trade-off between SSL+SL and SL, and propose a new MixTraining pipeline that adds a dedicated mixtraining phase between SSL and SL, yielding Pareto improvements in both accuracy and compute efficiency over the standard SSL+SL pipeline.

Modifications to Learning Objectives. A broad class of methods modifies learning objectives to improve generalization, robustness, or stability. One important direction explores the synergy between different objectives, such as combining self-supervised learning and supervised learning objectives. Prior studies in this space have focused on domain adaptation [281, 27], mitigating catastrophic forgetting [151, 249], and improving data efficiency [385, 189, 374, 388]. Another related line of work mixes up data from different sources or domains to bridge distribution gaps or regularize decision boundaries [387, 383, 346, 229, 409], often by constructing intermediate representations that incorporate information from multiple domains or tasks. Our work is inspired by these lines of research but differs in both purpose and design: we introduce a dedicated mixtraining phase between SSL and SL to smooth the transition between two learning different objectives, and we design this phase for compute efficiency by merging forward and backward passes over the shared backbone—an aspect not addressed in prior work.

Efficient Training of Deep Neural Networks. A substantial body of research has focused on reducing the computational and data costs of training deep neural networks while maintaining competitive performance. Model compression techniques [137, 138, 173, 104, 32, 223] reduce network size and complexity through pruning, weight sharing, quantization, or other architectural simplifications, thereby lowering both memory footprint and FLOPs for more efficient training and deployment. Parameter-efficient finetuning methods [165, 85, 139] freeze most of the backbone and introduce small trainable modules or low-rank adaptations, greatly reducing optimization cost while preserving downstream performance. Data-efficient training strategies [26, 255, 350, 29] accelerate convergence by prioritizing informative examples, filtering redundant or noisy data, or leveraging pretrained models for improved initialization. These approaches target model, parameter, or data-level efficiency, and are complementary to our pipeline-level strategy: MixTraining could be integrated with these techniques to further improve its efficiency.

3.3 Methodology

We briefly introduce the standard self-supervised learning and supervised learning (SSL+SL) pipeline in Section 3.3.1. We introduce our MixTraining framework in Section 3.3.2, which consists of its design intuition (Section 3.3.2) and operational procedure (Section 3.3.2).

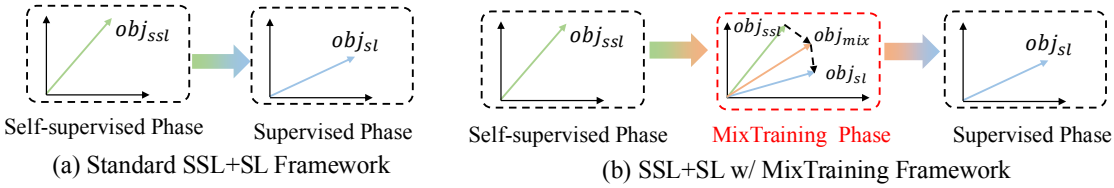


Figure 3.2: Comparison of MixTraining with the standard SSL+SL framework. (a) The standard SSL+SL framework features an abrupt transition from self-supervised objective ($\mathsf{obj}_{\mathsf{ssl}}$) to supervised objective ($\mathsf{obj}_{\mathsf{sl}}$). (b) Our MixTraining framework creates an dedicated mixtraining phase in the middle. The mixtraining phase optimizes towards a mixed objective ($\mathsf{obj}_{\mathsf{mix}}$), which enables a smooth transition from the self-supervised objective to the supervised objective.

3.3.1 Background: The Standard Self-supervised Learning and Supervised Learning Pipeline

The standard self-supervised learning and supervised learning pipeline consists of two separate phases: a *self-supervised learning (SSL)* phase and a *supervised learning (SL)* phase. In the self-supervised learning phase, a backbone model with a self-supervised learning head is trained to help the model learn general feature representations. Specifically, this process usually relies on learning from unlabeled data with reconstruction tasks [162, 161, 252] or predicting manually masked tokens [82, 143]. The backbone model is further refined in the supervised learning phase, together with a supervised learning head. This step adapts the model to downstream tasks, usually achieving better performances than directly training the downstream tasks. Fig. 3.2(a) shows the standard SSL+SL framework, which has now become the go-to approach for improving the performance of AI models [353, 254].

3.3.2 A New Framework: MixTraining

Design Intuition behind MixTraining

While the standard SSL+SL framework has achieved remarkable success, its self-supervised learning and supervised learning phases are completely separated, leaving limited opportunity for interaction or further optimization. To enable closer interactions between these two phases, we propose a novel MixTraining framework, as shown in Fig. 3.2(b), which effectively merges several self-supervised learning and supervised learning epochs into an additional *mixtraining phase*, featuring a smooth transition between learning objectives.

MixTraining introduces a new mixtraining phase that allows joint updates of self-supervised and supervised objectives, which is in contrast with the standard SSL+SL framework, where one *first* updates the self-supervised objective and *then* updates the supervised objective. At a high level, the benefits of this phase are rooted in the dedicated joint optimization objective. The joint objective can be easily understood as a weighted average of the self-supervised objective and the supervised objective to balance these two objectives. We next explain the design intuition behind the mixtraining phase to achieve both *computation gains* and *accuracy gains*.

Accuracy Gains. Since the self-supervised learning phase aims at learning general representations and the following supervised learning phase aims at learning task-specific information, intuitively, these two phases optimize the model in different directions. The standard SSL+SL framework features an abrupt change in optimization directions during the transition from self-supervised learning and supervised learning (Fig. 3.2(a)), which may

cause instability in model performance [258]. Indeed, there are also studies show that, in certain settings, supervised learning in the second phase can lead to worse model performance [292, 200]. In our MixTraining framework, the mixtraining phase creates a middle ground, i.e., a weighted combination of two objectives, allowing a rather smooth transition from the self-supervised learning objectives to the supervised learning objective, as illustrated in Fig. 3.2(b). We hypothesize that such a smooth transition avoids instability in phase transition, thus allowing the model to better adapt to the target task and achieve higher accuracy. We validate the smooth-transition intuition by measuring the cosine similarity of the last-backbone-layer gradients induced by different learning objectives. The average cosine similarity between SSL and SL objectives is only 0.018, indicating that these two objectives are nearly orthogonal. In contrast, the cosine similarity between SSL and MixTraining objectives is 0.791, and between SL and MixTraining objectives is 0.598, suggesting that the MixTraining objective indeed provides a smooth transition from SSL to SL. Our empirical results in Section 3.4 further support the hypothesis that smooth transitions lead to higher accuracy.

Computation Gains. In the standard SSL+SL framework, we first run self-supervised learning passes with data $(x_{\mathsf{ssl}}, y_{\mathsf{ssl}})$, and then run supervised learning passes with data $(x_{\mathsf{sl}}, y_{\mathsf{sl}})$. This process involves forward/backward passes of both $(x_{\mathsf{ssl}}, y_{\mathsf{ssl}})$ and $(x_{\mathsf{sl}}, y_{\mathsf{sl}})$, and strictly follows a *sequential order* to compute each sub-processes (top part of Fig. 3.3). In contrast, MixTraining aims to jointly optimize self-supervised and supervised objectives. Specifically, it "merges" $(x_{\mathsf{ssl}}, y_{\mathsf{ssl}})$ and $(x_{\mathsf{sl}}, y_{\mathsf{sl}})$ into a *mixed data* $(x_{\mathsf{mix}}, y_{\mathsf{mix}})$ and thus merging the separate forward passes over the backbone model into a single pass, and use its

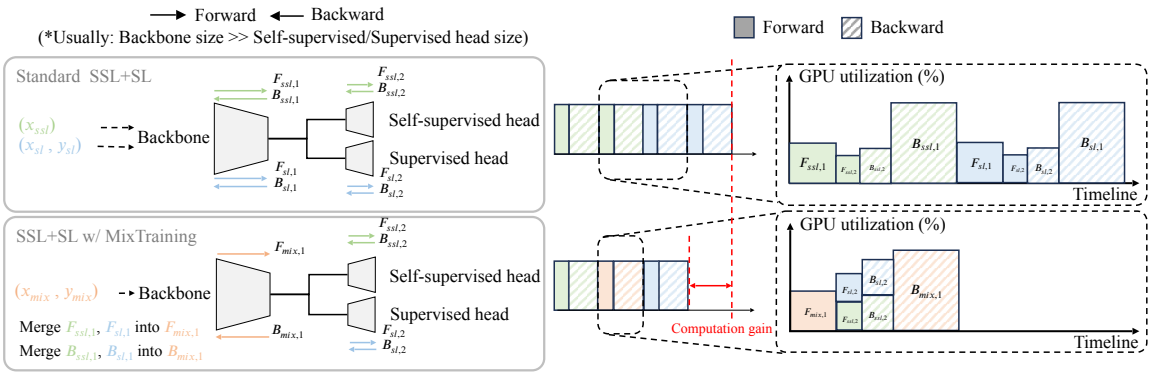


Figure 3.3: Comparison of MixTraining with the standard SSL+SL framework. MixTraining achieves computation gains over the standard SSL+SL framework. *Top: Standard* SSL+SL. Data first goes through a self-supervised learning pass ($F_{\mathsf{ssl},1} \to F_{\mathsf{ssl},2} \to B_{\mathsf{ssl},2} \to B_{\mathsf{ssl},1}$) and then goes through a supervised learning pass ($F_{\mathsf{sl},1} \to F_{\mathsf{sl},2} \to B_{\mathsf{sl},2} \to B_{\mathsf{sl},1}$). *Bottom:* MixTraining. We merge two forward passes ($F_{\mathsf{ssl},1}$ and $F_{\mathsf{sl},1}$) over the backbone model together into a single pass $F_{\mathsf{mix},1}$, and use its result for both self-supervised head and supervised head; the backward passes ($B_{\mathsf{ssl},1}$ and $B_{\mathsf{sl},1}$) are merged into $B_{\mathsf{mix},1}$ (bottom left). Our modifications reduce computation requirements and allow better parallelization (bottom right).

result for both self-supervised head and supervised head; from the computation aspect, we also merge the backward passes of self-supervised learning and supervised learning tasks over the backbone model into a single pass (bottom left part of Fig. 3.3). Since the size of the backbone model is usually much larger than the size of self-supervised and supervised heads [143, 96, 371], the merge of forward/backward passes over the backbone model allows us to reduce computation compared to the synchronous setting substantially. Additionally, the merge of forward/backward passes over the backbone model allows better parallelization of forward/backward passes over the self-supervised and supervised heads (right part of Fig. 3.3), which further speeds up the computation. Empirically, we find the runtime of a single merged mixtraining pass (0.317s) is lower than the combined runtime of an SL pass and an SSL pass (0.543s = 0.248s + 0.295s), validating our intuition.

The MixTraining Procedure

In this section, we introduce the operational procedure of our MixTraining framework in detail. Besides the number of self-supervised learning epoch e_{ssl} and the number of supervised learning epoch e_{sl}, MixTraining takes as input a hyperparameter mix-ratio $\rho \in [0, 1]$ to determine the number of self-supervised learning/supervised learning epochs e_{mix} to be merged into the mixtraining phase. We set

$$e_{\mathsf{mix}} = \lfloor \rho \min(e_{\mathsf{ssl}}, e_{\mathsf{sl}}) \rfloor. \tag{3.1}$$

Our MixTraining framework then operates by running (i) the vanilla self-supervised learning phase for $e_{\mathsf{ssl}} - e_{\mathsf{mix}}$ epochs, (ii) the mixtraining phase for e_{mix} epochs, and (iii) the vanilla supervised learning phase for $e_{\mathsf{sl}} - e_{\mathsf{mix}}$ epochs, as shown in Algorithm 1.

Since the self-supervised learning and supervised learning phases are standard, in the following, we mainly discuss the mixtraining phase. In the mixtraining phase, we (i) design a mixing function g to generate *a mixed dataset*, and (ii) design *a joint optimization objective* as supervision signal. We next highlight the design choice for these two parts.

The Mixed Dataset. The goal of creating a mixed dataset $\mathcal{D}_{\mathsf{mix}} = g(\mathcal{D}_{\mathsf{ssl}}, \mathcal{D}_{\mathsf{sl}})$ is to extract information stored in self-supervised learning dataset $\mathcal{D}_{\mathsf{ssl}} = \{x_i\}_i$ and supervised learning dataset $\mathcal{D}_{\mathsf{sl}} = \{(x_i, y_i)\}_i$, featuring a smooth transition between two learning objectives Fig. 3.2 (b). In the simple case where self-supervised learning and supervised learning use the same input (i.e., x_i), we can simply set $g(\mathcal{D}_{\mathsf{ssl}}, \mathcal{D}_{\mathsf{sl}}) = \mathcal{D}_{\mathsf{sl}}$. We remark that the importance of this simple case is usually overlooked: conducting self-supervised learning and supervised

Algorithm 1 The MixTraining Framework

Input: Self-supervised learning epoch e_{ssl}, Supervised learning epoch e_{sl}, MIX-RATIO $\rho \in [0, 1]$.

Initialize model parameters θ.
Calculate mixtraining epoch e_{mix} via Eq. (3.1).
▷ Vanilla Self-supervised Learning Phase
for $e = 1$ to $e_{\mathsf{ssl}} - e_{\mathsf{mix}}$ **do**
    Conduct vanilla self-supervised learning phase *w.r.t.* self-supervised loss $\ell_{\mathsf{ssl}}(x; \theta)$ to optimize θ.
end for
▷ MixTraining Phase
for $e = 1$ to e_{mix} **do**
    Optimize the model parameter θ with respect to the joint optimization objective as in Eq. (3.3).
end for
▷ Vanilla Supervised Learning Phase
for $e = 1$ to $e_{\mathsf{sl}} - e_{\mathsf{mix}}$ **do**
    Conduct vanilla supervised learning phase *w.r.t.* supervised loss $\ell_{\mathsf{sl}}(f(x), y; \theta)$ to optimize θ.
end for

learning on the same ImageNet dataset allows one to boost the top-1 classification accuracy from 82.5% to 84.9%, without using extra data [143].

We next discuss the general case where the self-supervised learning dataset is not the same as the supervised learning dataset, i.e., $\mathcal{D}_{\mathsf{ssl}} \neq \mathcal{D}_{\mathsf{sl}}$. Inspired by the mixup method in machine learning to improve the generalization and robustness to adversarial examples [387], we consider a *randomized* mixing function g, which randomly mixes up data points from both datasets. Specifically, we set

$$\mathcal{D}_{\mathsf{mix}} = g(\mathcal{D}_{\mathsf{ssl}}, \mathcal{D}_{\mathsf{sl}}) = \{(x_{\mathsf{mix}}, y_{\mathsf{sl}}) : x_{\mathsf{mix}} = \lambda\, x_{\mathsf{sl}} + (1 - \lambda)\, x_{\mathsf{ssl}}, (x_{\mathsf{sl}}, y_{\mathsf{sl}}) \in \mathcal{D}_{\mathsf{sl}}, x_{\mathsf{ssl}} \in \mathcal{D}_{\mathsf{ssl}}\}, \tag{3.2}$$

where for each $(x_{\mathsf{sl}}, y_{\mathsf{sl}}) \in \mathcal{D}_{\mathsf{sl}}$, we randomly draw a self-supervised data point x_{ssl} from $\mathcal{D}_{\mathsf{ssl}}$ and generate a mixup feature $\lambda\, x_{\mathsf{sl}} + (1-\lambda)\, x_{\mathsf{ssl}}$, where λ is a hyperparameter of user's choice (we set $\lambda = 0.5$ in our experiments to balance the contribution from both datasets). We adopt the supervised label y_{sl} to provide supervised signal since the self-supervised part typically doesn't require labels.

The Joint Optimization Objective. Let θ denote the model parameters. Let $\ell_{\mathsf{ssl}}(x;\theta)$ denote the self-supervised loss, e.g., MSE reconstruction loss for masked autoencoders [143], and let $\ell_{\mathsf{sl}}(f(x), y; \theta)$ denote the supervised loss, e.g., cross-entropy for image classification [199]. Let $\mathcal{D}_{\mathsf{ssl}}$ represent the SSL dataset, which contains examples $\{x_i\}_i$, and $\mathcal{D}_{\mathsf{sl}}$ represent the SL dataset, which also contains examples $\{(x_i, y_i)\}_i$. These datasets serve as the sources for self-supervised and supervised learning, respectively. The classical SSL+SL framework first optimizes $\min_\theta \mathbb{E}_{x,y\sim\mathcal{D}_{\mathsf{ssl}}}[\ell_{\mathsf{ssl}}(x;\theta)]$ and then optimizes $\min_\theta \mathbb{E}_{x,y\sim\mathcal{D}_{\mathsf{sl}}}[\ell_{\mathsf{sl}}(f(x), y; \theta)]$.

To integrate self-supervised and supervised learning objectives, MixTraining considers a weighted combination of these two objectives and optimizes the following goal:

$$\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}_{\mathsf{mix}}}[\alpha\, \ell_{\mathsf{ssl}}(x;\theta) + (1-\alpha)\, \ell_{\mathsf{sl}}(f(x), y; \theta)], \tag{3.3}$$

where $\mathcal{D}_{\mathsf{mix}}$ is the mixed dataset and $\alpha \in (0, 1)$ is a hyperparameter loss-ratio designed to balance the focus between learning general representations (by optimizing self-supervised loss $\ell_{\mathsf{ssl}}(x;\theta)$) and achieving specific target (by optimizing supervised loss $\ell_{\mathsf{sl}}(f(x), y; \theta)$). Eq. (3.3) can be naturally extended to include a continuously annealed loss weight α that interpolates between the SSL and SL objectives. For simplicity, we retain the current form

of Eq. (3.3); our ablation study in Section 3.4.2 shows that both linear and cosine annealing yield comparable performance.

3.4 Experiments

3.4.1 Setups

Datasets. We conduct experiments on standard computer vision datasets, including CIFAR-10 [198], CIFAR-100 [198], and TinyImageNet [205].

Models. Our model consists of three components: a shared backbone model, a classification head for SL, and a reconstruction head for SSL. For most of our experiments, we use the standard ViT-Tiny (ViT-T) [92, 361] as the backbone and the classification head. We adopt masked autoencoding (MAE) [143] as the reconstruction task and use an MAE decoder of depth 2 as the reconstruction head. In Section 3.4.4, we provide additional experiments with ResNet-18, where the reconstruction task is standard autoencoding (AE) and the reconstruction head is a two-layer transposed convolution [384].

Baselines. We evaluate the performance of our algorithm (Algorithm 1) against the following baselines:

- *Supervised learning (SL).* Conduct standard supervised learning on the backbone and the classification head with cross-entropy loss for e_{sl} epochs.

- *Self-supervised learning + supervised learning (*SSL+SL*).* Conduct self-supervised learning on the backbone and the reconstruction head with MSE loss for e_{ssl} epochs and then conduct standard supervised learning with cross-entropy loss for e_{sl} epochs.

The comparison between SL and SSL+SL reflects the compute-performance trade-off: SSL+SL achieves better performance at the cost of the added computation in the SSL-alone step. We aim to provide a better compute-performance trade-off with MixTraining, i.e., a Pareto improvement over the SSL+SL baseline. We adopt a standard set of augmentations following [146, 72, 50]. We remark that our goal is not to maximize benchmark accuracy, but to enable a controlled and reproducible comparison across different training pipelines.

Evaluation Metrics. For each method, we measure its performance by the accuracy on the downstream classification task and its computation cost by the training latency (i.e., the total wall-clock time). We report the average accuracy and latency over 4 runs with different random seeds. We calculate the speedups of our method as the ratio between the latency of SSL+SL and MixTraining.

Other Implementation Details. We conduct experiments across various data limitation levels by randomly select a fraction of p data points from the original dataset; we choose $p \in \{10\%, 25\%, 50\%, 75\%, 100\%\}$. In our main experiments, we set training epoch $e_{\mathsf{sl}} = e_{\mathsf{ssl}} = 100$, loss-ratio $\alpha = 0.5$, and mix-ratio $\rho = 0.5$.

3.4.2 Main Results

We present our main experiment results in this section. Results across different data limitation levels are provided in Section 3.4.2, and results with varying SSL and SL datasets are presented in Section 3.4.2.

Performance Analysis across Various Data Limitation Levels

In this section, we evaluate the performance of MixTraining across various data limitation levels. We conduct experiments on CIFAR-10, CIFAR-100, and TinyImageNet datasets and present results in Table 3.1. As expected, the comparison between SSL+SL and SL reflects a compute-performance trade-off: SSL+SL achieves better accuracy at the cost of higher training latency. Our MixTraining method achieves a Pareto improvement over the SSL+SL baseline: **MixTraining achieves higher accuracy and lower latency in 15 out of 15 settings.**

Compared to baselines, MixTraining achieves significant accuracy gains: for instance, on the full TinyImageNet dataset with the ViT-T model, MixTraining achieves 18.89% relative accuracy gain (8.81% absolute accuracy gain) over SSL+SL and 32.17% relative accuracy gain (13.50% absolute accuracy gain) over SL. The accuracy gains are more significant under limited data: for instance, on the TinyImageNet dataset and at data limitation level of 10%, MixTraining achieves 105.58% relative accuracy gain (10.78% absolute accuracy gain) over SSL+SL and 189.92% relative accuracy gain (13.75% absolute accuracy gain) over SL. MixTraining also saves more compute (reflected as training latency) compared to SSL+SL. For instance, on the full TinyImageNet dataset with the

Table 3.1: Accuracy and latency comparison using the ViT-T model. We evaluate performance across various data limitation levels $p \in \{10\%, 25\%, 50\%, 75\%, 100\%\}$. The highest accuracy in each setting is highlighted in **bold**). MixTraining achieves a Pareto improvement over the SSL+SL baseline, **achieving higher accuracy and lower latency in 15 out of 15 settings.**

Datasets	Methods	10%		25%		50%		75%		100%	
		Accuracy↑	Latency(s)↓	Accuracy↑	Latency(s)↓	Accuracy↑	Latency(s)↓	Accuracy↑	Latency(s)↓	Accuracy↑	Latency(s)↓
TinyImageNet	**SL**	7.24%	1102.42	20.91%	2619.14	30.36%	5181.44	36.98%	7709.21	41.96%	10257.12
	SSL+SL	10.21%	2382.06	21.78%	5813.27	34.30%	11449.43	42.73%	17296.22	46.65%	22917.29
	MixTraining	**20.99%**	1913.75	**31.43%**	4516.69	**42.77%**	8868.88	**49.30%**	13304.60	**55.46%**	17795.47
CIFAR-10	**SL**	53.40%	589.44	66.03%	1335.79	75.00%	2593.27	79.22%	3844.27	81.52%	5155.59
	SSL+SL	56.79%	1253.65	67.95%	2774.24	77.06%	5354.08	82.11%	7993.72	84.69%	10730.52
	MixTraining	**60.45%**	962.16	**72.61%**	2206.13	**79.95%**	4247.78	**83.97%**	6234.03	**87.13%**	8274.45
CIFAR-100	**SL**	19.05%	609.11	31.49%	1346.96	42.50%	2580.08	48.97%	3814.54	54.72%	5022.96
	SSL+SL	22.27%	1279.78	34.93%	2882.62	46.02%	5551.88	53.79%	8226.83	57.92%	10960.08
	MixTraining	**25.19%**	1010.94	**38.55%**	2226.92	**48.60%**	4298.75	**55.84%**	6354.76	**59.95%**	8457.93

ViT-T model, MixTraining acheives 1.29× speedup compared to SSL+SL. These results show that MixTraining provides a better compute-performance trade-off compared to the standard SSL+SL pipeline across various data limitation levels.

Self-Supervised Learning and Supervised Learning on Different Datasets

In this section, we evaluate the performance of MixTraining in settings where the self-supervised learning dataset and supervised learning dataset are different. We perform self-supervised learning on the TinyImageNet dataset and supervised learning on the CIFAR-10 or CIFAR-100 datasets. We present the results in Table 3.2. The comparison between SSL+SL and SL still reflects the compute-performance trade-off: SSL+SL achieves better accuracy at the cost of higher training latency. Our MixTraining method achieves a Pareto improvement over the SSL+SL baseline: **MixTraining achieves higher accuracy and lower latency on both settings.**

Compared to baselines, MixTraining achieves significant accuracy gains: for instance, on the CIFAR-10 dataset with the ViT-T model, MixTraining achieves 5.58%

Table 3.2: Accuracy and latency comparison with different self-supervised and supervised datasets using the ViT-T model. We use TinyImageNet as the self-supervised learning dataset and use CIFAR-10/CIFAR-100 as the supervised learning dataset. The highest accuracy in each setting is highlighted in **bold**. MixTraining achieves a Pareto improvement over the SSL+SL baseline, **achieving higher accuracy and lower latency in both settings.**

Methods	TinyImageNet to CIFAR-10		TinyImageNet to CIFAR-100	
	Accuracy↑	Latency(s)↓	Accuracy↑	Latency(s)↓
SL	81.76%	5203.76	54.22%	5149.24
SSL+SL	84.48%	16815.33	56.22%	16582.02
MixTraining	**89.19%**	12098.50	**58.49%**	13742.15

relative accuracy gain (4.71% absolute accuracy gain) over SSL+SL and 9.09% relative accuracy gain (7.43% absolute accuracy gain) over SL. In terms of computation cost (reflected as training latency), MixTraining also saves more compute compared to SSL+SL. For instance, on the CIFAR-10 dataset with the ViT-T model, MixTraining can achieve 1.39× speedup compared to SSL+SL. These results show that MixTraining can provide a better compute-performance trade-off compared to the standard SSL+SL pipeline, even when using different datasets for self-supervised learning and supervised learning.

MixTraining is More Powerful than Standard Data Mixups

To examine the advantages of MixTraining over standard data mixups [387], we further equip the SL and SSL+SL baselines with mixup data augmentation (Table 3.3) on both CIFAR-10 and CIFAR-100 datasets. While mixup improves the baselines' performance, our proposed MixTraining method consistently achieves the best performance. This demonstrates that MixTraining provides advantages that standard mixup alone cannot achieve; the gains stem from the unified MixTraining pipeline described in Section 3.3.2, rather than just from augmentation effects.

Table 3.3: Accuracy comparison on CIFAR-10 and CIFAR-100 with mixup augmentations applied to the SL and SSL+SL baselines (with ViT-T). We report results under the 100% data setting. The highest accuracy is highlighted in **bold**. MixTraining achieves the best performance over the baselines.

Datasets	**CIFAR-10**	**CIFAR-100**
SL w/ mixup	83.87%	58.96%
SSL+SL w/ mixup	85.81%	59.41%
MixTraining	**87.13%**	**59.95%**

Table 3.4: Ablation of linear and cosine annealing schedules for loss weight α in MixTraining. We conduct experiments on the CIFAR-10 dataset with the ViT-T model. All variants achieve comparable performance.

Methods	**10%**	**25%**	**50%**	**75%**	**100%**
Algorithm 1	60.45%	72.61%	79.95%	83.97%	87.13%
Algorithm 1 w/ linear	61.49%	72.21%	79.07%	84.82%	86.90%
Algorithm 1 w/ cosine	60.65%	72.03%	79.16%	83.40%	87.00%

Smooth Interpolation of α in the Mixtraining Phase

While Eq. (3.3) uses a fixed loss weight $\alpha \in [0, 1]$, it can be naturally extended to include a continuously annealed weight α that interpolates between the SSL and SL objectives. In Table 3.4, we conduct ablations with both linear and cosine annealing schedules. Since both variants achieve comparable performance to our original MixTraining approach, this indicates that the key advantage lies in the existence of the mixtraining phase itself, rather than in the specific scheduling of the weight α.

Impact of Training Epochs

We study the impact of varying training epochs e_{ssl}, e_{sl} on model accuracy and training latency. For simplicity, we set $e_{\mathsf{ssl}} = e_{\mathsf{sl}}$ and choose its value from {50,

Table 3.5: Parameter study on training epochs with full data using the ViT-T model. The highest accuracy in each setting is highlighted in **bold**. MixTraining achieves a Pareto improvement over the SSL+SL baseline, **achieving higher accuracy and lower latency in 9 out of 9 settings.**

Datasets	Computation (epoch)	SL		SSL+SL		MixTraining	
		Accuracy↑	Latency(s)↓	Accuracy↑	Latency(s)↓	Accuracy↑	Latency(s)↓
TinyImageNet	50	39.75%	4957.82	42.16%	10714.36	**50.96%**	8410.97
	75	41.74%	7560.40	44.22%	16498.39	**53.09%**	13111.49
	100	41.96%	10257.12	46.65%	22917.29	**55.46%**	17795.47
CIFAR-10	50	80.78%	2489.38	83.75%	5365.33	**85.60%**	4193.42
	75	81.59%	3818.88	84.48%	8070.96	**86.79%**	6515.22
	100	81.52%	5155.59	84.69%	10730.52	**87.13%**	8274.45
CIFAR-100	50	54.25%	2713.55	56.91%	5763.71	**58.67%**	4421.63
	75	54.71%	3881.84	56.95%	8104.50	**59.11%**	6459.23
	100	54.72%	5022.96	57.92%	10960.08	**59.95%**	8457.93

75, 100}.[1] We conduct experiments across various choices of data limitation levels $p \in \{10\%, 25\%, 50\%, 75\%, 100\%\}$. Due to space limitations, we present results for learning with full data ($p = 100\%$) in the main content.

We present results with full data in Table 3.5. We observe that, compared to SL and SSL+SL, MixTraining generally yields greater accuracy gains for added computation; for example, doubling the training epochs from 50 to 100 results in a larger improvement in accuracy. Interestingly, MixTraining with $e_{\mathsf{ssl}} = e_{\mathsf{sl}} = 50$ outperforms SL with $e_{\mathsf{sl}} = 100$ and SSL+SL with $e_{\mathsf{ssl}} = e_{\mathsf{sl}} = 100$ in terms of both accuracy and compute: MixTraining simultaneously achieves higher accuracy and lower training latency. MixTraining consistently acheives a Pareto improvement over the SSL+SL baseline: **MixTraining achieves higher accuracy and lower latency in all 9 out of 9 settings**. These improvements again demonstrate that MixTraining is a superior training pipeline compared to the standard SSL+SL approach.

[1]When $e_{\mathsf{ssl}} = e_{\mathsf{sl}} = 100$, all validation loss curves have converged. Some validation losses may not fully converge when the number of training epochs is reduced; however, this is intentional, as we aim to study compute-performance trade-offs under compute-constrained settings.

Table 3.6: Accuracy comparison in semi-supervised setting with 50% labeled data and 50% unlabeled data. We conduct experiments on the CIFAR-10 dataset with the ViT-T model. MixTraining achieves the **best** result over the baselines.

Metric	**SL**	**SSL+SL**	**MixTraining**
Accuracy↑	75.59	83.89	**87.35**

3.4.3 MixTraining in New Settings

Learning in Semi-Supervised Settings

We extend our experiments to the semi-supervised setting where 50% data are labeled and the other 50% data are unlabeled. In this setup, the SL baseline trains only on the labeled data, while the SSL+SL baseline first applies reconstruction-based SSL on all data and then performs supervised learning on the labeled subset. In our MixTraining framework, the SSL and SL phases remain the same as in the standard SSL+SL pipeline, while the intermediate mixtraining phase randomly mixes labeled data and train using the objective in Eq. (3.3). As shown in Table 3.6, MixTraining achieves the highest accuracy in this semi-supervised setting, with even larger gains over the baselines. This further highlights the effectiveness of MixTraining.

Learning with Data Noise

We simulate a real-world environment by adding random label noise to the dataset: for each data point, with probability $p = 0.2$, we randomly change its label. Table 3.7 shows that our MixTraining framework continues to achieve the best performance with data noise, verifying its robustness in practical settings.

Table 3.7: Accuracy comparison on CIFAR-10 (with ViT-T) with random data noise. We evaluate performance across various data limitation levels $p \in \{10\%, 25\%, 50\%, 75\%, 100\%\}$. The highest accuracy is highlighted in **bold**. MixTraining achieves the best performance over the baselines.

Methods	10%	25%	50%	75%	100%
SL	55.80%	64.64%	72.96%	78.33%	81.27%
SSL+SL	56.61%	68.45%	72.94%	80.98%	84.17%
MixTraining	**61.18%**	**70.44%**	**75.97%**	**84.01%**	**86.13%**

3.4.4 When MixTraining Helps and When It Doesn't

We conduct additional experiments with ResNet-18 on CIFAR-10, a setting where the SL baseline is already highly optimized. Across all data limitation levels, MixTraining consistently improves upon the SSL+SL baseline. However, when SSL+SL (with an added SSL stage) underperforms a strong SL baseline, MixTraining, since it also includes SSL, does not fully recover SL performance. In contrast, in regimes where SSL+SL improves over SL (e.g., under limited data), MixTraining further improves upon SSL+SL.

These results refine, but do not contradict, our main conclusions. MixTraining consistently improves upon SSL+SL and offers a favorable compute–performance trade-off; yet in settings where SL is already highly optimized and adding SSL hurts performance, MixTraining may not surpass SL.

3.5 Conclusion and Future Work

We introduced MixTraining, an innovative framework that interleaves multiple self-supervised learning (SSL) and supervised learning (SL) epochs within a unified training phase, enabling a smooth transition between the two learning objectives. By enhancing the

Table 3.8: Accuracy and latency comparison using the ResNet-18 model. We evaluate performance across various data limitation levels $p \in \{10\%, 25\%, 50\%, 75\%, 100\%\}$. The highest accuracy in each setting is highlighted in **bold** (if gain $\geq 0.3\%$). MixTraining consistently improves over SSL+SL but may not match the SL performance if SSL+SL degrades performance relative to a strong SL baseline.

Datasets	Methods	10%	25%	50%	75%	100%
CIFAR-10	**SL**	60.42	79.86	85.31	**87.97**	**92.05**
	SSL+SL	65.29	82.20	85.24	87.12	87.18
	MixTraining	**67.33**	**82.73**	85.30	87.63	87.91
CIFAR-100	**SL**	28.76	40.67	50.25	56.56	**77.87**
	SSL+SL	28.57	40.78	51.44	55.73	59.76
	MixTraining	**29.81**	**41.76**	**52.30**	**57.62**	61.54

synergy between SSL and SL, MixTraining achieves significant accuracy improvements while consolidating shared computation steps to reduce computational cost. Extensive experiments demonstrate that MixTraining offers a superior compute-performance trade-off compared to the conventional SSL+SL pipeline: MixTraining achieves substantial improvements in model accuracy while significantly reducing training latency.

While our work focuses on scenarios where practitioners need to train models from scratch due to data privacy constraints or domain-specific requirements, we believe MixTraining can also be leveraged in larger-scale settings when positioned as a *mid-training strategy*, rather than as a replacement for fully task-agnostic pretraining. As an initial validation of this idea, we conduct an additional experiment where a ViT-T model pretrained on TinyImageNet is adapted to CIFAR-10. Compared to directly fine-tuning the pretrained model (84.48% accuracy), applying MixTraining on the target dataset (as pretraining data is typically unavailable in this case) improves performance to 88.75%, indicating that MixTraining can preserve general representations while enhancing task-specific adaptation. We believe this mid-training perspective opens an interesting direction.

Chapter 4

R^3: On-Device Real-Time Deep Reinforcement Learning

4.1 Introduction

Deep Reinforcement Learning (DRL) has emerged as a promising field, showing a pervasive influence in various real-world applications. [256, 343, 24, 314, 318, 317, 131] One such application is autonomous vehicles (AVs), where DRL models need to adapt to ever-changing road and traffic conditions continually [183, 291]. The models must swiftly learn and retrain based on new data and evolving scenarios, maintaining their responsiveness and capability for immediate decision-making [14, 307]. Similarly, in robotic rescue, DRL models enable robots to navigate hazardous environments, locate survivors, and deliver assistance [6, 158]. These models must adapt to rapidly shifting and unpredictable conditions, requiring efficient on-the-spot training and retraining to ensure timely decision-making.

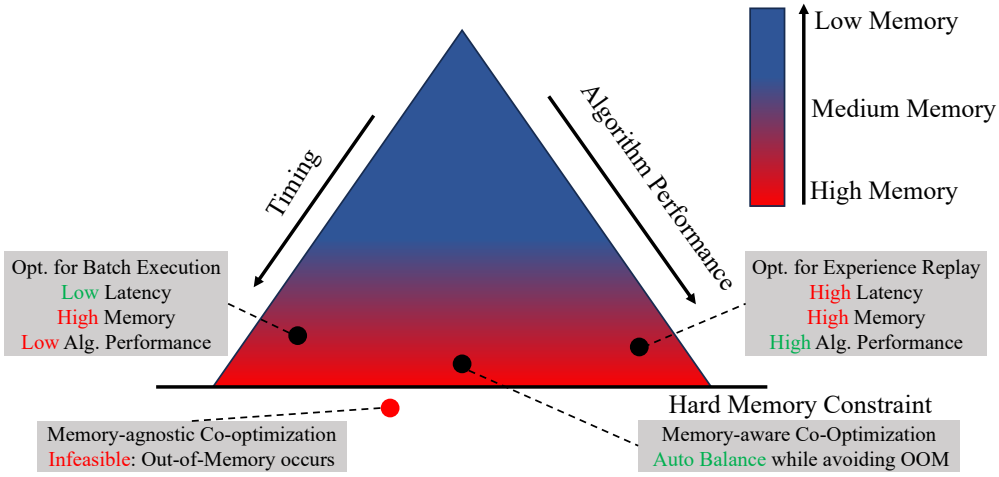


Figure 4.1: Visualization of optimization challenges of embedded deep reinforcement learning training. Arrow direction means better timing, better algorithm performance, and less memory usage. A higher red level represents higher memory pressure.

These examples underline the critical need for efficient On-Device DRL training with timing constraints, integrating (re)training within runtime inference to timely adapt to dynamic and evolving environments.

However, conducting On-Device DRL training is fraught with unique challenges. As depicted by Figure 4.1, these challenges are fundamentally tied to the need for co-optimizing two frequently conflicting objectives: latency and algorithm performance[1], a task made more complex by the memory constraints inherent to embedded devices. Intrinsically, the DRL training process involves two critical components: batch execution and experience replay, each representing its own dimension of trade-off. Batch execution primarily pertains to the batch size parameter, which forms a crucial junction between timing and memory trade-offs. A larger batch size might accelerate the training process but increase memory

[1]"Algorithm performance" in DRL is akin to "accuracy" in DNNs. In DNNs, accuracy gauges the correct prediction rate. However, in DRL, performance is more multifaceted, primarily involving the agent's capability to optimize cumulative reward over time, balancing exploration and exploitation. While accuracy can be relevant in some tasks, DRL performance typically entails a broader, more complex set of considerations.

consumption. On the other hand, experience replay entails a replay buffer size trade-off. Here, a larger buffer size can improve algorithm performance by providing more diverse experiences for training but can also cause memory issues due to the increased demand for storage. Optimizing these trade-offs independently could result in less-than-ideal outcomes, while co-optimizing them in a memory-agnostic manner may lead to significant memory pressure or even trigger Out-of-Memory (OOM) errors which in some severe cases cause the memory-limited embedded system (e.g., 16 GB memory for AGX Xavier) to fail. Therefore, a holistic memory-aware approach that addresses both dimensions of trade-off simultaneously becomes indispensable. To navigate this intricate multi-dimensional trade-off space and manage the inherent complexity of **R**eal-time Deep **R**einforcement Learning for Autonomous **R**obotics, in this paper, we present $\texttt{R}^3$, which is designed to tackle the complex problem of On-Device DRL training with careful consideration of timing, memory, and algorithmic performance. Our approach is composed of three innovative components. Firstly, we introduce a deadline-driven feedback loop, which dynamically assigns proper batch size for batch execution to balance memory and latency. Secondly, our approach includes efficient memory management, which applies task-specific memory optimizations to reduce memory footprint significantly. It allows for larger replay buffer sizes, enhancing algorithm performance in real-time embedded DRL while mitigating the risk of OOM occurrences. Finally, we incorporate a runtime coordinator, which synchronizes the interactions between the deadline-driven feedback loop, memory management, and other system components. This component, guided by heuristic analysis and a runtime profiler, dynamically adjusts memory resource reservations to meet hard resource constraints. These three components

working together, enable $\mathtt{R}^3$ to effectively balance the trade-offs of latency, memory, and algorithm performance in a holistic manner, ultimately achieving real-time On-Device DRL training.

Our approach, $\mathtt{R}^3$, has been deployed and tested across a range of DRL benchmarks [197, 25, 17] built upon different underlying frameworks [287, 2, 267, 39] to assess its efficacy with its application extending to three platforms, including a desktop and two embedded devices [268, 270]. Moreover, we highlight the practical usability of $\mathtt{R}^3$ by a further integration with a realistic autonomous car simulator [308]. Empirical results indicate that $\mathtt{R}^3$ achieves efficacy across diverse platforms, ensuring consistent latency performance and timing predictability with minimal overhead. Moreover, $\mathtt{R}^3$ showcases versatility by handling varied optimization goals and adapting to fluctuating systems scenarios. Notable achievements of $\mathtt{R}^3$ include:

- **Cross-platform Efficacy:** Our approach has demonstrated its efficacy in diverse environments across different platforms, including Classic Control [17], Atari [25], and DonkeyCar [197] environments. (Sec. 4.4.2)
- **Latency predictability:** Our proposed method consistently achieves timing predictability and satisfies deadlines across all benchmarks. (Sec. 4.4.2)
- **Practical usability:** Our user-friendly approach can be seamlessly integrated into existing practical complex deep reinforcement learning systems, such as DonkeyCar [197], without extensive modifications. (Sec. 4.4.3)

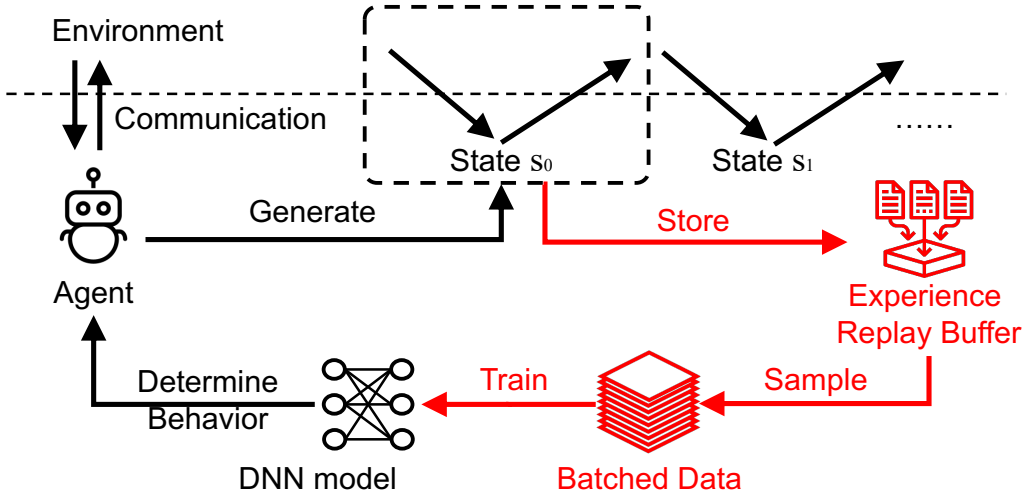


Figure 4.2: An overview of Deep Reinforcement Learning (DRL).

- **Low overhead:** The design and implementation of our method are highly efficient, resulting in negligible overhead on execution time and memory. (Sec. 4.4.4)

- **Versatility:** Our proposed framework exhibits a high degree of versatility, adeptly addressing diverse optimization goals and adapting to varying system scenarios. (Sec. 4.4.5)

4.2 Background and Motivational Case Study

In this section, we present a series of illustrative case studies to elucidate the unique challenges associated with embedded deep reinforcement learning. These examples provide valuable insights into the limitations of existing approaches and highlight the potential pitfalls when naively extending them to address the problem context under consideration.

4.2.1 Characteristics of Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is characterized by a combination of reinforcement learning (RL) algorithms and deep learning techniques (as shown in Figure 4.2),

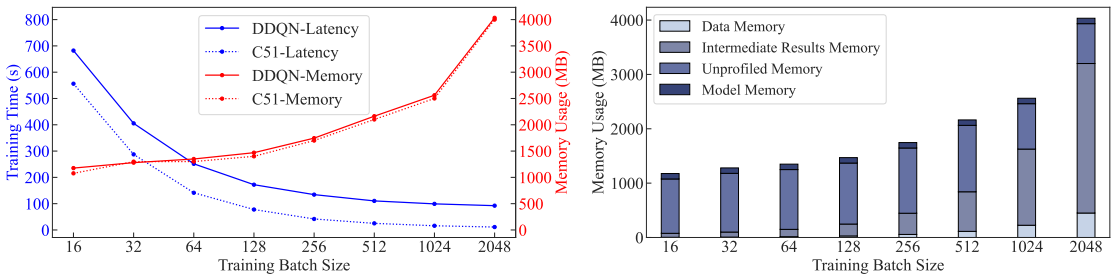
providing remarkable algorithm performance for extensive application prospects, particularly for intelligent robots [12, 59]. However, integrating DRL solutions into real-world autonomous embedded system-driven robots would be more challenging because robots are required to continuously, efficiently, and frequently train and retrain DRL models in order to adapt to new environments with strict timing and resource limitations. In such situations, meeting the real-time training deadlines is not merely a matter of budget adherence but a fundamental requirement to ensure system functionality, maintain up-to-date knowledge, and enhance accuracy. By emphasizing this, we make it clear that the timing sensitivity of DRL training extends beyond the offline workloads and is essential for a range of scenarios, especially those requiring real-time responses and continual learning.

To illustrate this need, consider the following real-world examples: (1) Autonomous navigation robots: DRL models must constantly adapt to dynamic road environments in the context of autonomous navigation robots. They are often required to learn and retrain based on new data and evolving environments. In such cases, meeting deadlines during the training phase ensures that the autonomous navigation robots' decision-making system remains responsive and capable of handling new situations. (2) Search and Rescue Robots: In emergencies like disaster relief, robots need to navigate unpredictable challenges like damaged buildings or shifting debris. Quick training and updates to their DRL models allow them to adapt on-the-fly, ensuring they can effectively locate and assist those in need. (3) Drone Surveillance: Drones face a variety of challenges, from changing terrains in a forest to fluctuating weather conditions. Real-time DRL training ensures drones navigate these challenges efficiently, avoiding mishaps and ensuring effective monitoring.

To thoroughly comprehend the implications of Deep Reinforcement Learning (DRL) in embedded systems, it is essential to focus on two pivotal components (red parts in Figure 4.2) of DRL training: batch execution and replay buffer [224, 256, 343, 131, 318]. Batch execution involves processing multiple data samples in a single operation, fostering effective learning. The replay buffer, on the other hand, serves as a repository for past experiences, which are sampled for conducting minibatch training [256, 248]. These components significantly impact not only the algorithm's application-level performance but also the system-level memory usage and latency [163, 314, 159]. This observation becomes particularly relevant in the context of real-time DRL systems, wherein training workloads consist of multiple time-bound tasks and subtasks, each encapsulating a complete DRL training episode. These tasks can be further divided into several subtasks, i.e., single-step-level minibatch training, with the flexibility for individual scheduling to meet specific end-to-end deadlines.

4.2.2 Balancing Data Parallelism and Memory Usage: A Focus on Training Batch Size

In this subsection, we investigate the trade-off space between parallelism and memory usage in varying DRL training batch sizes, concentrating on its implications for both system-level and application-level performance. All the data in case studies are based on Classic Control [17], which is a set of classical control games for DRL, capable of simulating realistic robots and complex mechanical systems. Classic Control problem emphasizes the importance of control theory concepts, which corresponds to the key control components in robotic scenarios contexts.



(a) Tradeoff for batch size (b) Memory breakdown

Figure 4.3: Balancing data parallelism and memory usage.

As depicted in Figure 4.3, we perform a comprehensive analysis of performance metrics based on the autonomous learning library [267], which offers insights into the impact of batch size on performance. Figure 4.3a highlights the trade-off space for training batch size, where data parallelism, the dominating factor affecting latency, can be improved with a larger training batch size, as long as it does not exceed the device's computing capability. However, as the training batch size increases, memory usage consistently grows for different replay-based DRL algorithms. In Figure 4.3b, we provide a memory breakdown analysis, which shows that data and intermediate result memory requirements increase significantly with larger batch sizes while model parameter memory remains minimal.

Consequently, the optimal batch size depends on the specific algorithm and the available computing resources. A thorough examination and optimization of batch size are necessary for more efficient and effective DRL training in resource-constrained situations.

Observation 1: The trade-off between parallelism and memory usage is vital for optimizing DRL training. Finding the proper balance through batch size adjustments can enhance algorithm performance while minimizing computing resource consumption.

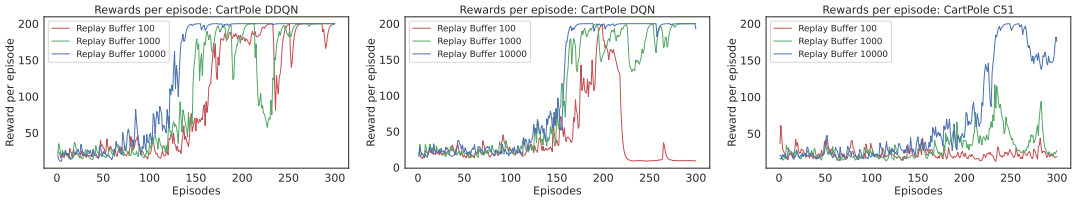


Figure 4.4: The deep reinforcement learning algorithm's performance gradually increases as the replay buffer size increases and convergence of the algorithms becomes faster while the memory usage increases significantly across different algorithms. The experiments are running based on Classic Control [17] CartPole environment. C51 refers to Categorical DQN.

4.2.3 Balancing Algorithm Performance and Memory Usage: A Focus on Replay Buffer Size

In this subsection, we explore the trade-offs between algorithm performance and memory usage in DRL training with respect to replay buffer size. Figure 4.4 illustrates the clear tradeoff space between algorithm performance, exemplified by convergence time, and memory usage in DRL scenarios utilizing replay buffers. As the replay buffer size increases, the algorithm performance generally improves, leading many DRL training approaches to adopt large buffer sizes to boost algorithm performance. Additionally, cumulative rewards gained from DRL training rise with larger replay buffer sizes.

Nonetheless, it is essential to acknowledge that expanding replay buffer sizes also entails increased memory usage, potentially causing out-of-memory (OOM) issues, particularly in memory-constrained embedded devices. Table 4.1 presents default settings for various DRL benchmarks and examines the peak memory allocation for the buffer. Notably, some default settings for widely-used DRL algorithms could trigger OOM in powerful NVIDIA embedded devices.

Table 4.1: Common resource-oblivious DRL training settings may cause serious out-of-memory (OOM) on resource-constrained embedded devices. "✓" refers OOM to happen.

Environment	Algorithm	Buffer Size	Batch Size	Peak Memory	Out-of-Memory	
					Xavier	Orin
Classic Control [17]	DQN [256]	1,000,000	64	404.3MB	✗	✗
	DDQN [343]	1,000,000	64	404.3MB	✗	✗
	C51 [24]	1,000,000	32	404.2MB	✗	✗
Atari [25]	DQN [256]	1,000,000	32	40384.5MB	✓	✓
	DDQN [343]	1,000,000	32	40384.5MB	✓	✓
	C51 [24]	1,000,000	32	40384.5MB	✓	✓
DonkeyCar [197]	DQN [256]	100,000	128	35522.6MB	✓	✓
	DDQN [343]	100,000	128	35122.6MB	✓	✓
	C51 [24]	100,000	64	33329.8MB	✓	✓

Observation 2: Balancing algorithm performance and memory usage for replay buffer size are crucial in DRL training, particularly for memory-constrained embedded devices. A thorough examination of characteristics and trade-offs enables the optimization of performance while meeting practical memory constraints.

4.2.4 Managing Complex Trade-offs under Practical System Scenarios: Balancing in the Three-dimensional Space

In light of the complexities of DRL and the trade-offs explored in the previous subsections, we now focus on the broader challenge of balancing conflicting characteristics in the three-dimensional space of DRL training under practical system scenarios.

As depicted in Figure 4.5, interestingly, patterns about the batch size and replay buffer size in different performance dimensions display vastly different behavior. This stark difference implies that DRL inherently involves conflicting characteristics. Simply co-optimizing some of these aspects may result in suboptimal solutions, underscoring the need for a more comprehensive and integrated approach. Moreover, practical challenges arise when considering resource-constrained embedded devices, where memory constraints

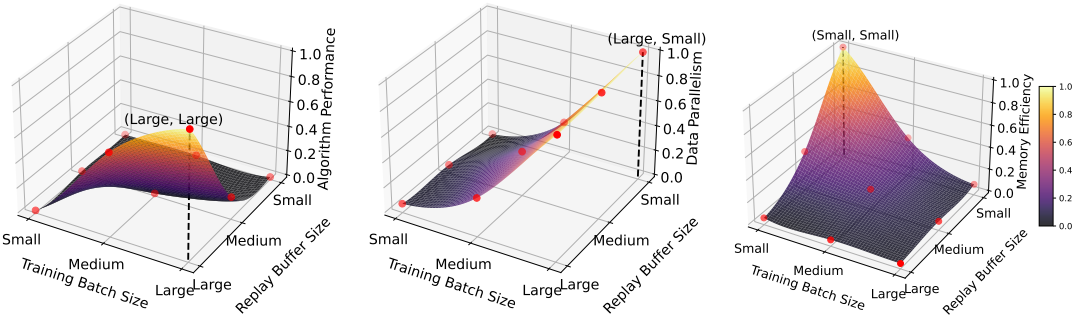


Figure 4.5: **No silver bullet.** Navigating the trade-offs: a three-dimensional analysis of performance metrics in embedded deep reinforcement learning and the challenge of out-of-memory issues. All data are normalized to the interval from zero to one to visualize the contribution of influence better.

may become a significant issue. Recall that simply combining the two 2-dimensional space optimization (as discussed in Sec.4.2.2 and Sec.4.2.3) could exacerbate out-of-memory (OOM) problems.

To address such challenges, it is essential to carefully consider all characteristics in the three-dimensional space and their trade-offs while also taking into account the practical constraints of embedded devices. By adopting such a holistic approach, DRL training can achieve near optimal performance while managing these conflicting characteristics under practical system scenarios.

Observation 3: Managing trade-offs between various DRL training characteristics is a complex task due to multiple conflicting factors. A comprehensive approach considering all characteristics and performance objectvies is necessary, particularly for resource-constrained embedded devices.

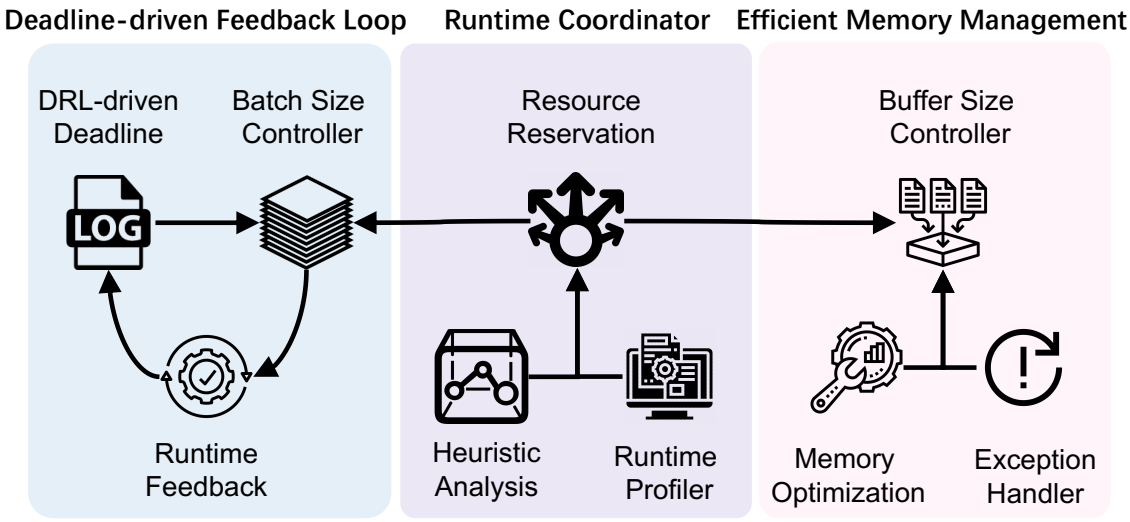


Figure 4.6: An overview of $\mathtt{R}^3$.

4.3 Methodology

4.3.1 System Overview

In response to the unique challenges of **R**eal-time deep **R**einforcement learning for **R**obotics outlined in Sec. 4.2, we propose a comprehensive framework $\mathtt{R}^3$, illustrated in Figure 4.6. This framework encompasses two main components: a deadline-driven feedback loop and efficient memory management, which together facilitate the balance between the two-dimensional trade-off of batch size and replay buffer size. Moreover, a runtime coordinator is introduced to manage the complex three-dimensional trade-off under resource-constrained scenarios, ensuring the adaptation of different performance goals without conflicts. Each component is discussed in further detail below:

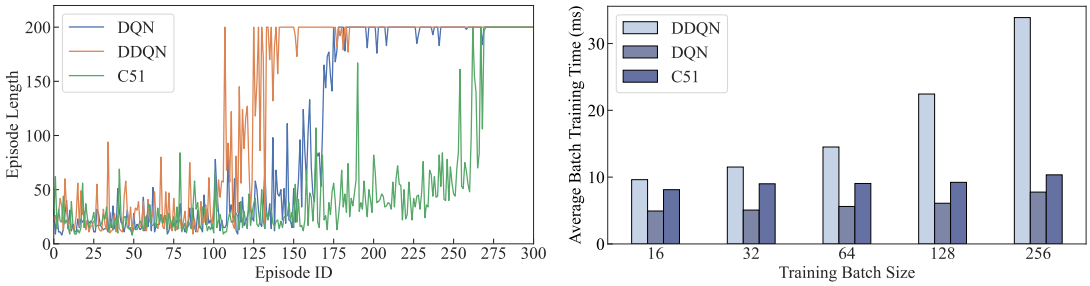
- **Deadline-driven Feedback Loop:** This component aims to balance memory constraints and latency (Sec. 4.2.2). It dynamically assigns an intermediate deadline for each training episode based on the computational dynamics nature of DRL (as illustrated in Figure 4.7).

Subsequently, it determines the optimal batch size that adheres to memory constraints while meeting the subtask deadline for the current training episode. If every subtask completes by its assigned intermediate deadline, the end-to-end deadline will be met.

- **Efficient Memory Management:** In addressing the challenges posed by memory-constrained embedded systems for DRL, efficient memory management is vital. Our proposed solution, corresponding to the trade-off discussed in Sec. 4.2.3, implements task-specific memory optimizations to significantly reduce memory footprint. This enables larger replay buffer sizes for enhanced algorithm performance in real-time embedded DRL. Furthermore, we design an exception handler to eliminate unexpected out-of-memory (OOM) occurrences. Based on these underlying subcomponents, the buffer size controller can decide on demand how to assign the replay buffer size.

- **Runtime Coordinator:** The runtime coordinator is essential for the seamless operation of $\mathtt{R}^3$, ensuring real-time DRL by synchronizing the interactions between the deadline-driven feedback loop, memory management, and other system components. Utilizing analytical modeling and a runtime profiler, the coordinator dynamically adjusts the resource reservation strategy in accordance with the actual system resource constraints. This guarantees that the runtime coordinator expertly maintains the delicate balance of factors required for achieving optimal DRL agent training.

By coherently integrating these three components, $\mathtt{R}^3$ provides a high-performance solution for training deep reinforcement learning agents in real-time. This enables rapid adaptation to dynamic and complex environments.



(a) Workload-induced dynamics (b) Batching-induced dynamics

Figure 4.7: The computational dynamic nature of DRL emphasizes the need for dynamic intermediate deadline assignments.

4.3.2 Deadline-driven Feedback Loop

As emphasized in the preceding case study (Sec. 4.2.2), achieving a balance between data parallelism and memory usage is vital for optimizing embedded DRL. Our goal is to meet end-to-end deadlines within hard memory constraints by adjusting the batch size. Intuitively, one possible approach is to partition subtasks and assign an appropriate intermediate deadline to each subtask. However, before designing a corresponding solution, it is necessary to consider how to partition the DRL workload and assign the intermediate deadline.

Although DRL can be naturally divided into episodes, the computational dynamics of DRL workload add complexity to the assignment of reasonable intermediate deadlines. As illustrated in Figure 4.7, we leverage an example of DRL classic control [17] to show this point in two folds. At the DRL workload level, the computational cost of each natural computational subdivision unit (episode) varies over time, as depicted in Figure 4.7a. The episode length was short at the beginning of DRL training; as the training progressed, the episode length gradually increased. Furthermore, examining finer-grained computation reveals that, since we aim to dynamically adjust the batch size, the computational cost of each batch (step) is also distinct, as depicted in Figure 4.7b. This computational nature presents

a further challenge to intermediate deadline assignments, which is hard to model without timing-consuming profiling. This implies that straightforwardly conducting a modeling-based optimization for all three dimensions may be infeasible across different DRL algorithms.

To address such a challenge, we propose an intuitive, practical deadline driven feedback loop mechanism that adapts to the varying computational dynamics of DRL workloads. The intuition is to directly adjust the batch size based on progress tracking without explicitly assigning intermediate deadlines. Specifically, we design to periodically assess the remaining time to dynamically adjust the training batch size incorporating temporal feedback information and thus meet the end-to-end deadline. Formally, as the DRL training process unfolds, two critical constraints often imposed are a data budget (B) and an end-to-end deadline (D). We keep track of the elapsed training time (t_i) and consumed budget (s_i) at the start of each episode i. We employ two trackers to monitor the progress of the DRL process: a timing tracker evaluates the ratio of elapsed training time to the given deadline, and a data tracker monitors the proportion of the consumed budget to the total budget. These ratios, denoted as a and b respectively, are given by $a = t_i/D$ and $b = s_i/B$.

These ratios serve as comparative indicators to guide the adjustment of the DRL process. If $a > b$, it indicates that the DRL training is lagging behind, necessitating acceleration by multiplying the current batch size by a scale factor c. Conversely, if $a < b$, it suggests the training is on track, thereby allowing for deceleration by dividing the batch size by the scale factor c. However, to maintain the stability of the DRL process, it is crucial to prevent the batch size from becoming excessively small or large. [256, 318, 224] As such, we confine the batch size within a specified range as per established DRL practices as follows:

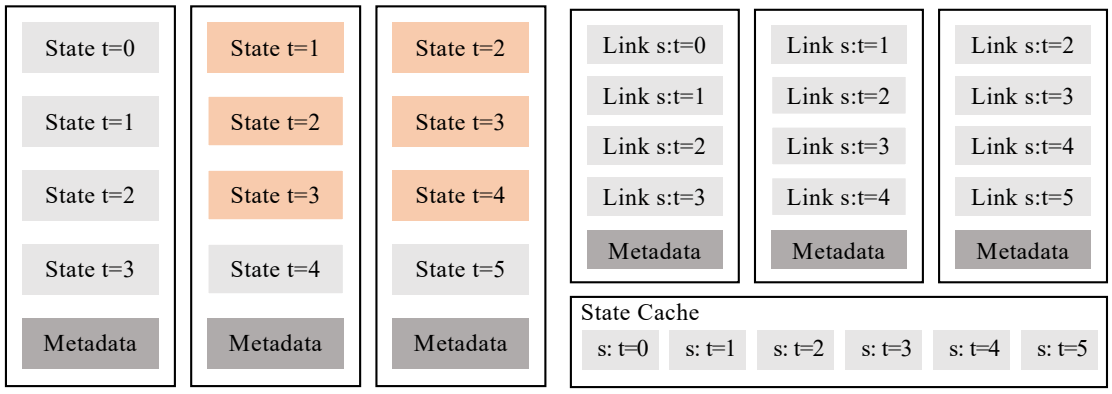
$$b_{i+1} = \min(\max(b_i, b_{\min}), 4 * b_{\min}). \tag{4.1}$$

Ultimately, it is critical to ensure that the DRL process complies with the hard memory constraint given by:

$$b_{i+1} = \min\left(b_{i+1}, M_{batch} \cdot \frac{b_{base}}{M_{base}}\right). \tag{4.2}$$

Two key components in this context are M_{base} and M_{batch}, both playing essential roles in memory reservation for batch execution during the scaling of minibatch size. The M_{base} represents the memory reserved for batch execution considering a base minibatch size, denoted as b_{base}. This reservation is determined by the initial episode hyperparameter configuration, thereby serving as a reference point for subsequent adjustments. Conversely, M_{batch} signifies the memory currently reserved for batch execution, a value that can dynamically vary in accordance with system constraints and requirements.

To further enhance the responsiveness, we integrate granular step-level adjustments within our feedback loop, enabling meticulous progress tracking at each training step for precise monitoring of elapsed time and budget expenditure. The detailed structure of this fine-grained deadline feedback loop aligns with the episode-level setup, facilitating seamless implementation without extensive modifications. Although this granular control promotes rapid adjustment to computational dynamics, it potentially introduces additional overhead that may affect training performance. Hence, the selection between episode-level



(a) Dummy replay buffer (b) Efficient replay buffer

Figure 4.8: Replay buffer memory optimization. The orange color indicates redundant data in the dummy replay buffer.

and step-level adjustments is contingent upon specific DRL task requirements and available resources.

4.3.3 Efficient Memory Management

As demonstrated in Sec. 4.2.3, a larger replay buffer enhances algorithm performance but also demands more memory. The performance of DRL algorithms relies heavily on the quantity of data, which in turn depends on available memory resources. This is particularly relevant in resource-constrained embedded systems, where memory resources are limited and must be meticulously managed to avoid out-of-memory (OOM) issues. To address this challenge, we propose an efficient memory management scheme that contains memory optimization, an exception handler, and an application-level buffer size controller.

First, to target real-time embedded systems with stringent memory constraints, we propose underlying system-level DRL-task-specific optimizations for efficient memory management. First, we conduct a finer-grained analysis of the replay buffer and observe that the current caching mechanism of DRL training is inefficient. As illustrated in Figure 4.8a,

each data frame is stored in a tuple data structure (states, metadata) in the replay buffer, recording multiple time steps. It has to be admitted that such a dummy implementation can avoid frequent memory accesses and thus achieve higher parallelism, however, it would result in significant memory redundancy due to duplicated data shown in orange color. Ideally, each time step should maintain only one copy of data in the replay buffer. Based on this observation, we design and implement a simple yet efficient caching method that significantly reduces the memory footprint of the replay buffer, allowing for larger buffer sizes within memory-constrained embedded devices. As shown in Figure 4.8b, we use a practical example of the Atari Breakout benchmark [256] to illustrate the ideal memory efficiency of our method. For each data frame of the dummy replay buffer implementation, each data frame stores 4 consecutive state information (i.e., 4 moments of RGB images with size (210,160), occupying size 4x210x160x3=403,200 bytes). In contrast, the metadata, containing action, reward, and done flag, occupies very little memory space (9 bytes) in total. Therefore, we use a contiguous memory for storing state information, while only storing soft links (4 bytes for each link) in the data frame. When the corresponding data frame is required, the corresponding tuple data is filled on demand. Our efficient replay buffer can ideally achieve about 75% less memory assumption to support the same length as a dummy replay buffer as in Figure 4.8, because in most data frames, three out of four pieces (orange) of state information are replaced with lightweight soft links.

Let M_{replay} represent the total memory reserved for the replay buffer, S_{state} represent the size of one state, $S_{metadata}$ represent the metadata size, and N_{frames} represent the number of frames stored in a single data frame (in the Atari Breakout example, $N_{frames} =$

4). The replay buffer size r_i for training episode i can be calculated as Eq. 4.3, where S_{link} denotes the size of a soft link, which is used to reference the state information in the contiguous memory:

$$\begin{aligned} r_i = max\{M_{replay}, (N_{frames} + i - 1) * S_{state} + \\ (N_{frames} - 1) \cdot i \cdot S_{link} + i \cdot S_{metadata}\} \end{aligned} \tag{4.3}$$

Directly applying this efficient caching method could reduce memory redundancy; however, it could introduce substantial overhead since it necessitates more memory access, potentially harming data parallelism in DRL training. Inspired by recent work [175], we optimize the training process with non-blocking data prefetching to hide latency. These task-specific optimizations enable real-time embedded DRL training to achieve larger replay buffer sizes for improved algorithm performance while minimizing side effects on data parallelism. To handle the possibility of unexpected out-of-memory (OOM) issues caused by memory interference or other unforeseen circumstances, we propose to design an exception handler mechanism. This mechanism can monitor memory usage during runtime and promptly react when an OOM exception is detected. The handler can temporarily scale down the memory reservations for both batch execution and replay buffer, which allows the system to recover gracefully from the OOM situation. Furthermore, it can record the OOM events and adjust the memory allocation strategy accordingly, ensuring the system remains robust under memory interference.

Specifically, we show part of the code example of a standard DDQN training step (as shown in Algorithm 2). We use a multi-threading technique to hide the latency of data

Algorithm 2 An example of DRL training

```
1: function Train
2:     if ShouldTrain() then
3:         (states, actions, rewards, next_states, weights) ← ReplayBuffer.Sample(b);
4:         values ← Q(states, actions);
5:         next_actions ← arg max_a Q_no_grad(next_states);
6:         targets ← rewards + γ · Q_target(next_states, next_actions);
7:         loss ← Loss(values, targets, weights);
8:         Q.Reinforce(loss);
9:         ReplayBuffer.Update();
10:    end if
11: end function
```

prefetching (line 4) and thus achieve a better training step latency. Based on the above optimizations, the buffer size controller could calculate the current maximum replay buffer size based on the given memory limit and dynamically allocate or free memory at runtime.

4.3.4 Runtime Coordinator

The Runtime Coordinator in $\mathtt{R}^3$ is a pivotal component devised to address the intricate triadic trade-off between algorithm performance, memory efficiency, and data parallelism (as outlined in Sec. 4.2.4). This component serves to orchestrate the execution of various system aspects, including the deadline-driven feedback loop and efficient memory management, thereby enabling smooth real-time DRL training. The coordination is executed through heuristic analysis, taking into consideration the outputs from the aforementioned system components alongside system-level resource constraints and performance objectives.

Memory reservations M_{batch} and M_{replay} are adjusted using a heuristic formulation that factors in the system's current performance and memory constraints. For episode i, the episode runtime is denoted as t_i, and the cumulative reward is R_i. Two scaling factors, α and β, are employed to fine-tune the memory reservations based on the system's performance:

$$\alpha = \frac{\gamma t_i}{\sum_{j=1}^{\gamma} t_{i-j}}$$
$$\beta = \frac{\gamma R_i}{\sum_{j=1}^{\gamma} R_{i-j}} \tag{4.4}$$

Here, t_i is the runtime of episode i, R_i is the cumulative reward at the i-th episode, γ is a hyperparameter, and $R_{i-\gamma}$ is the cumulative reward γ steps prior to the i-th episode. γ's role is to control the reward comparison span. By modulating γ, the algorithm can be adjusted to focus on short-term or long-term reward trends, influencing the training process and potentially enhancing overall performance. α and β track latency and algorithm performance dynamically. To this end, the memory reservations for batch execution and replay buffer are then updated based on α and β :

$$M_{batch}^{new} = M_{batch} \cdot (1 + \max(\alpha - 1, 0) \cdot (1 - \min(\beta, 1)))$$
$$M_{replay}^{new} = M_{replay} \cdot (1 + \min(\alpha, 1) \cdot \max(1 - \beta, 0)) \tag{4.5}$$

Here, M_{batch}^{new} and M_{replay}^{new} represent the updated memory reservations for batch execution and replay buffer, respectively. This adjustment mechanism increases batch memory if the episode runtime is shorter than the subtask deadline and the cumulative reward is increasing relative to the training loss, while reducing replay buffer memory. To ensure total memory usage does not exceed system constraints, the sum of M_{batch}^{new} and M_{replay}^{new} must be checked against the available memory budget. If it exceeds the limit, memory reservations are proportionally scaled down to fit within the constraints. Let's denote M as the total available memory. We want to proportionally distribute M between M_{batch} and M_{replay}. First, we calculate the proportion of each reservation to the total desired memory,

i.e., the sum of M_{batch}^{new} and M_{replay}^{new}. Then, we allocate the memory in proportion to these ratios. The resulting equations are:

$$\begin{aligned} M_{batch}^{final} &= M \cdot \frac{M_{batch}^{new}}{M_{batch}^{new} + M_{replay}^{new}} \\ M_{replay}^{final} &= M \cdot \frac{M_{replay}^{new}}{M_{batch}^{new} + M_{replay}^{new}} \end{aligned} \tag{4.6}$$

Here, M_{batch}^{final} and M_{replay}^{final} are the final memory reservations for batch execution and replay buffer, respectively. These equations ensure that the total memory used is exactly M, and that it is distributed in proportion to the desired reservations for batch execution and the replay buffer.

Let's consider different scenarios to illustrate the behavior of the memory reservation system:

- **Timing Inefficiency**: When $\alpha > 1$ (indicating an episode runtime exceeding its subtask deadline), there is an increase in the batch memory allocation (M_{batch}^{new}). This allocation is unaffected by performance ($\beta \geq 1$), as efficient operation negates the need for additional batch processing memory.

- **Performance Deterioration**: An increase in replay buffer memory allocation (M_{replay}^{new}) is observed when $\beta < 1$, signifying performance deterioration. Temporal efficiency ($\alpha \leq 1$) neutralizes the adjustment to batch memory (M_{batch}^{new}), as an efficient runtime eliminates the need for more replay buffer memory.

- **Maintaining Optimal Performance**: Under optimal conditions of both runtime and performance ($\alpha \leq 1$, $\beta \geq 1$), the memory reservations for batch execution and replay

buffer remain constant. This indicates the system is running efficiently, thus no need to adjust memory allocations.

- **Balanced Trade-off**: In the event of both inefficient runtime ($\alpha > 1$) and performance deterioration ($\beta < 1$), there is a compensatory increase in both M_{batch}^{new} and M_{replay}^{new}. Despite this, the overall memory usage must remain within system constraints, necessitating a proportional reduction in memory reservations if required.

These illustrative cases demonstrate the adaptability of the memory reservation system in response to different performance conditions. By dynamically adjusting M_{batch} and M_{replay}, the system can address diverse challenges and maintain a balanced trade-off between timing performance and algorithm performance, all within the constraints of the available memory budget.

4.3.5 Algorithm Details

The overall workflow of the $\texttt{R}^3$ framework is presented in Algorithm 3. For each DRL training episode, the batch size is initially determined in accordance with Eq. 4.2 in Sec. 4.3.2. Subsequently, the replay buffer size is computed by Eq. 4.3 in Sec. 4.3.3. Following this, a series of replay buffer size adjustments, memory garbage collection, and synchronization operations are executed on-demand to maintain system correctness. Next, the coarse-grained algorithm is trained according to the standard process, while the fine-grained algorithm slightly adjusts the batch size of each training step. Training progress is checked afterward to avoid overspending the training budget. It's worth mentioning that inspired by early stopping techniques [375], once the training meets the algorithm performance goal(K consecutive

Algorithm 3 $\mathtt{R}^3$ framework

Input: hyperparameter m; hyperparameter γ; end-to-end deadline D; training budget B; maximal training episode N; Data frame information I; training budget C;
Initialize memory reservation $\{M_{batch}, M_{replay}\}$ with m;
Initialize spent training cost c with 0;
for each episode $i \in N$ **do**
    Batch size $b_i \leftarrow$BatchSizeControl(D, B, M_{batch});
    Replay size $r_i \leftarrow$ ReplayControl(I, M_{replay});
    if $i > 0$ and $r_i < r_{i-1}$ **then**
        ShrinkReplayBuffer(r_i);
        GarbageCollection();
    else
        ExpandReplayBuffer(r_i);
    end if
    Synchronization();
    if IsCoarseGrained() **then**
        Training Logs $s \leftarrow$ VanillaTraining();
    else
        Training Logs $s \leftarrow$ DynamicTraining();
    end if
    Update spent training cost $c \leftarrow c + s[cost]$;
    if $c >= C$ or meet early exit condition **then**
        exit();
    end if
    Update memory reservation $\{M_{batch}, M_{replay}\} \leftarrow$ RuntimeCoordinator $(s, \gamma, \{M_{batch}, M_{replay}\})$
end for

episodes rewards consistently no smaller than the targeted algorithm performance R^*), the training process will exit early to minimize latency. Lastly, based on the acquired training logs, the runtime coordinator dynamically adjusts the memory resources allocated to batch execution and replay buffer management by Eq. 4.6. This enables the $\mathtt{R}^3$ framework to effectively balance the trade-offs among data parallelism, memory usage, and algorithm performance, resulting in a more efficient and adaptive system.

Table 4.2: Hardware platforms used in our experiments.

	Desktop	**NVIDIA AGX Xavier**	**NVIDIA AGX Orin**
CPU	Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz	8-core NVIDIA Carmel Armv8.2 64-bit CPU	8-core Armv8.2 Cortex-A78AE 64-bit CPU
GPU	NVIDIA GTX 3060	NVIDIA Volta GPU	NVIDIA Ampere GPU
Memory	DDR4 16GBx2	16GB LPDDR4x	32GB LPDDR5
Storage	1TB SSD	32GB eMMC	64GB eMMC

4.4 Evaluation

4.4.1 Experimental Setup

Testbeds. We choose three different platforms aiming to demonstrate the cross-platform compatibility of our design as shown in Table 4.2. These platforms comprise one desktop configuration along with two NVIDIA embedded platforms, motivated by the widespread adoption of NVIDIA hardware in deployed autonomous systems, particularly in the domains of autonomous driving [187, 195] and robotics [293, 269, 271, 273].

Benchmarks. To thoroughly evaluate our solution, we have selected three deep reinforcement learning benchmark environments, as detailed in Table 4.3. These environments encompass various application areas and are widely used in the research community. Note that we evaluate two representative DRL benchmarks, Atari [256] and Classic Control [17], integrated with widely used autonomous-learning-library(ALL) [267]. We use these Classic Control and Atari to evaluate the overall effectiveness and versatility of $\mathtt{R}^3$. Additionally, to demonstrate the robustness of our solution in real-world practical scenarios, we further conduct a practical case study by integrating our approach into autonomous navigation scenarios, empowered by a high-resolution simulator DonkeyCar [197]. In our experiments,

Table 4.3: Deep reinforcement learning benchmark environments used in our experiments.

Benchmark ID	Description
Atari [256]	A well-known environment for video game-based deep reinforcement learning research, featuring a collection of Atari 2600 games.
Classic Control [17]	A set of classical control games for deep reinforcement learning research, capable of simulating robots and complex mechanical systems.
DonkeyCar [197]	A practical simulated deep reinforcement learning environment for training autonomous driving agents and deploying to real-world platforms.

we assess the performance of three well-known and widely implemented DRL algorithms: DQN [256], DDQN [343], and C51 [24]. These algorithms provide a comprehensive comparison of our solution's efficacy and adaptability across different learning methods and application domains. We set hyperparameter b_{min} strictly following the MAX-A preset settings, m refers to each platform's maximal available memory, γ to 4. We set hyperparameter maximal episode number N to 10000 for all environments, data budget the same as MAX-A preset setting, and early exit parameter K to 10 for all benchmark, R^* to 200, 300, and 1000 for Classic Control, Atari, and DonkeyCar.

Metrics. This paper evaluates two main types of metrics. The first set reflects latency predictability, which evaluates the throughput by end-to-end latency and real-time performance indicators using task deadline miss rates. Moreover, end-to-end deadlines for DRL training are based on the worst-case execution time (WCET), and task deadlines are based on the proportional intermediate deadline assignment. The second set of metrics corresponds to algorithm performance, measuring widely adopted maximal and average cumulative rewards.

Baselines. We compare $\mathtt{R}^3$ to the following approaches:

- **MAX-A**: To maximize the algorithm performance, we directly adopt preset hyperparameters in the autonomous learning library (ALL) [267], which is an object-oriented novel deep reinforcement learning (DRL) library designed for PyTorch [288]. The creators have incorporated high-quality reference implementations of modern DRL algorithms and provided ease of use API, and pre-defined hyperparameter sets for best algorithm performance.

- **MAX-P**: MAX-P, or maximal data parallelism, denotes a high-efficiency mode that enables concurrent data processing to achieve peak performance. For fair comparisons, we implement MAX-P strictly following the implementation of ALL [267] to exploit the computational resources and maximize throughput in DRL training scenarios.

- $\texttt{R}^3$: Our proposed solution is implemented at various DRL training granularities: $\texttt{R}^3_{episode}$ for coarse-grained episode-level and $\texttt{R}^3_{step}$ for fine-grained step-level implementation.

Scheduling Policy. In On-Device DRL training, there is only one DRL training job running at a time because the training task is extremely resource-demanding for both computation and memory resources, and the GPU resource is often limited (e.g., typically only one GPU device available for most desktop and embedded settings). Therefore, such scenarios cannot be optimized through multi-task scheduling. We thus applied FIFO scheduling in all cases. Deadline miss rates are calculated task-wise based on the proportional intermediate deadline assignment policy.

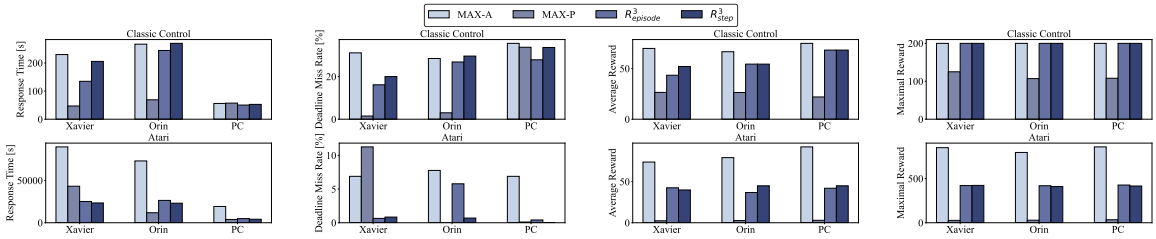


Figure 4.9: Overall effectiveness of $\mathtt{R}^3$ on the C51 algorithm evaluated on four different resource-constrained intelligent robotic systems. $\mathtt{R}^3$ auto balances throughput, timing correctness, and algorithm performance.

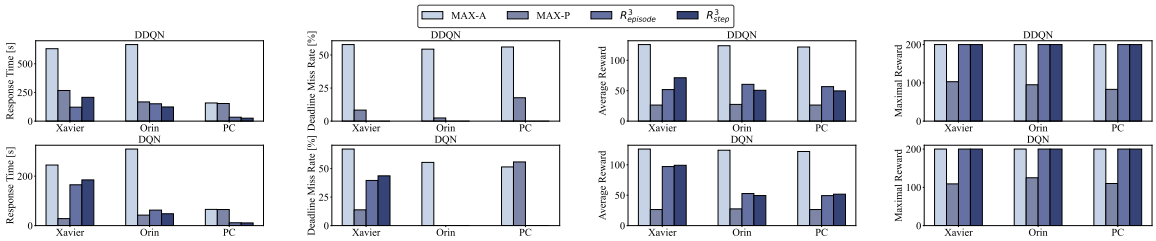


Figure 4.10: $\mathtt{R}^3$ evaluated on more different DRL algorithms on the Classic Control benchmark.

4.4.2 Overall Effectiveness

We measure the overall effectiveness of $\mathtt{R}^3$ across three platforms. Since our design concerns latency predictability and algorithm efficacy, we measure both constraints and compare against strong baselines based on widely used DRL evaluating benchmarks autonomous-learning-library (ALL) [267].

Latency predictability. Figure 4.9 shows timing performance comparisons. In the Classic Control benchmark, $\mathtt{R}^3$ outperforms MAX-A, with average execution time improvements of 39.3%, 8.2%, and 9.0% on Xavier, Orin, and PC. This improvement is attributed to our memory management, which increases memory access slightly but enhances timing optimization. On the PC, latencies are similar, maybe due to the high computation capabilities of desktop GPU meeting the relatively small benchmark Classic Control. Other platforms show notable timing differences, indicating the efficacy of our solution on embedded

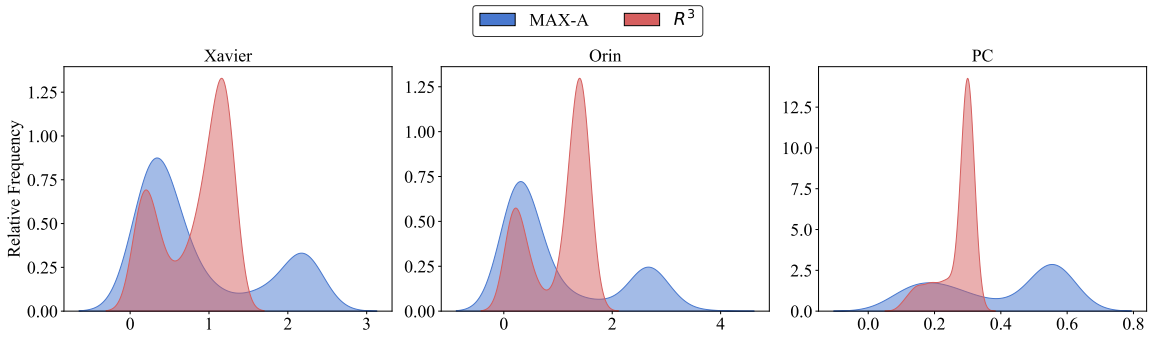


Figure 4.11: Episode-level response time histogram across different platforms. $\mathtt{R}^3$ ensure significantly better timing correctness than MAX-A qualitatively.

systems. The deadline miss rate further emphasizes this, with $\mathtt{R}^3$ improving over MAX-A by 15.0%, 1.7%, and 7.8% on Xavier, Orin, and PC. On the Atari benchmark, $\mathtt{R}^3$ balances timing correctness and algorithm performance, outperforming MAX-A by 71.9% and 5.8%. However, MAX-P on Xavier is slower, possibly due to the Atari benchmark's longer duration and Xavier's older Linux kernel affecting CPU scheduling. Figure 4.11 breaks down response times, showing $\mathtt{R}^3$ consistently outperforms MAX-A in timeliness. The empirical results demonstrate that our proposed $\mathtt{R}^3$ qualitatively has significantly better timeliness than MAX-A, avoiding potentially high-tailed latency.

Algorithm Efficacy. The last two columns of Figure 4.9 display algorithm performance comparisons. On the Classic Control benchmark, $\mathtt{R}^3$ almost obtains a very close to optimal algorithm performance (MAX-A) by on average 100.0% and 82.7% on average rewards and maximal rewards, while outperforming MAX-P by a large margin on average by 76.5% and 133.6% on average rewards and maximal rewards, respectively. Note that the theoretical upper bound of maximal reward for Classic Control is 200; both MAX-A and $\mathtt{R}^3$ consistently reach this maximal value during training. This quantitatively validates the effectiveness of our co-optimization, especially for training a usable DRL algorithm under strict time

and memory constraints. On the Atari benchmark, R^3 auto-balances the timing correctness and algorithm performance. Specifically, R^3 achieves on average by 49.7% and 50.2% on the average rewards and maximal rewards than MAX-A, but largely outperforms on average by 1521.1% and 1297.8% on the average rewards and maximal rewards than MAX-P. This supports the validity R^3 and further evidences the resilience of our design. Furthermore, to demonstrate quantitatively the algorithm performance between different methods, we plot the per-episode cumulative rewards during the training process, as shown in Figure 4.12. It can be observed that MAX-P runs out of data budget too early (green curves), so the training lasts only a very small number of episodes and therefore leads to very poor algorithm performance. In contrast, our proposed R^3 selects proper bath size in the batch execution, thus reaching algorithm performance consistently better than MAX-P and nearly approaching empirically optimal MAX-A.

Evaluated on more DRL algorithms. Our method, R^3, was further evaluated on two additional algorithms (refer to Figure 4.10). Specifically, for the DDQN algorithm, R^3 performs better than MAX-A for execution time by 73.9%, 79.4%, and 81.1% on Xavier, Orin, and PC. Also, R^3 performs better than MAX-P for maximal rewards by 94.2%, 110.5%, and 140.1% on Xavier, Orin, and PC. Additionally, R^3 performs better than MAX-P for average rewards by 94.2%, 133.1%, and 141.0% on Xavier, Orin, PC. Additionally, for the DQN algorithm, R^3 performs better than MAX-A for execution time by 133.1%, 102.0%, and 101.3% on Xavier, Orin, and PC. Also, R^3 performs better than MAX-P for maximal rewards by 83.5%, 83.8%, and 60.0% on Xavier, Orin, and PC. Additionally, R^3 performs better than MAX-P for average rewards by 256.8%, 84.3%, and 68.3% on Xavier, Orin,

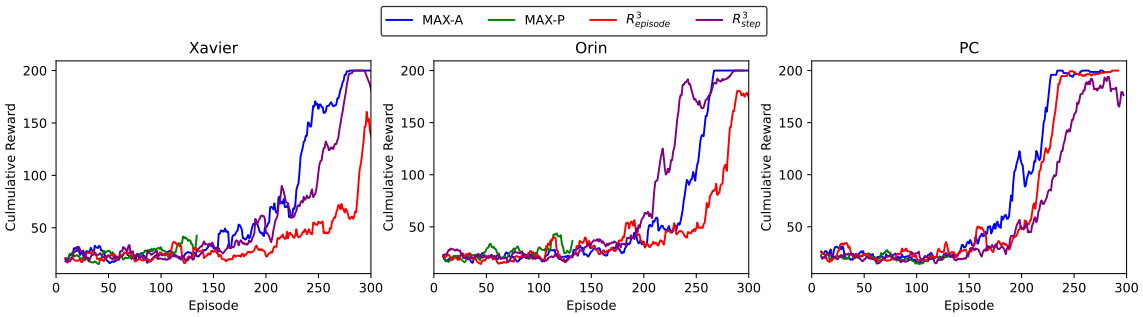


Figure 4.12: Episode-level reward curves across different platforms. The algorithm performance of R^3 is consistently better than MAX-P and close to MAX-A (empirical optimal).

Table 4.4: Deadline miss rate on DDQN and DQN algorithms. The best values are in bold.

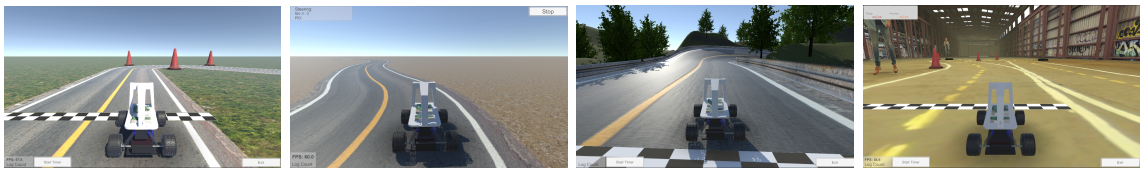
	Solution	Xavier	Orin	PC
DDQN	MAX-P	8.3%	2.4%	17.7%
	MAX-A	57.9%	54.5%	56.0%
	R$^3_{episode}$	**0.0%**	**0.0%**	**0.0%**
	R$^3_{step}$	**0.0%**	**0.0%**	**0.0%**
DQN	MAX-P	**13.8%**	**0.0%**	55.6%
	MAX-A	67.1%	55.4%	51.4%
	R$^3_{episode}$	39.6%	**0.0%**	**0.0%**
	R$^3_{step}$	43.6%	**0.0%**	**0.0%**

and PC. Our efficient memory optimization yields supreme latency performance of R^3, even outperforming MAX-P in DDQN and DQN. Note that R^3 exhibited a markedly low deadline miss rate for both DRL algorithms, suggesting its adaptability and efficacy in enhancing real-time performance (see Table 4.4).

Cross-platform overall effectiveness: R^3 effectively addresses all challenges outlined in Sec. 4.2 for resource-constrained intelligent robotic systems. It auto-balances throughput, timing correctness, and algorithm performance in DRL training across different platforms.

4.4.3 A Practical Case study: Autonomous Navigation via DRL

To empirically evaluate the effectiveness of our proposed approach, R^3, in real-world DRL applications for robotics, we conducted a comprehensive case study centered



(a) Generated track (b) Generated road (c) Mountain track (d) Warehouse track

Figure 4.13: DonkeyCar [197] driven high-resolution autonomous navigation simulator for evaluating DRL algorithm.

on autonomous navigation. We selected DonkeyCar [197] as the platform for this case study, a high-resolution environment based on the x86 Unity3D autonomous navigation simulator [132], as displayed in Figure 4.13. DonkeyCar was chosen due to its relevance in investigating the deployment of DRL-based control systems in real-world robotics contexts. Several real-world autonomous driving cars are built upon DonkeyCar library [44, 308] based on TensorFlow framework [2]. To successfully incorporate DRL training into this platform, we implement a highly optimized networking module that facilitates the seamless execution of DRL training natively on advanced embedded systems (Xavier and Orin).[2] The integration of these robust systems underscores the real-world feasibility and performance of $\texttt{R}^3$ in robotic contexts, highlighting its potential for wider adoption in various robotic applications.

DonkeyCar [197] processes 60 FPS high-resolution RGB images as streaming camera inputs, raising a significant memory challenge when conducting On-Device DRL training. Table 4.5 presents the results of the DDQN algorithm implementation on DonkeyCar, showcasing a range of evaluation metrics. Specifically, we report latency, FPS to describe throughput, data budget consumption percentage to describe the relative percentage of

[2]Due to specific compilation challenges, we were unable to prepare the DRL training library for the x64 PC environment. Table 4.5 does not include the evaluation of DonkeyCar on the PC.

Table 4.5: Quantitative results on DonkeyCar simulator. DPR implies data processing rate, Reward$_{max}$ implies maximal episode rewards, and Reward$_{avg}$ implies average episode rewards. * implies OOM occurs.

Platform	Solution	Latency(↓)	DPR(↑)	Budget(↑)	R_{max}(↑)	R_{avg}(↑)
Xavier	MAX-P	64.7*	1423.8	0.2%	116.6	59.8
	MAX-A	3619.4*	742.0	41.9%	1343.2	157.3
	$R^3_{episode}$	7525.9	850.4	100.0%	532.7	79.1
	R^3_{step}	7429.1	861.5	100.0%	544.1	80.2
Orin	MAX-P	4918.0	1301.3	100.0%	29.4	7.3
	MAX-A	8341.0*	653.7	85.2%	1303.8	156.9
	$R^3_{episode}$	8416.18	760.4	100.0%	598.6	67.0
	R^3_{step}	8205.23	780.0	100.0%	588.5	70.2

training completion, and maximal rewards R_{max} and average rewards R_{avg} to describe algorithm performance.

Results demonstrate that R^3 adeptly balances both latency predictability and algorithm efficacy without incurring out-of-memory (OOM) problems on both platforms; while OOM occurs for MAX-A on both platforms and MAX-P on Xavier. R^3's data processing rate (DPR) exceeds MAX-A by an average of 16.6%, while R^3 lags behind MAX-P by an average of 40.4%. This is because R^3 provides a smaller batch size than MAX-P. In terms of algorithm performance, R^3 significantly outperforms MAX-P, with maximal and average rewards better by 872.9% and 1140.3% in the challenging autonomous navigation task. Still, R^3 trades off certain algorithm performance by 57.2% and 52.8% on maximal and average rewards, mainly due to reserving smaller memory for replay buffer compared to MAX-A.

A deeper dive into R^3's memory efficiency, as shown in Figure 4.14, reveals its strengths. MAX-A heavily consumes memory in data storage due to high-res RGB images in its replay buffer. As this buffer expands, memory use grows uncontrollably, causing OOM errors. In contrast, MAX-P's memory consumption spikes in batch execution, storing

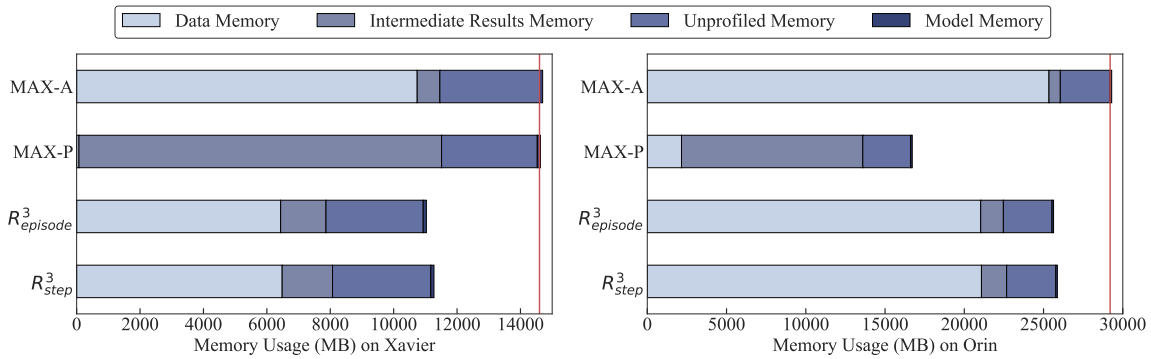


Figure 4.14: Memory breakdown on DonkeyCar evaluated on two embedded devices. The red line represents the OOM point.

numerous intermediate activations, leading to OOM errors on Xavier. However, $\mathtt{R}^3$ optimally manages memory in both data storage and batch execution. Its innovative replay buffer design allows for a larger size within the same memory constraints, enhancing algorithmic output. $\mathtt{R}^3$ also dynamically adjusts batch sizes during training, striking a balance between timing requirements and memory constraints, ensuring effective DRL training within these boundaries.

Pratical usability: This case study demonstrates the integration effectiveness of $\mathtt{R}^3$ on practical robotic applications. $\mathtt{R}^3$ significantly reduces the memory footprint by efficient memory optimization, ensuring the usability of $\mathtt{R}^3$ in embedded robotic DRL training scenarios.

4.4.4 Overhead Analysis

We further break down the overhead of $\mathtt{R}^3$ for different platforms regarding execution time and memory usage.

Execution Overhead. Table 4.6 details the average execution overhead for each component: deadline-driven feedback loop (Feedback), replay memory management (MM), and runtime

Table 4.6: Detailed average runtime execution overhead (s) and percentage of applying $\mathtt{R}^3$ for each benchmark.

Platform	Benchmark	Feedback	MM	Coord
Xavier	Classic Control	0.30 (0.15%)	0.11 (0.05%)	0.07 (0.04%)
	Atari	3.68 (0.01%)	10.22 (0.01%)	0.16 (0.01%)
	DonkeyCar	0.35 (0.01%)	41.95 (0.56%)	0.09 (0.01%)
Orin	Classic Control	0.14 (0.05%)	0.29 (0.11%)	0.06 (0.02%)
	Atari	1.55 (0.03%)	11.04 (0.02%)	0.08 (0.01%)
	DonkeyCar	0.22 (0.01%)	115.8 (1.41%)	0.04 (0.01%)
PC	Classic Control	0.05 (0.09%)	0.10 (0.19%)	0.01 (0.02%)
	Atari	0.99 (0.01%)	8.93 (0.02%)	0.04 (0.01%)
	DonkeyCar	N/A	N/A	N/A

Table 4.7: Detailed memory overhead and percentage of applying $\mathtt{R}^3$ for each benchmark.

	(a) Overhead upon Framework			(b) Overall Overhead Ratio		
	Feedback	MM	Coord	Xavier	Orin	PC
Classic Control	5 KB	160 KB	3 KB	0.001%	0.001%	0.001%
Atari	50 KB	15.6 MB	10 KB	0.975%	0.488%	0.488%
DonkeyCar	12 KB	1.6 MB	15 KB	0.010%	0.005%	0.005%

coordinator (Coord) across platforms. The system's execution overhead remains under 1.43%, underscoring our method's efficiency. The feedback loop has a minimal overhead, peaking at 3.68s in Atari. The runtime coordinator also maintains a low overhead, at most 0.16s and 0.01%. Conversely, memory management has a higher overhead, reaching 115.8s and 1.41% on Orin, due to increased memory access in our strategy. Yet, this is counterbalanced by memory optimization techniques like non-blocking data prefetching, enhancing the overall system performance.

Memory Overhead. Table 4.7 showcases the memory overhead of $\mathtt{R}^3$, both in absolute terms and as a percentage. The overhead remains below 1% on all platforms, making $\mathtt{R}^3$ ideal for memory-limited autonomous systems. While emphasizing memory efficiency, $\mathtt{R}^3$ may compromise latency. Future work might focus on minimizing memory access or enhancing caching strategies for better memory management.

Low overhead: $\mathtt{R}^3$'s efficient implementation introduces low overhead, making it a lightweight deployment solution for optimizing On-Device DRL training.

4.4.5 Adaptability to Different System Scenarios

As a complement to the overall effectiveness, we investigate the adaptability of the proposed $\mathtt{R}^3$. We focus on the adaptation of both variable deadline and variable computational interference in the following angles.

Adaptability to Variant Deadlines. We examine the adaptability of $\mathtt{R}^3$ to variant end-to-end deadlines in Figure 4.15. This figure depicts the outcomes, demonstrating that $\mathtt{R}^3$ can dynamically adjust to variant end-to-end deadlines. Thus it can sustain a highly predictable task-level tail latency within a defined range of computational interference. This evidences the resilience of $\mathtt{R}^3$ as a solution to different timing constraints.

Adaptability to Variant Computational Interference. We examine DRL training based on $\mathtt{R}^3$ alongside a computationally demanding background workload, namely FFmpeg video decoding, to explore the adaptability of $\mathtt{R}^3$ to computational interference. We varied the interference workload intensity by modifying the video's output resolution. Specifically, 720P, 1080P, and 2K decoding resolutions were employed to represent light, heavy, and malicious interference. Figure 4.15 depicts the outcomes, demonstrating that $\mathtt{R}^3$ can dynamically adjust to computational workload, thereby sustaining a highly predictable task-level tail latency within a defined range of computational interference. Thus, it ensures the fulfillment of end-to-end deadlines.

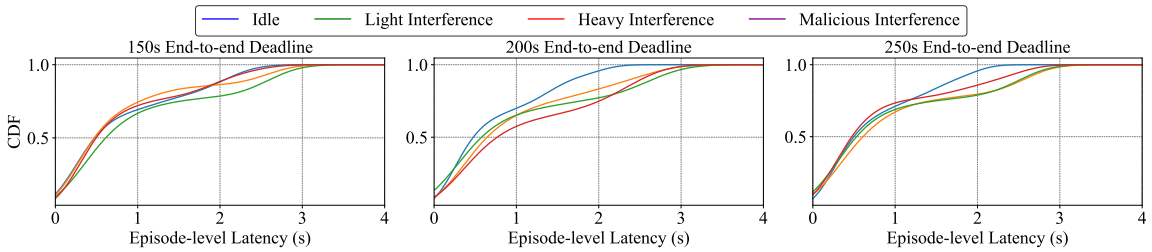


Figure 4.15: Episode-level latency CDF for varying deadline configuration and interference workloads intensity on Xavier.

Versatility: R^3 consistently maintains its effectiveness across different system scenarios, including various deadline settings and interference workloads. This adaptability ensures robustness and resilience, regardless of timing or computational constraints.

4.5 Related Work and Discussion

Deep Reinforcement Learning (DRL) has gained significant attention due to its ability to learn complex tasks and adapt to dynamic environments, making it particularly suitable for recent robotics applications [125, 291, 183, 149, 264] and future robots with complex deep learning techniques [319, 214, 5, 65, 64, 111, 112, 58, 57, 56, 218, 62]. The foundation of DRL rests on DQN's introduction [256] and its subsequent extensions tailored to robotics, including DDQN[343], C51 [24], and others. While newer algorithms such as PPO [318], TRPO [317], SAC [131] have emerged, the real-time capabilities of DRL algorithms largely remain an untapped domain. Notably, while a few studies propose new DRL algorithms that can provide faster response times [339, 299, 113], R^3 could optimize performance of given DRL algorithms without modifying the algorithms themselves. Recent research has explored dynamic adjustments of training parameters, such as batch size [265] and replay buffer size [103], to enhance algorithmic performance. However, these approaches,

being agnostic to system constraints, risk causing catastrophic out-of-memory (OOM) errors as illustrated in Figure 4.1. In contrast, by considering the unique properties of system constraints and DRL, R^3 effectively avoids OOM errors while achieving the goal of efficient real-time DRL training.

Recently, real-time ROS schedulers have emerged equipped with a suite of optimization strategies [179, 211, 338, 33, 337, 66, 34]. Our emphasis remains mainly on application-layer application for DRL training, but the potential of integrating more ROS optimizations remains an exciting prospect for future exploration.

Although considerable progress has been made in addressing real-time deep learning inference [21, 184, 365, 20, 406, 22, 263, 175, 176, 174], unfortunately, adapting these techniques directly to the DRL training scenario poses new challenges since the complexity and resource-demanding characteristics of training. This paper empirically explores these challenges and introduces a novel framework for On-Device DRL training.

Limitations of R^3. R^3 provides optimization in a static hardware context, but there still exists space to get performance gains from software-hardware synergy [390]. Moreover, R^3 focus is on DRL training that utilizes replay buffers, leaving the optimization for replay-buffer-free DRL algorithms [315] unaddressed. Additionally, despite validations on the DonkeyCar simulator [197] for R^3, physical-world implementations may introduce additional unpredictability, such as sensor noise, dynamic lighting conditions, unpredictable environmental changes, and unmodeled physical interactions. To understand how well R^3 works in the real world, a promising future direction is actual autonomous systems integration[150, 148, 118, 102, 1].

4.6 Conclusion

This study introduces the novel framework, R^3, which is specifically designed to ensure timing predictability for executing deep reinforcement learning (DRL) training workloads in GPU-enabled autonomous embedded systems. The comprehensive nature of R^3 allows for the seamless optimization of both timing and algorithm performance while simultaneously adhering to stringent memory constraints. This is achieved by incorporating a thorough understanding of DRL workload characteristics as well as utilizing real-time system feedback information. Through rigorous experimentation, we have demonstrated that R^3 consistently showed a significant reduction in OOM errors, effective latency handling, and maintenance of competitive algorithm performance, marking a stride in the realm of embedded real-time DRL.

Chapter 5

Orion: Adaptive Memory Management for On-Device OCL

5.1 Introduction

Online continual learning (OCL) addresses the challenge of learning from non-stationary, streaming data while mitigating catastrophic forgetting [52, 294, 234, 51, 10, 194, 222, 333]. This capability is critical for robotics and other embodied AI systems that must adapt in real time in dynamic environments such as transportation, agriculture, and defense [262, 347, 348, 48, 133]. Unlike offline retraining on fixed datasets, OCL updates the model incrementally in a single pass over new experiences, making both algorithmic behavior and systems design central to practical deployment [245, 9].

OCL performance is commonly characterized by *plasticity* (fast adaptation to new experiences) and *stability* (retention of prior knowledge), together with *training latency*,

i.e., how quickly the model incorporates new data [117]. On-Device deployment adds a fourth, implicit axis: *memory*. Recently, replay-based methods such as ER [52] revisit stored experiences and are effective across diverse settings, including resource-constrained platforms [294]. Memory selection (GSS) favors informative samples [10]. Constraint- and regularization-based methods, e.g., GEM [234], AGEM [51], EWC [194], and LwF [222], preserve past knowledge via gradient projections or parameter consolidation. Beyond algorithms, system-level efforts remain limited: LifeLearner targets hardware-level optimization for meta-continual learning [202], Ekya optimizes *offline* CL pipelines on multi-GPU servers [28], and Latent Replay focuses on algorithmic changes without systems co-design [289]. Yet none of these efforts jointly address runtime memory constraints, algorithmic efficacy, and system-level co-design for *On-Device* OCL. This gap is critical, as autonomous robots and edge devices increasingly require real-time adaptation under strict hardware budgets in the field, far from data centers and without opportunity for offline retraining.

More challenging, the trade-off space of On-Device OCL is inherently conflicted: batch scaling reduces latency but inflates memory, larger buffers help stability, but risk OOM, and stronger optimization improves plasticity while increasing compute and memory traffic. Figure 5.1 summarizes this "no silver bullet" landscape and motivates an adaptive, memory-aware runtime. Critically, because memory pressure and workload complexity shift dynamically as the number of OCL experiences grows, any static offline configuration could be either too conservative and waste resources early, or too aggressive and cause OOM failures later. This necessitates an online, self-adaptive memory manager.

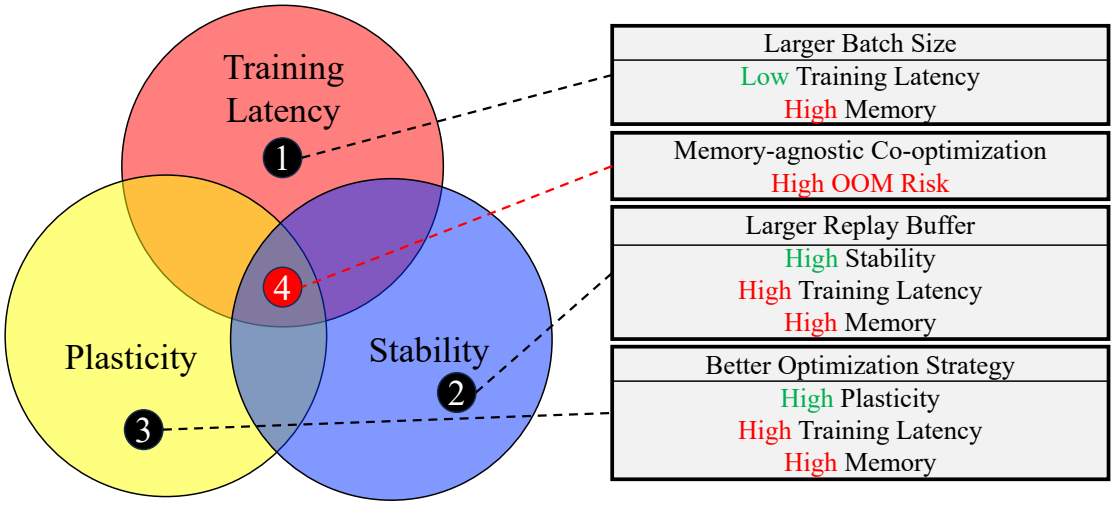


Figure 5.1: **No silver bullet.** Visualization of the complex tradeoff space among three key performance metrics of On-Device OCL. Memory-agnostic co-optimizing training latency, plasticity, and stability could easily raise out-of-memory (OOM) concerns under stringent memory constraints.

Contributions. This work introduces *Orion*, a self-adaptive memory management ecosystem for On-Device OCL that addresses the challenge of balancing training latency, plasticity, and stability under stringent memory constraints. Managing these trade-offs is non-trivial due to OS-level allocation and runtime variability; doing so robustly requires tight integration of application- and system-level decisions, especially on embedded platforms.

At its core, *Orion* introduces `URGE`, a unified runtime indicator that dynamically optimizes the OCL process without manual intervention. While URGE's formulation is intentionally pragmatic and interpretable, inspired by Liebig's Law of the Minimum [275], we demonstrate it is robust across diverse benchmarks, hardware platforms, and OCL algorithms. By integrating performance metrics (plasticity, stability, latency), balancing memory usage with system constraints to prevent OOM errors, and adapting to workload changes using time-dependent thresholds, `URGE` enables autonomous memory reallocation across OCL components. This self-adaptive mechanism ensures efficient resource utilization,

enabling *Orion* to maintain OCL algorithmic performance while adhering to strict resource constraints. In addition, we prototype a practical system for real-world deployment by extending the `Avalanche-lib` library [46, 233] with system-level optimizations tailored for embedded and edge platforms. A key module is our unified, multi-threaded data prefetcher that leverages idle CPU resources to proactively load streaming and replay data, reducing training bottlenecks. All enhancements are implemented as transparent modules, enabling rapid deployment of *Orion* across diverse hardware, including ARM64-based embedded devices and edge servers.

Implementation and Evaluation. *Orion* is evaluated on standard OCL benchmarks [198, 232, 231, 160] and four representative algorithms [52, 10, 234, 51], across platforms ranging from high-performance edge servers to memory-constrained embedded devices [268, 270]. We further demonstrate practicality in an autonomous driving case study built on `EndlessCL-Sim` [160] with a navigational TurtleBot 3.

Key highlights of *Orion* include:

- **Overall Effectiveness and Versatility:** *Orion* was evaluated on one edge server and two GPU-enabled autonomous embedded systems, demonstrating its ability to balance training latency, plasticity, and stability in OCL training. It adapts to different platforms, benchmarks, OCL algorithms, and user preferences while consistently meeting memory constraints. *Orion* achieved an average 12.13× speedups w.r.t. training latency compared to online baselines, with minimal trade-offs in other performance metrics, specifically, an average plasticity and stability reduction of less than 10% compared to the offline brute-force Oracle baseline (Sec. 4.4.2).

- **Practical Usability:** A robotic case study validated the practicality and efficacy of *Orion* in real-world OCL scenarios. It successfully auto-balances training latency, plasticity, and stability, without encountering any OOM errors during deployment (Sec. 5.5.3).
- **Low Overhead:** *Orion* exhibits low execution overhead ranging from 0.47% to 2.10%, almost negligible memory overhead between 0.016% to 0.042%, and negligible energy overhead of less than 0.1% across a rich set of OCL scenarios (Sec. 4.4.4).

5.2 Background

5.2.1 Online Continual Learning

Online Continual Learning (OCL) enables systems to incrementally learn from streaming data, adapting to new information while retaining prior knowledge [245]. Unlike traditional deep learning, which relies on static datasets and offline retraining, OCL is essential for dynamic, real-world applications where data evolves, and full dataset retraining is impractical. OCL processes sequential experiences, new batches of data, by incrementally updating model parameters and balancing adaptation to new data with retention of prior learned knowledge. In real-world robotic applications, OCL provides a lightweight mechanism for adapting to new lighting or terrains in autonomous navigation, recognizing novel objects on the fly [289], updating person re-identification models under varying conditions [377], and continuously refining physical manipulation skills [207].

OCL introduces unique challenges due to its online workload characteristics. It cannot rely on multiple runs or offline optimization methods, thus inherently requiring *fast, adaptive system-application coordination*. Unlike offline methods, OCL demands rapid,

incremental learning, which makes it particularly challenging for resource-constrained devices where low training latency is critical [9].

Following state-of-the-art OCL algorithms [117, 294], we evaluate On-Device OCL performance using the following metrics:

- Plasticity: the ability to adapt to new data.
- Stability: the ability to retain accuracy on previously learned data.
- Training Latency: The speed of adaptation to new data, where higher latency may indicate slower adaptation and reduced accuracy [117].
- Memory: adherence to stringent memory constraints, which is crucial for On-Device deployment.

Formally, given a dataset $\mathcal{D}$ and experience number $\mathcal{N}$, we define plasticity (P), and stability (S) for the $\mathcal{K}$-th experience ($\mathcal{K} \leq \mathcal{N}$) as follows: Plasticity (P) indicates the ability to adapt to new knowledge: $P = \frac{1}{\mathcal{K}} \sum_{i=1}^{\mathcal{K}} \text{acc}_i$, where acc_i is the test accuracy on the i-th experience. Stability (S) indicates the ability to maintain previously learned knowledge while integrating new knowledge:

$$S = \begin{cases} 1 & \text{if } \mathcal{K} = 1, \\ 1 - \frac{1}{\mathcal{K}-1} \sum_{i=1}^{\mathcal{K}-1} \mathcal{F}_i & \text{if } \mathcal{K} > 1. \end{cases}$$

where $\mathcal{F}_i$ measures how much knowledge about previous experiences is lost after learning the i-th experience. This equation also considers that when $\mathcal{K} = 1$, there are no prior experiences to evaluate forgetting, so stability is set to one. Plasticity and stability can be assessed

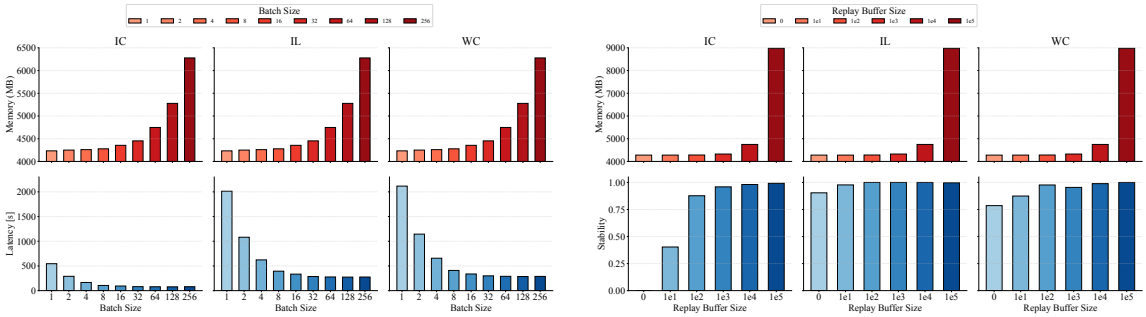
immediately after training in each experience, without needing to complete the entire OCL process.

To gain a comprehensive understanding of the performance of On-Device OCL, it is essential to focus on two key OCL components: batch execution and replay buffer. Batch execution involves processing multiple data samples in a single operation, fostering effective learning. On the other hand, the replay buffer serves as a repository for past experiences, which are sampled for conducting mini-batch training. These components significantly impact algorithmic performance, that is, plasticity and stability, and system-level memory usage and training latency. Specifically, we explore the impacts of key OCL parameters on the above metrics, including (1) training batch size, which defines the amount of data used in each mini-batch training; (2) replay buffer size, which determines the amount of data stored for revisitation; and (3) optimization strategies tailored for enhancing plasticity.

5.3 Motivation

This section thoroughly examines the critical trade-offs in On-Device OCL under stringent memory constraints, highlighting specific challenges through a series of realistic case studies. Understanding and quantifying these trade-offs provides valuable insights into how to effectively co-optimize plasticity, stability, and resource usage, guiding the development of more robust and efficient deployable solutions.

On-Device OCL must jointly balance *training latency*, *plasticity*, and *stability* under tight memory budgets. We study three levers that practitioners frequently adjust in practice



(a) Effect of training batch size on OCL. (b) Effect of replay buffer size on OCL.

Figure 5.2: Impact of key hyperparameter choices on OCL metrics under memory constraints.

Table 5.1: Integrating OCL plugins into ER. Best results are marked. Subscripts show differences from the baseline.

Metrics	Training Latency (s)				Plasticity				Memory (MB)			
Setting	w/o Opt.	GEM	EWC	GEM+EWC	w/o Opt.	GEM	EWC	GEM+EWC	w/o Opt.	GEM	EWC	GEM+EWC
IC	best73.06	$147.40_{+74.34}$	$147.18_{+74.12}$	$215.13_{+142.07}$	0.98	$best0.99_{+0.01}$	$best0.99_{+0.01}$	$best0.99_{+0.01}$	best4100	4195_{+95}	4200_{+100}	4207_{+107}
IL	best263.28	$505.00_{+241.72}$	$537.28_{+274.00}$	$771.61_{+508.33}$	0.94	$0.94_{+0.00}$	$best0.98_{+0.04}$	$0.93_{-0.01}$	best4250	4347_{+97}	4352_{+102}	4360_{+110}
WC	best268.88	$527.16_{+258.28}$	$560.63_{+291.75}$	$817.82_{+548.94}$	0.97	$0.89_{-0.08}$	$0.93_{-0.04}$	$best0.97_{+0.00}$	best4180	4270_{+90}	4281_{+101}	4301_{+121}

(batch size, replay buffer size, and optimization plugins), and find that memory is the shared bottleneck that frequently drives failures on embedded platforms.

We first vary the training batch size and measure its effect across the Incremental Class (IC), Incremental Learning (IL), and Weather Classification (WC) scenarios. As illustrated in Figure 5.2a, batch size dictates a severe trade-off between training latency and memory consumption. While small batch sizes (e.g., 1 to 16) keep memory usage manageable at roughly 4.2 GB, they incur prohibitive training latencies, exceeding 2000 seconds in the IL and WC scenarios, which effectively prevents real-time robotic adaptation. Conversely, scaling the batch size to 256 drastically reduces latency to under 200 seconds, but inflates memory consumption well beyond 6 GB. On edge devices equipped with 4 to 8 GB of unified memory, such aggressive batching guarantees Out-Of-Memory (OOM) failures.

Next, we analyze the impact of replay buffer size. Figure 5.2b demonstrates a clear pattern of diminishing returns. Stability improves dramatically as the buffer size

expands from 0 to 10^3, successfully mitigating catastrophic forgetting. However, beyond 10^4, stability saturates at nearly 1.0. Meanwhile, memory usage remains relatively stable (4.2 to 4.7 GB) up to 10^4 but experiences a massive, abrupt spike to approximately 9 GB at 10^5. Allocating excessively large buffers yields negligible performance benefits while ensuring memory exhaustion on resource-constrained platforms, underscoring the need for precise, context-aware sizing rather than naive maximization.

Finally, we examine the deployment overhead of integrating state-of-the-art algorithmic plugins (GEM and EWC) into the baseline Experience Replay (ER) algorithm. As detailed in Table 5.1, these plugins successfully enhance model plasticity (e.g., from 0.94 to 0.98 in IL) and help preserve stability. However, this algorithmic superiority is not free on the edge. The computational complexity of calculating gradient projections (GEM) or Fisher information matrices (EWC) doubles or even triples the training latency (e.g., IC latency jumps from 73 s to over 215 s with GEM+EWC). Furthermore, maintaining these auxiliary structures consistently inflates memory usage by roughly 90 to 121 MB across scenarios. This demonstrates that purely algorithmic OCL advancements can severely penalize system-level responsiveness if deployed indiscriminately.

In summary, these empirical case studies reveal that OCL optimization goals are inherently coupled and frequently conflicting. Maximizing learning performance (plasticity and stability) via larger batches, expanded buffers, or advanced plugins fundamentally degrades system performance (latency and memory footprint). Because edge devices operate under strict, non-negotiable hardware ceilings, there is no static, universally optimal configuration. Instead, realizing deployable On-Device OCL demands an intelligent, dynamic

co-optimization strategy that treats hardware constraints and algorithmic efficacy as a joint, multi-objective problem.

Takeaway: Taken together, these three patterns all draw from the same memory pool. When batch size increases, when the buffer grows, or when heavier plugins are enabled, the combined footprint often exceeds device capacity, which leads to OOM and unstable training. Managing these trade-offs is a systems problem. Operating system memory allocation interacts with algorithm choices; static or memory blind tuning is brittle. We need a fast and self-adaptive memory manager that coordinates batch processing, replay buffers, and plugin usage to avoid OOM while preserving stability and training efficiency. The next section develops this design.

5.4 System Design

5.4.1 Design Overview

Our proposed system, named *Orion*, is a holistic framework designed to optimize OCL systems through self-adaptive runtime memory management, balancing system performance and resource constraints. Importantly, *Orion* is transparent to the online continual learning process and can be seamlessly integrated into existing OCL frameworks. This ensures that training and evaluation steps remain consistent with established practices in the field, aligning with the methodologies used in state-of-the-art continual learning studies [10, 51]. Specifically, *Orion* maintains transparency between data and experience splits in benchmark evaluations, adhering to the protocols followed in prior continual learning works.

As shown in Fig. 5.3, *Orion* employs a hierarchical control framework guided by a unified, system-aware indicator (`URGE`). This indicator integrates multiple performance metrics to provide a comprehensive evaluation of the system's state and serves as the foundation for decision-making. Intuitively, `URGE` captures the trade-offs between system performance and memory resource availability. A high `URGE` value indicates that resources are abundant but the system is underperforming, suggesting both the opportunity and urgency for optimization without significant risk of overloading memory. Conversely, a low `URGE` value reflects either: (1) the system is performing sufficiently well and does not require further optimization, or (2) resource constraints that make further optimization impractical or risky due to potential OOM errors. (Detailed calculation and application of `URGE` is in Sec. 5.4.2.)

Orion implements a hierarchical control framework which operates at two levels: (1) Coarse-grained control (Algorithm 1, lines 3-5): At this level, `URGE` is calculated to guide high-level decisions about the overall optimizing direction of the OCL system. (2) Fine-grained control (Algorithm 1, lines 6-20): This level involves a more granular approach to optimizing system performance. After each experience of the OCL process, `URGE` is calculated and incorporated with a time-dependent threshold to make adaptive memory management decisions, as detailed in Section 5.4.3.

Beyond its adaptive control logic, *Orion* features practical engineering enhancements to address OCL deployment bottlenecks. Notably, we prototype a unified, multi-threaded data prefetching module that leverages idle CPU resources to mitigate data loading latency

and improve end-to-end training performance. All system-level optimizations are implemented as transparent, drop-in modules within our prototype, ensuring compatibility with the standard Avalanche pipeline and minimal engineering overhead for end users. Details of these engineering contributions are described in Section 5.4.5.

5.4.2 The Design of `URGE`

The design of `URGE` is inspired by the Liebig's Law of the Minimum (also knowns as the "Buckets effect") [275], which posits that growth is limited not by the total resources available but by the scarcest resource (limiting factor). In this context, `URGE` ensures that the system achieves a balance among key metrics: plasticity, stability, and latency while minimizing memory usage. The "Buckets effect" analogy highlights the intuition behind the design of `URGE`. Specifically, when plasticity is high, stability is high, latency is too low (indicating overly fast training), or when memory usage risks an OOM (out-of-memory) error, the `URGE` decreases, leading to more conservative memory allocation (Algorithm 1 line 11-13). Conversely, under more favorable conditions, such as lower memory usage risk, the `URGE` increases, allowing for more aggressive memory allocation (Algorithm 1 line 7-9). We will elaborate on the precise design of `URGE` later.

Formally, let P_t and S_t represent the plasticity and stability at time t, respectively. Let P_{th} and S_{th} be the corresponding thresholds. Let L_t represent the training latency at

time t, and L_{th} be the latency threshold. M_t is the memory usage at time t, M_{max} is the maximum allowed memory. We define the `URGE` at time t as

$$\begin{aligned}\texttt{URGE}_t = \frac{1}{1+e^{k_p(P_t-P_{th})}} \cdot \frac{1}{1+e^{k_s(S_t-S_{th})}} \\ \cdot \frac{1}{1+e^{-k_l(L_t-L_{th})}} \cdot \frac{1}{1+e^{k_m(M_t-M_{max})}}\end{aligned} \tag{5.1}$$

where k_p, k_s, k_l, and k_m are scaling factors that control the sensitivity of the `URGE` to changes in plasticity, stability, training latency, and memory pressure, respectively. The terms in the `URGE` equation (Eq. 5.1) reflect the system's state. If plasticity is low, the first term is high; If stability is low, the second term is high; If training latency is high, the third term is high. These scenarios highlight how different factors may influence `URGE`, and consequently, the configuration of the OCL system. Intuitively, when any of these terms is high, i `URGE` increases, signaling the need to optimize plasticity, stability, or training latency, if sufficient memory is available. The system responds by increasing the batch size, enlarging the replay buffer, or enabling advanced optimization strategies to address these deficiencies (Algorithm 1, line 7-9).

Memory usage is critical in preventing OOM errors. When memory usage is high, the fourth term in Eq. 5.1 decreases, penalizing the `URGE`. In response, the system decreases the batch size and replay buffer size and disables advanced optimization strategies to free up memory resources, minimizing the risk of OOM errors (Algorithm 1, line 11-13).

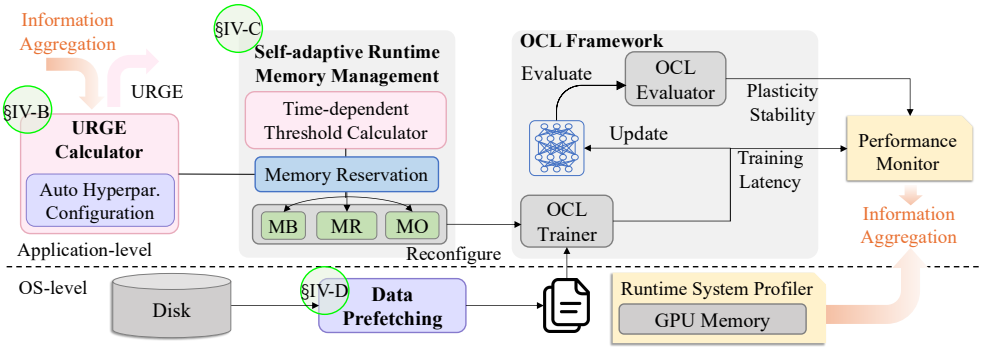


Figure 5.3: Design overview of *Orion*.

5.4.3 Self-adaptive Memory Management

This section explains how *Orion* leverages `URGE` for self-adaptive memory management. By dynamically adjusting memory allocation in response to performance metrics, resource constraints, and workload complexity, *Orion* ensures efficient and robust optimization for On-Device OCL systems.

Time-dependent Threshold. While performance metrics such as stability and plasticity are only accessible after the completion of training for one experience rather than during training, they remain instrumental for guiding fine-grained control. This is particularly relevant in OCL, where workloads dynamically evolve. As the OCL progresses, each experience typically exhibits increasing training latency due to the growing computational complexity required to retain old knowledge. This dynamic is distinct from the relatively homogeneous patterns in training latency observed during each training epoch of the standard Deep Neural Network fine-tuning process [121]. To account for this, we introduce

a time-dependent threshold, $\texttt{Thr}_t$, that decreases over time to encourage a more aggressive optimization policy as training advances:

$$\texttt{Thr}_t = \texttt{Thr}_0 \cdot e^{-\delta t} \tag{5.2}$$

where $\texttt{Thr}_0$ is the initial threshold value, and δ is a constant decay rate hyperparameter that controls how quickly the threshold decreases over time.

Unified URGE-based Memory Management. We propose a unified memory manager for application-aware memory allocation and reallocation dynamically. The memory manager uses URGE and system memory availability to make decisions about memory reservations for each OCL component and reactively adjust application-level control knobs (batch size, replay buffer size, and optimization strategies).

The memory allocated for batch processing can be dynamically adjusted based on URGE:

$$MB_t = \begin{cases} MB_0, & \text{if } t = 0 \\ MB_{t-1} \cdot (1 + \alpha \cdot (\texttt{URGE}_t - \texttt{Thr}_t)), & \text{if } t > 0 \end{cases} \tag{5.3}$$

where MB_{t-1} is the batch processing memory at the previous time step, and α is a scaling factor that controls the sensitivity of the batch processing memory to changes in the URGE. If URGE exceeds the time-dependent threshold, the batch processing memory will be increased by a factor of $(1 + \alpha \cdot (\texttt{URGE}_t - \texttt{Thr}_t))$ to accelerate the learning process. For example, if $\alpha = 0.1$, $\texttt{URGE}_t = 0.8$, and $\texttt{Thr}_t = 0.7$, then the batch processing memory will increase by a factor of $1 + 0.1 \cdot (0.8 - 0.7) = 1.01$. After that, batch size at time t could be calculated by $B_t = \lfloor \frac{MB_t}{M_{batch}} \rfloor$, where M_{batch} is memory consumption for each batch data. M_{batch} could be

easily accessed offline, based on the characteristics of continual learning environments and tasks. This value is invariant to hardware devices.

The replay buffer memory is managed based on the URGE in a similar manner:

$$MR_t = \begin{cases} MR_0, & \text{if } t = 0 \\ MR_{t-1} \cdot (1 + \beta \cdot (\texttt{URGE}_t - \texttt{Thr}_t)), & \text{if } t > 0 \end{cases} \tag{5.4}$$

where MR_{t-1} is the replay buffer memory at the previous time step, and β is a scaling factor that controls the sensitivity of the replay buffer memory to changes in the URGE. If URGE is higher than the threshold, the replay buffer memory will increase by multiplying a factor of $(1 + \beta \cdot (\texttt{URGE}_t - \texttt{Thr}_t))$ to allow for more diverse samples during training and improve stability. For example, if $\beta = 0.2$, $\texttt{URGE}_t = 0.8$, and $\texttt{Thr}_t = 0.7$, then the replay buffer memory will increase by multiplying by a factor of $1 + 0.2 \cdot (0.8 - 0.7) = 1.02$. After that, the replay buffer size at time t could be calculated by $R_t = \lfloor \frac{MR_t}{M_{df}} \rfloor$, where M_{df} is memory consumption for each data frame. Note that this calculation is conservative in avoiding OOM because although the replay buffer may not be immediately filled up, actual memory consumption could be lower than the calculation. M_{df} could be easily accessed offline and is invariant to hardware devices.

Memory allocation for optimization strategies is managed as follows:

$$MO_t = \begin{cases} MO_{advanced}, & \text{if } \texttt{URGE}_t \geq \texttt{Thr}_t \\ MO_{default}, & \text{if } \texttt{URGE}_t < \texttt{Thr}_t \end{cases} \tag{5.5}$$

Algorithm 4 `URGE`-based Self-adaptive Memory Management for OCL

Input: OCL dataset D; initial memory allocations MB_0, MR_0, MO_0; initial threshold $\texttt{Thr}_0$; scaling factors α, β; decay rate δ
Initialize: $MB_t \leftarrow MB_0$, $MR_t \leftarrow MR_0$, $MO_t \leftarrow MO_0$, $\texttt{URGE}_0 \leftarrow 1$, $t \leftarrow 0$
for each experience e in D **do**
    Execute training with current MB_t, MR_t, and MO_t
    Compute $\texttt{URGE}_t$ using Eq. (5.1)
    Compute $\texttt{Thr}_t$ using Eq. (5.2)
    if $\texttt{URGE}_t > \texttt{Thr}_t$ **then**
        $MB_t \leftarrow MB_{t-1} \cdot (1 + \alpha \cdot (\texttt{URGE}_t - \texttt{Thr}_t))$
        $MR_t \leftarrow MR_{t-1} \cdot (1 + \beta \cdot (\texttt{URGE}_t - \texttt{Thr}_t))$
        $MO_t \leftarrow MO_{advanced}$
    else
        $MB_t \leftarrow MB_{t-1} \cdot (1 - \alpha \cdot (\texttt{Thr}_t - \texttt{URGE}_t))$
        $MR_t \leftarrow MR_{t-1} \cdot (1 - \beta \cdot (\texttt{Thr}_t - \texttt{URGE}_t))$
        $MO_t \leftarrow MO_{default}$
    end if
    Reconfigure the training system based on updated memory allocations
    Launch training for experience e
    Prefetch data for the next experience
    $t \leftarrow t + 1$
end for

where $MO_{advanced}$ represents the memory consumption of more advanced optimization strategies that can improve the model's ability to learn new knowledge, and $MO_{default}$ represents the memory consumption of less memory-intensive optimization strategies.

Note that memory management for optimization is particularly challenging because it is correlated to batch size and replay buffer size in the above and could be hard to estimate precisely. Therefore, we roughly assume that all the unprofiled memory measured at the system level in OCL, except for batch processing memory and replay buffer memory, is consumed by optimization. Specifically, we estimate the relationship between $MO_{advanced}$ and $MO_{default}$ as $MO_{advanced} = kMO_{default}$, where k is a factor that can be determined rapidly at runtime.

5.4.4 Automatic Hyperparameter Configuration

To automatically set the hyperparameters k_p, k_s, k_l, and k_m based on a user-specified order of importance, we propose a rule-based method. Users provide the importance order as a list of metrics, e.g., [memory, plasticity, stability, training latency]. *Orion* then assigns weights to each metric based on its position in the list, with the last factor receiving a weight of 1, the second-to-last a weight of 2, and so forth. For instance, if the user specifies the importance order as [memory, plasticity, stability, training latency], the weights for each metric are $(w_m, w_p, w_s, w_l) = (4, 3, 2, 1)$. We then normalize the weights by $k_* = \frac{w_*}{\sum w}$ to ensure that $k_m + k_p + k_s + k_l = 1$. After normalization, (k_m, k_p, k_s, k_l) is set to be (0.4, 0.3, 0.2, 0.1). This rule-based method ensures that the `URGE` is more calculated considering user-specified preferences. Note that the time complexity of calculating `URGE` at runtime is $O(e)$ where e is the experience number for OCL, as it is only computed at the boundary between training experiences. This ensures that we can quickly assess `URGE` during the OCL process.

5.4.5 System Prototype Implementation

Our OCL toolchain is developed as an extension of the Avalanche [46] continual learning library, with targeted adaptations for ARM64-based embedded platforms. All system-level optimizations are designed as transparent modules, ensuring full compatibility with the standard Avalanche pipeline and imposing minimal engineering overhead for end users. These enhancements are evaluated in Sec. 5.5, where we demonstrate their impact on training latency under real-world OCL workloads. To address data loading

bottlenecks common in continual learning scenarios, we introduce a unified, multi-threaded data prefetching module. Operating in parallel with the main training loop, the CPU proactively loads and stages both streaming and replay samples ahead of time, while the GPU remains dedicated to model training. Our implementation unifies the disparate data access patterns of raw and replay buffers in OCL, leveraging Python threading to ensure data is always available at iteration boundaries and avoiding blocking overhead. As described in Algorithm 4, this co-optimized pipeline reduces end-to-end training latency and maximizes system resource utilization. All modifications require no changes to Avalanche's core logic and supporting rapid deployment across diverse hardware platforms.

5.5 Evaluation

5.5.1 Experimental Setup

Testbeds. Our testbeds include three platforms exhibiting different memory constraints: an edge server with A4500 GPU and two GPU-enabled embedded devices, Xavier, and Orin, which are widely used in autonomous driving [187, 195] and robotics [293, 269, 271, 273].

OCL algorithms and benchmarking dataset. To ensure a comprehensive evaluation of *Orion*, we assess the performance metrics on four prominent OCL algorithms, including ER [52], GSS [10], GEM [234], and AGEM [51], based on ResNet-20 [146] DNN architecture. These algorithms represent various OCL approaches, ensuring a comprehensive assessment of *Orion* across various settings. We evaluate *Orion* on a set of widely studied OCL benchmarks involving different sizes and different OCL tasks as detailed in Tab. 5.2.

.

Table 5.2: Statistics of benchmark datasets used in this work. IC: Incremental Class, IL: Incremental Illumination, WC: Weather Change, NI: New Instances, NC: New Classes, NIC: New Instances and Classes.

Dataset	Size	Task	# Experience	# Images	Image Size
SplitCIFAR10 [198]	Small	IC	10	60,000	32×32
SplitCIFAR100 [198]	Small	IC	10	60,000	32×32
CORe50 [232, 231]	Medium	NI	8	164,866	32×32
	Medium	NC	9	164,866	32×32
	Large	NIC	79	164,866	32×32
`Endless-Sim` [160]	N/A	IC	4	49,134	32×32
	N/A	IL	5	181,899	32×32
	N/A	WC	5	188,536	32×32

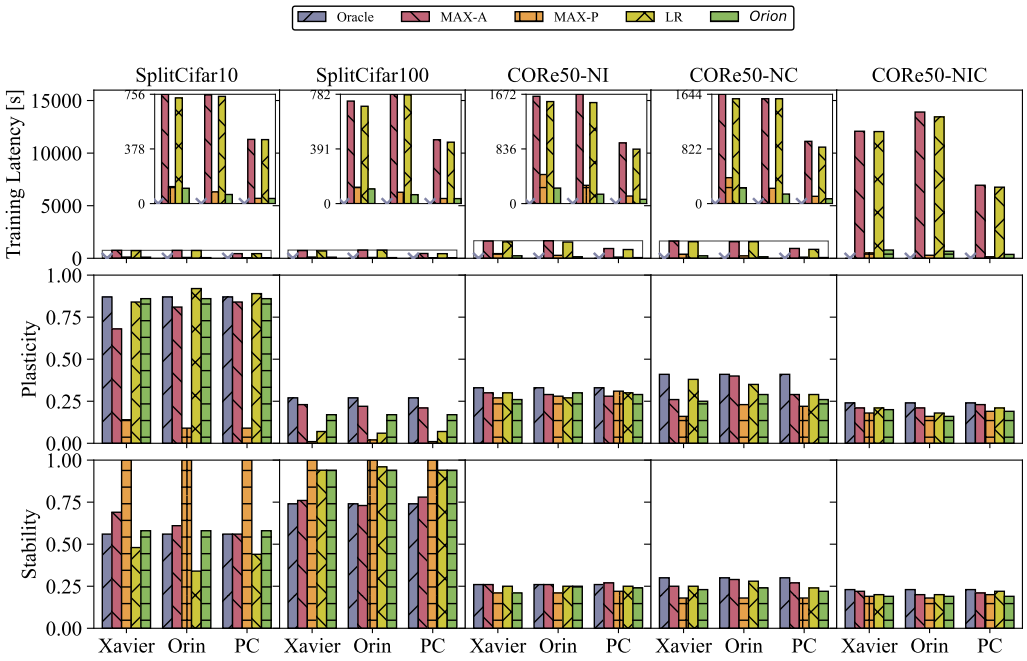


Figure 5.4: Overall effectiveness of *Orion* using the ER algorithm evaluated on five benchmarks. Crosses indicate OOM.

Robotic case study. To further demonstrate the practicality of *Orion* in realistic scenarios, we conduct a robotic case study in the context of autonomous driving using `Endless-Sim` [160], under incremental class learning (IC), incremental illumination conditions (IL), and weather changes (WC).

Baselines. We compare *Orion* to the following approaches:

- **LR [289]:** A state-of-the-art baseline leveraging latent replay for real-time continual learning in robotics. It reduces computations by replaying latent DNN results instead of raw data, using default `Avalanche-lib` parameters.
- **MAX-A [46]**: MAX-A uses preset values for training parameters in `Avalanche-lib` for empirical good plasticity and stability in general OCL scenarios.
- **MAX-P [46]**: a high-efficiency implementation version of `Avalanche-lib`, targeting optimized training latency and maximizing throughput with a large batch size.
- **Oracle**: We conduct an offline parameter search by brute force to ensure optimized plasticity and stability.

Implementation details. We implement MAX-A [46] by using default configurations for batch processing and replay buffer in `Avalanche-lib`, enabling both optimization plugins to boost performance. MAX-P [46] is implemented by modifying `Avalanche-lib` using a large batch size and a small replay buffer size. We also fully implement LR [289] and integrate it with `Avalanche-lib`. For Oracle, we conduct an extensive parameter search offline, sweeping from training batch size {16, 32, 64, 128, 256, 512, 1024} and replay buffer size {10, 100, 1000, 10000, 100000, 1000000}. The offline running set number is 42 in total.

5.5.2 Overall Effectiveness

This section evaluates the effectiveness of *Orion* across three platforms, benchmarks, and OCL algorithms. The evaluation is divided into three subsets: (1) performance across benchmarks using a state-of-the-art OCL algorithm, (2) performance across representative OCL algorithms on a single benchmark, and (3) performance under various user-specified

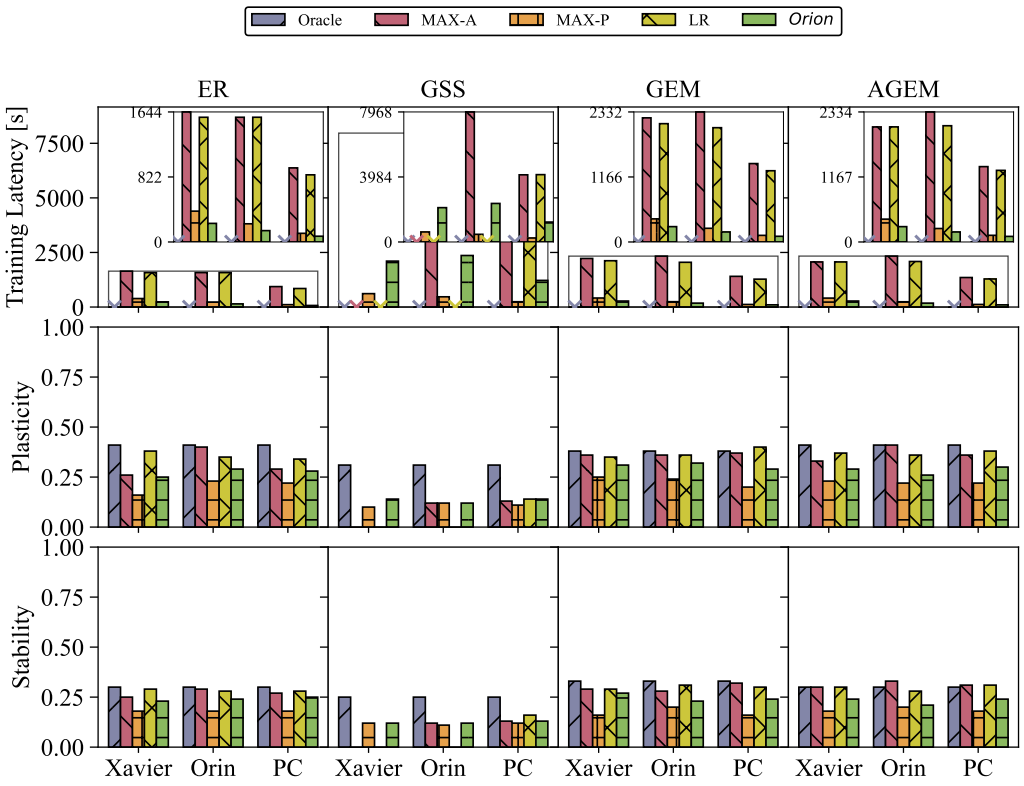


Figure 5.5: Overall effectiveness of *Orion* on four OCL algorithms on the CORe50-NC benchmark. Crosses indicate OOM.

preferences for different metrics, highlighting trade-offs between training latency, plasticity, and stability.

Performance across Benchmarks.

We evaluate the performance of *Orion* across five benchmarks of varying sizes. We focus ER [52] herein since it is the most representative replay-based OCL algorithm among the four algorithms.

Training Latency. As seen in Fig. 5.4, *Orion* runs significantly faster by an average of 392.04× compared with the offline approach Oracle. This huge improvement is due to Oracle requiring extensive 42 offline runs, while *Orion* could optimize the OCL systems in a

Table 5.3: Overall effectiveness of *Orion* under different user preferences on ER algorithms on NVIDIA Jetson AGX Xavier. The arrow directions indicate better performance metrics. The best results are highlighted in bold. Differences compared to the best results are written in the subscription.

Baselines	**Prefer Latency**	**Balanced**	**Prefer P/S**
Metrics	Training Latency [s] ↓		
SplitCIFAR10	best88.21	$106.06_{+17.85}$	$117.61_{+29.40}$
SplitCIFAR100	best86.72	$104.65_{+17.93}$	$117.07_{+30.35}$
CORe50-NI	best209.44	$237.39_{+27.95}$	$277.53_{+68.09}$
CORe50-NC	best219.81	$235.02_{+15.21}$	$357.24_{+137.43}$
CORe50-NIC	best519.89	$786.18_{+266.29}$	$1097.33_{+577.44}$
Metrics	Plasticity ↑		
SplitCIFAR10	$0.51_{-0.35}$	best0.86	best0.86
SplitCIFAR100	$0.08_{-0.10}$	$0.17_{-0.01}$	best0.18
CORe50-NI	$0.27_{-0.05}$	$0.26_{-0.06}$	best0.32
CORe50-NC	$0.24_{-0.10}$	$0.25_{-0.09}$	best0.34
CORe50-NIC	$0.17_{-0.07}$	$0.20_{-0.04}$	best0.24
Metrics	Stability ↑		
SplitCIFAR10	$0.55_{-0.16}$	$0.58_{-0.13}$	best0.71
SplitCIFAR100	$0.94_{-0.02}$	$0.94_{-0.02}$	best0.96
CORe50-NI	$0.21_{-0.06}$	$0.21_{-0.06}$	best0.27
CORe50-NC	$0.21_{-0.05}$	$0.23_{-0.03}$	best0.26
CORe50-NIC	$0.15_{-0.11}$	$0.19_{-0.07}$	best0.26

single run. Compared with online approaches, *Orion* also demonstrates supreme latency performance. *Orion* exhibits significantly lower training latency than both MAX-A and LR across all benchmarks, achieving an average speedup of 12.13× and 11.69×, respectively. Furthermore, *Orion* even outperforms MAX-P in training latency on small and medium benchmarks (4 out of 5 benchmarks), achieving an average 1.43× speedup due to *Orion*'s efficient implementation and seamless integration within the OCL systems. On the large CORe50-NIC benchmark, *Orion* incurs slightly higher latency, averaging 116.05% of MAX-P, which is optimized solely for training latency.

Plasticity and Stability. On small benchmarks, *Orion* trades off more on plasticity by 12.0% compared to MAX-A but outperforms stability than MAX-A by 19.9%. *Orion* also

outperforms LR on plasticity and stability on average by 52.3% and 37.4%. According to MAX-P, *Orion* significantly outperforms it by 9.43× on plasticity, but notably, tradeoffs on the stability of MAX-P on average by 24.0%. These counter-intuitive results could explained by the definition plasticity, which indicates the ability to learn new knowledge and stability to resist forgetting[1], MAX-P learns very little new knowledge (low plasticity value near to zero) that can forgotten, so it should have higher stability. Compared with offline baseline Oracle, *Orion* performs competitively on small benchmarks. For SplitCIFAR10, *Orion* achieves only a 1.15% plasticity gap compared to Oracle while surpassing it by 3.57% in stability. These results highlight that *Orion*'s online adaptability allows it to perform near or even beyond the offline baseline on smaller benchmarks.

On medium and large benchmarks, *Orion* demonstrates strong performance in plasticity and stability. It outperforms MAX-P by an average of 15.6% in plasticity and 11.9% in stability. However, compared to MAX-A and LR, *Orion* trades off plasticity/stability by 11.2%/16.1% and 13.1%/5.7%, respectively, while achieving much faster adaptation. Oracle shows a more significant advantage on medium and large benchmarks, achieving an average plasticity of 0.28 and stability of 0.33, compared to *Orion*'s 0.22 and 0.30, despite this, *Orion* remains competitive in stability, staying within 10% of Oracle on average. The trade-offs in plasticity become more pronounced as benchmark size increases, suggesting that online methods like *Orion* could benefit from parameter tuning and further refinement for larger-scale scenarios.

[1]Definitions of forgetting may vary across continual learning research. For consistent comparison, all forgetting measurements in this paper are directly adopted from `Avalanche-lib` [46, 233].

Performance across OCL Algorithms.

Fig. 5.5 shows the performance of the evaluated methods across four representative OCL algorithms with heterogeneous computation complexity and memory usage. We used the CORe50-NC because it is medium-sized and represents typical resource requirements for OCL benchmarks.

Training Latency. *Orion* achieves an average speedup of 267.64× over the offline Oracle, which requires 42 runs, by optimizing OCL systems online in a single self-adaptive run. Notably, Oracle encounters OOM errors in all algorithms except ER due to its high memory requirements, making it impractical for resource-constrained scenarios. Compared to online approaches, *Orion* achieves significant average speedups of 8.87×, 8.31×, and 1.13× over MAX-A, LR, and MAX-P, respectively, demonstrating efficient adaptation across diverse OCL algorithms. For GSS, the most computationally demanding algorithm, MAX-A, and LR face OOM errors on the Xavier platform, and LR encounters OOM errors on the Orin platform. In contrast, *Orion* successfully executes GSS in all scenarios, albeit with a 349.05% higher training latency compared to MAX-P. These results highlight *Orion*'s versatility in handling challenging scenarios, ensuring execution across all OCL algorithms without OOM errors, even under stringent resource constraints.

Plasticity and Stability. *Orion* maintains strong plasticity and stability across all four OCL algorithms. It significantly outperforms MAX-P by 28.1% in plasticity and 27.6% in stability, while trading off plasticity/stability by 13.0%/16.6% and 15.9%/13.8% compared to MAX-A and LR, respectively, with much faster adaptation. Compared to Oracle, *Orion* demonstrates greater robustness and usability, avoiding the critical reliability

issues that hinder Oracle in practical settings, where it encounters 11 OOM errors across 42 runs. Importantly, *Orion* successfully executes all algorithms while achieving comparable performance to Oracle, with an average plasticity/stability trade-off of 9.5% and 8.3%, respectively, making it a reliable alternative under memory constraints.

The GSS algorithm, with the highest memory complexity, causes OOM errors for MAX-A and LR on the Xavier platform and for LR on the Orin platform. In contrast, *Orion* successfully executes GSS without OOM errors, outperforming MAX-P in plasticity/stability by 40.0%/0.0% on Xavier and 0.0%/9.1% on Orin. On an edge server with richer resources, *Orion* achieves the best plasticity among all online baselines, outperforming MAX-P by 8.3% and matching MAX-A, while trading off stability against LR by 18.75%. These findings indicate that *Orion* effectively balances plasticity and stability across a wide range of hardware setups and resource conditions. This adaptability underscores *Orion*'s ability to perform reliably and efficiently across diverse OCL scenarios, making it a practical and robust solution for real-world deployments.

Performance across user-specific preferences.

Approach is designed to be versatile in handling different constraint scenarios based on user-specified preferences for certain performance metrics. To evaluate the adaptability of *Orion* to different user preferences, we assess its performance under three scenarios: (1) Prefer Latency, where the user prioritizes training latency; (2) Balanced, where the user assigns equal importance to all metrics; and (3) Prefer P/S, where the user prioritizes plasticity and stability. We focus on the ER algorithm and evaluate its performance on Xavier across five benchmarks. Tab. 5.3 presents the results of *Orion* under these three

user preference scenarios. When the user prefers low training latency, *Orion* achieves the lowest latency across all benchmarks, with an average latency of 224.81 seconds. However, this comes at the cost of lower plasticity and stability, with average values of 0.25 and 0.42, respectively. In the balanced scenario, *Orion* strikes a good balance between training latency, plasticity, and stability. It achieves an average training latency of 293.86 seconds, which is 30.7% higher than the Prefer Latency scenario. However, it maintains acceptable plasticity and stability, with average values of 0.32 and 0.43, respectively. When the user prefers high plasticity and stability (Prefer P/S), *Orion* achieves the best plasticity and stability across all benchmarks, averaged by 0.36 and 0.52. However, this comes at the cost of higher training latency, with an average of 393.36 seconds, 1.34× higher than the balanced scenarios. These results demonstrate *Orion*'s ability to adapt to different user preferences and optimize its performance.

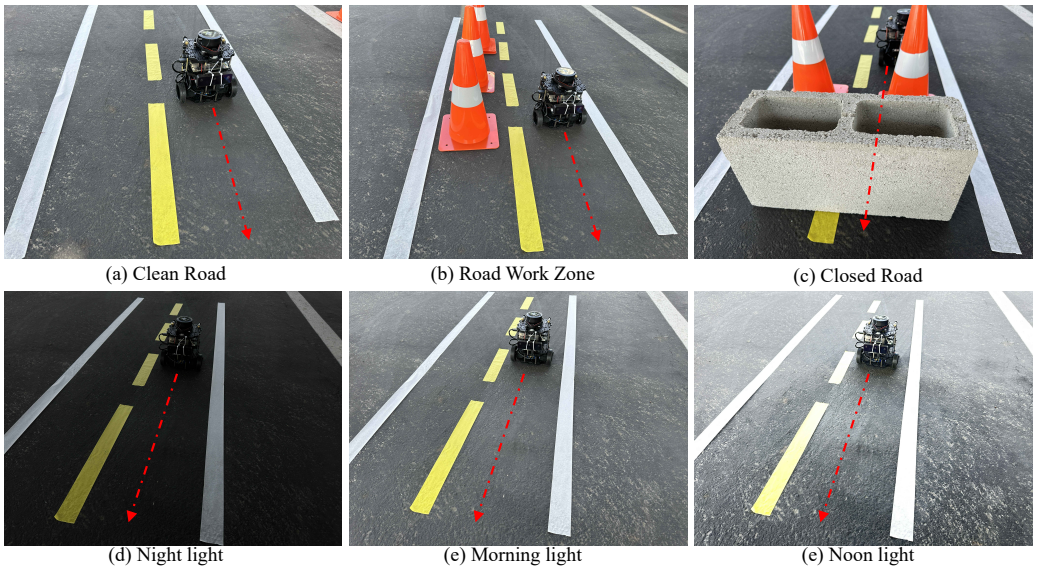
Overall effectiveness and versatility: *Orion* effectively auto-balances training latency, plasticity, and stability across various benchmarks, OCL algorithms, and user-specified preferences while consistently adhering to memory constraints on different hardware platforms. This versatility ensures optimal performance tailored to specific system requirements and user needs.

5.5.3 Robotic Case Study

To evaluate the practicality of *Orion*, we conducted a case study using a Jetson-enabled autonomous navigation robot, built on the Turtlebot3 Burger [309] with an NVIDIA Jetson motherboard and a high-resolution camera for real-time data processing and navigation (Fig. 5.6). Using `Endless-Sim` [160], we tested *Orion* in three realistic OCL scenarios

Table 5.4: Robotic case study results on NVIDIA Jetson Xavier. ✗ indicates out-of-memory error occurs. The arrow directions indicate better performance metrics. IC: Incremental Class, IL: Incremental Illumination, WC: Weather Change.

Algorithms	ER				GSS				GEM				AGEM			
Baselines	**MAX-A**	**MAX-P**	**LR**	***Orion***	**MAX-A**	**MAX-P**	**LR**	***Orion***	**MAX-A**	**MAX-P**	**LR**	***Orion***	**MAX-A**	**MAX-P**	**LR**	***Orion***
Metrics	Training Latency [s] ↓															
IC	203.45	✗	204.30	111.98	✗	✗	✗	436.07	233.24	✗	234.98	122.29	229.82	✗	235.74	111.02
IL	990.80	✗	995.93	416.97	✗	✗	✗	1169.56	1195.18	✗	1201.09	444.75	1177.83	✗	1182.66	432.37
WC	1040.45	✗	1035.50	435.02	✗	✗	✗	1183.83	1240.66	✗	1235.39	463.93	1239.03	✗	1235.22	454.81
Metrics	Plasticity ↑															
IC	0.99	✗	0.25	0.97	✗	✗	✗	0.86	0.76	✗	0.25	0.76	0.84	✗	0.25	0.74
IL	0.98	✗	0.99	0.81	✗	✗	✗	0.98	0.97	✗	0.95	0.90	0.98	✗	0.95	0.80
WC	0.95	✗	0.96	0.85	✗	✗	✗	0.89	0.91	✗	0.96	0.74	0.93	✗	0.81	0.79
Metrics	Stability ↑															
IC	0.92	✗	0.99	0.36	✗	✗	✗	0.38	1.00	✗	0.99	0.68	0.85	✗	1.00	0.83
IL	1.00	✗	0.99	0.91	✗	✗	✗	1.00	1.00	✗	1.00	1.00	1.00	✗	1.00	1.00
WC	1.00	✗	1.00	0.99	✗	✗	✗	0.90	1.00	✗	1.00	1.00	1.00	✗	1.00	1.00



(a) Clean Road (b) Road Work Zone (c) Closed Road

(d) Night light (e) Morning light (e) Noon light

Figure 5.6: A realistic case study based on an NVIDIA Jetson GPU-enabled autonomous navigation robot built on Turtlebot3. (a) (b) (c) exhibit incremental class scenarios (IC), (d) (e) (f) exhibit incremental light scenarios (IL). Red lines indicate the robot's moving directions.

(Tab. 5.4). The results show that *Orion* effectively balances training latency, plasticity, and stability without any OOM errors. In contrast, baseline methods experienced 3, 12, and 3 OOM errors for MAX-A, MAX-P, and LR, respectively. These findings demonstrate *Orion*'s suitability for autonomous robotics applications with stringent memory constraints.

To gain deeper insights into *Orion*'s handling of memory-constrained scenarios, we performed a detailed breakdown and time profiling of memory usage for GSS, as shown in

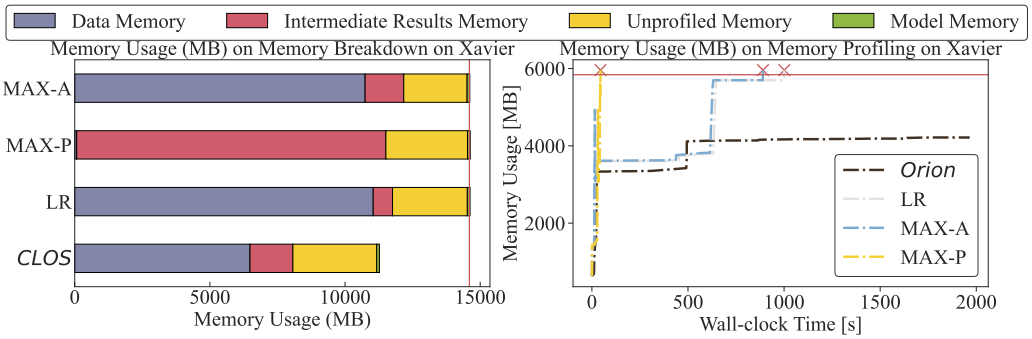


Figure 5.7: *Left:* Memory breakdown on GSS algorithm in the case study. The red line represents Xavier's maximum memory. *Right:* System-level memory usage profiling for GSS algorithm. The red line represents Xavier's maximum memory. Red crosses indicate the OOM point.

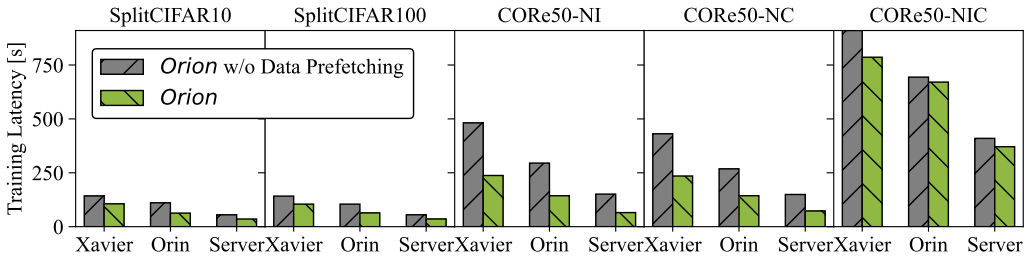


Figure 5.8: Ablation study of data prefetching.

Fig. 5.7. The left plot shows the memory breakdown by usage categories, revealing that MAX-A and LR heavily consume memory in data storage, while MAX-P uses excessive memory for intermediate activations, leading to OOM errors. In contrast, *Orion* auto-balances memory usage, ensuring smooth execution without exceeding available memory. The right plot in Fig. 5.7 presents a system-level memory usage profile over time for GSS. MAX-P quickly runs out of memory during batch executions, while MAX-A and LR last longer but eventually encounter OOM errors due to uncontrolled storage. Conversely, *Orion* maintains a stable memory footprint throughout training, demonstrating its practicality in real-world OCL applications.

Practical usability: *Orion* effectively auto-balances training latency, plasticity, and stability without out-of-memory errors in practical robotic scenarios.

5.5.4 Ablation Study

Data prefetching optimizes data loading without altering training parameters or the DNN model, thus preserving plasticity and stability unchanged. As shown in Fig. 5.8, data prefetching reduces training latency by 32.3%, 36.7%, and 37.6% on Xavier, Orin, and Server, respectively, demonstrating greater benefits for platforms with higher computational capabilities. The ablation studies demonstrate the effectiveness of *Orion*'s modular design and system prototyping. The seamless integration of these components enables *Orion* to achieve substantial gains in training latency performance while maintaining acceptable plasticity and stability.

5.5.5 Overhead Analysis

Execution Overhead. As shown in Tab. 5.5, the overall execution overhead of *Orion* remains below 2.1% of the total execution time, demonstrating its efficiency. Overhead from the `URGE` calculator, finer-grained controller, and data prefetching scales linearly with the number of experiences. There is some unprofiled execution overhead while remaining nearly unchanged, coming from the OCL framework itself.

Memory Overhead. Tab. 5.7 shows that *Orion* maintains a memory overhead of less than 1.0% across all platforms. Minor overhead arises from score calculation, finer-grained control, and selective approximation, with data prefetching contributing the most.

Table 5.5: Average runtime execution overhead [ms] and percentage of *Orion* across benchmarks on Xavier.

Benchmark	Calculator	Control	Prefetch	Overall
SplitCIFAR10	18 (0.02%)	2 (0.01%)	190 (0.18%)	2210 (2.10%)
SplitCIFAR100	18 (0.02%)	2 (0.01%)	195 (0.18%)	2215 (2.10%)
CORe50-NI	14 (0.01%)	8 (0.01%)	156 (0.07%)	2178 (0.93%)
CORe50-NC	17 (0.01%)	8 (0.01%)	176 (0.07%)	2201 (0.94%)
CORe50-NIC	144 (0.02%)	16 (0.01%)	1540 (0.20%)	3700 (0.47%)

Energy Overhead. Tab. 5.6 shows that *Orion* maintains a memory overhead of less than 0.1% across all platforms. Negligible overhead arises from all components.

Low overhead: *Orion*'s highly optimized implementation introduces negligible computational, memory burdens, and energy consumption, establishing it as a lightweight, deployable solution for sustainable On-Device OCL.

5.5.6 Discussions

Limitations. This work focuses on replay-based OCL under stringent memory constraints, excluding non-replay-based approaches [194, 222]. We do not address OCL for natural language processing tasks, large-scale datasets like ImageNet [78] and CLOC [190], or infinite streams [376, 395], which remain future work due to resource limitations. Following existing continual learning work [52, 294, 234], we assume data labels are available for supervised learning, as unsupervised continual learning remains a theoretical challenge. Notably, server-based solutions using large, highly accurate "golden models" [28] could provide annotations, but they are impractical for On-Device memory-constrained scenarios due to high memory resource demands.

Generality and Broader Impact. *Orion* is highly adaptable across diverse datasets, models, and hardware. Built atop `Avalanche-lib` [46, 233], it seamlessly integrates with

Table 5.6: Average energy execution overhead [Joules] of *Orion* across benchmarks on Xavier.

Benchmark	Calculator	Control	Prefetch	Overall
SplitCIFAR10	0.27	0.03	2.85	33.15
SplitCIFAR100	0.27	0.03	2.93	33.23
CORe50-NI	0.21	0.12	2.34	32.67
CORe50-NC	0.26	0.12	2.64	33.02
CORe50-NIC	2.16	0.24	23.10	55.50

Table 5.7: Memory overhead of *Orion*.

	(a) Overhead Breakdown on Module			(b) Overall Overhead Ratio		
Benchmark	Calculator	Control	Prefetch	Xavier	Orin	Server
SplitCIFAR10	1 KB	10 KB	3.2MB	0.041%	0.021%	0.016%
SplitCIFAR100	1 KB	10 KB	3.2MB	0.041%	0.021%	0.016%
CORe50-NI	1 KB	8 KB	3.2MB	0.041%	0.021%	0.016%
CORe50-NC	1 KB	9 KB	3.2MB	0.041%	0.021%	0.016%
CORe50-NIC	2 KB	79 KB	3.2MB	0.042%	0.021%	0.017%

scalable DNN training solutions [180, 13, 76, 323, 278] for both traditional and resource-constrained environments. Beyond navigational robotics, *Orion*'s modular and transparent architecture readily extends to other On-Device scenarios, including video analytics [28], fault diagnosis [191], user personalization [53, 86, 290], home automation [290, 404], and smart wearables [316]. Furthermore, *Orion* is orthogonal to and fully compatible with existing efficiency techniques. At the model level, it supports automated porting [127], model merging [280], compression [41, 77, 305, 396, 116, 358, 349], and quantization [186, 321, 300]. At the system level, it can integrate memory-saving optimizations like gradient checkpointing [123, 242], micro-batch execution [167, 351], recomputation [60, 351], and selective layer updates [357, 332, 356]. While our current evaluation relies on the standard ResNet-20 backbone for fair algorithmic comparison, validating our self-adaptive memory policies across a broader spectrum of modern model architectures remains important future work. Ultimately, we hope the engineering insights and design trade-offs will guide improvements

in other prominent OCL frameworks [84, 93, 266], facilitating the practical, widespread deployment of continual learning systems.

5.6 Related Work

Online Continual Learning (OCL) addresses the challenge of sequential learning from streaming data [52, 294, 234, 51, 10, 194, 222, 347, 348, 48, 333, 301]. Replay-based methods, such as ER [52], mitigate catastrophic forgetting by revisiting past experiences during new task learning[52, 294, 51, 234, 10, 9]. These methods perform well across diverse settings and are particularly effective in resource-constrained environments [294]. GEM [234] and AGEM [51] preserve prior knowledge using gradient-based constraints, while GSS [10] enhances memory efficiency by storing informative samples. These approaches balance memory retention and adaptability. In practice, OCL serves as an efficient framework for continuous robotic adaptation. It enables autonomous systems to seamlessly adjust to changing lighting and terrains, perform real-time novel object recognition [289], dynamically update person re-identification models across shifting environments [377], and incrementally perfect physical manipulation tasks [207].

Few studies explore system-level optimization for continual learning. For instance, LifeLearner [202] focuses on hardware-level optimization for meta-continual learning, addressing a distinct problem scope. Ekya [28] optimizes offline continual learning, e.g., iCaRL [301], for video analytics on high-end multi-GPU servers via dynamic workload scheduling, targeting different systems and applications. Latent Replay [289], which only modifies algorithms without considering system-level optimization, serves as a strong baseline in our evaluation.

Unlike these prior efforts, which either isolate algorithmic improvements from hardware realities or target resource-rich offline environments, *Orion* explicitly bridges the gap between OCL algorithms and edge system constraints. To the best of our knowledge, *Orion* is the first comprehensive framework to dynamically co-optimize latency, energy efficiency, and learning efficacy for online continual learning on embedded platforms. This enables sustainable, real-time robotic adaptation without exceeding the strict memory constraints inherent to low-power SoCs.

5.7 Conclusion

This paper presents *Orion*, a holistic solution for managing training latency, plasticity, and stability in On-Device OCL for memory-limited autonomous embedded systems. Our study highlights the trade-offs required in OCL systems on resource-constrained platforms. *Orion* effectively auto-balances these trade-offs while adhering to memory constraints. We evaluate *Orion* across state-of-the-art replay-based OCL algorithms, diverse benchmarks, and three heterogeneous hardware platforms, demonstrating its effectiveness and versatility. Although we focus on replay-based OCL algorithms, we believe the design principles of *Orion* can extend to other continual learning systems, laying a foundation for future On-Device OCL systems in intelligent systems.

Part II

Reducing Latency and Deadline Misses in Real-Time Learning

Chapter 6

RED: Systematic Real-Time Scheduling for Robotic Environmental Dynamics

6.1 Introduction

For complex cognitive analysis, intelligent robotic applications increasingly rely on co-executing *multiple* deep neural networks (DNNs), often for *correlated* tasks, known as multi-task inference [101, 247, 364, 201]. These DNNs, each trained for a specific inference task, are often run simultaneously on resource-constrained robotic platforms. Examples include personal robots that recognize sounds and places [208], autonomous vehicles that see the world from the front, side, and rear views [364], and home hubs that recognize emotions through both speech and facial expression [393, 155].

One imminent challenge for performing multi-task inference in robotic systems is their strict resource and real-time constraints [370, 394, 38, 206, 119, 120]. As modern deep learning algorithms are notoriously memory- and computation-hungry, it is challenging to deploy multiple DNNs on resource-constrained hardware platforms while satisfying strict real-time constraints. Weight sharing across deep neural networks is a class of newly developed methods for increasing drastically the efficiency of multi-task inference [157, 155, 208]. Memory usage can be reduced effectively as parameters are shared across DNNs, which is only possible when these DNNs are intended for correlated inference tasks. However, task correlation is commonplace as many DNNs in multi-task inference use the same input to infer distinct labels (single-input, multi-output) (SIMONet), merge complementary inputs to output a single label jointly (multi-input, single-output) (MISONet), or even a combination of both (multi-input, multi-output) (MIMONet).

Intelligent robotic applications often face another challenge: *environmental dynamics*. Intelligent robots expect to effectively negotiate dynamic, unpredictable environments crowded with moving mechanical elements and objects. Environment-induced dynamics such as moving obstacles could easily transform the computing demand (e.g., new tasks) and structure of workloads (e.g., precedence constraints among tasks) during runtime, negatively impacting overall system performance. Existing solutions rely on runtime scheduling to cope with the changing environment and runtime variations [119, 120]. However, as illustrated in Sec. 6.2, the problem herein is rather challenging due to (*i*) dynamically changing workload (Sec. 6.2.1), (*ii*) MIMONet architecture (Sec. 6.2.2), (*iii*) asynchronous inference (Sec. 6.2.3).

To this end, we propose MIMONet, a systematic real-time scheduling approach designed to enable multi-task DNN workloads in resource-constrained robotic systems that can adaptively handle environmental dynamics and satisfy real-time constraints (i.e., end-to-end deadline of an operation). The center of MIMONet is a deadline-based scheduler leveraging an intermediate deadline assignment policy, which could efficiently handle changing workloads and asynchronous inference challenges. This scheduling framework also flexibly supports MIMONet architecture (multi-input multi-output neural nets) [214] for memory-efficient deployment.

Leveraging the MIMONet-driven scheduling framework mentioned above appears to be a feasible and practical solution. However, a critical observation we had is that such a general scheduling framework does not consider the unique properties of MIMONet and may yield suboptimal performance. Specifically, MIMONet features a weight-shared architecture which can be exploited to reduce computation overhead further but may introduce new runtime optimization challenges.

To consider and exploit this feature unique to MIMONet, MIMONet develops a novel workload refinement and reconstruction process to make the scheduling framework compatible with MIMONet and fully explore its efficiency. An on-demand synchronization is integrated with this scheduling framework aiming to improve the overall system performance by reducing synchronization overhead. We implemented MIMONet on four embedded platforms pervasively used in robotic applications, including NVIDIA Jetson Nano, TX2, AGX Xavier, and AGX Orin, which exhibit different computing capabilities and architectural

characteristics. We evaluate MIMONet under a wide range of workloads and settings that navigation robots may experience.

Our main contributions and results are as follows. 1) To the best of our knowledge, we are the first to enable the effective execution of weight-shared DNNs with deadline-driven schedulers. 2) We designed MIMONet, a unified system that enables efficient multi-task inference on resource-constrained robotic platforms, dealing with real-time constraints in dynamic environments. This holds the potential for more advanced and efficient robotic applications in the future. 3) Our experimental results demonstrate:

- **On-Device effectiveness**: The MIMONet demonstrates significantly lower deadline miss rates by 40.5%, providing an average of 24.7% reduction in response time and making a 34.8% improvement in quality of experience as compared to the EDF baseline. (Refer to Sec. 6.4.2)

- **Practical usability:** MIMONet can be seamlessly integrated with widely-used robotic middleware ROS 2 and existing systems like NVIDIA IoT AI [171], requiring no extensive modifications. Compared to the EDF scheduler built upon ROS2, MIMONet provides 32.7% less response time, and lower deadline miss rates by 67.3%. (Refer to Sec. 6.4.3)

- **Robust adaptability**: MIMONet is capable of automatic adaptation to varying end-to-end deadline settings and diverse quality of experience requirements within acceptable ranges. (Refer to Sec. 6.4.4)

- **Low overhead**: MIMONet's efficient design and implementation incurs low offline profiling overhead and runtime overhead. (Refer to Sec. 6.4.5)

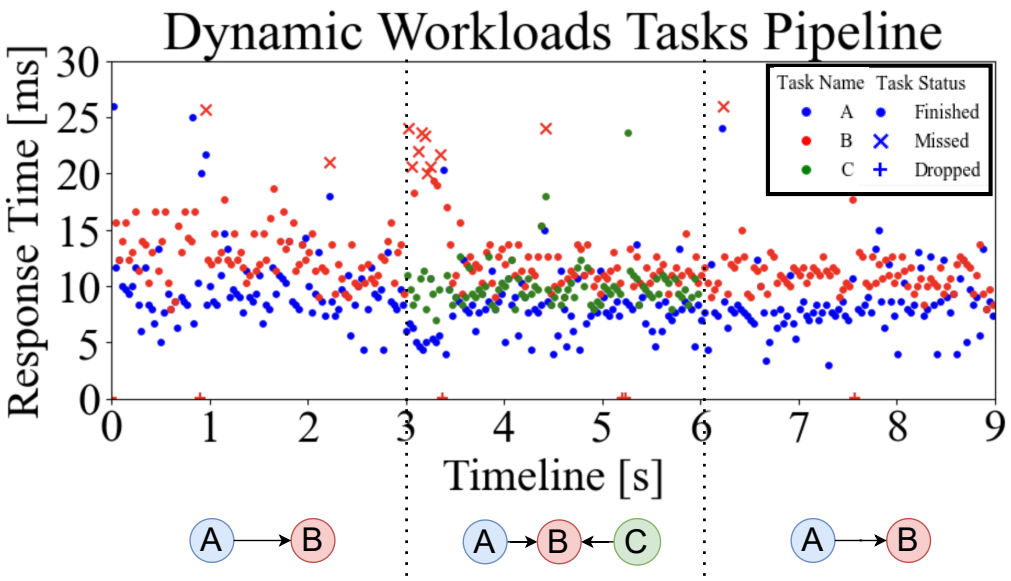


Figure 6.1: Robotic environmental dynamics induces changes of workloads structures. When environmental dynamics increase, the rate of missing deadlines, tasks dropped rate, and observed execution time of specific tasks dramatically increases, and vice versa.

6.2 Background and Motivational Case Study

In this section, we present several motivating case studies to examine and comprehend the unique challenges associated with operating multi-input multi-output deep neural network (MIMONet) driven robotic systems.

6.2.1 Challenges due to Dynamically Changing Workloads

Robotic systems often function in intricate environments that demand the ability to adapt to dynamically evolving and unpredictable situations. A case in point involves unexpected obstacles, requiring real-time workload adjustments to maintain timing correctness.

Table 6.1: This table shows deep learning-based models for navigating robot tasks. The Model column lists the specific models and the sources that describe the model architecture. The dataset column indicates the dataset to be used for each model. For some models, we reduced the resolution of the original model to fit the context of the embedded scenario. We choose YOLOv5n as the minimal version for the object detection task and scale the channel number to 1/4 official UNet.

Task	**Model**	**Dataset**	**FLOPs**	**Params**	**Mem**$_{arm}$	**Mem**$_{x86}$
Lane Detection	ENet [286]	KITTI [115]	4.111G	3.666M	3329MB	3796MB / 2073MB
Segmentation	UNet [310]	KITTI [115]	2.655G	1.943M	3359MB	3752MB / 1261MB
Cruise Control	AutoPilot [36]	Dave [282]	0.465G	2.489M	3680MB	3864MB / 1139MB
Object Detection	YOLO [302]	KITTI [115]	0.283G	5.286M	3593MB	3754MB / 1159MB
Multi-tasks	MIMONet [214]	KITTI [115]	6.704G	10.465M	3718MB	3739MB / 1611MB

A test scenario was established with a refitted TurtleBot3 powered by NVIDIA Jetson Nano, navigating from point A to point B and back, encountering random obstacles en route. The navigation process, composed of three tasks from the model table, evolved in three phases. Initially, the navigation comprised two dependent tasks: cruising control and segmentation. Upon detecting obstacles, an object detection task was added, causing a substantial increase in deadline misses. When the obstacles cleared, the structure reverted to the initial phase, and the rate of deadline misses declined.

The pivotal challenge, thereby, lies in the dynamic environments where robotic systems operate, inducing changing workload structures. These shifts necessitate real-time adaptive decisions to maintain the end-to-end deadline. The case study data is illustrated in Figure 6.1 and Table 6.1.

As demonstrated in Figure 6.1, the navigation process, encapsulating three key tasks from Table 6.1, unfolds in three distinct phases. The initial phase naturally includes two interdependent tasks: cruising control followed by segmentation. Upon encountering an obstacle at a specific juncture (t=3), the object detection task is necessitated, and subsequently incorporated into the workload Directed Acyclic Graph (DAG). Once the

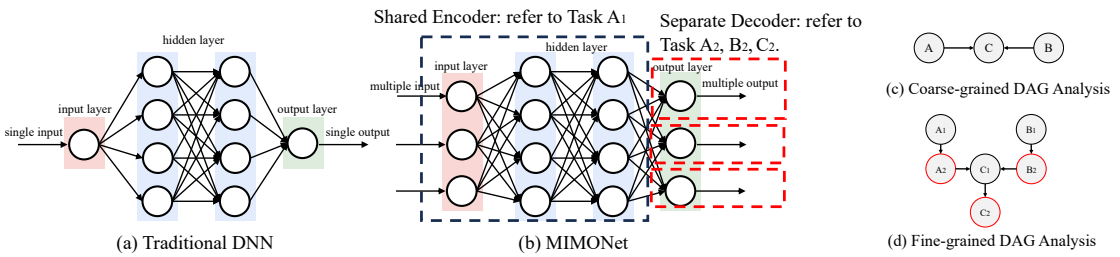


Figure 6.2: The architecture of MIMONet compared to traditional DNN. The structural characteristics of MIMONet (partitionable components) naturally facilitate finer-grained DAG partitioning. Shared encoder inference in (b) marked with a black dashed box refers to a black node in (d), and each separate decoder in (b) marked with a red dashed box refers to a red node in (d).

environment clears at t=6, the DAG structure reverts to its original configuration. A, B, and C represent cruising control, segmentation, and object detection. The deadline for A, B, and C is set to 50ms, 30ms, and 50ms, based on their execution times as well as a pre-defined end-to-end deadline of 130ms.

Figure 6.1 reveals a significant escalation in deadline misses during the second phase, coinciding with the unexpected integration of the object detection task into the workload DAG. Specifically, the miss and drop rate for task B surges from 3.3% to 30.0%, resulting in an average deadline missing rate of 13.0% within the interval (3,6). Despite this, the system's GPU remains underutilized during this period. Of note, if task B (the sink node in the DAG) fails to meet its deadline, it leads to a violation of the operation's end-to-end deadline. The removal of task C at t=6 sees a drop in the rate of missed and dropped deadlines for task B to 2.2%.

Challenge 1: Robotic systems frequently operate in dynamic environments that induce fluctuating workload structures. Consequently, robotic system software is required to make real-time adaptive decisions in response to these changing workloads to ensure the end-to-end deadline is met.

6.2.2 Challenges due to the MIMONet Architecture

In response to the dynamic challenges posed by robotic systems, advanced computational techniques have become crucial. One area of considerable progress is the evolution of deep neural networks (DNNs). The past decade has witnessed a dramatic expansion of computational capabilities, contributing significantly to the success of DNNs. The field of self-supervised learning-based Large Language Models (LLMs) has notably improved algorithm performance, primarily attributed to their remarkable generalizability, facilitating smooth application across various tasks. Numerous applications founded on these models, including CLIP [297], DINOv2 [277], GPT-4 [276], and Bard [340], have attracted significant attention. These applications leverage the core technique of MIMONet (Multiple-Input-Multiple-Output Deep Neural Networks), an advanced DNN architecture enabling simultaneous processing of multiple inputs and outputs for efficient and versatile model training (Figure 6.2). The superior performance of MIMONet is rooted in its ability to decipher complex patterns and dependencies within input data, thereby fostering improved representations and adaptability across a wide spectrum of tasks.

Building on the triumphs of DNNs, the MIMONet architecture has showcased its potential in diverse applications, such as intelligent robots operating within resource-limited embedded systems. Despite variations in implementation, MIMONet architectures manifest common workload characteristics, presenting considerable challenges when deployed in such environments. These characteristics include:

- **Memory-Efficient Architecture:** MIMONet is specifically designed to optimize memory usage, making it highly suitable for environments with resource constraints. Even in

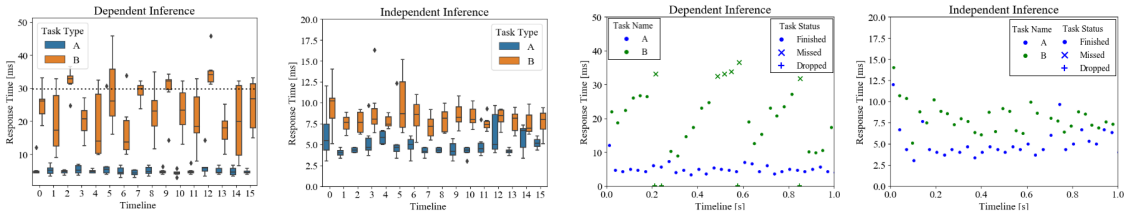


Figure 6.3: Data dependency leads to periodic system's real-time performance degradation in asynchronous inference scenarios. The first two figures present a statistical analysis of end-to-end response times under the two scenarios. The last two figures summarize the task instances that meet, miss, or are dropped by the system.

its simplest form (as depicted in Figure 6.2), MIMONet requires only a modest number of additional parameters to facilitate multi-input multi-output support, as compared to traditional DNNs. Table 6.1 highlights a selection of critical tasks along with their respective standard models and datasets, underpinning the DNN-driven navigation robots. It is noteworthy that employing MIMONet, as opposed to multiple distinct DNNs, can substantially reduce memory consumption on both ARM-based and x86-based platforms. This observation underscores the memory-efficient design of MIMONet, making it inherently suitable for memory-constrained embedded systems.

- **Partitionable Components:** As depicted in Figure 6.2, unlike a traditional DNN (Figure 6.2(a)) that accepts a single input and generates a single output, the network architecture of MIMONet (Figure 6.2(b)) is designed to accept multiple inputs and generate multiple outputs within a single DNN simultaneously. This architectural design boasts a naturally partitionable structure, allowing the inference process to be decomposed into dependent smaller, manageable subtasks that can be scheduled. This feature introduces increased flexibility and adaptability in complex environments, as the subtasks can be dynamically allocated and reconfigured in response to changing environments. Figure 6.2 further presents a simple showcase to illustrate the potential for MIMONet-enabled

optimization based on its structural characteristics. This naturally invites finer-grained DAG partitioning while introducing additional optimization challenges, compared to simply MIMONet-agnostic DAG analysis (see Figure 6.2(c)). Specifically, the inference subtask in the black dashed box part, as in Figure 6.2(b), corresponds to the black node of the DAG in Figure 6.2(d) (A_1, B_1, C_1), while the inference in the red dashed box corresponds to the red node of the DAG (A_2, B_2, C_2).

Challenge 2: The MIMONet architecture, with its memory-efficient design and partitionable structure, provides a robust solution for intelligent embedded systems. However, it simultaneously presents new runtime optimization challenges that stem from the need to dynamically manage and schedule multiple subtasks in real time in response to changing environmental conditions.

6.2.3 Challenges due to Asynchronous Inference

In the context of deep learning, asynchronous inference refers to the execution of DNN model inferences without a fixed, synchronized order or timing. Instead of processing data in a strict sequence or at consistent intervals, asynchronous inference allows tasks to be processed based on the availability of resources, task dependencies, or other dynamic factors. This flexibility provides advantages in terms of resource utilization and responsiveness, but it also introduces complexities with respect to task coordination, timing, and system stability.

Specifically, we conduct a case study to attribute complexities in DNN-based robotic systems by introducing asynchronous inference through a two-task asynchronous inference scenario, as shown in Figure 6.3.

In this scenario, a TurtleBot3 navigates an obstacle-ridden, hazardous environment, guided by an exploration mission driven by MIMONet systems. The scenario necessitates the constant operation of the object detection module and includes two tasks: Task A, overseeing cruising control, and Task B, managing object detection. Task A operates at a frequency of 30Hz (1/30s) while Task B runs at 33Hz (1/33s), showcasing a subtle variance in frequency data streams. Both tasks have a deadline of 30ms.

This case study explores the implications of asynchronous inference under two conditions: when Tasks A and B are interdependent and when they are not. As Fig. 6.3 illustrates, in the absence of interdependency, both tasks demonstrate a uniformly distributed performance pattern. However, the introduction of task interdependency results in a significant increase in deadline misses, highlighting the challenges posed by asynchronous inference.

Challenge 3: The operational framework of a Directed Acyclic Graph (DAG) can spawn asynchronous inference due to task dependencies, potentially causing cascading delays or system failures. This complexity is further intensified when dealing with MIMONet components, which naturally exhibit task interdependence post-partitioning, thereby exacerbating the challenges associated with asynchronous inference. Therefore, it is imperative for robotic system software to incorporate this asynchronicity into runtime resource management decisions.

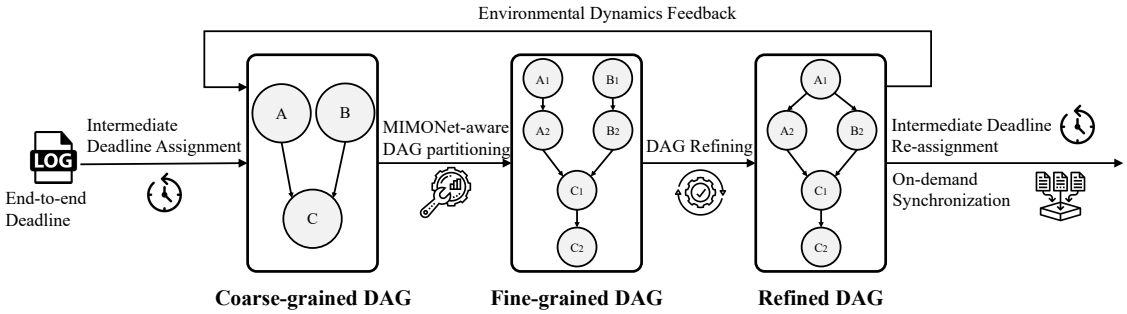


Figure 6.4: Overview of MIMONet.

6.3 System Design

6.3.1 Overview

In response to the intricate challenges of ensuring timing correctness for operating robots in complex dynamic environments, as discussed in Sec. 6.2, we introduce MIMONet, a comprehensive MIMONet-driven DAG scheduling framework. MIMONet inference highly relies on GPU resources. Since most embedded platforms are equipped by robots containing one GPU, this study targets single-GPU autonomous embedded platforms, e.g., NVIDIA Jetson Nano, TX2, Xavier, Orin, etc. Note that our proposed method can be further extended to multi-GPU settings. As illustrated in Figure 6.4, MIMONet consists of three primary components: (*i*) A simple yet highly effective intermediate deadline-based scheduler that efficiently adapts to dynamically changing workloads (Challenge 1), (*ii*) A MIMONet-driven DAG refinement and re-assignment policy that resolves issues arising from the MIMONet structure (Challenge 2), and (*iii*) An on-demand synchronization mechanism that significantly enhances overall performance under asynchronous inference system scenarios (Challenge 3). Each component functions in conjunction to address unique challenges and promote optimal system operation, with detailed discussions following below:

- **Intermediate Deadline Assignment Policy:** The heart of our framework is a robust policy that assigns intermediate deadlines to tasks. This policy is designed to be simple yet highly effective in dealing with the dynamically changing workloads prevalent in complex robotics environments. It ensures efficient scheduling and timely completion of tasks, thereby maintaining the operational correctness of the robots.

- **MIMONet-Driven DAG Refinement and Deadline Re-Assignment:** To tackle the issues stemming from the MIMONet structure (Challenge 2), our framework incorporates a process of DAG refinement and deadline re-assignment driven by the MIMONet. This component refines the task nodes, adjusting the granularity of tasks and their interconnections according to the weight-shared architecture of MIMONet, which in turn enhances system efficiency against dynamic environmental changes.

- **On-demand Synchronization:** The last key component of our framework is an on-demand synchronization system aimed at significantly improving the overall performance via largely reducing synchronization overhead. By allowing on-demand synchronization, this system ensures that tasks are executed in harmony with the changing dynamics of the environment, thereby minimizing latency and enhancing the timeliness and effectiveness of robotic operations.

Through the seamless integration of the three core components—intermediate deadline assignment policy, MIMONet-driven DAG refinement, and on-demand synchronization system, MIMONet emerges as a high-performance solution to maintain timing correctness for robotic operations in complex environments. As depicted in Figure 6.4, this framework adroitly handles dynamically changing workloads, resolves challenges intrinsic

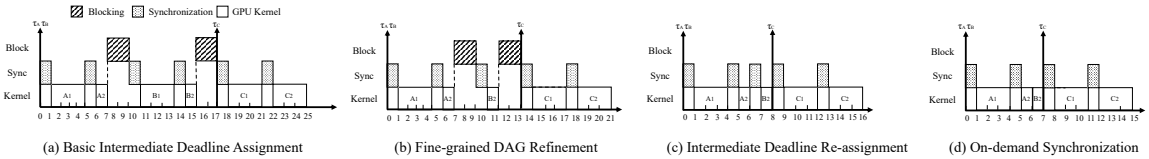


Figure 6.5: An illustrative example to apply each component of our design to a MIMONet-driven DAG scheduling. **(a)** Basic intermediate deadline assignment enables incorporation with real-time scheduler and further optimization. **(b)** Fine-grained DAG refinement reduces computation costs by merging some tasks. **(c)** Intermediate deadline reassignment reduces blocking time at runtime. **(d)** On-demand synchronization reduces unnecessary synchronization overhead.

to the MIMONet structure, and significantly improves overall performance by minimizing synchronization overhead. The holistic design of MIMONet, therefore, offers an efficient system for managing the demands of real-time operations in changing environments, ensuring optimal performance and effective robotic operations

6.3.2 Intermediate Deadline Assignment

Achieving the end-to-end deadline of a robotic operation, represented by a Directed Acyclic Graph (DAG) task, necessitates an efficient deadline assignment strategy. To this end, we propose an intuitive intermediate deadline assignment approach that assigns an intermediate deadline to each node in the DAG task. Essentially, by ensuring each node meets its intermediate deadline, we guarantee the completion of the overall operation within the end-to-end deadline.

With the above understanding of MIMONet (as in 6.2.2), we can explore how it interacts with Directed Acyclic Graphs (DAGs) and apply this to a practical example. Consider a DAG τ_i with a relative end-to-end deadline D, comprising k nodes $\tau_i^1, \ldots, \tau_i^k$. We assign an intermediate deadline D_i^k to each node. Each node τ_i^k has a height $H(\tau_i^k)$,

representing the length of the longest path from any source node to τ_i^k. By allocating an intermediate deadline $D_i^{H_j}$ to all nodes of height H_j and ensuring $\sum D_i^{H_j} = D_i$, the end-to-end-deadline is met if each node in the DAG completes by its assigned intermediate deadline. The partitionable nature of MIMONet's design allows for such tasks to be carried out effectively and efficiently while meeting these intermediate deadlines.

The challenge lies in identifying the optimal intermediate deadline assignment strategy. While equal or proportional assignment to each node's execution time are both intuitive strategies, MIMONet adopts the latter. Our choice is backed by both empirical evidence from our evaluations and the logical inference that a proportional assignment strategy increases the likelihood of nodes meeting their assigned deadlines. Under an equal assignment strategy, nodes with longer execution times could more easily breach their assigned deadlines.

For instance, consider a coarse-grained DAG shown in Figure 6.4 with an end-to-end deadline of 120ms, where nodes A, B, and C have execution costs of 20ms, 20ms, and 40ms, respectively. Given the dependencies among these nodes, we would assign intermediate deadlines to A, B, and C of 40ms, 40ms, and 80ms, respectively. Thus, if all nodes complete within their intermediate deadlines, the end-to-end deadline is met. Note that a node in the DAG commences execution immediately once all its predecessors have completed. Similarly, in the partitioned DAG in Figure 6.5**(a)**, the proportional intermediate deadline can also be assigned directly and work in suboptimal.

While our adoption of proportional intermediate deadline assignment may seem straightforward, it proves to be highly effective and lays the groundwork for the subsequent

components of our framework, namely the MIMONet-driven DAG scheduling and on-demand synchronization.

6.3.3 MIMONet-driven DAG Scheduling: Refinement and Re-Assignment

The scheduling of DAG tasks within MIMONet involves two primary components: fine-grained DAG refinement and MIMONet-driven intermediate deadline reassignment.

Fine-grained DAG Refinement

Weight-shared architectures, such as MIMONet employed in MIMONet, enable improved system efficiency by sharing parameters across multiple simultaneous tasks as illustrated in Sec. 6.2.2. This structure, comprising a shared encoder followed by several separate decoders, maintains inference accuracy while achieving a smaller overall model size [47, 244].

However, this parameter-sharing approach presents unique scheduling challenges. Tasks executed simultaneously can share the intermediate output from the shared encoder, reducing computation costs, as illustrated in Figure 6.5**(b)**. Conversely, tasks that start asynchronously cannot share these results, increasing overall computation. It is optimal to delay early tasks slightly to enforce synchronous execution, thereby saving computation and reducing latency.

To address such a challenge, we propose a MIMONet-driven DAG structure refinement. This approach involves performing a finer-grained DAG analysis, leveraging the shared encoder characteristics, thereby significantly reducing computational redundancy and costs. We split each inference sub-task in MIMONet into two dependent parts (encoder

Algorithm 5 MIMONet DAG structure refining

1: **Input:** Task Dependency Graph G
2: **Output:** Fine-grained Task Dependency Graph G', Task List S, Merged Task $\widetilde{T}$
3: Initialize: Refining granularity hyperparameter γ.
4: $G' = \emptyset$
5: **while** $G \neq \emptyset$ **do**
6: Conduct topological sort on G
7: $S = \{v \in V : \text{indegree}(v) = 0\}$
8: $\widetilde{T} = \textsc{DynamicMerge}(S, \gamma)$
9: $G' \leftarrow G' \cup \widetilde{T}$
10: $G \leftarrow G \setminus S$
11: **end while**
12: **return** G'

inference/decoder inference) and perform semantic-preserving reconstruction of finer-grained DAGs under certain constraints, thus reducing computation cost and taking advantage of parallelism. Algorithm 1 details this refining process.

Let $G = (V, E)$ be a topologically sorted directed graph where V is the set of nodes and E is the set of edges. Define the indegree of a node v as:

$$\text{indegree}(v) = |\{u \in V : (u, v) \in E\}|$$

. The set S is formed by taking all nodes in V with indegree(0) (Line 7). Following that, a dynamic merge operation is conducted (Line 8). The objective of this merge operation is to maximize the efficiency of task execution in a MIMONet architecture, through leveraging the shared encoder characteristics to minimize computational redundancy and costs. Note that only sub-tasks exhibiting release time differences within γ will be merged.

MIMONet-driven Intermediate Deadline Re-Assignment

Following intermediate deadline assignment to nodes within a DAG task, the primary objective is to schedule all nodes to meet their deadlines, thereby guaranteeing the fulfillment of the DAG's end-to-end deadline. To achieve this, MIMONet utilizes the Earliest Deadline First (EDF) scheduling methodology, renowned for its efficacy within both soft and hard real-time system contexts [369]. This strategy, based on EDF scheduling, schedules and prioritizes nodes according to their allocated intermediate deadlines. Despite its straightforward nature, this EDF-focused scheduling technique competently addresses major challenges such as dynamic alterations and refinements in workload structure.

Our preliminary studies revealed a distinctive optimization potential in the realm of deadline re-assignment. Specifically, we found that recalculating intermediate deadline assignments at each scheduling point significantly enhances the system's overall average response time performance and minimizes the deadline miss ratio. This is particularly noticeable when a node's execution speed substantially deviates from its estimated execution costs. Furthermore, as illustrated in Figure 6.5**(c)**, re-assignment also reduces blocking time[1], thereby decreasing overall end-to-end latency. Hence, in such circumstances, re-assigning intermediate deadlines can more accurately reflect the real-time workload demand within the system.

Our method dynamically adjusts to the addition or removal of a node from the DAG task by rapidly recalculating the intermediate deadline assignment and scheduling

[1]Blocking time is mainly caused by acquiring a lock on shared memory used by NVIDIA Jetson embedded devices.

nodes according to their newly assigned deadlines. Despite adding a layer of complexity, this dynamic re-assignment assures optimal resource utilization and increases the probability of adhering to end-to-end deadlines.

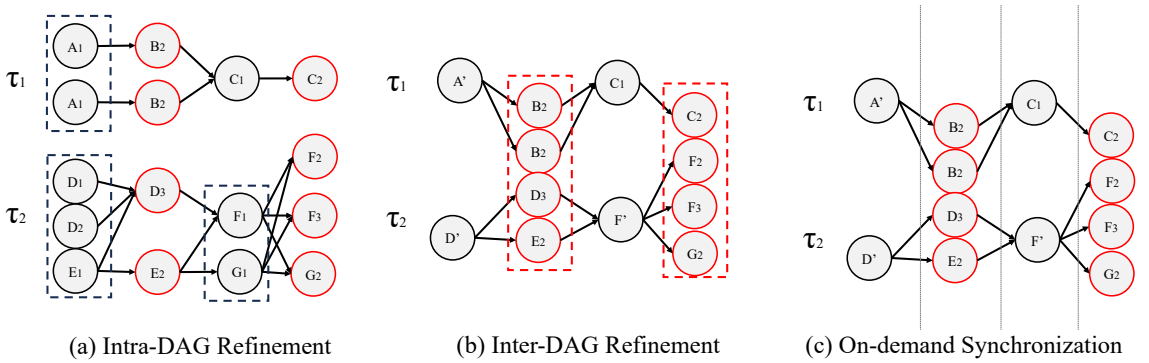
6.3.4 On-demand Synchronization

Synchronization plays a pivotal role in orchestrating our framework, facilitating effective information sharing throughout the system. However, a naively implemented synchronization strategy can introduce significant overhead, hampering the system's overall performance. Therefore, we leverage an on-demand synchronization strategy to mitigate this issue.

An intuitive, periodic synchronization strategy calls for regular, pre-scheduled synchronization intervals, irrespective of the system's state or needs. This strategy can result in high overhead, as unnecessary synchronization operations may be carried out even when there are no significant changes in the system state. These redundant operations consume valuable computational resources, leading to inefficiencies.

On the other hand, an on-demand synchronization strategy allows for more efficient resource usage. This strategy triggers synchronization only when necessary, i.e., when significant changes in the system state or meeting specific conditions. The synchronization operations are thus tied directly to the system's needs, reducing redundant operations and conserving computational resources. This results in a more responsive system with lower overhead, as illustrated in Figure 6.5**(d)**.

We use a multi-DAG parallel execution scenario to illustrate further combining on-demand synchronization with other components, as in Figure 6.6. In this example, within



(a) Intra-DAG Refinement (b) Inter-DAG Refinement (c) On-demand Synchronization

Figure 6.6: A multi-DAG parallel execution example. Intra-DAG refinement aims to merge nodes in the computation graph, thus reducing the computational cost. Inter-DAG refinement increases the parallelism of non-mergeable nodes by batch execution. On-demand synchronization further boosts latency performance by removing unnecessary synchronization overhead.

each DAG, we conduct intra-DAG refinement to reduce computation costs by merging some nodes as illustrated in Sec. 6.2.2. Beyond one single DAG, we parallel-execute unmerged independent nodes with the same heights to maximize parallelism. By default, synchronization happens on the edges of nodes. We conduct synchronization only when all the nodes with the same heights finish, reducing the synchronization costs.

Global and Local Synchronizers Implementation. To effectively implement on-demand synchronization in our framework, we introduce a global synchronizer and multiple local synchronizers. The global synchronizer is responsible for system and application-level coordination. Specifically, it periodically invokes the scheduler to dispatch control signals to task handlers, which then request computing resources based on these signals. Upon completion of an inference request, task handlers relay feedback to the scheduler. At a lower level, each task handler has a dedicated local synchronizer maintaining its internal state. We abstract the handler state into a finite state machine, as depicted in Figure 6.7. The local synchronizer periodically reviews the state of this finite state machine. When a

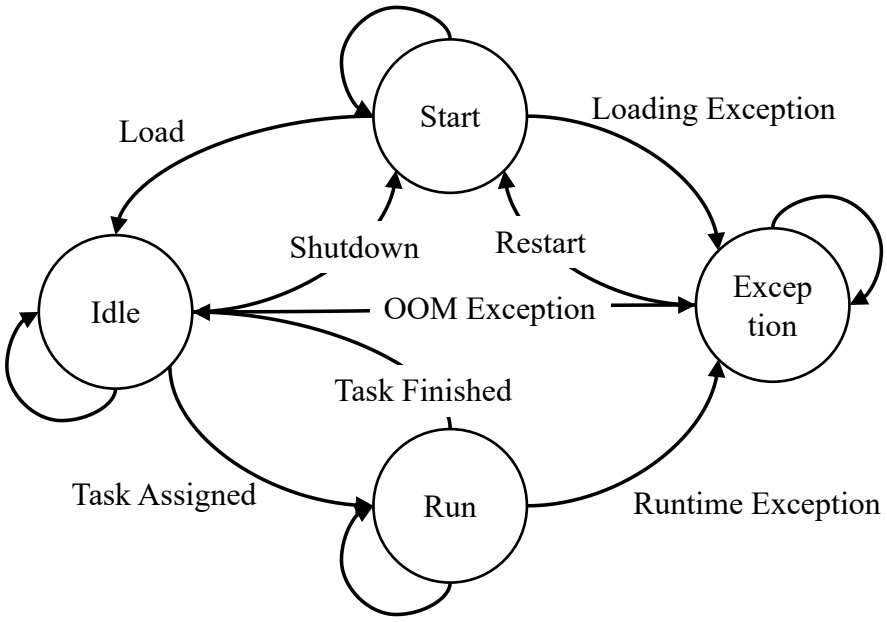


Figure 6.7: Abstract design of finite state machine for handlers. "OOM" refers to out-of-memory.

state transition occurs, the local synchronizer sends this information to the controller to inform subsequent scheduling decisions. The global and local synchronizers are implemented based on a spin-lock design, which ensures rapid response times and minimizes overhead, while slightly increasing CPU consumption. This implementation strikes a balance between efficiency and performance, significantly enhancing the system's overall responsiveness and effectiveness. Based on all modules mentioned above, the overall integrated task orchestration algorithm is presented in Algorithm 6.

6.4 Evaluation

In this section, we test our full implementation of MIMONet on top of the PyTorch framework with an extensive set of evaluations. We explore its overall effectiveness under different deadline configurations, from tight to relaxed deadlines, across several hardware

Algorithm 6 Integrated Task Orchestration Algorithm

Input: DAG Task τ, End-to-end deadline D, MIMONet Model M, Timing Requirements T_{req}
Output: Refined DAG Task τ', Scheduled Task Execution
$\tau' \leftarrow$ IntermediateDeadlineAssignment(τ, D)
$\tau' \leftarrow$ MIMONetDAGRefinement(τ', M)
InitializeSynchronizers(T_{req})
while not all tasks τ_i in τ' are completed **do**
    OnDemandSynchronization()
    $\tau_{next} \leftarrow$ NextTaskToSchedule(τ')
    $\tau' \leftarrow$ MIMONetDAGReAssignment(τ', τ_{next})
    ExecuteTask(T_{next}) ▷ Execute the scheduled task
end while

platforms with distinct characteristics (Sec. 6.4.2). This evaluation aims to demonstrate the general usability and flexibility of MIMONet in a wide range of scenarios. Next, we delve into a practical case study, integrating MIMONet with ROS2 on NVIDIA IoT AI, to understand its effectiveness and adaptability in real-world, complex situations (Sec. 6.4.3). This highlights how seamlessly our solution can be integrated with ROS, an aspect we introduced in the introduction. Additionally, we conduct a parameter study to demonstrate MIMONet's flexibility in handling varying deadline settings (Sec. 6.4.4). We then evaluate the computational overhead of the MIMONet framework (Sec. 6.4.5), providing insight into its operational costs. We conclude the evaluation with a discussion on the outcomes and implications of our analysis (Sec. 6.4.6).

6.4.1 Experimental Setups

This section details a comprehensive evaluation setup, complete with varying hardware platforms, a diverse set of tasks, and a range of deadline configurations, designed to thoroughly test the MIMONet framework under a variety of conditions. This breadth of

Table 6.2: Hardware platforms used in our experiments.

	Nano	**TX2**	**Xavier**	**Orin**
CPU	4-core ARM Cortex-A57 CPU @ 1.43GHz	6-core ARM Cortex-A57 CPU @ 1.70GHz	8-core Armv8.2 Carmel CPU @ 2.03GHz	12-core Armv8.2 Cortex-A78AE @ 2.20GHz
GPU	NV Maxwell GPU	NV Volta GPU	NV Volta GPU	NV Ampere GPU
Memory	4GB LPDDR4	8GB LPDDR4	16GB LPDDR4x	32GB LPDDR5
Storage	16GB eMMC 5.1	32GB eMMC 5.1	32GB eMMC 5.1	64GB eMMC 5.1

testing helps ensure that MIMONet can deliver consistent, high-quality performance across many potential use-cases in robotic navigation.

Testbeds. We use pre-trained models or train DNNs following the original paper's setting on a server with Intel(R) Xeon(R) CPU E5-2650 and a GeForce RTX 2080 Ti GPU. We conduct forward inference experiments on four NVIDIA autonomous embedded platforms as shown in Table 6.2. These platforms are widely used in robotics research [293] applied as the mainboard of various well-known industrial robots, e.g., Duckiebot [269], SparkFun Jetbot [271], Waveshare Jetbot [273], etc.

Metrics. We evaluate our system's performance primarily through three critical metrics: latency, deadline miss rate, and Quality of Experience (QoE) score. Latency is the time taken for a request to be processed and returned, with lower times indicating faster responses. The deadline miss rate is the proportion of tasks that fail to meet their specified deadlines, which we aim to minimize. Lastly, we define the QoE score (detailed in Eq.6.1) motivated by [201], providing a comprehensive measure of user satisfaction in soft real-time systems by considering aspects such as responsiveness, reliability, and overall service quality.

$$QoEScore(r) = \frac{1}{1 + e^{\lambda}[max(0, C_r - S_r)]} \tag{6.1}$$

Here, C_r represents the execution time of the inference task r, and S_r is the slack of task r. λ is a hyper-parameter controlling latency tolerance. This metric refers to the exponential distribution, and for this study, we set λ to 1. By closely monitoring and optimizing these metrics, we ensure a seamless and efficient user experience.

Baselines. We implement the MIMONet scheduling framework (refer Sec. 6.3.1), which leverages a single MIMONet model to adhere to memory constraints in robotic embedded deployment. We compare MIMONet against these baselines[2]: (1) **EDF** [226]: a strict version of the EDF scheduler; (2) **MIMONet-FG**: fine-grained MIMONet partitioning with proportional deadline assignment [283] following EDF scheduling; (3) **MIMONet-IDA**: an optimized intermediate deadline assignment (IDA) based on MIMONet-FG serving as a robust baseline; and (4) **MIMONet**: the comprehensive MIMONet implementation with fine-grained DAG partitioning, optimized IDA, and an on-demand synchronization mechanism.

Minibenchmark. To examine the temporal correctness and throughput performance of the scheduling approach in a dynamic robotic environment, we establish a mini-benchmark employing deep learning tasks as in Table 6.1. As detailed in Table 6.3, we configure a sequential three-stage task setting for both environments, each predicated on completing

[2]Recall that most of the existing DAG scheduling methods could not handle dynamically changing workloads and thus are not applicable to our problem scope. To our knowledge, one feasible solution to handle the dynamically changing workload scenario is the classical intermediate deadline assignment method designed for scheduling DAGs [283, 226], which we evaluated as one of the baselines.

Table 6.3: Minibenchmark design for evaluating multi-DNN based robots in dynamic environments. "L", "S", "C", and "O" refer to lane detection, segmentation, cruise control, and object detection, respectively.

	Stage1	**Stage2**	**Stage3**
Obstacle-free cruising	L	S	C
Obstacle cruising	L	S+O	C

the previous stage. The end-to-end task involves sequential execution of the obstacle-free cruising task set and obstacle cruising task set. Each task set is executed 10 times in the benchmark. The end-to-end deadline for each deep learning task instance, based on the platform-wise worst-case execution time (WCET), is set under two configurations: tight and loose. For our mini benchmark, the tight and loose deadlines were established as 9815ms and 11325ms for Nano; 8400ms and 10080ms for TX2; 6500ms and 7315ms for Xavier; and 2400ms and 3600ms for Orin. We set hyperparameter γ as 100ms. Note that for almost all these benchmark workloads, we observe that GPU utilization reaches nearly 100% consistently. This implies that modern DNN inference is extremely GPU-intensive and can naturally exploit the maximum parallelism provided by GPU hardware employed on most embedded devices.

6.4.2 Overall Effectiveness

In this section, we conduct a comprehensive evaluation study of MIMONet, taking into account all its components. Our design aims to optimize system throughput, real-time performance, and quality of experience (QoE).

Throughput. As depicted in Figure 6.8, MIMONet consistently outperforms the baseline approaches under all variant deadline settings. The average latency is reduced by 24.7%,

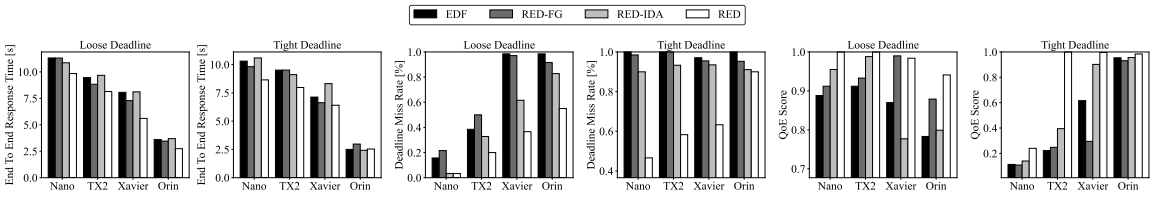


Figure 6.8: Overall effectiveness of MIMONet evaluated on four different resource-constrained intelligent robotic systems. MIMONet optimizes throughput, real-time performance, and QoE, demonstrating substantial improvements over baseline methods.

17.3%, and 14.2% compared to EDF, MIMONet-IDA, and MIMONet-FG, respectively. These results demonstrate the overall throughput effectiveness of MIMONet. Furthermore, MIMONet-FG and MIMONet-IDA both outperform EDF in seven out of eight settings. This further illustrates that each subcomponent of our design could almost always result in throughput performance improvements. Delving into the details, we observe that MIMONet-FG and MIMONet-IDA significantly outperform EDF in all loose deadline settings. However, under tight deadline settings, they outperform EDF in three out of four cases correspondingly. This observation suggests that while the subcomponents of MIMONet can improve throughput in most cases, their effectiveness might be more pronounced in scenarios with less stringent deadlines. The reduced effectiveness under tight deadline settings could be due to the increased scheduling complexity or the limited optimization opportunities in such constrained environments. To further demonstrate the effectiveness of our proposed MIMONet, we show the histogram of task-level response time in the case of tight and loose deadline setting, respectively. As depicted in Figure 6.9, we plot the response time histogram of the proposed MIMONet compared with EDF. MIMONet's holistic efficient design offers significantly faster responses qualtitavely.

Real-time Performance. Figure 6.8 illustrates the deadline miss rate for different approaches tested on four embedded devices. The results reveal that our proposed MIMONet

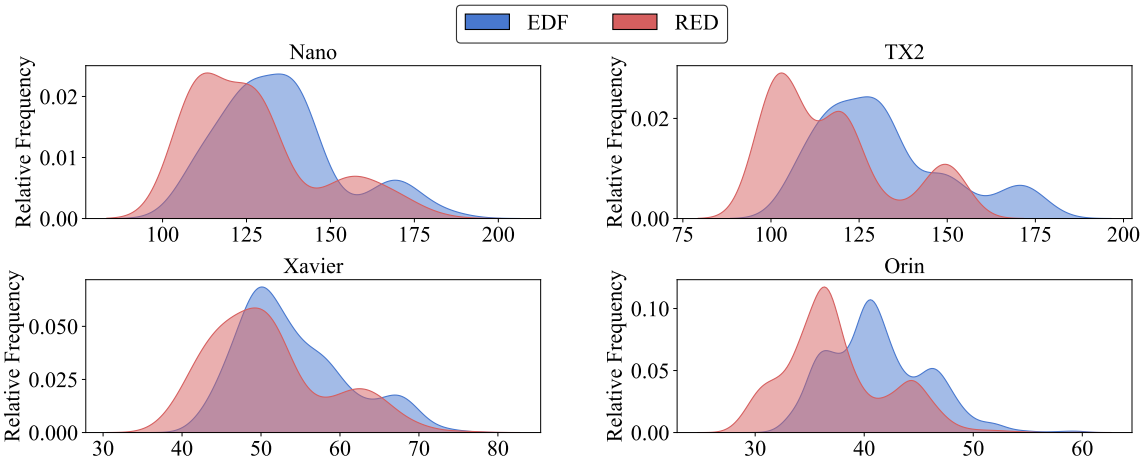


Figure 6.9: Response time histogram of MIMONet across different embedded platforms compared to EDF.

achieves superior real-time performance, surpassing the baselines considerably. Specifically, MIMONet significantly outperforms the baseline approaches under nearly all settings, with an average deadline miss rate lower than that of EDF, MIMONet-IDA, and MIMONet-FG by 40.5%, 37.8%, and 37.5%, respectively. Interestingly, we observe that merely applying finer-grained DAG partitioning (MIMONet-FG) and further implementing optimized intermediate deadline assignments (MIMONet-IDA) results in a marginal improvement over the EDF baseline, with increases of 4.8% and 4.4%, respectively. This marginal improvement can be attributed to various factors, one of the most significant being the high synchronization overhead. MIMONet's on-demand synchronization mechanism effectively reduces this overhead, leading to markedly better real-time performance.

Quality of Service. In the embedded system scenario, imposing a stringent deadline on the heavy MIMONet workload often results in missed deadlines, as shown in Figure 6.8. This section investigates the potential of MIMONet in enhancing the quality of experience (QoE) across various settings, following the methodology outlined in [201]. Figure 6.8 presents the

QoE for different approaches evaluated on four embedded devices. The findings demonstrate that MIMONet consistently achieves superior QoE, significantly outperforming the baseline methods. In particular, MIMONet surpasses the QoE scores of EDF, MIMONet-IDA, and MIMONet-FG on average by 34.8%, 34.2%, and 15.9%, respectively. In the loose deadline setting, MIMONet exceeds others by a relatively smaller margin on average (13.1%, 9.8%, and 1.4%). Conversely, in the tight deadline setting, the baseline candidates exhibit lower QoE, with MIMONet significantly outperforming them on average by 76.4%, 84.7%, and 40.4%. Notably, in specific scenarios such as TX2-tight, EDF's QoE score is a mere 0.222, while MIMONet achieves a QoE of 1.0, which is 3.50 times higher than the EDF baseline. In summary, MIMONet consistently delivers superior QoE across various settings in MIMONet workload systems, outperforming baseline methods by substantial margins, particularly under tight deadline constraints.

On-Device overall effectiveness: MIMONet effectively addresses all challenges outlined in Sec.6.2 for resource-constrained intelligent robotic systems. It optimizes throughput, real-time performance, and QoE in MIMONet workload systems, demonstrating substantial improvements over baseline methods, especially under tight deadline constraints.

6.4.3 A Practical Case Study on ROS2

We implemented a practical case study on the Robot Operating System 2 (ROS2) utilizing the open-source NVIDIA IoT AI framework [171]. Specifically, we focused on a realistic robotic application involving a camera input with multiple inference outputs, including lane detection, segmentation, cruise control, and object detection. And we

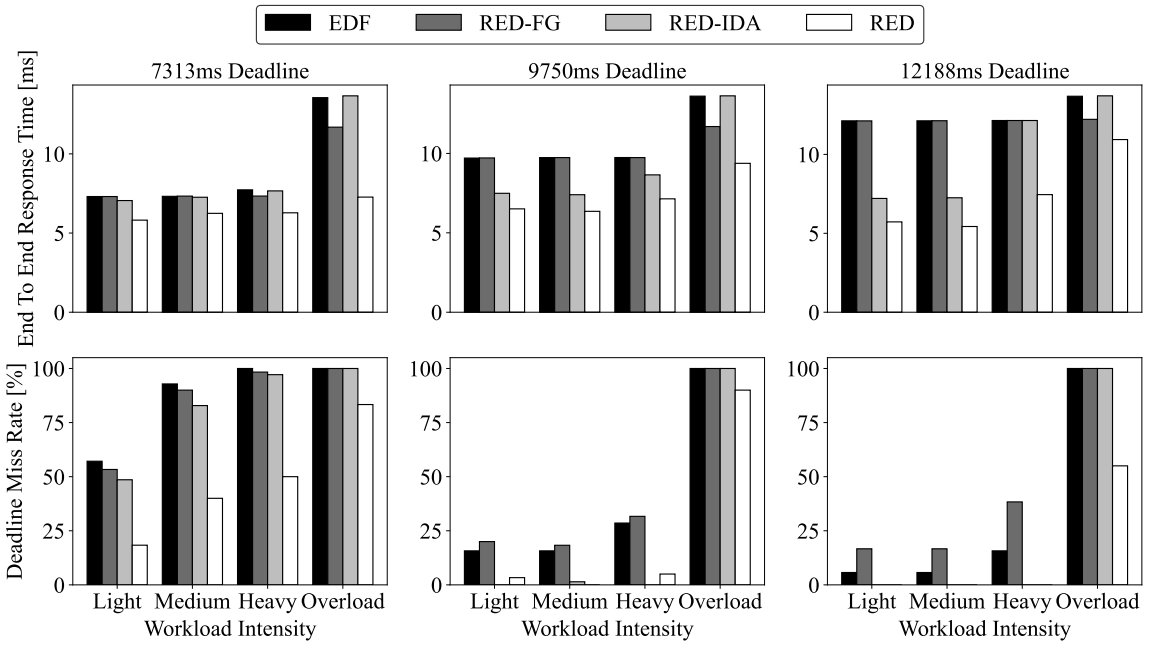


Figure 6.10: A practical case study on ROS2 involving a camera input with multiple inference outputs on Xavier.

comprehensively evaluate the effect of MIMONet working under different GPU interference intensities.

To support MIMONet's inference, we modified the open-source library provided by NVIDIA. As shown in Figure 6.10, unlike traditional single-task DNNs requiring deployment across multiple ROS nodes, we deployed the MIMONet model on a single ROS node. This modification significantly reduced the communication overhead intrinsic to ROS while enhancing the controllability of each module. Additionally, we simulate a camera mounted on a car, abstract as a node in ROS2. The MIMONet model processed this data to produce outputs for several deep learning models, as mentioned before. To conduct a fair comparison, we fully implement EDF, RED, and its ablations built on ROS2, strictly following our design.

As depicted in Fig. 6.10. Quantitative results from this case study demonstrated the superior performance of the MIMONet model deployed on a single ROS node compared

Table 6.4: End-to-end latency of different approaches under variant deadline settings on Xavier. A larger deadline-setting ID indicates a looser deadline. The best values are in bold.

Deadline Setting ID	1	2	3	4	5	6
EDF	5869	6807	6865	7760	8737	9694
RED-FG	5883	6006	6803	7856	8723	9688
RED-IDA	4950	4929	6691	7465	8313	9045
RED	**4455**	**4435**	**4456**	**4500**	**5930**	**6779**

to traditional single-task DNNs deployed on multiple nodes. Compared to the EDF scheduler built upon ROS2, MIMONet provides consistently lower deadline miss rates on average by 67.3% and 32.7% lower response time.

These significant improvements underline the practical usability of MIMONet models in integrating with real-world ROS2 systems. We emphasize that in most interference scenarios, MIMONet integrated with ROS2 consistently has better timing performance than all the ablations and baseline EDF by a large margin, further evidencing the practical usability of MIMONet.

Practical Usability: This case study demonstrates the practical deployment of *Orion* on ROS2-based robotic applications. *Orion* significantly reduces latency and improves deadline compliance, highlighting the usability of *Orion* in complex robotic navigation scenarios.

6.4.4 Adaptability under Different Scenarios

Adaptability to variant deadlines. To investigate the adaptability of the MIMONet system to various end-to-end deadlines, we conducted a parameter study on the NVIDIA Jetson AGX Xavier platform. This study evaluates the system's performance under different deadlines, ranging from tight to loose. In real-world scenarios, end-to-end deadlines can change due to environmental dynamics, and a robust system should adapt to these changes

to meet the required deadlines. As depicted in Table. 6.4, MIMONet outperforms the baseline approaches under all deadline settings, with an average latency less than EDF, MIMONet-IDA, and MIMONet-FG, on average by 23.2%, 20.1%, and 17.1%, respectively. This suggests that MIMONet can adapt to varying end-to-end deadlines by applying a MIMONet-aware intermediate deadline assignment policy integrated with dynamic DAG reconstruction, even in the face of environmental dynamics. Moreover, by incorporating components of MIMONet, the end-to-end response time gradually decreases, significantly outperforming the EDF baseline. This observation further supports the idea that each component of our proposed MIMONet contributes to increasing throughput and fully utilizing system resources to achieve a fast response time.

Adaptability to variant QoE requirements. To assess the versatility of our tool in the context of diverse QoE requirements, we maintained the workload intensity at a constant level while adjusting the hyperparameter λ of the QoE metrics, as defined in Eq. 6.1. This experiment was performed across four unique embedded devices using a stringent tight deadline benchmark as defined in Sec. 6.4.1. As depicted in Table 6.5, our MIMONet consistently outperforms the alternatives across an extensive range of hyperparameter λ values, varying from 0.001 to 10. This result holds true under constant workload conditions, demonstrating the adaptability of our tool to different system settings.

Robust Adaptability: MIMONet demonstrates exceptional adaptability under varied end-to-end deadlines and QoE requirements. It consistently outperforms baselines across multiple platforms and scenarios, showcasing its robustness and efficacy in dynamic environments.

Table 6.5: QoE scores of different approaches under variant hyperparameter settings across four embedded platforms. Larger λ indicates higher QoE requirements. The best values are in bold.

	λ Value	0.001	0.01	0.1	1.0	10.0
Nano	EDF	0.256	0.254	0.238	0.113	0.001
	RED-FG	0.248	0.246	0.230	0.108	0.001
	RED-IDA	0.306	0.304	0.285	0.140	0.002
	RED	**0.461**	**0.460**	**0.437**	**0.240**	**0.004**
TX2	EDF	0.420	0.418	0.399	0.222	0.003
	RED-FG	0.473	0.470	0.448	0.248	0.001
	RED-IDA	0.638	0.636	0.615	0.395	0.001
	RED	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Xavier	EDF	0.748	0.746	0.735	0.616	0.003
	RED-FG	0.525	0.523	0.501	0.295	0.005
	RED-IDA	0.946	0.945	0.942	0.902	0.500
	RED	**0.999**	**0.999**	**0.998**	**0.996**	**0.900**
Orin	EDF	0.973	0.972	0.971	0.954	0.878
	RED-FG	0.953	0.952	0.951	0.931	0.893
	RED-IDA	0.973	0.973	0.971	0.956	0.900
	RED	**0.990**	**0.990**	**0.989**	**0.984**	**0.960**

6.4.5 Overhead Analysis

Memory Overhead. Our method demonstrates exceptional memory efficiency, with minimal overhead, as detailed in Table 6.6. As expected, our highly efficient implementation ensures low memory overhead values for both the scheduler and synchronization components, with the scheduler being particularly modest. However, it is noteworthy that the on-demand synchronization components slightly compromise memory efficiency to enhance latency through reduced context switching between processes. As discussed in Sec. 6.3.4, on-demand synchronization is implemented to decrease code design space coupling, thereby increasing reusability. This approach leads to increased memory overhead due to complex inter-process communication. Future work may consider a potentially more memory-efficient alternative, such as implementing our solution in a multithreading manner, though this may trade off

Table 6.6: Detailed Memory overhead and percentage of applying *Orion* for each minibenchmark in this paper.

	(a) Overhead Raw Value		(b) Overhead Ratio			
	Scheduler	Synchronization	Nano	TX2	Xavier	Orin
Tight	5 KB	40.1 MB	1.00%	0.50%	0.25%	0.13%
Loose	3 KB	36.9 MB	0.92%	0.46%	0.23%	0.12%

Table 6.7: Average runtime overhead of *Orion* are shown as the first row. Profiling overhead is shown as the second row.

	Nano	TX2	Xavier	Orin
Runtime Overhead (ms)	27.3	26.9	17.4	1.2
Profiling Overhead (s)	51.1	34.3	19.1	16.1

implementation complexity and reusability. Notably, the overhead ratios consistently remain low across four different hardware platforms and configurations, evidencing the memory efficiency of our approach.

Execution Overhead. In addition to memory overhead, the execution overhead of our proposed method is also critical. Table 6.7 shows the runtime execution overhead of MIMONet and the profiling overhead for each testing platform. The first line of table showcases the remarkably low runtime execution overhead due to the efficient implementation of the *Orion* framework. The scheduling overhead for NVIDIA Jetson Orin is significantly lower, potentially due to its utilization of a newer Linux kernel version. The main source of execution time overhead arises from scheduling decision-making. One potential strategy to reduce such overheads in future work involves employing finer-grained locks, especially when adapting to more stringent resource-constrained scenarios. Compared to latency data, the runtime overhead remains reasonably low, further underscoring the effectiveness of our approach. The second line of the table examines the profiling overhead of MIMONet. The offline profile overhead is deemed acceptable, even on the Jetson Nano, which displays the largest

absolute overhead value of merely 51.1 seconds (approximately 1 minute). Importantly, for a fixed system configuration, the offline profile requires only a one-time effort, thus easing its integration into continuous integration and deployment of MIMONet.

Low overhead: Our method demonstrates not only impressive memory efficiency but also low execution overhead across various platforms. The efficient implementation of the *Orion* framework, as well as the modest overhead incurred by the scheduler and synchronization components, contribute significantly to these outcomes.

6.4.6 Discussion

Solution Scalability. The scalability of our proposed solution is primarily hindered by the computational constraints of the devices in use. In instances where training the MIMONet from scratch is impractical because of limited computing resources, leveraging pre-trained models can significantly alleviate this issue. These models, often trained on extensive datasets, can serve as efficient encoders for the MIMONet model, thereby greatly reducing the required training time. For instance, state-of-the-art pre-trained visual transformer (ViT) models on the ImageNet dataset [79] which consists of over 14M images can be utilized as shared encoders for navigation robot tasks. Additionally, the low-coupling design of the MIMONet's controller and handler modules offers flexibility, allowing the framework to be conveniently modified for decentralized deployment through the integration of a network communication module.

Future Work. Looking ahead, there are several promising directions for enhancing the functionality and adaptability of our framework. Beyond the weight-sharing approach employed in this work, integrating existing strategies for deploying deep learning models

on embedded devices, such as model compression [74, 214], pruning, and quantization, could further optimize our system. These enhancements could be designed to be both configurable and reusable. On the application front, integrating state-of-the-art structures, such as Transformers-based large language models (LLMs) [386, 65, 212, 63], into MIMONet could potentially enhance accuracy. In terms of device compatibility, the landscape of AI-oriented embedded devices is rapidly evolving, with many now equipped with function-specific computational devices like Field-Programmable Gate Arrays (FPGAs) and Neural Processing Units (NPUs). This trend towards heterogeneous Systems-on-a-Chip (SoCs) suggests another future direction: optimizing MIMONet's strategy to account for the unique characteristics of these heterogeneous computing devices with integrating processor scheduling [354, 220]. This enhancement would expand the range of embedded devices that can deploy MIMONet On-Device.

6.5 Related Work

Real-time DAG scheduler. DAG-based workloads have received much attention in both server systems [8, 312, 362, 43, 283] and embedded systems [367, 31]. Theoretical multi-processor DAG scheduling has been explored dating back to Graham's bound in 1969 [122]. Recently, more advancements have been achieved in real-time DAG scheduling and analysis of parallel workloads [342, 18, 177, 178, 210, 209, 250, 31, 147]. However, to our knowledge, most existing DAG schedulers cannot handle MIMONet and dynamically changing workloads, thus being in-applicable to our problem. A classical real-time DAG

scheduling approach, the intermediate deadline assignment [283, 226], can be leveraged to handle dynamically changing workloads, which we evaluated as one of the baselines.

Real-time ROS scheduler. Several real-time schedulers have been proposed to handle various tasks in the Robot Operating System (ROS) environment in recent years, encompassing dynamic priority scheduling, latency optimization, feedback-based scheduling, multicore awareness, and resource-aware scheduling. [179, 211, 338, 33, 337, 66, 34] Although prior research has significantly advanced ROS-based real-time scheduling, these works mainly target traditional tasks, not accounting for the unique demands of Multiple-Input Multiple-Output Deep Neural Networks (MIMONet). Our work distinctively addresses this gap, proposing an innovative scheduler specifically designed for MIMONet workloads, thereby catering to the unique requirements of real-time DNN execution in a ROS environment.

Real-time DNN Inference. Recent advancements in real-time DNN research have significantly improved the performance of low-latency solutions. Various strategies and frameworks have been developed to optimize the trade-off between performance and accuracy. For instance, dynamic approximation strategies, supervised streaming and scheduling frameworks, and energy-efficient execution approaches have been proposed to boost real-time performance [406, 22]. Also, some research efforts have focused on creating operating systems specifically designed for DNN execution, exploring theoretical schedulability, and developing methods for the multi-DNN inference that maximize the use of available CPU and GPU resources [184, 365]. However, despite these significant strides, to our knowledge, a research gap persists in the domain of real-time systems for Multiple-Input Multiple-Output (MIMO)

DNNs. This paper addresses this gap by formulating and resolving corresponding problems in this sphere.

Weight-Shared DNNs. Weight-sharing techniques enhance the feasibility of deploying Deep Neural Networks (DNNs) on low-memory devices, applicable in both single-task [381, 37] and multi-task setups [157, 155, 208]. In the former, different DNN variants provide varied workload-accuracy trade-offs for the same task, whereas in the latter, parameter-sharing occurs among related tasks, achieved through methods like cross-model quantization or fine-tuning.

Intelligent Robotic Systems Recent intelligent robotic systems are expected to be capable of multi-tasking, corresponding to deep learning models [58, 57, 56, 218, 62]. Several researchers focus on embedded training [378] or inference [293] of deep learning models in robotic scenarios. In both scenarios, the intelligent robotic systems are supposed to be efficient, light-weighted, and mostly required to ensure timing correctness under soft or hard real-time constraints for safety-critical or strongly interactive scenarios such as multi-robot communication [264, 149], UAV trajectory planning [370], geo-localization [394], robotic tracking [38], human-computer interaction [206], autonomous driving [150, 148], etc.

6.6 Conclusion

In this paper, we introduce MIMONet, a comprehensive framework tailored for multi-task DNN inference on resource-restricted robotic systems, designed to adaptively navigate **R**obotic **E**nvironmental **D**ynamics under real-time constraints. Central to MIMONet is a deadline-driven scheduler incorporating an intermediate deadline assignment policy,

adept at managing evolving workloads and asynchronous inference amidst unpredictable environments. Furthermore, MIMONet effectively supports the deployment of MIMONet (multi-input multi-output neural networks), addressing memory limitations and exploiting the unique weight-sharing architecture of MIMONet. Consequently, MIMONet devises an innovative workload refinement and reconstruction process, enabling compatibility with MIMONet and optimizing efficiency. The comprehensive evaluation of MIMONet underscores its effectiveness concerning throughput, timing correctness, practical usability, adaptability, and low overhead. However, it's worth noting that the current design might be best suited for a specific DNN structure (i.e., MIMONet). Our implemented DAG scheduling algorithm, in its present form, is relatively simple and practical. We plan to leverage interesting and relevant ideas from many real-time DAG scheduling approaches proposed recently and further enhance the performance of *Orion* in the near future.

Chapter 7

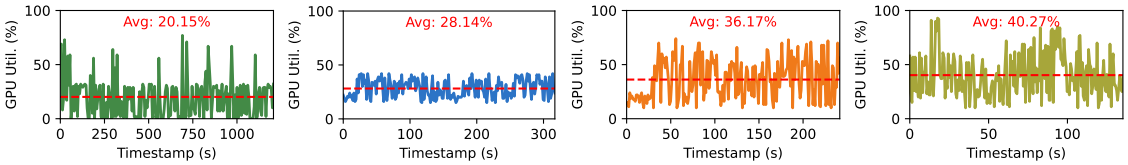
Real-Time Continual Learning on Embedded GPUs

7.1 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 [289], person re-identification [377], and robotic manipulation [207]. 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 [52, 10, 234, 51] 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. 7.1a), or *alternate* them at a coarse fixed cadence, recovering some throughput but still wasting parallelism (Fig. 7.1b). 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. 7.1c-d). Cloud-derived OCL frameworks such as Ekya [28], RECL [188], and AdaInf [324] pursue this idea, but assume multi-GPU server isolation and operate at tens-of-seconds-to-minute granularity, far beyond sub-second embedded inference



(a) Avalanche (b) Default Alternation (c) AOCL$_{basic}$ (d) AOCL

Figure 7.1: **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.

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 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. 7.5, 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. 7.1), per-task inference feasibility and queue backlog-freeness at the controller's operating point so steady-state operation is deadline-safe by construction (Prop. 7.2), and an upper bound on the transient adaptation-window latency during safeguard fallback (Cor. 7.1).

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 [46, 233] framework (2100 GitHub stars) and deployed on Nano and Orin devices widely adopted hardware platform in robotic applications [187, 195, 293, 269, 271, 273]. We will open-source this first 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 deadline miss rate at 13 out of 15 testing scenarios. We further validate AOCL through two case studies: an autonomous driving simulator [160] 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 offline workload profiling or 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 transient deadline-miss bound during its safety fallback, 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× higher training throughput (average 5.15×) over state-of-the-art OCL baselines while sustaining the lowest deadline-miss rate at 13 out of 15 testing scenarios 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.

7.2 Background: Online Continual Learning

Online Continual Learning (OCL) enables models to update incrementally as new data arrives, without retraining from scratch [52, 10, 51, 146]. Unlike static, offline deep learning, where models are trained once on fixed datasets, OCL operates under non-stationary input distributions caused by environment shifts, sensor changes, or evolving user behavior [160]. Such data drift can rapidly degrade performance if the model is not adapted, but incremental updates risk catastrophic forgetting. Replay-based methods such as Experience Replay (ER) [52] mitigate this by revisiting stored samples to preserve prior knowledge. This

continual adaptation is essential in domains like robotics, autonomous vehicles, and smart sensors [262, 347, 133], where environmental conditions change unpredictably, and embedded platforms impose tight memory, compute, and power constraints. Standard offline retraining is impractical in such settings due to high compute cost, memory usage, privacy concerns, and unpredictable execution time. OCL offers a lightweight, On-Device alternative for online incremental learning in tasks such as object recognition [289], person re-identification [377], and manipulation [207].

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

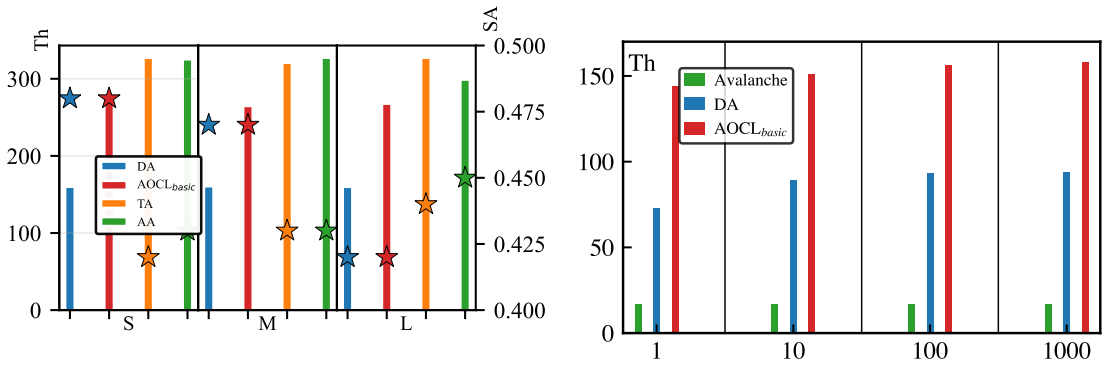
$$SA(T) = \frac{1}{T} \sum_{i=1}^{T} \mathbb{I}(\hat{y}_i = y_i), \tag{7.1}$$

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 [52, 10].

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

$$QPS = \frac{\#\text{ training samples processed}}{\text{elapsed time (s)}} \tag{7.2}$$

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



(a) Alternation Method (b) Alternation Interval

Figure 7.2: Impacts of alternation in OCL systems.

SA by capturing the system's ability to balance learning effectiveness with computational responsiveness under constrained resources.

7.3 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.

7.3.1 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) [52] across three benchmarks of increasing scale: SplitCIFAR10 (small), CoRe50-NC (medium), and CoRe50-NIC (large). We evaluate three execution modes: (1) Sequential OCL (Avalanche [46]), (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.

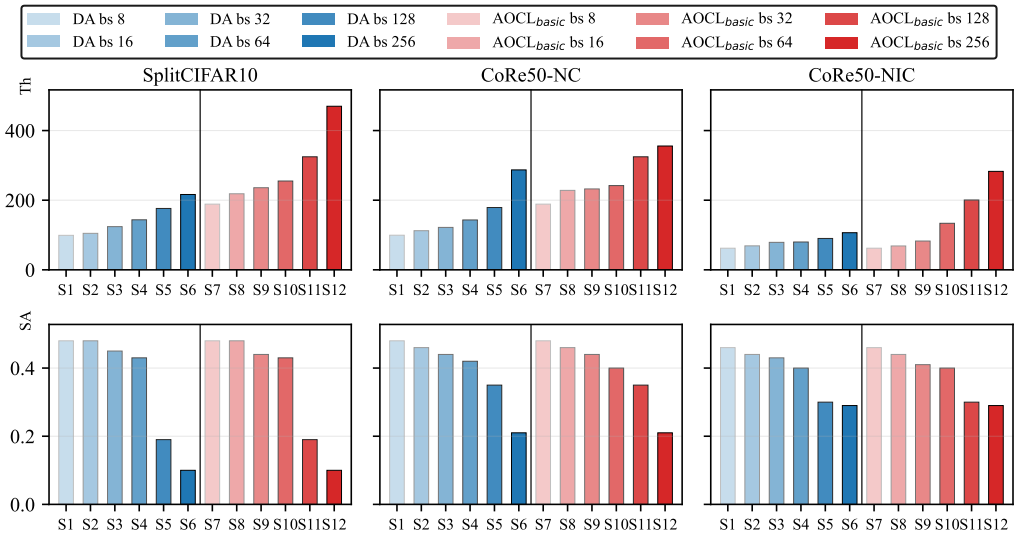


Figure 7.3: Impacts of batch size in OCL systems.

Fig. 7.1 contrasts GPU utilization under four execution modes. The sequential baseline Avalanche shows bursty training spikes with long low-utilization inference phases (mean 29%). Default Alternation interleaves training and inference to reduce blocking, but scheduler overhead suppresses peaks and leaves the mean roughly unchanged (28%). In contrast, AOCL$_{basic}$ overlaps training and inference, filling idle windows and lifting the average utilization to about 36% with smoother traces. The full AOCL adds lightweight, self-adaptive coordination, further raising the mean to about 40% and stabilizing utilization. Higher and steadier GPU utilization at a fixed power budget translates directly into higher throughput and better support for continuous inference on embedded devices. This motivational case study indicates that concurrent execution of the inference and training stages could better utilize GPU resources on embedded devices, thus introducing better parallelism opportunities.

7.3.2 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. 7.2(a)). We compare four alternation policies: DA and $\texttt{AOCL}_{basic}$ are application-agnostic concurrent policies, whereas TA (throughput-oriented alternation) and AA (accuracy-oriented alternation) add application-aware approximations atop $\texttt{AOCL}_{basic}$ to reduce redundant computation and steer the objective. With a fixed 0.1 s alternation interval, $\texttt{AOCL}_{basic}$ 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. 7.2(b)). We sweep four alternation intervals (1, 10, 100, 1000ms) for DA and $\texttt{AOCL}_{basic}$, 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, $\texttt{AOCL}_{basic}$ demonstrates similar patterns. However, based on additional measurements, the optimal of $\texttt{AOCL}_{basic}$ 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. 7.3). We evaluate six batch sizes (8–256) under DA (S1-S6) and $\texttt{AOCL}_{basic}$ (S7-S12). Larger batches reliably lift QPS (e.g., on SplitCIFAR10, $\texttt{AOCL}_{basic}$ 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.

7.3.3 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,

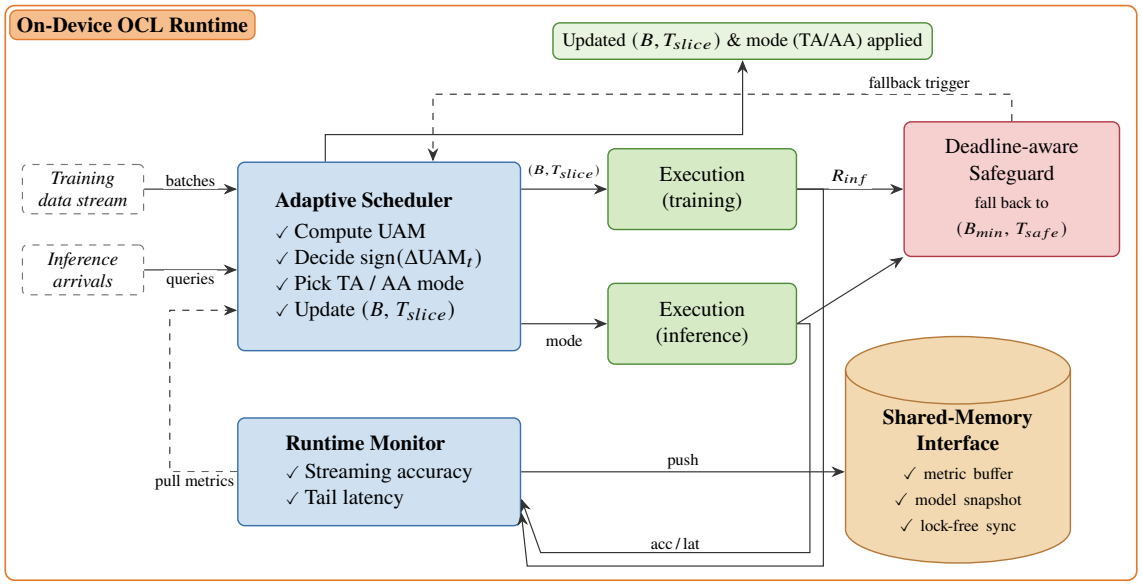


Figure 7.4: High-level design of the AOCL toolchain.

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. 7.5); and *(iii) dynamic workload mix*, input rate, sensor modality, and learning progress all shift online, requiring runtime resource management. The next section introduces lightweight, application-aware scheduling that dynamically balances training and evaluation in response to both model progress and device-level signals.

7.4 Methodology

In this section, we present the design and implementation of a comprehensive On-Device OCL toolchain, specifically developed for efficient deployment on resource-constrained embedded platforms. Our toolchain dynamically orchestrates concurrent training and evaluation processes in response to online streaming accuracy and latency metrics. This ensures robust application-system co-optimization, delivering reliable performance in dynamic and latency-sensitive embedded environments.

7.4.1 System Overview

Fig. 7.4 illustrates the high-level design of our OCL toolchain, which seamlessly integrates two concurrent processes: *training*, responsible for incremental model updates using incoming data streams, and *evaluation*, tasked with continuous monitoring of streaming accuracy. The core of our toolchain is a lightweight, adaptive scheduler that dynamically coordinates these processes based on a unified performance metric. To achieve this, our scheduler employs a shared-memory interface for efficient and low-latency communication between the training and evaluation processes, enabling rapid system adjustments. Leveraging continuously monitored runtime performance metrics, the scheduler dynamically adjusts key parameters such as batch sizes and execution alternation interval. Furthermore, it intelligently alternates between latency-oriented and accuracy-oriented goal, effectively balancing timely model adaptation and sustained inference quality.

7.4.2 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:

$$\text{UAM}_t = \omega \cdot SA_t - (1 - \omega) \cdot \hat{L}_t, \tag{7.3}$$

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.

The incremental change in UAM is computed as:

$$\Delta \text{UAM}_t = \text{UAM}_t - \text{UAM}_{t-1}, \tag{7.4}$$

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.

7.4.3 Design Rationale: Why a Heuristic 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 re-tuning. Hyperparameter sensitivity study in the supplementary files further demonstrates that UAM is not sensitive to hyperparameters.

- ***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. 7.5 by construction and extends to additional knobs (e.g., DVFS, multi-stream placement) by adding axes to the discrete state.

7.4.4 Dynamic Configuration

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

Batch Size Update:

$$B_{new} = \text{clip}\big(\text{round}\big(B_{old} \cdot (1 + \gamma \cdot \text{sign}(\Delta\text{UAM}_t))\big), \\ B_{min},\, B_{max}\big), \tag{7.5}$$

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:

$$T^{new} = \text{clip}\Big(T_q \cdot \text{round}\Big(\tfrac{T^{old} \cdot (1+\eta \cdot \text{sign}(\Delta\text{UAM}_t))}{T_q}\Big), \\ T_{min},\, T_{max}\Big), \tag{7.6}$$

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[1].

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 +

[1]Sensitivity study of ω, γ, η is in supplementary files.

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. (7.6) 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. (7.5) and the binary mode component $\in \{\text{TA}, \text{AA}\}$, the controller state space is finite.

7.4.5 Dynamic Alternation

We build two adaptive alternation policies *on top of* $\texttt{AOCL}_{basic}$: 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

$$\frac{E_{curr}}{E_{total}} \leq \alpha, \tag{7.7}$$

then reverts to the standard $\texttt{AOCL}_{basic}$ 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

$$SA_t \leq \delta_{acc}, \tag{7.8}$$

and switches back to $\texttt{AOCL}_{basic}$ 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}.

7.4.6 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. 7.3 is only influenced by latency.
- Generalize to accuracy-oriented scenarios: Occurs if set $\omega = 1$, where Eq. 7.3 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.

7.4.7 Algorithm Details

Algorithm 7 formally describes our adaptive scheduler. The scheduler initializes parameters and computes the initial UAM value from initial accuracy and latency (line 1) before entering the main control loop (line 2). During each runtime tick, streaming accuracy and latency metrics are collected from shared memory (line 3). The deadline-aware safeguard (lines 4–8) then checks whether the worst-case inference response time $R_{\text{inf}}(B)$ exceeds the deadline D_{inf} or whether the measured tail latency exceeds $L_{\text{tail}}^{\max}$; if so, the scheduler immediately falls back to the safe configuration $(B_{\min}, T_{\text{safe}})$, switches to AA, and skips the rest of the iteration to avoid further deadline violations. Otherwise, the scheduler computes the current UAM (line 9) and its incremental change ΔUAM_t (line 10). If performance improves ($\Delta\text{UAM}_t > 0$), the batch size and alternation interval are increased and the scheduler switches to TA (lines 11–14); otherwise, both configurations decrease and AA is selected (lines 15–19). Updated configurations are asynchronously propagated to the training

Algorithm 7 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. (7.3)
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. (7.3)
10: Compute incremental ΔUAM_t using Eq. (7.4)
11: **if** $\Delta\text{UAM}_t > 0$ **then**
12: Updated B via Eq. (7.5) ▷ Increase batch size
13: Updated T via Eq. (7.6) ▷ Increase alternation interval
14: Switch to Throughput-oriented Alternation (TA) based on Eq. (7.7)
15: **else**
16: Updated B via Eq. (7.5) ▷ Decrease batch size
17: Updated T via Eq. (7.6) ▷ Decrease alternation interval
18: Switch to Accuracy-oriented Alternation (AA) based on Eq. (7.8)
19: **end if**
20: Asynchronously propagate updates to training and evaluation processes
21: **end while**

and evaluation processes, maintaining minimal overhead and continuous operation (line 20). The loop (line 21) repeats, continuously adapting configurations to optimize performance.

7.5 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.

7.5.1 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 sign(ΔUAM) with strictly bounded outputs and under a finite state space.

7.5.2 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. 7.4.4. 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 sign(ΔUAM_t)) interval of at most $|\mathcal{S}|$ ticks, and the bang-bang controller with clipping is input-to-state stable (ISS) with respect to bounded ΔUAM perturbations in the standard sense of [331, 181]: the state trajectory remains in the compact set $\mathcal{S}$ for any bounded

disturbance sequence. Both statements follow directly from the system construction; the substantive per-axis one-sided sweep-time claim is the content of Prop. 7.1 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 7.1 (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*

$$K_{transient} \leq |\mathcal{S}| = N_B \cdot N_T \cdot 2 \tag{7.9}$$

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. 7), the number of ticks the controller needs to drive* B *(resp.* T*) from one boundary to the opposite boundary is bounded by*

$$K_B^{transient} \leq \left\lceil \log_{1+\gamma}\left(B_{max}/B_{min}\right) \right\rceil + 1 \quad \textit{ticks sweep time}, \tag{7.10}$$

$$K_T^{transient} \leq \left\lceil \log_{1+\eta}\left(T_{max}/T_{min}\right) \right\rceil + 1 \quad \textit{ticks sweep time}, \tag{7.11}$$

Eqs. (7.10)–(7.11) 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. 7.9 follows from the same pigeonhole argument as the bounded-state-space remark (Sec. 7.5.2): among the first $|\mathcal{S}|+1$ states the trajectory must repeat. For Eq. (7.10), under sustained $\mathrm{sign}(\Delta\mathrm{UAM}_t) = +1$ from $B^{(0)} = B_{min}$, Eq. (7.5) yields $B^{(k+1)} \geq \mathrm{round}\big((1+\gamma)B^{(k)}\big) \geq (1+\gamma)B^{(k)} - \frac{1}{2}$ before clipping. Iterating gives $B^{(k)} \geq B_{min}(1+\gamma)^k - \frac{1}{2}\sum_{i=0}^{k-1}(1+\gamma)^i$, so $B^{(k)} \geq B_{max}$ 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. (7.10); 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. (7.11) follows by an identical argument for T with η in place of γ and T_q as the rounding granularity instead of 1, yielding $K_T^{transient} \leq \lceil \log_{1+\eta}(T_{max}/T_{min}) \rceil + 1$. □

7.5.3 Feasibility and Backlog-Freeness for Continuous Inference

Recall from above that 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 $P_{inf} = 33.3\,\mathrm{ms}$ 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:

$$R_{inf} = C_{inf}(B) + C_{block}(B) \tag{7.12}$$

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:

$$C_{inf}(B_{min}) + C_{block}(B_{min}) \leq D_{inf} \tag{7.13}$$

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. 7.5 and 7.6), we state utilization as a function of both:

$$U(B, T) := \frac{C_{inf}(B)}{P_{inf}} + \frac{C_{train}(B)}{T} \leq 1. \tag{7.14}$$

Eq. 7.14 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

$$T_{\text{feas}}(B) \;=\; \frac{C_{train}(B)}{1 - C_{inf}(B)/P_{inf}}\,, \tag{7.15}$$

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

Proposition 7.2 (Boundary Feasibility and Backlog-Freeness)**.** *Under the task model of Sec. 7.5, the following two results hold:*

(a) ***(Boundary feasibility.)*** *If Eq. (7.13) holds, then at the boundary configuration $B = B_{min}$ the worst-case response time of τ_{inf} satisfies $R_{inf}(B_{min}) \le D_{inf}$. 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. (7.14) 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. (7.13) gives $R_{inf}(B_{min}) = C_{inf}(B_{min}) + C_{block}(B_{min}) \le D_{inf}$, so τ_{inf} meets its deadline at the smallest reachable batch B_{min}. At larger B, $R_{inf}(B)$ increases monotonically in B (Sec. 7.5.1) and may exceed D_{inf}; Eq. (7.13) 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. (7.13) guarantees the destination B_{min} satisfies the deadline. (b) Eq. (7.14) 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 [260] or introspective per-job placement [366], and with microsecond-scale preemptive GPU schedulers such as REEF [135]. We will leave multi-GPU extension as future work. While Prop. 7.2 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. 7.1 supplies this bound by collapsing the per-window WCET sum to its boundary value at $B = B_{max}$ under monotonicity in B. The details are in the supplementary file.

Corollary 7.1 (Adaptation Window Latency Bound)**.** *For any $B \in [B_{min}, B_{max}]$, assuming $C_{inf}(B)$, $C_{train}(B)$, $C_{block}(B)$ are non-decreasing in B, the per-window latency bound attains its maximum at $B = B_{max}$ for every reachable controller configuration. This follows immediately under H1 the wall time is bounded by the per-task WCET sum, and monotonicity in B makes each summand maximal at $B = B_{max}$.*

7.6 Evaluation

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

Table 7.1: 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 [46]	○	○	○
DA	●	○	○
Ekya [28]	●	◐	◐
RECL [188]	●	◐	◐
AOCL$_{basic}$	●	◐	●
AOCL	●	●	●

7.6.1 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. 7.1.

- **Avalanche [46]:** 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 [28]:** originally for offline continual learning on multi-GPU servers; adapted to the embedded OCL setting.

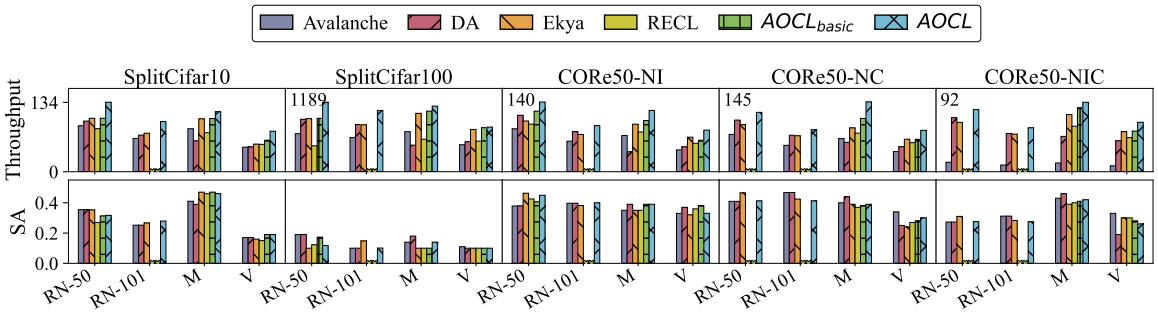


Figure 7.5: Overall effectiveness of AOCL on different deep learning models on the ER algorithm across benchmarks. The top row reports training throughput (QPS); the bottom row reports streaming accuracy (SA). Each cell shows four backbones: RN-50 (ResNet-50), RN-101 (ResNet-101), M (MobileNetV1), and V (ViT-Tiny). Crosses (×) at $y = 0$ mark cells where the baseline failed due to out-of-memory (OOM) errors.

- **RECL [188]:** originally for autonomous-driving and video-analytics continual learning on multi-GPU servers; adapted to the embedded OCL setting.

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. 7.1). AOCL support all three features natively (●) via fine-grained slicing, deadline tracking, and contention-aware scheduling.

7.6.2 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. 7.6 reports ResNet-20 training throughput across five benchmarks. On Nano (Orin), AOCL delivers 4.57× (5.72×) average throughput over Avalanche, peaking at 7.18× (11.61×) 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;

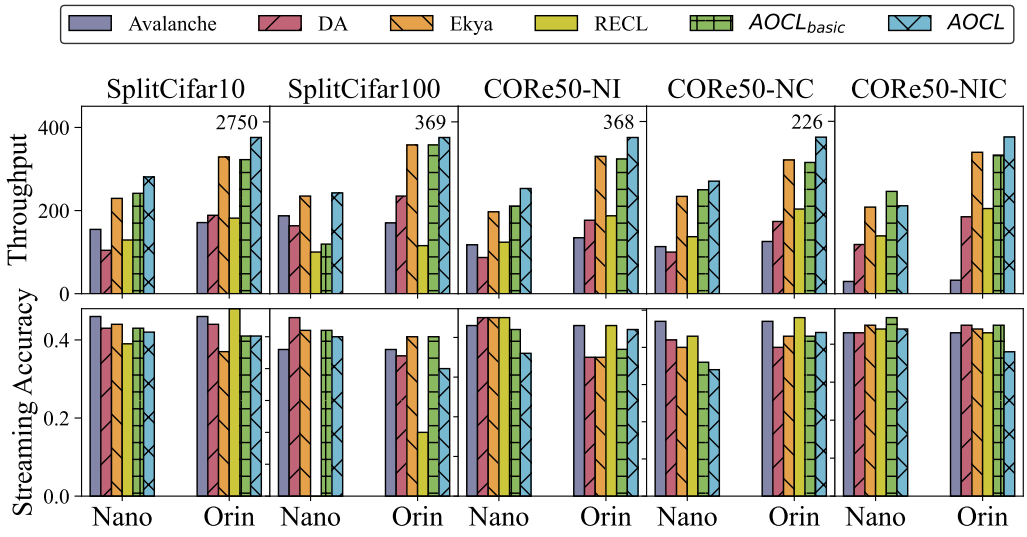


Figure 7.6: Overall effectiveness of AOCL using the ER algorithm evaluated on five benchmarks on ResNet-20 model.

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 7.7 reports throughput across ER, GSS, GEM, and AGEM on CORe50-NC. On Nano with ResNet-20, AOCL delivers 1.80× on GEM and 2.29× on AGEM over Avalanche, with 1.76× (GSS) and 1.58× (GEM) over the static AOCL$_{basic}$. On Orin, it achieves 1.65× on GSS and 3.80× 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 (as shown in the

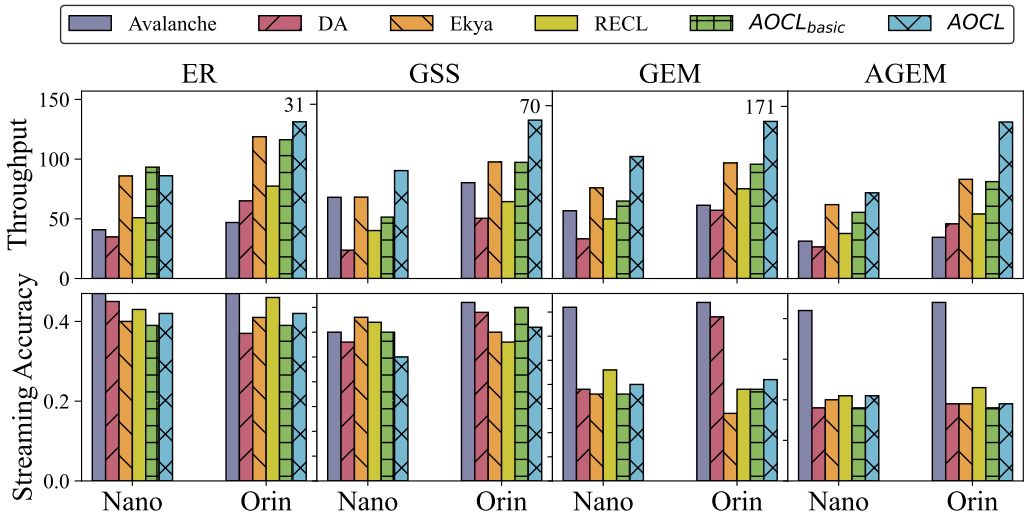


Figure 7.7: Overall effectiveness of AOCL on four OCL algorithms on the CORe50-NC benchmark.

supplementary file), roughly 31× and 18× the ER cost, nearly 19× 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 7.5. On ResNet-50, ResNet-101, MobileNetV1, and ViT-Tiny, AOCL achieves 1.21×, 1.21×, 1.16×, and 1.20× 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. 7.13) 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

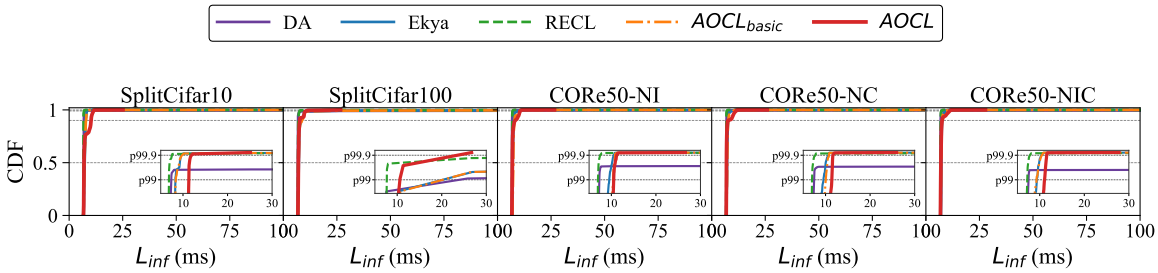


Figure 7.8: Per-frame inference-latency CDFs on Jetson Orin under sustained ER training with ResNet-20 on five benchmarks. The x-axis is capped at 100 ms to expose method separation in the typical-frame regime; horizontal dashed lines mark the $p50/p90/p99/p99.9$ percentile reference levels. The lower-right 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.

B_{max}=16 (ResNet-101) imposed to honor Prop. 7.2. 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.

7.6.3 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 (same five benchmarks and settings as Sec. 7.6.2; Avalanche is omitted as it never co-runs training and inference).

Tab. 7.2 reports deadline-miss rates (DMR) at three application-relevant tiers: 16/33 ms match 60/30 FPS sensor cadences, and 100 ms is the sub-second envelope of larger inference pipelines [272]. DA's sequential alternation forces inference to wait for an entire

training step, yielding 0.50–1.43% DMR across all benchmarks/tiers and a > 1 s CDF tail in Fig. 7.8. Among the concurrent methods, AOCL attains the strictly lowest DMR at 33 and 100 ms on every benchmark, stays within 0.04% of Ekya/$\texttt{AOCL}_{basic}$ at 16 ms on small benchmarks, and keeps overall DMR $< 0.5\%$, empirically validating Prop. 7.2: the controller's active retreat of B when $R_{inf}(B) > D_{inf}$ holds inference within the deadline while preserving the throughput gains of Sec. 7.6.2.

The L_{inf} CDFs in Fig. 7.8 corroborate this. DA's $p99.9$ exceeds 845 ms, appearing as a flat segment near CDF $= 0.99$. RECL is tightest at $p99$ (≤ 7.66 ms) but at the cost of substantial training throughput (Sec. 7.6.2); AOCL stays competitive (10.5–11.4 ms, below the 16 ms line). Above $p99.9$, AOCL keeps the tightest tail (11.2–23.5 ms) while other baselines reach 100 ms, explaining its dominance at the 33 and 100 ms DMR tiers.

7.6.4 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.7.9b) and physical environments (Fig.7.9a). 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 [160], 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. 7.3a shows AOCL reaches up to

Table 7.2: 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**%

Table 7.3: 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.

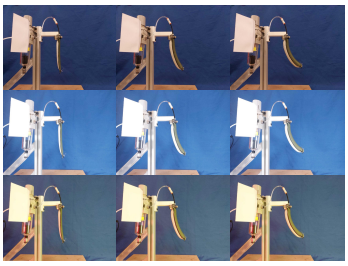
Baseline	Training Throughput [QPS] ↑						Streaming Accuracy ↑					
	ILC			SS			ILC			SS		
	IC	IL	WC	IC	IL	WC	IC	IL	WC	IC	IL	WC
Avalanche	2.13	0.51	0.49	0.52	0.35	0.34	**0.97**	**0.99**	**0.99**	**0.97**	**0.94**	**0.94**
DA	3.73	1.01	0.97	0.77	0.59	0.46	0.48	0.99	0.91	0.96	0.83	0.86
Ekya	6.30	1.73	1.72	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

1.80 QPS on ILC and 1.09 QPS on the heavier SS (both in IL), corresponding to 1.36–3.12× speed-ups over the sequential Avalanche oracle with absolute accuracy drops ≤ 0.02 (≥ 0.97 in most ILC cells; best or second-best on SS).



(a) OCL-driven soft robotic system prototype under dynamic actuator pressure and lighting conditions.

(b) High-fidelity navigation simulation with dynamic environmental factors (lighting, weather) for On-Device OCL.

Soft Robotic Case Study. We further evaluate `AOCL` on a Jetson Orin soft-robot prototype (Fig. 7.9a) under two OCL scenarios: IC (classifying varying actuator pressure levels) and IL (illumination changes on the vision stream). In Tab. 7.3b, 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 ≈2.3 pp of the sequential baseline in streaming accuracy.

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.

Table 7.4: 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

Table 7.5: 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

7.6.5 Ablation Study

We systematically evaluate each component of AOCL using ER across five benchmarks (Tab. 7.4). AOCL$_{\text{basic}}$ achieves moderate throughput at moderate accuracy. Adding TA pushes throughput at modest accuracy cost (e.g., up to 2500.00 QPS on SplitCIFAR100); AA maximizes accuracy on three of five benchmarks at lower throughput (e.g., 0.45 on SplitCIFAR10 and CORe50-NIC, 181.40 QPS on CORe50-NIC). The full adaptive AOCL jointly attains the best throughput (e.g., 375.94 vs. 326.80 QPS on SplitCIFAR10 over TA) while keeping accuracy within 0.03 of the best per benchmark. The adaptive controller, therefore, not only aggregates TA/AA benefits but also smooths their trade-offs across OCL paradigms.

7.6.6 System Overhead Analysis

We profile the per-component runtime, memory, and energy footprint of `AOCL` on Jetson Orin across five benchmarks. Tab. 7.5 shows two patterns. (i) Per-step cost is dominated by D-Buffer (model snapshot and pointer swap), with UAM and the scheduler contributing ≤ 20 ms on every benchmark except CORe50-NIC; the runtime scales with the number of adaptation windows rather than model size, so the four steady benchmarks stay within 156–215 ms while CORe50-NIC, with the most reconfigurations, reaches UAM/Scheduler/D-Buffer of 144/16/1,540 ms. (ii) Energy and memory remain tight: 32.7–55.5 J board energy and a flat 1.68–1.72 MB peak footprint regardless of benchmark, since the runtime keeps only a fixed-size shadow tensor and a small UAM ring buffer. Overall, the runtime layer adds ≤ 1.7 s of cumulative cost per stream and consumes well under 0.1% of the device's 5–15 W active envelope.

7.7 Related Work

Online continual learning (OCL) provides opportunities to update models from streams while mitigating catastrophic forgetting [52, 294, 234]. Algorithmic approaches include rehearsal [52, 301], gradient constraints [234, 51, 10], and regularization [194, 222]. Surveys consolidate these streaming algorithms [347, 348, 48]. 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 [46] for diverse OCL algorithms.

Despite extensive algorithmic advancements, system-level optimizations specifically designed for OCL remain underexplored. Ekya [28] optimizes offline continual learning methods such as iCaRL [301] for video analytics tasks on powerful multi-GPU servers requiring offline profiling. RECL [188] focuses on continual learning video analytics and autonomous driving scenarios on multi-GPU server scenarios, requiring complex model selections. AdaInf [324] alters model structures for multi-model continual learning on multi-GPU servers. LifeLearner [202] targets optimization on hardware-level meta-continual learning. Their multi-GPU, offline-centric designs significantly differ from our strict embedded scenarios. `AOCL` is the first concurrent OCL runtime specifically tailored for embedded platforms, bridging algorithmic efficiency with resource-constrained system-level practicality.

Real-time GPU scheduling has been widely explored [99, 363, 184, 355]. 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.

7.8 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.

Part III

Deployable On-Device Learning Ecosystems in Uncertain, Resource-Constrained Environments

Chapter 8

Genie: Smart ROS-based Caching for Connected Autonomous Robots

8.1 Introduction

Autonomous robots are rapidly expanding from a research-oriented subject to real-world applications, with autonomous vehicles being a particular and promising representation [329, 330, 142, 130]. Companies such as Waymo already started deploying fully autonomous taxi fleets in a limited number of areas [142, 130]. However, several outstanding questions remain when it comes to accuracy [216], latency [215, 216], timing-predictability [215, 216], and energy efficiency [405].

The problem addressed in this paper is that of latency given the Size, Weight, and Power (SWaP) constraints of the computing platform used in autonomous vehicles. Specifically, existing low-power autonomous embedded platforms such as the latest NVIDIA Jetson

Xavier cannot execute crucial intelligence tasks such as object detection and localization in a real-time fashion. To remedy this, some existing work suggests that various server clusters can be strategically located close to the autonomous vehicle to aid in computational tasks [246].

In collaboration with Fujitsu, a leader in edge computing and networking, this work addresses the pivotal challenges in integrating edge computing with autonomous vehicle technologies. The primary obstacle is the necessity for edge computing infrastructure, particularly remote server clusters, to be in close proximity to the vehicles to mitigate communication costs, rendering the use of conventional, energy-intensive data center equipment both impractical and costly. Consequently, there's a compelling need for affordable, low-power alternatives.

Key Insight: Utilizing edge servers with locality-aware caching can reduce latency significantly if results can be reused effectively, making it feasible to deliver enhanced performance and provide additional, valuable environmental information to autonomous vehicles.

Nonetheless, existing solutions (reviewed in detail in Sec. 8.5) lack three crucial design features that are required for practical deployment of a locality-aware caching method: (1) incompatibility with the Robot Operating System (ROS), (2) reliance on parameter servers, or a central cache, and (3) lack of specific usability for autonomous vehicles.

Contributions: To address these issues, we introduce Genie, a distributed ROS-based interface for object-oriented caching and data sharing, constructing a 3D object map of the edge server's surroundings: (1) ROS-based Transparent Caching: Genie intercepts ROS communications for caching without modifying software. (2) Distributed Cache Construction:

An algorithm allows Genies to communicate, enhancing their local caches. (3) 3D Object Map: Identifies and caches useful real-world objects for autonomous vehicles, reducing redundant computation and providing extra information.

Implementation and Evaluation: We implemented Genie on a vehicular edge computing platform with Jetson TX2s and AGX Xaviers, using Autoware [187] as a representative ROS-based system. Our evaluation shows that Genie reduces latency by 82% on average, with a peak improvement of 95%. Object reusability is effective, with 31% reusable objects on average and up to 67%. A confidence score demonstrates improved data quality through multi-car information gathering.

8.2 Background and Motivation

Autonomous Vehicles (AVs) rely on a multi-stage computational process involving sensors like cameras and LiDAR to ensure safe driving decisions [120]. This pipeline, which includes perception, localization, detection, prediction, planning, and control, must meet stringent timing constraints to maintain safety. However, state-of-the-art deep learning models impose heavy computational demands, and even advanced edge devices struggle to keep pace, forcing trade-offs in data and algorithm selection.

Our evaluation of six cutting-edge object detection models [303, 45] across various GPU-enabled devices (Nano, AGX, Orin) and an edge server with an A4500 GPU shows significant speedups. For instance, YOLOv8s and DETR-ResNet-101 achieve speedups of 5.02x and 10.37x on the A4500 compared to the Nano, illustrating that advanced GPU-equipped edge servers can substantially enhance computational efficiency by reusing results.

Table 8.1: Runtime/speedup of autonomous driving models on different devices (Nano, AGX, Orin) and edge server GPU (A4500). OOM indicates an out-of-memory error. The baseline for speedup is the slowest successful execution among devices.

Model	Nano	AGX	Orin	A4500
YOLOv8s	27.60 / 1.00x	21.20 / 1.30x	13.73 / 2.01x	5.50 / 5.02x
YOLOv8m	60.90 / 1.00x	48.45 / 1.26x	17.50 / 3.48x	7.10 / 2.46x
YOLOv8l	92.00 / 1.00x	85.50 / 1.08x	26.20 / 3.51x	9.70 / 2.70x
DETR-ResNet-50	307.28 / 1.00x	230.39 / 1.33x	112.26 / 2.74x	30.91 / 9.94x
DETR-ResNet-101	422.51 / 1.00x	336.96 / 1.25x	145.92 / 2.90x	40.73 / 10.37x
DETR-ResNet-101-DC5	OOM / N/A	747.30 / 1.00x	316.52 / 2.36x	86.13 / 8.68x

Note that for some small models like YOLOv8s, even all evaluated devices provides acceptable latency, but this model has the lowest accuracy.

The advent of edge servers presents a significant opportunity to augment computational efficiency. By offloading tasks to edge servers, AVs can speed up data processing and avoid system failures, such as out-of-memory errors observed with DETR-ResNet-101-DC5 on less capable hardware.

This motivates the incorporation of edge servers or powerful embedded devices as distributed caches to help AVs meet stringent timing constraints, enhance safety-critical processes, and improve overall performance.

8.3 Design

8.3.1 Design Considerations

Design Goals. First and foremost, we would like to provide a design that can improve latency substantially when using low-power edge servers. In the process of our design, we would like to create an architecture that can improve decision-making accuracy by providing

smarter, as well as more reliable, information to autonomous vehicles to offer an incentive to connect to edge services that are going to be imminent. To make our design practical, we design and implement everything around the ROS framework [296], which is the most prominent framework used to implement autonomous vehicles. However, similar techniques can also be applied to other frameworks as long as they follow a modular peer-to-peer communication approach. Finally, we note that in all of our design decisions, the features that are added will always remain optional since the car will carry all the necessary computing modules for full autonomy.

Proposed Edge Benefits. Fig. 8.1 shows a scenario where a car is connected to the edge. As is evident in the figure, the necessary setup for autonomous vehicles is for the vehicle itself to have every service required for full autonomy (depicted as Node X to Node D). This is a requirement because the connection to the edge-based server could be cut off at any given moment. In the scenario of Fig. 8.1, the network provider has created a duplicate of a service (Node Y) that already exists on the car. The red arrows in the figure indicate that every message coming out of Node X to Node Y is duplicated both on the edge and on the car. This is done by replicating all the subscribed ($T_S(Y)$) and published ($T_P(Y)$) topics of Node Y to the edge. The results are then fed into Node Z. In this configuration, it is possible for Node Z to receive duplicated messages, and has to identify and discard one of the duplicates (since the services are identical). This is the most common scenario where the edge can be useful. For example, Autoware relies on object detection for tracking and sign detection functionality. However, slow object detection is still serviceable without a remote connection. The value of the edge here is to run faster object detection and thus,

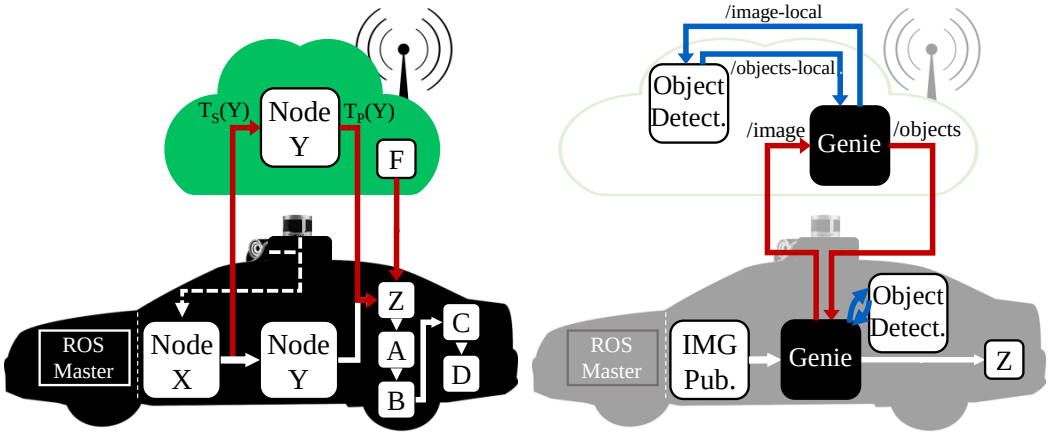


Figure 8.1: Left: an autonomous car connected to the edge. Right: an example of applying Genie in an autonomous vehicle.

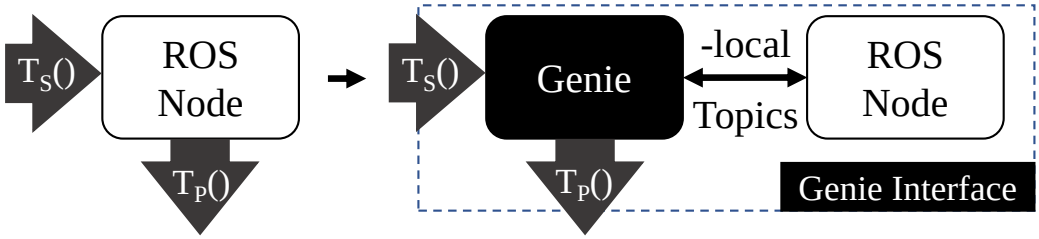


Figure 8.2: Overview of the Genie.

increase the overall decision-making accuracy of Autoware. The other potentially useful scenario is for the edge to have an extra service that would improve accuracy when available, depicted as Node F in Fig. 8.1. For example, edge devices can offer a vision assistant for the car (as we shall discuss in Sec. 8.4.6). In both scenarios, the connection to the edge shall remain optional.

8.3.2 Genie ROS Node for Non-Intrusive Caching

With the previously mentioned design goals in mind, the master node inside the vehicle, together with all the service nodes located on the car and the nodes offered by the edge, creates a virtual private network to facilitate peer-to-peer communication between all nodes (which is typical in ROS). In this section, we would like to design a system that can

enable efficient caching between these inter-network ROS nodes (we discuss caching across different virtual private networks of ROS in Sec. 8.3.3).

The main problem to address here is that of building and storing the cache. This problem is not as intuitive as it seems. Imagine Autoware, where ROS nodes are already implemented and deployed. Even though Autoware is open-source, other ROS-based autonomous driving software may not be. Thus, we would like to store the cache in a non-intrusive way without modifying the source code of Autoware's ROS nodes. For that, we present the concept of Genie, an encapsulation made for ROS nodes, presented in the right of Fig. 8.1. As is evident in the figure, a ROS node is encapsulated in a Genie interface conceptually. This is done by attaching an additional generic Genie ROS node developed by us to each ROS node except the master. This attachment procedure is done by automatically detecting the ROS topics subscribed to and published by that node (depicted as $T_S()$ and $T_P()$ in Fig. **??**), switching them to topics affixed with `-local` so that they would only send and receive messages to the Genie node, and exposing the original topics by the Genie node itself. To other ROS nodes, this new Genie will act like the encapsulated node. However, before passing on any messages from other ROS nodes, the Genie node can check the local cache. Fig. 8.1 shows this design applied to the Object Detection service of an autonomous vehicle.

8.3.3 Distributed Collective Cache

Fig. 8.3 shows a scenario where two cars are connected to the edge. The existence of the ROS virtual network, depicted as VN1 (Virtual Network 1) and VN2, means the "experiences" of the cars would be local to the nodes that are in their respective virtual

network. If any cache exists on the edge or on the car, it would only be aware of the information that has been seen by the car itself. For example, with a simple application of Genie, if a vehicle were to be looping around a city block, it could have reduced computation the next time it arrives at the same spot because much of the environment has been seen before. This can be particularly useful for daily commutes, which is the most common use-case for vehicles [261]. Nonetheless, this situation is limited.

What is more desirable, however, is for the vehicle to receive additional information from other vehicles (i.e., use their experiences). Since each car has its virtual private network by design, this would be an inherently distributed system. Before exploring this type of collective cache construction from distributed entities, we first present a unified interface that is recognized by all members of the system (i.e., topics that all Genie ROS nodes are aware of).

Cache Sharing Interface. As part of our design goal, we would like to expand the Genie ROS node so that it is capable of communicating with other Genie nodes in a distributed fashion. To achieve that, each Genie will expose an additional set of topics with a `-remote` label affixed for the topics detected on the encapsulated node. To clarify, imagine the scenario of Fig. 8.3. As we discussed in Sec. 8.3.2, the existing "/image" and "/objects" topics will be changed to ‘‘`/image-local`’’ and ‘‘`/objects-local`’’ and the original topics will be taken over by the Genie node. To inform other Genies that such services exist, the Genie node also publishes messages selectively on the ‘‘`/image-remote"` and ‘‘`objects-remote"` topics. Likewise, the Genie node will subscribe to the same set of topics to receive information from other remote Genie nodes. The simplicity of these interfaces

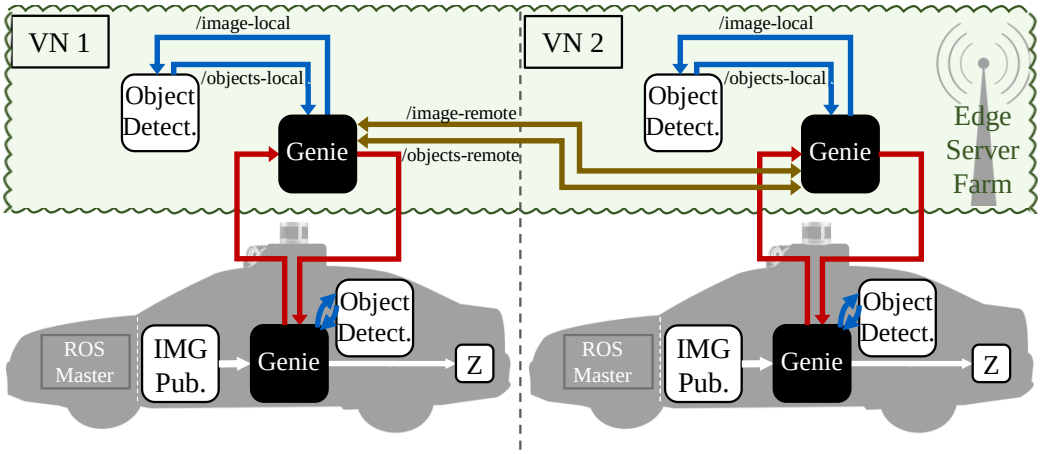


Figure 8.3: An example of applying Genie to object detection across autonomous vehicles.

means that if a Genie has encapsulated an object detection node, it can understand and communicate with other Genies that are also object detection nodes, or understand nodes that work with a subset of the same topics. This design would make sharing functionalities across the network less sophisticated.

Collective Cache Construction. Alg. 8 depicts how the cache is populated in detail, using the aforementioned interfaces. Whenever a message arrives on a topic, it is checked against the database (line 12). If the entry does not exist, the message is forwarded both on the `-local` version (line 13) and the `-remote` of that topic. If the entry exists in the database (line 16), the stored results are retrieved (line 17) and directly returned to the sender (line 19). Sending messages on the `-remote` topics constitutes broadcasting the message on the edge network if the Genie is on the edge, where other Genies with the same functionality have subscribed to the same set of topics (whereas sending message on `-remote` for local car-based Genies uploads the workload to the remote Genie). Upon receiving a message from neighboring Genies, the same procedure of cache miss (line 12) and cache hit (line 16) is followed. If the Genie has an entry for the request, the results will be sent back

Algorithm 8 Genie ROS Node Distributed Cache Procedure

Require: $M < header, data, T >$ ▷ The message M with a header, data, topic.
Require: $T < name, type >$ ▷ Topic T with a name and a type.
Require: H_{DB} ▷ Database for all the hashmaps.
Require: $PUBLISH(M, T)$ ▷ Publishes a message M on the topic T.
1: **function** BUILDHASHMAP(T)
2: hashmap$_T$ = ⟨type_of(T), {}⟩
3: H_{DB}.add(hashmap$_T$)
4: **end function**
5: **function** ONMESSAGEARRIVAL(M, T)
6: **if** $T \notin H_{DB}$ **then**
7: BUILDHASHMAP(T) ▷ Create a new hashmap for never-seen topics.
8: **end if**
9: cacheEntry = LookUpByHeader(hashmap$_T$, M.header)
10: **if** cacheEntry ≠ null **then** ▷ Message is an answer to our previous query.
11: ENHANCEDCACHENEWDATA(M) ▷ See Algorithm 9
12: cacheEntry.second = M.data ▷ Store it in the hashmap as value.
13: **return**
14: **else if** LOOKUPLOCALCACHE(hashmap$_T$, M = null) **then** ▷ **Cache miss.**
15: PUBLISH(M, T + "-local") ▷ Share with the encapsulated node.
16: PUBLISH(M, T + "-remote") ▷ Share with remote Genie.
17: hashmap$_T$.add(M, {}) ▷ Add message as key with empty value.
18: **else**
19: cacheEntry = LOOKUPLOCALCACHE(hashmap$_T$, M) ▷ **Cache hit.**
20: ENHANCEDCACHEBOOSTDATA(cacheEntry) ▷ See Algorithm 9
21: PUBLISH(cacheEntry.second, cacheEntry.second.T)
22: **end if**
23: **end function**

to the sender directly (line 19). We shall explain line 9 and line 18 in our enhanced cache design, discussed in Sec. 8.3.4.

8.3.4 Driving-specific Caching

For autonomous vehicles, using an object map (i.e., a map where individual records are of specific detected objects such as a traffic light) instead of a generic message-based ROS cache can have several benefits. In this section, we detail a design that is specific to object caching.

Smart Cache Specific to Autonomous Driving. To facilitate our design, we have identified key data structures required in autonomous driving, which include raw images, LIDAR point clouds, and 3-dimensional objects. Specifically in Autoware, objects include

every information needed to enable the computation-heavy perception module, eventually leading to autonomous driving decisions. To reiterate, storing objects instead of raw data such as images has several benefits: First, the objects are already-processed information, and contain more useful data whereas raw data must be processed first. Second, objects have substantially smaller storage and communication overhead because they are orders of magnitude smaller than raw data. Third, an object can be more easily translated into 3-dimensional space. As an example, for two cars that are moving in opposite directions, raw image data cannot be useful because images cannot be easily rotated and translated on their z-axis. Meanwhile, the three-dimensional coordinates of objects can be translated in any direction (location translation is one of the main techniques used in autonomous vehicles already [105]). For example, a traffic light with location $< x_1, y_1, z_1 >$ relative to car 1 can be translated to $< x_1', y_1', z_1' >$ for car 2 based on their ground locations. Thus, we use objects exclusively to build our driving-specific cache in addition to the message-based caching proposed in Sec. 8.3.3.

Location-based High Confidence Object Map. As keen readers have noticed, lines 9 and 18 in the previous Alg. 8 call upon an enhanced cache before storing and retrieving data. Alg. 9 depicts the implementation for those two aforementioned functions. First and foremost, upon data arrival, the function `EnhancedCacheNewData` is called. Each object in the received objects array is checked against the $OBJECT\_MAP$ database of type $< location, objects >$ in line 4. We use absolute object location as the key to the object map since absolute location is the only identifying characteristic of an object that multiple cars can agree on. If not in the database, line 7 would add the object.

A key design decision was to decide on how the database treats object confidence. A key feature of machine learning techniques such as DNN used for object detection is that they can calculate a score, indicating for example how confident they are in labeling an object as a tree. This value is usually between 0 and 1 with 1 being 100% sure. In this paper, we have decided to create a high-confidence map, meaning that the Genie will only share objects that are above a certain threshold with the cars requesting information. This decision was because information retrieved from the edge has to be reliable. To facilitate that, the threshold C_T for the high-confidence object map is set to be the 60th percentile of the confidence range in our design.

The initial confidence of a stored object could be lower than this threshold (for example, 0.3 instead of 0.65). However, if multiple cars see and detect the same object (potentially from various angles), the confidence in that object could be improved over time. For example, a traffic light is stationary. Even with an initial low score, many cars will see and recognize it. Thus, we would update the confidence of a seen-before object (line 5 of Alg. 9). We use gradient ascent to update the score.

Finally, Genie will call `EnhancedCacheBoostData` whenever it is returning a set of objects to the sender. This function checks for existing objects at relevant locations, and adds them to the list of objects to be returned to the sender.

Algorithm 9 Enhanced Semantic Cache

Require: $TYPES = \{objects\}$ ▷ Recognized type for the enhanced cache.
Require: $M < header, data, T >$ ▷ The message M.
Require: OBJECT_MAP $< location, objects >$ ▷ Object map.
Require: C_T ▷ Confidence threshold for the database (between 0 and 1).

```
1: function EnhancedCacheNewData(M)
2:     if type_of(M) ∈ TYPES then
3:         for object ∈ M.data.objects do
4:             if O = OBJECT_MAP[object.location] exists then
5:                 O.C = O.C_l + λ × (O.C_l − M.C_l)
6:             else
7:                 OBJECT_MAP[M.data.location].add(object)
8:             end if
9:         end for
10:    end if
11: end function
12: function EnhancedCacheBoostData(cacheEntry)
13:    if type_of(M) ∈ TYPES then
14:        for object ∈ cacheEntry.value.objects do
15:            for stored_object ∈ OBJECT_MAP[object.location] do
16:                if stored_object.C_l ≥ C_T then
17:                    cacheEntry.value.objects.add(stored_object)
18:                end if
19:            end for
20:        end for
21:    end if
22: end function
```

8.4 Evaluation

8.4.1 Experimental Setup

Devices. We evaluate our design's efficacy in tail latency, reusability, and value-added accuracy using an edge server cluster modeled after our industry partner's platform. The cluster includes three NVIDIA AGX Xaviers and three NVIDIA Jetson TX2s. The Jetson TX2s represent cars running the full autonomous driving software plus the ROS master, while the AGX Xaviers serve as remote edge servers. For heterogeneous setups, we add a server with an Intel Xeon E5-2650 CPU and NVIDIA Quadro RTX 4000.

Configuration. Cars and edge servers are co-located to eliminate network delays, focusing solely on computational latency effects from our caching method. We use Autoware, an

open-source autonomous driving software gaining industry traction [359]. Other ROS-based alternatives like those from BMW and Baidu share similar architectures [4, 391].

Data. The KITTI Vision Benchmark suite [114] provides raw data for playback from vehicle sensors, including four camera streams, a LIDAR point cloud, and precise location.

Measurements. Metrics include tail latency, image reusability (cache hits/total cache requests for image cache), object reusability (cache hits/total cache requests for object map), and confidence boost (average total boost for object map).

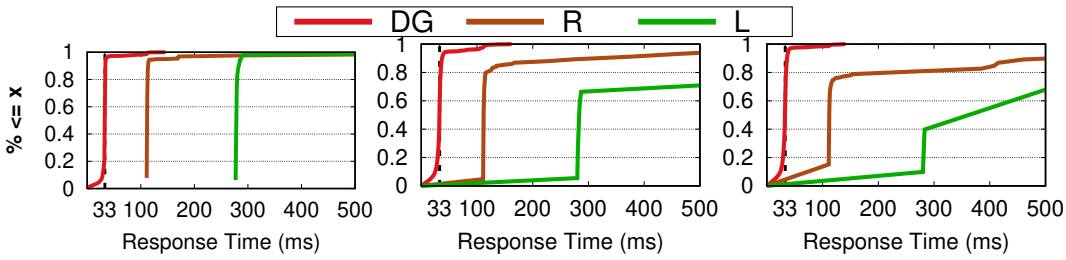
Scenarios. We evaluate scenarios with 1, 2, and 3 cars to explore different levels of complexity. Increased car involvement enhances collective cache information.

Homogeneous vs. Heterogeneous Cluster. We test clusters with low-power devices and add a powerful server for heterogeneous configurations. The Quadro-based server acts as one of the remote machines in heterogeneous tests.

Genie Case Study. We perform a case study demonstrating how Genie aids vision-assisted cars through communication with vision-enabled remote Genies.

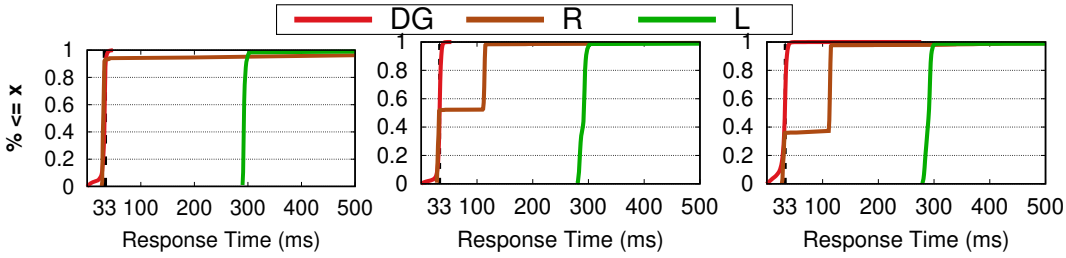
8.4.2 Tail-Latency Performance

Fig. 8.4 shows the cumulative distribution function (CDF) of response times for Genie (DG) versus ROS remote (R) and local (L) execution of the Autoware Vision Detector (AVD) across scenarios with 1, 2, and 3 cars in homogeneous and heterogeneous edge clusters. The 33ms deadline is marked with a dashed line. Genie outperforms both remote and local executions of AVD, achieving an average improvement of 82%, consistently meeting real-time demands with stable response times due to efficient caching and selective execution



(a) 1 car. (b) 2 cars. (c) 3 cars.

(i) Homogeneous setting.



(d) 1 car. (e) 2 cars. (f) 3 cars.

(ii) Heterogeneous setting.

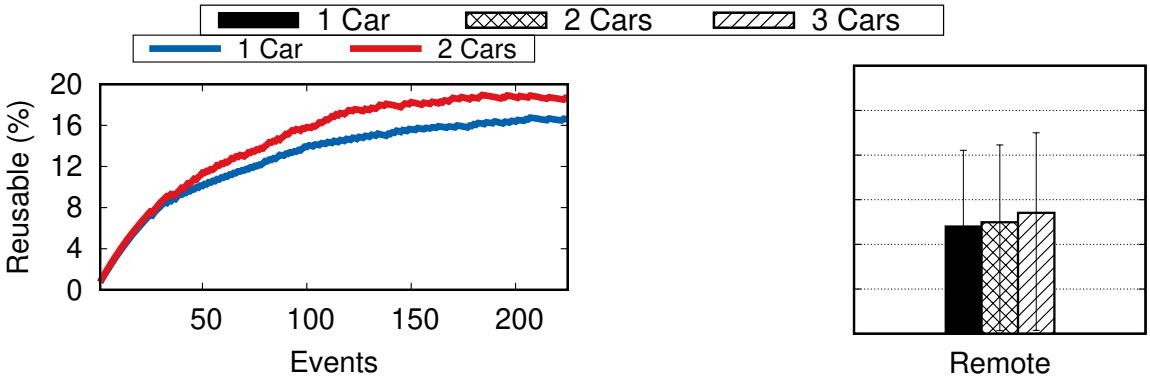
Figure 8.4: Cumulative distribution function for the response time of Genie(DG), versus remote (R) and local (L) ROS execution on homogeneous and heterogeneous configurations.

of algorithms. Adding more cars does not significantly impact latency, even with network communication, due to the wired Ethernet setup.

In the 1-car heterogeneous scenario, the server (R) performs similarly to Genie initially, but as cars increase, remote execution experiences higher latency, showcasing Genie's advantage in maintaining consistent performance. Genie's execution time remains stable across low-power embedded and powerful heterogeneous servers, thanks to its efficient CPU operations, adaptable to both ARM and Intel CPUs [204]. This ensures Genie's reliability across various architectures.

8.4.3 Image Caching

In this section, we measure the reusability of the average image cache under DG for the three scenarios and both on the homogeneous and heterogeneous server clusters.



(a) Homogeneous setting. (b) Hetero. setting.

Figure 8.5: Image reusability ratio for local Genies and remote Genies under the three scenarios.

The results are depicted in Fig. 8.5. The results are divided into Local, where the data is recorded on the cars (i.e., TX2) versus Remote, where the data is recorded from the remote servers (i.e., AGX). As is evident in the figure, Remote Genie nodes are capable of reusing more data because they can communicate with other cars in the case of 2-car and 3-car scenarios. In the case of the 1 Car scenario, the remote Genie has faster hardware and thus can collect many more objects compared to the local Genie. Fig. 8.5 also shows the error bars. As is evident in the figure, the maximum reusability on the local cache for the 2 cars and 3 cars scenario is quite high because the cache can be inflated initially from the indirect communication with the remote Genie of other cars.

Finally, the faster server in the heterogeneous architecture has enabled the collective cache to gather processed images faster, leading to higher average and maximum reusability.

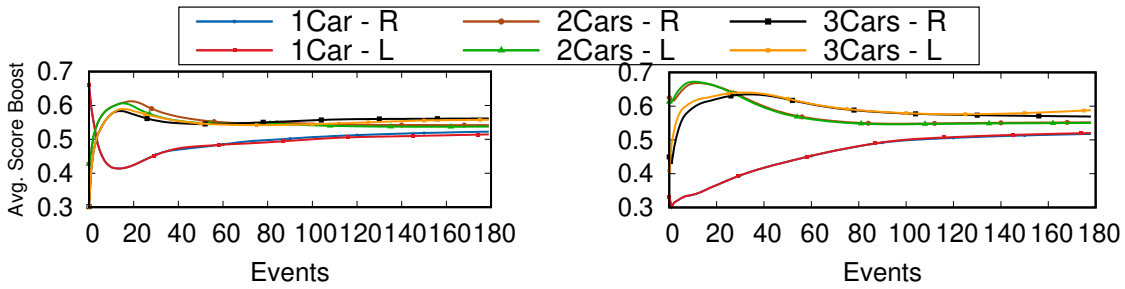
8.4.4 Object Caching

As discussed in Sec. 8.3.4, object mapping is particularly beneficial for autonomous driving, despite being less general and scalable than our message-based cache. Fig. 8.5

illustrates the average object reusability, defined as the ratio of cache hits on objects to total requests. The reusability rate for object caching is significantly higher than for image caching, averaging 31% and reaching a maximum of 67%, which is four times greater than image cache rates. In homogeneous scenarios, object reusability decreases with more cars because each location can contain a large number of objects. While more cars increase the number of available objects, not all meet the high confidence threshold from Sec. 8.3.4, slightly reducing overall reusability but enhancing data quality. For the 1-car scenario, the local object map shows better reusability due to selective uploads that avoid unnecessary requests to the remote server when objects are cached locally. In heterogeneous scenarios, the powerful server significantly enhances object reusability, especially with more cars. The server can process four times as many images (and thus objects) compared to the Jetson AGX Xavier, overcoming the limitations seen in the homogeneous setup and leading to increased reusability with additional vehicles.

8.4.5 Confidence Boost

We measured the confidence boost of our method across various scenarios, as shown in Fig. 8.6. The cumulative average scores results are compared for 1, 2, and 3 cars in both homogeneous and heterogeneous architectures. Confidence boosts are recorded for both remote (R) and local (L) Genies. The y-axis shows the running average confidence addition per object, while the x-axis represents the number of recorded events. Local Genies have generally lower confidence due to limited communication. The 3-car scenario shows a higher confidence boost compared to 2-car and 1-car scenarios, and the heterogeneous architecture achieves a higher cumulative average due to faster processing by the powerful server. Over a



(a) Homogeneous setting.

(b) Heterogeneous setting.

Figure 8.6: Cumulative average for the score (confidence) boost of all objects in remote and local Genies for all scenarios.

long period, the more cars that are present in the system, the higher the quality of the data on the remote Genies will become. Moreover, the heterogeneous architecture can achieve a slightly higher cumulative average value due to the faster processing of the powerful server.

Benefits of Distributed Cache versus Local Cache. Fig. 8.5 and Fig. 8.6 demonstrate the superiority of distributed caching in enhancing information sharing and reusability among nodes, a feature unattainable with local caches in ROS. Additionally, our distributed cache maintains minimal overhead, averaging 8.8ms (for a mix of cache hits and cache misses).

8.4.6 Case Study: Vision-Assisted Driving

In our case study on vision-assisted driving, we explored the potential of enhancing autonomous vehicles that navigate without cameras, relying instead on LIDAR and high-definition maps for localization, and point cloud-based detectors and radars for obstacle avoidance [203]. We introduced phantom Genies in our Genie design to assess if data from camera-equipped vehicles could benefit these non-vision-based systems. Our coordinator server identifies vehicles lacking Autoware Vision Detector and assigns them local and remote phantom Genies, facilitating data sharing among vehicles. For instance, in a two-car setup,

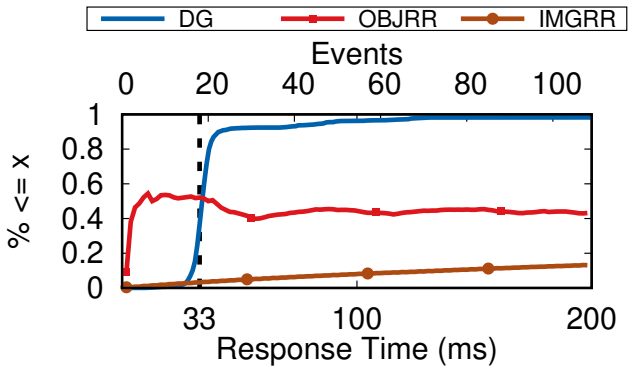


Figure 8.7: Vision-assisted driving: the CDF for time (ms), object reuse rate (OBJRR) based on events for the second car with no object detection (the image reuse rate (IMGRR) from the first car is also shown for reference).

one vehicle can utilize data from the other, which is illustrated in our findings. As shown in Fig. 8.7, although the response time for a car using this shared information is slightly increased, it remains under 41ms or 24FPS, demonstrating the viability of this approach despite the communication delay.

8.5 Related Work

Caching and Intelligent Caching. The concept of reusing computation to minimize costs is well-established. Techniques like FoggyCache enable close-quarters devices to reuse redundant computations [128, 94], but depend on centralized servers and lack driving-specific benefits. Similar methods exist for cloud environments [71, 136], relying on centralized APIs [3, 73] to manage caching, applicable to AR/VR [327], storage systems [11], mobile apps [243, 95], and robotics [214]. A few concurrent works like FogROS [54], FogROS2 [169], and Schafhalter et. al [313] also focus on improving latency performance in robots by using cloud hardware.

AI-based Collaborative Sensing. AI is expected to make great impacts on various robotic fields, including home services [379], healthcare [239, 238, 237, 236], and transportation [240, 241, 69, 402], etc., in 2030 according to Stanford's report [335]. Deep learning approaches have become prominent in autonomous driving [90, 89]. Approaches like collaborative sensing [235, 124] focus on individual devices or central units for task distribution, as seen in Potluck [129] and Darwin phones [253]. This contrasts with our decentralized approach. EMP [397] and AutoCast [295] proposed collaborative perception leveraging the edge to improve overall accuracy via decentralized data sharing among vehicles. Genie has the potential to integrate these solutions in ROS-based scenarios.

Real-Time Autonomous Driving. Reusability in distributed systems is uncommon in real-time contexts. Caching offers significant potential for meeting latency requirements, as seen in embedded platforms using approximation [19], scheduling [98, 215, 221, 213], memory management [7, 216], and software-hardware co-design [389]. Our solution Genie focuses on autonomous vehicle caching and introduces unique challenges, such as non-intrusive cache management and collaborative caching. Furthermore, we plan to adapt Genie to more safety-critical multi-robot scenarios [401, 109, 108, 110, 382].

8.6 Conclusion

This paper addresses three key challenges identified by an industry partner that impede the practical implementation of edge computing for connected autonomous vehicles. We developed Genie, demonstrating its effectiveness through implementation and evaluation

with realistic workloads and contemporary platforms. Future work aims to extend Genie beyond autonomous driving and adapt it to ROS 2 for more modern robotic system solutions.

Chapter 9

EOCL: Energy-Aware On-Device Online Continual Learning

9.1 Introduction

Modern intelligent robots operate in dynamic, open-ended environments on embedded GPUs and low-power SoCs, where compute, memory, and energy budgets are tight and time-varying. Sensor inputs drift (e.g., lighting, weather, and motion) [160] induced accuracy performance drops make offline retraining impractical due to latency, cost, and privacy. Online continual learning (OCL) incrementally updates models from streams, improving robustness to non-stationarity and mitigating catastrophic forgetting, eliminating the need for costly full retraining and enabling online adaptation [52, 10, 234, 51]. While such algorithmic advances have significantly improved the adaptability of learning models for robotics, the gap between algorithm design and deployable On-Device execution remains

substantial. Bridging this gap is non-trivial, as embedded robotic systems must integrate OCL within constrained compute and energy budgets, execute in real time, and remain robust under dynamic environmental conditions.

Specifically, deploying OCL On-Device introduces three fundamental technical challenges. (i) *Resource constraints and system coupling:* compute efficiency depends jointly on system- and application-level knobs, creating a high-dimensional trade-off space under tight power/compute/memory budgets; static or energy-oblivious settings leave substantial headroom untapped. (ii) *Dynamic environment:* non-stationary data streams, workload intensity shifts, and ambient factors (lighting, weather, motion) induce concept drift, *invalidating previously optimal configurations* and requiring timely adaptation. This represents arguably the most critical barrier to practical embedded OCL, since a single dramatic environmental change can render a carefully tuned system inefficient or non-functional. (iii) *Performance requirements:* embedded robots must sustain real-time responsiveness and streaming accuracy while respecting energy/thermal limits; heavy profiling or multi-trial searches are infeasible, mandating lightweight, sample-efficient runtime control.

Recent continual-learning systems such as Ekya [28] and RECL [188] achieve strong concurrent scheduling on multi-GPU servers, but rely on large-batch or offline coordination at coarse granularity, which does not transfer to embedded-critical systems. More fundamentally, *energy optimization*, pivotal for long-duration autonomy on battery-powered robots, remains largely unexplored in the OCL systems literature. A deployable solution must therefore sustain streaming accuracy and responsiveness while operating within energy limits under non-stationary conditions.

Problem Statement. We consider the runtime optimization problem of minimizing total energy consumption under latency and accuracy constraints in an On-Device OCL pipeline. (1) at each learning window, choose a feasible setting of system- and application-level knobs that satisfies current constraints; and (2) when environmental changes happen, promptly reassess the trade-offs and update the configuration with minimal samples and disruption.

Approach Overview. To this end, we present *EOCL*, a deployability-first runtime framework for On-Device OCL whose design is grounded in a principled Model-Based Optimization (MBO) formalism. Rather than relying on ad-hoc tuning, *EOCL* formalizes the deployment problem as a multi-objective online optimization over system and application knobs and introduces a recency-aware MBO engine, specifically a Gaussian-process surrogate with Expected Improvement acquisition, that continuously learns the energy, latency, and accuracy trade-off surface. To address the critical challenge of adapting to dramatic environmental shifts, *EOCL* employs a sliding-window surrogate that selectively forgets outdated observations and re-explores recent configurations, achieving sample-efficient, real-time re-adaptation after drift. We further design a concurrent On-Device execution pipeline integrated with the open-source Avalanche framework [46], supporting mainstream OCL algorithms and online metric collection.

This paper makes the following key contributions:

- **Problem formalization and motivation.** We formally define On-Device OCL deployment as an online constrained optimization problem, highlighting the tight coupling between system-level Dynamic Voltage and Frequency Scaling (DVFS) control and learning dynamics.

- **Methodology: Recency-aware MBO-driven Approach.** We propose a lightweight, formalism-guided model-based optimizer that jointly tunes CPU/GPU DVFS configuration and batch size online. The optimizer minimizes energy while maintaining latency and streaming accuracy, providing a principled and generalizable framework for embedded continual learning. A concept-drift-aware variant with a recency-weighted sliding-window surrogate enables sample-efficient re-adaptation to dramatic physical-environment changes, addressing a fundamental open challenge for On-Device OCL.

- **System and implementation design.** We implement *EOCL* as a concurrent OCL runtime built atop open-sourced tool Avalanche, supporting dynamic instrumentation, metric fusion, and fine-grained control on Jetson-class devices.

- **Evaluation.** We evaluate *EOCL* on two Jetson platforms across five benchmarks, four OCL algorithms, and a high-fidelity robotic navigation case study with dynamic environmental factors. Results show *EOCL* cuts energy by *up to* 26% vs. the strongest online schedulers (*on average* 19%) and by *up to* 70% vs. RECL (*on average* 52%), with $< 2\%$ absolute accuracy drop on medium/large workloads and in our robotic case study, and minimal latency overhead. Under sudden and incremental environmental changes, our proposed method sustains these savings and achieves the lowest CER among online methods, indicating steadier control and better adaptivity to environmental changes.

9.2 Background

Modern edge intelligence systems are increasingly deployed in dynamic, open-ended environments using resource-constrained platforms such as embedded GPUs and

low-power SoCs. In these settings, where both the input data distribution and available system resources fluctuate over time, models must learn continuously. To address these constraints, online continual learning (OCL) has emerged as a promising technique, enabling deep learning models to incrementally self-adapt directly on the device. For example, in real-world robotic applications, OCL provides a lightweight mechanism for adapting to new lighting or terrains in autonomous navigation, recognizing novel objects on the fly [289], updating person re-identification models under varying conditions [377], and continuously refining physical manipulation skills [207]. However, successfully deploying OCL on edge robots requires balancing three critical metrics, *latency*, *accuracy*, and *energy efficiency*, to ensure sustainable, real-time operation. While prior OCL studies have extensively examined accuracy and forgetting mitigation [52, 10, 146], energy optimization has remained largely overlooked despite its essential role in long-term, deployable OCL-driven robotic autonomy.

Techinically, OCL enables models to update incrementally as new data arrives without full retraining [52, 10, 51]. Unlike static, offline learning on stationary datasets, OCL operates under *non-stationary* and *streaming* conditions driven by environmental shifts, changing illumination, or sensor noise [160]. Such conditions can lead to rapid performance degradation if adaptation is delayed, yet aggressive updates risk catastrophic forgetting. Replay-based methods such as Experience Replay (ER) [52] and memory-efficient regularization approaches attempt to alleviate forgetting. Such continual adaptation is essential in domains like robotics, autonomous vehicles, and smart sensors [262, 347, 133], where environmental conditions change unpredictably and embedded platforms (e.g., NVIDIA Jetson-class SoCs) impose tight memory, compute, and power constraints. Standard offline

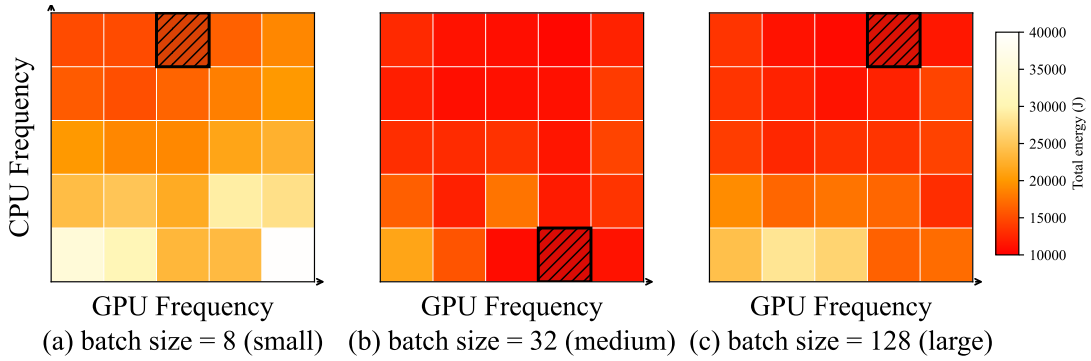


Figure 9.1: Visualization of complex reconfigurable space among three key control knobs of On-Device OCL. The diagnostic shadow indicates the most energy-efficient CPU/GPU frequency configuration for each training batch size.

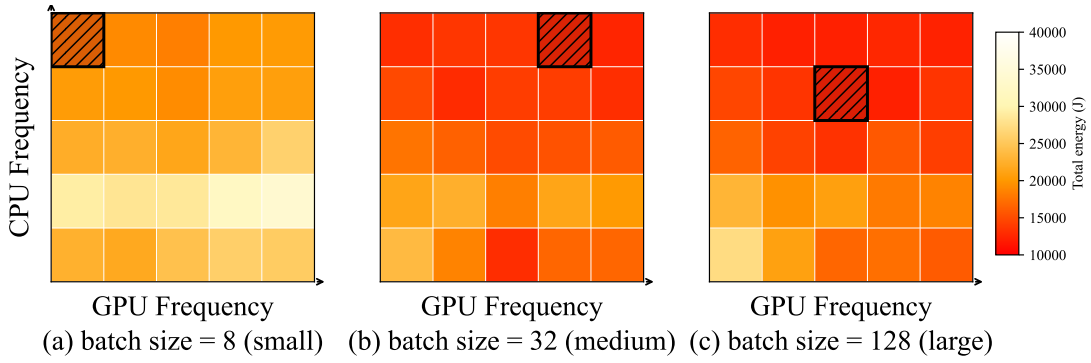
retraining is impractical in such settings due to high compute cost, memory usage, privacy concerns, and unpredictable execution time.

9.3 Motivation

To understand where energy-efficiency headroom comes from in On-Device online continual learning, we conduct two concise motivational studies on an NVIDIA Jetson AGX Orin. One standard online continual learning workloads, CORe50-NIC, is used in these studies. Specifically, we vary three key control knobs that are universally available on On-Device OCL, i.e., CPU frequency, GPU frequency, and training batch size, and compare four schedulers for handling environmental changes-induced sudden concept drift scenarios.

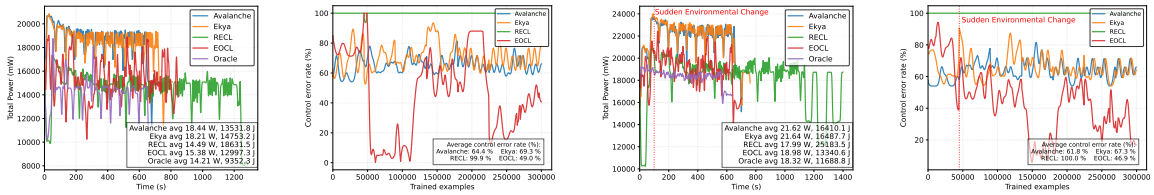
9.3.1 Challenges of Online Optimization in Large and Complex Reconfiguration Space

Fig. 5.1 visualizes a 5×5 sweep over CPU and GPU frequency for three batch sizes: (a) b=8 (small), (b) b=32 (medium), and (c) b=128 (large). Each cell shows total SoC



(a) batch size = 8 (small) (b) batch size = 32 (medium) (c) batch size = 128 (large)

Figure 9.2: Visualization of complex reconfigurable space among three key control knobs of On-Device OCL under sudden environmental changes.



(a) Energy consumption and power trace vs. time with no environmental changes.

(b) Control error rate (CER) vs. trained examples with no environmental changes.

(c) Energy consumption and power trace vs. time with sudden environmental changes.

(d) Control error rate (CER) vs. trained examples with sudden environmental changes.

Figure 9.3: Power trace and control error rate analysis of online continual learning workloads.

energy; darker (red) indicates lower energy and brighter (yellow/white) higher energy. The diagonally hatched cell marks the most energy-efficient CPU/GPU pair at that batch size. Complementarily, Fig. 9.2 repeats the same sweep under a sudden environmental change, revealing that both the location of the optimum and the overall energy landscape *re-shape*.

It is clear that the hatched optima shift with batch size and can also relocate when the environment changes; choosing a single static DVFS point leaves substantial savings untapped, often tens of percent and sometimes nearly $\sim 2\times$. Moreover, given the inherently online nature of OCL workloads (Sec. 9.2), fine-grained offline profiling is impractical; when sudden changes occur, pre-profiled settings become stale and previously

"optimal" configurations could be drifted. This heterogeneity motivates a lightweight online optimization strategy that adaptively learns a sample-efficient surrogate of the black-box energy surface over system- and application-level configurations.

Takeaway 1: The energy–optimal CPU/GPU DVFS configuration for On-Device OCL is batch-size dependent and drifts with both system state and environmental conditions; therefore, a lightweight online optimization strategy is required. The optimizing strategy should be sample-efficient, broadly applicable across workloads/hardware, and generalizable to non-stationary operating conditions.

9.3.2 Challenges of Handling Sudden Environmental Changes

Fig. 9.3 contrast two streaming settings on Jetson AGX Orin: a steady environment and a sudden changing environmental. We compare four online schedulers, Avalanche [46] (fixed, performance-oriented policy max clocks), Ekya [28] and RECL [188] (state-of-the-art schedulers that are energy-oblivious), and *EOCL* (our proposed method) against an *Oracle* found by offline grid search. We discuss results in two-fold: energy consumption and control error rate.

Energy Consumption

Stable environment. As demonstrated in Fig. 9.3a, all methods settle into quasi-stationary power envelopes, but EOCL converges to a visibly lower operating band than Avalanche/Ekya and substantially below RECL. The legend summaries avgerage power and total energy, showing the ranking that *EOCL* is is the closest to Oracle. Specifically, *EOCL* is the closest

online method to Oracle and *shrinks the gap-to-Oracle* by 12.8% vs. Avalanche, 32.5% vs. Ekya, and 60.7% vs. RECL.

Sudden environmental change. At the sudden enviromental change point (Fig. 9.3c),, Avalanche and Ekya *overshoot*, they retain high DVFS and flat power near the pre-drift envelope, while RECL *undershoots* and recovers slowly, lengthening execution. In contrast, *EOCL* promptly re-optimizes its DVFS/batch configuration, dropping to a lower post-drift power band and avoiding long tails. The resulting total energy is the best among online methods and approaches the offline Oracle.Specifically, *EOCL* shrinks the energy gap to Oracle by 65.0% vs. Avalanche, 68.0% vs. Ekya, and 87.8% vs. RECL.

Control Error Rate (CER)

While the raw control signals (batch size, CPU frequency index, GPU frequency index, and scheduling mode) describe how each scheduler steers the device over time, direct comparison across methods is challenging since these control knobs differ in units and scales. We therefore normalize them into a unified *control vector* and evaluate a progress-aligned *control error rate (CER)* relative to an offline Oracle obtained via dense grid search over batch size and DVFS to minimize energy on the same stream.

Let n index trained examples in the stream. For method m, define the normalized control vector:

$$\mathbf{u}_m(n) = \left[\frac{b_m(n)}{b_{\max}},\ \frac{\mathrm{cpu}_m(n)}{\mathrm{cpu}_{\max}},\ \frac{\mathrm{gpu}_m(n)}{\mathrm{gpu}_{\max}},\ s_m(n)\right], \tag{9.1}$$

where b_m is the instantaneous training batch size, cpu_m and gpu_m are the selected DVFS indices for CPU and GPU, and $s_m(n) \in \{0, 1\}$ indicates the scheduling decision (0: concurrent

training and inference; 1: non-concurrent). The maximal available values $b_{\max}$, $\mathrm{cpu}_{\max}$, and $\mathrm{gpu}_{\max}$ used for normalization.

With weights $\mathbf{w}=[\alpha,\beta,\gamma,\delta]$, all set to unity by default to treat each knob equally, we define an inner product over the control space $\mathcal{U}$ as:

$$\begin{aligned}\langle \mathbf{u}_1, \mathbf{u}_2 \rangle_{\mathcal{M}} =& \alpha u_1^{(b)} u_2^{(b)} + \beta u_1^{(\mathrm{cpu})} u_2^{(\mathrm{cpu})} \\ &+ \gamma u_1^{(\mathrm{gpu})} u_2^{(\mathrm{gpu})} + \delta u_1^{(s)} u_2^{(s)},\end{aligned} \tag{9.2}$$

which induces the semi-norm $\|\mathbf{u}\|_{\mathcal{M}} = \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle_{\mathcal{M}}}$. The aggregated control signal (control sum) is then expressed compactly as

$$y_m(n) = \langle \mathbf{w}, \mathbf{u}_m(n) \rangle_{\mathcal{M}}. \tag{9.3}$$

The Oracle target $y_{\mathrm{Oracle}}(n)$ is defined analogously from the grid-searched configuration $(b^\star, \mathrm{cpu}^\star, \mathrm{gpu}^\star, s^\star)$ and remains constant over n.

Given a common prefix of $N^\star$ training examples, we measure the *pointwise deviation* between the control trajectories of method m and the Oracle as an $\mathcal{M}$-induced norm:

$$d_m(n) = \|\mathbf{u}_m(n) - \mathbf{u}_{\mathrm{Oracle}}(n)\|_{\mathcal{M}}, \quad n = 1, \ldots, N^\star, \tag{9.4}$$

and normalize it by the largest instantaneous deviation across all methods to obtain the control error rate:

$$\mathrm{CER}_m(n) = 100 \cdot \frac{d_m(n)}{\max_{m' \in \mathcal{M}} d_{m'}(n)} \in [0, 100]. \tag{9.5}$$

The overall average CER is reported over the shared prefix:

$$\overline{\mathrm{CER}}_m = \frac{1}{N^\star} \sum_{n=1}^{N^\star} \mathrm{CER}_m(n)\ (\%). \tag{9.6}$$

This induced-norm formulation ensures geometric consistency across heterogeneous control spaces and is invariant to throughput scaling. It directly quantifies how closely a scheduler tracks the Oracle's control policy (0% = Oracle-like, 100% = worst-case deviation).

Stable environment. As demonstrated in Fig. 9.3b, CER traces align with the power patterns: *EOCL* maintains a markedly lower CER than Avalanche/Ekya, indicating closer tracking of the oracle control trajectory even without drift. *EOCL* achieves an average CER of 47.3%, substantially lower than Avalanche (64.4%, −26.6% improvement), Ekya (69.3%, −31.8%), and RECL (99.9%, −52.6%). *EOCL*'s curve exhibits rapid convergence and consistently remains below the baselines, indicating tighter alignment with the oracle control policy and more stable adaptation.

Sudden environmental change. At the sudden enviromental change point (Fig. 9.3d), CER spikes sharply for Avalanche/Ekya/RECL after the dashed line but quickly collapses for *EOCL* and remains low thereafter, evidencing fast adaptation and stable policy tracking. Quantitatively, *EOCL* attains an average CER of 46.9%, compared with Avalanche's 61.8% (24.1% improvement), Ekya's 67.3% (-30.3%), and RECL's 100.0% (-53.%). After the sudden drift point, *EOCL* rapidly stabilizes below 50% CER, while all baselines remain oscillatory or saturated near their pre-drift levels. This highlights *EOCL*'s strong resilience and rapid recovery in non-stationary conditions, maintaining performance closest to the oracle control trajectory despite abrupt environmental shifts.

Takeaway 2: In realistic OCL deployments, *sudden* environmental changes trigger concept drift; energy-oblivious schedulers often overshoot or react too slowly, degrading efficiency. *EOCL*'s concept drift-aware characteristics make it suitable for these realistic scenarios by (i) searching the joint DVFS–batch space sample-efficiently and (ii) especially effective in continuously re-optimizing promptly upon detecting drift to remain energy-efficient under nonstationary environmental scenarios.

9.4 Methodology

This section first formalizes the configuration-tuning problem for On-Device OCL and then introduces our model-based optimization (MBO) approach, $EOCL_s$, together with its concept-drift–aware variant, $EOCL_{cd}$, which dynamically tracks environmental changes.

9.4.1 Problem Formulation

We consider On-Device OCL on an embedded platform with tunable compute and training parameters. Our original knob set in Eq. 9.1 included a binary *scheduler* choice, *parallel* (enable multicore/multithreading and accelerator concurrency) versus *sequential* (single-threaded execution). On our target platform and workloads, parallel execution reliably improves throughput and *energy-to-solution* via higher concurrency and "race-to-idle" effects, a behavior reported broadly for multicore/accelerator systems and GPU spatial multitasking [30]. Sequential mode substantially reduces utilization, increasing wall-time and often total

energy. To avoid expanding the mixed discrete search space with a categorical factor that is consistently dominated in practice, we set the scheduler to *parallel* by default and let

$$\boldsymbol{x} \;=\; \big(u_{\mathrm{cpu}},\, u_{\mathrm{gpu}},\, b\big)$$

denote the configuration, where $u_{\mathrm{cpu}} \in \{0, \ldots, C-1\}$ and $u_{\mathrm{gpu}} \in \{0, \ldots, G-1\}$ index discrete DVFS (Dynamic Voltage and Frequency Scaling) states for the CPU and GPU, respectively, and $b \in \mathbb{N}$ is the training batch size. Frequency/voltage states directly influence dynamic power (approximately $P \propto C_{\mathrm{sw}} V^2 f$), and batch size is known to affect both throughput and energy per update in deep-learning workloads [259, 380].

At discrete time t, let $\hat{E}_t(\boldsymbol{x})$ denote the (random) energy consumed by performing one OCL *learning step* under configuration $\boldsymbol{x}$ (including forward/backward passes, optimizer updates, and any replay/checkpoint I/O). Our best-effort objective is *per-step* energy minimization without prescribing long-horizon budgets or peak-power caps:

$$\begin{aligned} \tilde{\boldsymbol{x}}_t^{\star} &= \arg\min_{\boldsymbol{x} \in \mathcal{X}} \; \mathbb{E}\Big[\hat{E}_t(\boldsymbol{x})\Big]\,, \\ \mathcal{X} &= \{0, \ldots, C-1\} \times \{0, \ldots, G-1\} \times \mathbb{N}. \end{aligned} \tag{9.7}$$

Per-step minimization can favor very small batches (few samples per step), which lowers instantaneous energy but may increase the *total* energy to process a dataset. Therefore, we consider a *per-sample* objective to capture efficiency at the data level. Let $n_t(\boldsymbol{x})$ be the

number of samples processed in the step at time t under $\boldsymbol{x}$ (typically $n_t(\boldsymbol{x}) = b$). Define the per-sample energy:

$$E_t(\boldsymbol{x}) \;=\; \frac{\hat{E}_t(\boldsymbol{x})}{n_t(\boldsymbol{x})},$$

We formulate the optimization problem as tracking a time-varying minimizer:

$$\boldsymbol{x}_t^\star = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \; \mathbb{E}[E_t(\boldsymbol{x})]\,. \tag{9.8}$$

Evaluating a candidate configuration $\mathbf{x} = (u_{\text{cpu}}, u_{\text{gpu}}, b)$ requires executing an actual On-Device OCL step. Consequently, the objective $E_t(\mathbf{x})$ is an expensive, noisy black box influenced by complex, hard-to-model hardware dynamics like thermal coupling and DVFS-induced memory traffic.

To efficiently track $\boldsymbol{x}_t^\star$, we employ Model-Based Optimization (MBO). Instead of exhaustive online profiling, MBO fits a probabilistic surrogate to past measurements. This surrogate predicts both the expected energy and the uncertainty of untried configurations, intelligently guiding the search by balancing exploration and exploitation (e.g., via Expected Improvement). We specifically choose a Gaussian Process (GP) surrogate because it provides the closed-form uncertainty estimates necessary for our low-dimensional, sample-constrained space. To naturally handle environmental nonstationarity (drift) and bound the GP's cubic computational scaling, we apply a simple sliding window over recent evaluations (Sec. 9.4.3). This ensures the optimization overhead remains strictly predictable while continuously tracking the moving objective without requiring complex temporal kernels.

9.4.2 Model-Based Optimization

In this subsection, we present our detailed Model-Based Optimization (MBO) approach $EOCL_s$. First, we cast configuration tuning as the black-box *minimization* problem

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}),$$

where $f : \mathcal{X} \subset \mathbb{R}^p \to \mathbb{R}$ is expensive to evaluate. MBO replaces direct optimization of f with an inexpensive surrogate that is iteratively refined from past evaluations.

Moreover, we use a Gaussian-process (GP) surrogate. The procedure is:

1. **Initialization.** Draw an initial design $\mathcal{D}_0 = \{\boldsymbol{x}_i\}_{i=1}^k$ (e.g., a Latin hypercube over $\mathcal{X}$) and observe $\boldsymbol{y}_0 = \{y_i\}_{i=1}^k$ with $y_i = f(\boldsymbol{x}_i)$.

2. **Surrogate fitting.** Fit a GP $\hat{f}$ on $(\mathcal{D}_t, \boldsymbol{y}_t)$.

3. **Acquisition construction.** Define an acquisition function $\mathrm{acq}(\boldsymbol{x}; \hat{f})$ that trades off exploration and exploitation using the surrogate's mean and uncertainty.

4. **Candidate selection.** Choose the next configuration

$$\hat{\boldsymbol{x}}_{t+1} = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathrm{acq}(\boldsymbol{x}; \hat{f}).$$

5. **Evaluation and update.** Evaluate $y_{t+1} = f(\hat{\boldsymbol{x}}_{t+1})$ and augment the dataset: $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{\hat{\boldsymbol{x}}_{t+1}\}$, $\boldsymbol{y}_{t+1} = \boldsymbol{y}_t \cup \{y_{t+1}\}$.

Repeat Steps 2–5 until the evaluation budget is exhausted; the final configuration is the input with the lowest *observed* objective value.

With a GP surrogate, a popular acquisition function is the expected improvement (EI). Using a Gaussian process as a surrogate yields a Gaussian posterior with mean $\hat{\mu}(\boldsymbol{x})$ and standard deviation $\hat{s}(\boldsymbol{x})$ at each point $\boldsymbol{x}$. Accordingly the expected improvement can be derived as follows:

$$\begin{aligned} \mathrm{EI}(\boldsymbol{x}) &= \mathbb{E}(max(\hat{\mu}(\boldsymbol{x}) - y_{\max}), 0) \\ &= (\hat{\mu}(\boldsymbol{x}) - y_{\max})\Phi(\frac{\hat{\mu}(\boldsymbol{x}) - y_{\max}}{\hat{s}(\boldsymbol{x})}) + \hat{s}(\boldsymbol{x})\phi(\frac{\hat{\mu}(\boldsymbol{x}) - y_{\max}}{\hat{s}(\boldsymbol{x})}) \end{aligned} \tag{9.9}$$

where Φ is the distribution, and ϕ is the density function of the standard Gaussian distribution and $y_{\max}$ is the best observed value in $\boldsymbol{y}$ so far. This classical formulation of MBO yields one proposal $\hat{\boldsymbol{x}}$ in each iteration.

9.4.3 Model-Based Optimization for Concept Drift

Towards concept drift scenarios, that a optimal configuration at certain time may become suboptimal due to environmental changes, we propose a concept-drift variant $EOCL_{cd}$ that dynamically tracks environmental changes.

In our setting, the configuration that minimizes power consumption is not fixed but evolves with the environment and device state; this yields a *time-varying* black-box objective. Such a dynamic optimization problem is often discussed under *concept drift*. Concretely, we model both the objective and its optimizer as functions of time,

$$\boldsymbol{x}_t^* = \operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{X}} f_t(\boldsymbol{x}).$$

so that a configuration that was optimal at time t may become suboptimal later as illumination, motion patterns, background load, temperature, or harvested power change. The methodological implication is to *track* rather than "solve once": the optimizer must rapidly re-identify $\boldsymbol{x}_t^*$ as conditions shift, while still being sample-efficient. Prior work in time-varying Bayesian optimization (BO) formalizes this via dynamic regret and proposes mechanisms such as restarts and sliding windows to handle nonstationarity.

We extend classical MBO with a *recency-weighted* surrogate and online incumbent selection. At time t, the Gaussian-process surrogate is fit on a sliding window of recent evaluations. This windowing intentionally re-explores previously visited regions to detect shifts in f_t, and induces the standard bias–variance trade-off: a larger t_Δ provides more data but adapts more slowly to sudden changes; a smaller t_Δ reacts quickly but increases modeling variance. Because repeated evaluations of the same $\boldsymbol{x}$ can yield different outcomes y as the environment evolves, observations are treated as noisy. To account for the additional insecurity the *nugget effect* [311] is estimated and added to the GP. Accordingly, augmented EI (AEI) and related noisy-EI variants are applied in [306].

In this work, given the windowed surrogate, we maintain a set of evaluations with fixed size, i.e., $\mathcal{D}_\omega$. The number of the evaluated configurations is defined by the window size ω. The set is operated as a first-in-first-out queue, that is, once a new evaluation is added, the oldest evaluation is deleted to maintain the fixed size. In each iteration, we construct a standard acquisition function and select the next point:

$$\hat{\boldsymbol{x}}_{t+1} = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathrm{EI}(\boldsymbol{x})$$

then observe $y_{t+1} = f_t(\hat{\boldsymbol{x}}_{t+1})$ and update $\mathcal{D}_{\omega,t+1}$. We still use classical expected improvement, EI, for simplicity and low overhead. While AEI and related noisy-EI are available, EI remains competitive and inexpensive in practice.

At each step we publish an online incumbent $\hat{\boldsymbol{x}}_t^*$ consistent with the current surrogate (best predicted mean among evaluated points, and we assess tracking quality against the time-varying optimum via standard online criteria (e.g., instantaneous objective gap or dynamic regret over time). In effect, $EOCL_{cd}$ follows the moving energy-minimizing configuration with best effort, emphasizing recency without imposing explicit long-horizon energy budgets or peak-power caps.

9.4.4 Scope of analysis and guarantees

We note that EOCL combines a principled optimization core with deployment-driven engineering choices. The core MBO formulation, a GP surrogate with EI acquisition, is a standard Bayesian optimization framework for expensive black-box optimization, and its stationary setting is well studied in the literature. For time-varying objectives, prior work has shown that restart- and sliding-window-based BO can achieve dynamic-regret guarantees under bounded drift assumptions. In contrast, several system parameters in EOCL, including the control interval Δ and the fixed sliding-window size ω, are selected empirically to balance adaptivity and runtime overhead on embedded devices. We therefore position EOCL as a practical, best-effort runtime for resource-constrained On-Device OCL rather than a method with formal worst-case guarantees on energy or latency under arbitrary environmental changes. Extending EOCL with formal regret bounds under deployment-specific drift models remains an important area for future work.

9.5 System Prototyping, Deployment, and Metrics

This section outlines our end-to-end On-Device OCL deployment, embedded system prototyping, and the streaming data preparation pipeline. We summarize implementation details, instrumentation, and metrics, including end-to-end latency, streaming accuracy, and energy consumption. We then describe the optimization controls: passive control knobs adjusted online by our MBO controller, and active OCL application-specific adaptations that reshape the input stream.

9.5.1 System Prototyping

Runtime integration. Our prototype is built on Avalanche [46] over PyTorch[288] and ported to ARM64 for embedded deployment, integrated as a thin, non-intrusive plugin that leaves models, optimizers, datasets, and checkpoints unchanged. Both variants of proposed methods are implemented based on BoTorch [15]. We use double-buffered queues between training and inference. Inference executes on an exclusive GPU stream, while training runs on a separate CUDA stream. Internal inference and training pipelines remain those of the original OCL framework.

Controller, Communication, and Reconfiguration. A lightweight, independent controller communicates with Avalanche's training and evaluation workers via a lock-free, multiprocessing-backed shared dictionary to eliminate serialization overheads. Workers publish throughput, accuracy, and queue metrics after each batch. Every $\Delta = 10$ s, the controller atomically reads this snapshot and performs one MBO step over a sliding window $\omega = 90$ for $EOCL_{cd}$ (Sec. 9.4.3). This specific Δ was chosen to carefully balance rapid

adaptation against the non-trivial CPU and energy overheads of frequent GP surrogate updates; it perfectly fits Jetson-class platforms where learning steps take 0.5–2 s and environmental shifts unfold over tens of seconds. The controller then notifies workers via a single `SIGUSR1` signal to apply the proposed `reconfigure(dvfs_cpu, dvfs_gpu, batch)` updates using CUDA-safe callbacks at step boundaries. This strictly preserves process isolation and Avalanche's pipeline without modifying user code. Latencies are measured using a monotonic host clock bracketing the ingest-to-commit window (Sec. 9.5.3), with all signals recorded in an append-only log for optimizer consumption. For deployments with tighter real-time constraints, dynamically adapting Δ (e.g., shortening during drift, lengthening when stable) and scaling to multi-device MBO coordination remain key directions for future work.

OCL-Specific Active Optimization. We decouple optimization into MBO-tuned *passive controls* (e.g., CPU/GPU DVFS, batch size; Sec. 9.4) and workload-specific *active levers* that reshape the input stream. Active levers improve latency, accuracy, and energy by dynamically adjusting sensor rate, resolution, exposure, and training frequencies based on scene dynamics and model confidence (e.g., downsampling during low motion, or triggering label acquisition only under high uncertainty and adequate thermal headroom). A coarse-grained outer loop governs these active levers at a slower cadence, seamlessly complementing the inner MBO loop that continuously tunes passive knobs for real-time adaptation.

9.5.2 On-Device Deployment and Data Preparation

Streaming Pipeline. Our embedded system executes a five-stage On-Device streaming pipeline: capture, preprocess, inference, training, and optimizer. Each sample is timestamped

on arrival and passed through resizing, normalization, and optional denoising before entering a bounded MPMC queue. Inference generates predictions using the current model, and training performs OCL model updates. The optimizer stage executes the MBO controller, adjusting control knobs, and conducts unified logging for optimization. Inference and training run concurrently under the fixed execution policy of Avalanche, and their algorithmic internals remain unmodified.

Progress Monitoring, System Profiling, and Logging. We expose OCL progress without modifying algorithm logic by overriding the framework's data loader to emit lightweight statistics (e.g., trained examples, step/epoch indices). A callback aggregates these into a read-only global snapshot that downstream components use to correlate progress with latency and energy. On-Device system profiling runs the Jetson-native `tegrastats` at 1 Hz to record timestamps, CPU/GPU frequencies, and power; we also collect memory usage and temperature. To support future extensions, profiling is routed through an abstraction layer with pluggable backends and a counter registry, with `tegrastats` as the default implementation. At each tick, system readings are aligned with application counters (e.g., queue depths, step times, number of trained example) and fused into a unified, time-ordered log consumed by the optimizer and evaluation scripts. For thread safety, aggregation is out-of-place: immutable per-tick records are appended to an append-only buffer so readers observe consistent snapshots without locks.

9.5.3 Metrics Computation

End-to-end latency. We report *pipeline-level* OCL latency (training and inference combined), not per-item latency. Using software timestamps, for each analysis window W_k,

let $t^{\mathrm{sw}}_{\mathrm{min}}(W_k)$ denote the earliest sample-ingress time and $t^{\mathrm{dec}}_{\mathrm{max}}(W_k)$ the latest commit time (prediction or learning step) observed in the window. The end-to-end latency[1] is

$$L^{\mathrm{ocl}}(W_k) \;=\; t^{\mathrm{dec}}_{\mathrm{max}}(W_k) - t^{\mathrm{sw}}_{\mathrm{min}}(W_k).$$

Streaming accuracy (SA). In our learning task setting, streaming accuracy is computed strictly following the Avalanche framework's implementation. For a window W_k,

$$\mathrm{Acc}(W_k) \;=\; \frac{1}{|W_k^{\ell}|} \sum_{i \in W_k^{\ell}} \mathbf{1}\{\hat{y}_i = y_i\},$$

where W_k^{ℓ} are items with labels available under the framework's labeling policy or tolerance[2].

Energy consumption. We sample power $P(t)$ from device rails at a fixed period Δt and compute per-window energy and average power as

$$E(W_k) \;=\; \sum_{t \in W_k} P(t)\,\Delta t.$$

---

[1]Although end-to-end latency can span 10^2–10^4 s for a full stream pipeline, the processing time for a single training example is typically sub-second to a few seconds. We report the pipeline-level metric to expose differences across OCL benchmarks and algorithms, capturing queuing, overlap, and scheduler effects that per-item timings miss.

[2]Streaming accuracy is task-defined; for other learning tasks or OCL scenarios, the definition may change accordingly.

Control Error Rate (CER). To quantify how closely a method's control trajectory follows an Oracle policy, we additionally report the *Control Error Rate* (CER) as formulated earlier in the motivation (Sec. 9.3.2).

All statistics are computed from the append-only, time-ordered log described in Sec. 9.5.2, avoiding any in-place updates during aggregation.

9.6 Evaluation

9.6.1 Experimental Setup

Control Knobs for Optimization. To achieve better energy-efficiency of On-Device OCL, we continuously adjust and optimize three key control knobs for system-/application-level, i.e., CPU frequency, GPU frequency, and training batch size, as discussed in Sec. 9.3.1. For more details, please refer to the supplementary files.

Testbeds. Our evaluation employs two GPU-enabled embedded platforms that represent distinct hardware capabilities: NVIDIA Jetson AGX Orin and NVIDIA Jetson Nano Orin. These platforms are frequently utilized in real-world applications such as autonomous driving [187, 195] and robotics [293, 269, 271, 273]. Detailed hardware specifications are shown in the supplementary file.

OCL Algorithms, DNN Architecture, and Benchmarking. We assess *EOCL* across four representative OCL algorithms: ER [52], GSS [10], GEM [234], and AGEM [51]. We evaluate our systems across representative DNN architecture ResNet-20 [146]. Our evaluations involve multiple standard OCL benchmark datasets, spanning different-scale

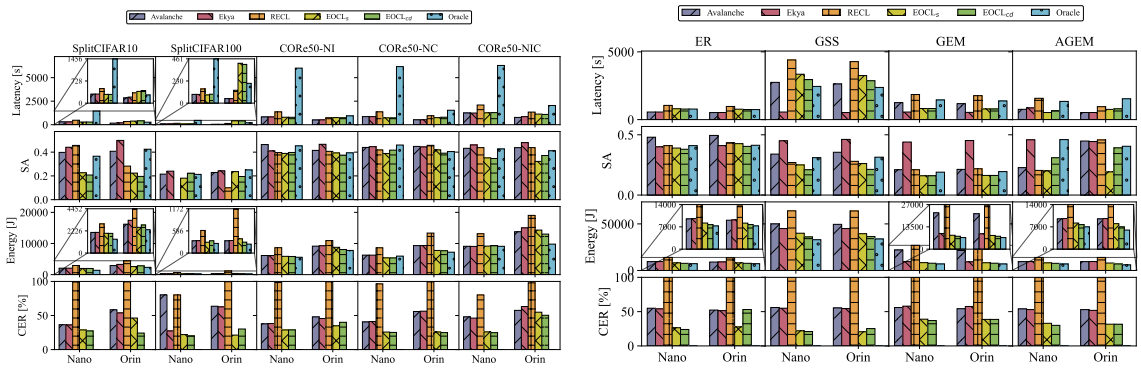
incremental learning scenarios: SplitCIFAR10 (small), SplitCIFAR100 (small), CORe50-NI (medium), CORe50-NC (medium), and CORe50-NIC (large). These experimental settings encompass a diverse range of continual learning scenarios, ensuring comprehensive benchmarking. More details of the benchmarks are described in the supplementary files.

Baselines. We compare *EOCL* with the following methods:

- **Avalanche [46]:** Avalanche is a prominent open-source library designed specifically for continual learning, widely adopted for fast prototyping, training, and reproducible evaluation of OCL algorithms. Avalanche received over 2000 GitHub stars, underscoring its broad acceptance in the community. We have improved it to support concurrent execution of training and evaluation for fair comparisons. Avalanche employs a max-frequency DVFS strategy by default, i.e., it runs all CPU and GPU cores at the highest available frequency throughout execution. This "race-to-idle" policy is a viable strategy for computation-intensive workloads because completing work at peak frequency maximizes throughput and allows the processor to enter low-power idle states sooner.

- **Ekya [28]:** Ekya is a state-of-the-art approach originally designed for concurrent offline continuous learning on multi-GPU servers. We adapt Ekya to fit embedded online continual learning scenarios. Implementation details for adaptation are provided in the supplementary material.

- **RECL [188]:** RECL is a state-of-the-art approach for concurrent continual learning in autonomous driving scenarios, video analytics on multi-GPU servers. We also adapt RECL to fit embedded online continual learning scenarios. Implementation details for adaptation are provided in the supplementary material.



(a) Evaluated on five benchmarks (ER algorithm).

(b) Evaluated across four OCL algorithms (CORe50-NC).

Figure 9.4: Overall effectiveness of *EOCL*. SA refers to streaming accuracy.

- **Oracle:** We conduct a sufficient grid-search for 75 configurations for each experimental setting and report the most energy-efficient configuration as Oracle. Oracle is a best-effort approximation because the configuration space is continuous and combinatorially large, and exhaustive search over the full space is prohibitively expensive. The grid discretizes this space at representative points, but it may not capture the global optimum. Despite this limitation, the grid-search Oracle provides a reasonable and reproducible reference point for comparing online methods, and we report it transparently as a practical upper bound on offline optimization quality.

- ***EOCL***: Our proposed solution is implemented at two different variants based on MBO: $EOCL_s$ leverages MBO for general scenarios without applying sliding windows, $EOCL_{cd}$ applies MBO-CD tailored to address drastic environmental changes induced sudden concept drifts.

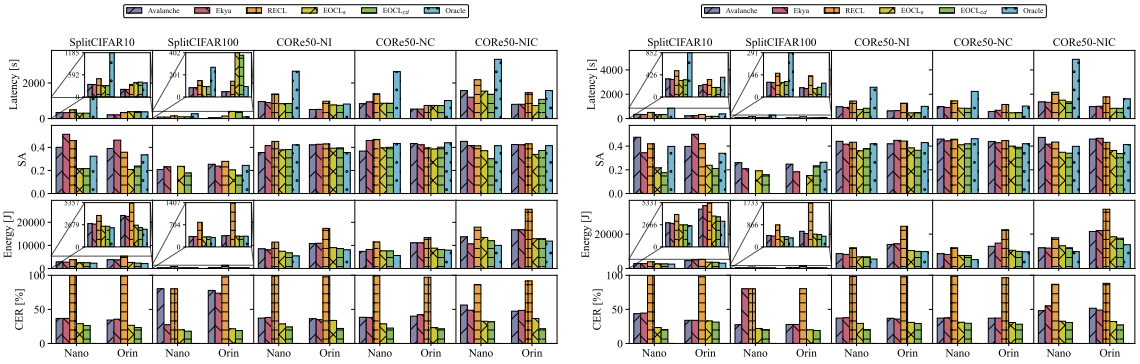
9.6.2 Overall Effectiveness

Overall Effectiveness across OCL Benchmarks. Our proposed framework consistently outperforms existing baselines like Avalanche, Ekya, and RECL by delivering significant energy savings and superior system stability across multiple benchmarks. For example, on the complex CORe50 tasks, our control-driven variant reduces energy consumption by up to 42 percent compared to RECL and 18 percent compared to Avalanche, requiring as little as 7.7 kilojoules. While we trade a moderate amount of training latency to achieve these savings, taking about 750 to 800 seconds compared to the roughly 520 seconds of the fastest baselines, we maintain competitive streaming accuracy within a few percentage points of the leaders. Most notably, our framework drastically improves reliability by achieving a Control Error Rate of just 20 to 30 percent, whereas competing methods suffer failure rates ranging from 40 to nearly 100 percent. Interestingly, while our drift-aware variant is essential for rapidly adapting to sudden environmental shifts by intentionally discarding obsolete historical data, our full-history variant can actually perform slightly better in highly stable conditions by leveraging all available past data to more accurately map the system's energy surface.

Overall effectiveness across OCL algorithms. Our framework demonstrates strong overall effectiveness across diverse continuous learning algorithms by prioritizing massive energy savings and system stability. When benchmarked against baselines like Avalanche, Ekya, and RECL on edge hardware, our method consistently slashes energy consumption. For example, on the computationally heavy GEM algorithm, our approach consumes only 7.8 kilojoules compared to RECL's 26.2 kilojoules, and similarly reduces GSS energy costs from

nearly 64 kilojoules down to 36 kilojoules. This remarkable efficiency is achieved through a deliberate, moderate trade-off in training latency, operating slightly slower than the aggressively fast baselines. However, this pace does not compromise learning quality, as our framework maintains highly competitive streaming accuracy just fractions behind the baseline leaders. Most importantly, our dynamic adaptation drastically improves system reliability, achieving a Control Error Rate of just 21 to 31 percent across most algorithms, effectively halving the 52 to 100 percent error rates suffered by competing methods. Ultimately, our system delivers fundamentally lower energy consumption and steadier control without meaningfully sacrificing model accuracy.

Sensitivity to sliding window size. The sliding window size ω=90 is a design parameter that controls the bias–variance tradeoff of the recency-weighted surrogate. A smaller window (e.g., ω=30) reacts faster to drift but increases GP modeling variance due to fewer training points, potentially causing oscillatory control decisions. A larger window (e.g., ω=180) provides a more stable surrogate but adapts more slowly to sudden shifts, retaining outdated observations that mislead the optimizer. We chose ω=90 based on preliminary experiments on CORe50-NIC, where it provided a practical balance between adaptation speed and surrogate stability. While a full ablation study across all benchmarks and drift patterns is beyond the scope of this work, we note that the performance of $EOCL_{cd}$ remained robust when ω was varied between 60 and 120 in our preliminary runs (less than 3% variation in total energy). A systematic study of adaptive window sizing could be an important direction for future work.



(a) Sudden changes. (b) Incremental changes.

Figure 9.5: Adaptivity under sudden and incremental system environmental changes.

Overall effectiveness: *EOCL* consistently optimizes energy efficiency across benchmarks and OCL algorithms on both embedded GPU platforms while maintaining competitive latency and streaming accuracy.

9.6.3 Adaptivity under System Environmental Changes

To evaluate our method's adaptability to dynamic system environments, we utilize a CPU spin-lock approach. This effectively simulates system-level changes, specifically, the fluctuating resource availability and compute contention commonly experienced in multi-tenant embedded systems.

Adaptivity under sudden system environmental changes. We evaluated how robust our framework is when the system environment changes abruptly, such as when background tasks suddenly monopolize CPU cores. Across all benchmarks, our control-driven variant proved highly reliable, achieving the lowest control error rate. For instance, it maintained error rates between 19 and 23 percent, drastically outperforming baselines that suffered error rates ranging from 34 to nearly 100 percent. This stability translates directly to significant energy savings. On the CIFAR10 benchmark, our method consumed only 2.33

kilojoules compared to the 3.69 to 5.36 kilojoules used by competing baselines, with similar proportional reductions seen on the larger CORe50 tasks. While our approach trades a slight increase in latency to achieve these savings, its streaming accuracy remains highly competitive on the complex CORe50 datasets. Ultimately, the control-driven variant is the most dependable choice for real-world edge deployments where sudden, unpredictable resource contention is common.

The noticeable accuracy drops on the smaller SplitCIFAR datasets occur for two main reasons. First, these data streams are exceptionally short, meaning every system reconfiguration has an outsized impact on training, leaving the optimizer with too few steps to adapt before the task ends. Second, our current design treats accuracy as an evaluation metric rather than a strict optimization constraint. On our medium and large-scale benchmarks, the accuracy stays within 2 percent of the leading baselines, which is an excellent trade-off for the massive energy savings we achieve. However, for safety-critical deployments where strict performance guarantees are mandatory, future work will integrate streaming accuracy as a hard constraint within the optimization loop to prevent the system from ever selecting sub-optimal configurations.

Adaptivity under incremental system environmental changes. We evaluated the system's robustness under incremental environmental stress by gradually saturating CPU cores to simulate an escalating background workload. Under these worsening conditions, our control-driven variant demonstrated unmatched stability, achieving the lowest control error rates across all hardware benchmarks. For example, it successfully suppressed error rates to just 19 percent on CIFAR100 and 27 percent on the complex CORe50-NIC task, easily

Figure 9.6: High-fidelity navigation simulation with dynamic environmental factors for On-Device OCL evaluation.

outperforming competing methods like Ekya and RECL which suffered failure rates ranging from 50 to nearly 90 percent. This superior control translates directly into leading energy efficiency, as the control-driven method required under 18 kilojoules to complete the heavy CORe50-NIC benchmark, substantially undercutting all baseline alternatives. Furthermore, this efficiency does not compromise speed; our method actually outpaced standard baselines on the larger CORe50 tasks, finishing training runs hundreds of seconds faster. While there is a slight, expected trade-off in raw streaming accuracy compared to offline models, the control-driven approach ultimately provides the most practical and dependable balance of minimal energy consumption, rapid adaptation, and rock-solid reliability as system contention steadily escalates.

Adaptivity: Across both sudden and incremental system environmental changes, *EOCL*, especially the control-drift-aware variant $EOCL_{cd}$, demonstrates consistently stronger adaptivity than alternatives. It sustains better control (lower CER), uses less energy, and maintains competitive or improved latency, making it well-suited for resource-volatile, dynamically changing deployments.

Table 9.1: Robotic case study on NVIDIA Jetson AGX Orin-driven autonomous navigational simulation. The best values of online methods are highlighted in bold.

Baseline	Energy ↓			Latency ↓			Streaming Accuracy ↑			Control Error Rate [%] ↓		
	IC	IL	WC	IC	IL	WC	IC	IL	WC	IC	IL	WC
No System Environmental Changes												
Avalanche	4228.67	15341.71	15814.81	186.00	604.00	611.00	**0.9894**	0.9955	0.9876	54.90	50.00	55.60
Ekya	4209.51	15181.55	15723.83	**184.00**	**589.00**	**605.00**	0.9836	0.9821	0.9599	57.40	49.90	56.50
RECL	5660.14	17964.44	19135.49	360.00	975.00	1046.00	0.9829	**0.9971**	0.9811	99.50	100.00	99.60
$EOCL_s$	**4076.77**	**14892.60**	15107.91	385.00	1162.00	1132.00	0.9728	0.9953	0.9561	38.70	33.90	47.20
$EOCL_{cd}$	4078.97	16444.37	**15001.89**	385.00	1143.00	1091.00	0.9728	0.9967	**0.9893**	**27.30**	**21.30**	**24.30**
Oracle	1124.23	14800.62	13643.40	263.00	1053.00	762.00	0.9720	0.9959	0.9759	0.00	0.00	0.00
Sudden System Environmental Changes												
Avalanche	4669.07	17401.68	18146.33	**186.00**	**593.00**	616.00	0.9802	0.9936	0.9654	38.30	35.39	35.66
Ekya	4668.00	17488.07	18223.18	188.00	604.00	**598.00**	0.9780	0.9960	0.9830	42.12	30.74	38.04
RECL	6928.80	21881.68	23365.95	363.00	977.00	1052.00	**0.9878**	**0.9967**	0.9810	99.88	98.78	96.83
$EOCL_s$	4458.61	17113.72	18321.01	386.00	752.00	1080.00	0.9747	0.8880	**0.9879**	43.68	51.40	44.83
$EOCL_{cd}$	**4448.70**	**16893.46**	**17900.54**	383.00	744.00	1080.00	0.9785	0.9957	0.9843	**32.63**	**29.46**	**24.34**
Oracle	3852.00	14615.55	15395.17	362.00	1167.00	1203.00	0.9784	0.9944	0.9770	0.00	0.00	0.00
Incremental System Environmental Changes												
Avalanche	3855.73	21077.23	21641.23	**139.00**	732.00	747.00	0.4887	0.9931	0.9558	34.03	36.30	38.15
Ekya	3892.70	17945.68	22209.93	141.00	608.00	783.00	0.4884	0.9960	0.9623	36.58	35.54	37.48
RECL	7429.80	29070.26	29680.15	352.00	1275.00	1291.00	0.9865	**0.9969**	**0.9865**	97.76	98.64	99.23
$EOCL_s$	3951.39	17261.38	17956.52	187.00	571.00	589.00	0.9758	0.9851	0.9839	44.21	40.05	49.57
$EOCL_{cd}$	**3838.03**	**17163.15**	**17582.67**	182.00	**564.00**	**573.00**	0.9726	0.9933	0.9860	**31.53**	**30.32**	**30.05**
Oracle	3770.22	15261.25	16462.73	360.00	1209.00	1276.00	0.9756	0.9951	0.9484	0.00	0.00	0.00

9.6.4 Pratical Case Studies

We evaluate our framework's robustness against system-level resource contention and natural data distribution shifts using an autonomous navigation stack on NVIDIA Jetson hardware. Employing the Endless-Sim environment, we simulated practical challenges, such as changing weather, evolving lighting, and novel objects, while injecting background CPU contention either suddenly or incrementally to mimic multi-tenant edge environments.

Across all operating regimes, our framework consistently delivers the most energy-efficient performance. The control-driven variant specifically excels in unstable environments, achieving the lowest Control Error Rate among all tested methods. For instance, under both sudden and incremental resource shocks, it maintains an error rate of roughly 24 to 32 percent, demonstrating significantly faster recovery from system disturbances than competing baselines. While it occasionally trades a marginal amount of training latency to maintain

this stability, it preserves near-perfect streaming accuracy, peaking at over 99 percent in illumination-shifting scenarios. Furthermore, during gradual environmental changes, it not only minimizes energy and control errors but also achieves leading training speeds, completing complex weather and lighting adaptations in under 10 minutes. Ultimately, our control-driven design sustains online learning with remarkably lower energy consumption and superior system stability as real-world environments evolve.

Pracality: *EOCL* maintains competitive streaming accuracy and end-to-end latency while significantly optimizing energy consumption, showing its readiness for On-Device embedded continual learning. This underscores *EOCL*'s effectiveness in addressing practical constraints in real-world embedded continual learning deployments.

9.6.5 Overhead Analysis

As demonstrated in Tab. 9.2, *EOCL* maintains a small GP surrogate (kernel parameters, Cholesky factor) and a bounded buffer of recent evaluations. For $EOCL_{cd}$, the history is a fixed sliding window; $EOCL_s$ uses a full-history surrogate in our deployment, so it requires more memory and latency overhead. To reduce MBO's overhead, we execute controller runs out-of-band every Δ=10 s and apply DVFS and batch updates asynchronously at iteration boundaries, so the OCL training and inference pipeline are not blocked. Both variants have acceptable memory overhead (less than 6.01%) and latency overhead (less than 4.50%). Because ω is fixed, the optimizer state is bounded; with $\omega = 90$, GP fitting and acquisition maximization remain lightweight enough to execute asynchronously every $\Delta = 10$ s without stalling the OCL loop.

Table 9.2: On-Device overhead of two variants of *EOCL* on Jetson AGX Orin and Jetson Nano Orin. Memory refers to additional resident memory relative to an Avalanche baseline. M and L refer to memory and latency, respectively.

	Jetson AGX Orin		Jetson Nano Orin	
Variant	**M [MB/%]**	**L [s/%]**	**M [MB/%]**	**L [s/%]**
$EOCL_s$	492 / 1.57	18.6 / 4.22	492 / 6.01	23.3 / 4.50
$EOCL_{cd}$	484 / 1.55	16.4 / 4.03	484 / 5.91	18.0 / 4.00

Low overhead: The added memory and latency from MBO are minimal and consistently outweighed by the energy savings and stability benefits reported in Sec. 9.6.3 and Sec. 4.4.2.

9.6.6 Discussion and Limitations

While our framework advances energy-aware OCL on embedded systems, it operates within specific boundaries. First, NLP tasks, massive datasets (e.g., ImageNet [78], CLOC [190]), and infinite streams [376, 395] currently exceed typical embedded capacities and are not addressed. Additionally, consistent with prior systems research [52, 294, 234, 28], we assume supervised learning, leaving unsupervised OCL to the broader machine learning community. Architecturally, to strictly minimize runtime overhead on edge devices, we intentionally omit an explicit concept drift detector. Instead, the system implicitly handles environmental non-stationarity via a recency-weighted sliding-window surrogate and a fixed, asynchronous control interval. This lightweight, deployability-first design safely absorbs short-term burst loads without ever stalling the main training pipeline, providing actionable, low-overhead mechanisms for safe concurrency and energy efficiency. Integrating explicit drift detectors, adaptive window sizing, and dynamic control intervals remains a key direction for future work.

9.7 Related Work

9.7.1 Online Continual Learning and System Optimization

Online continual learning (OCL) updates models from non-stationary streams while mitigating catastrophic forgetting [52, 294, 234, 51, 10, 194, 222, 347, 348, 48, 333, 301]. To achieve this under computational budgets, various algorithmic paradigms have been proposed, including replay-based methods like ER [52, 294] and GSS [10], as well as gradient-constrained approaches like GEM [234] and AGEM [51]. Our work is orthogonal and transparent to these algorithmic OCL paradigms; rather than modifying the learning algorithms themselves, we target the *system* dimension, specifically, how training and inference are co-scheduled and batched On-Device, to boost throughput while preserving streaming accuracy.

Despite extensive algorithmic progress, system-level OCL optimizations remain underexplored, with prior work largely targeting different scopes or hardware. For example, LifeLearner [202] focuses on hardware optimizations for meta-continual learning. Meanwhile, Ekya [28] and RECL [188] target video analytics on powerful multi-GPU servers, employing dynamic profiling for offline methods like iCaRL [301] or complex model-selection safety checkers, respectively. For consistency and practical deployment, we implement our concurrent execution pipeline atop the extensible Avalanche framework [46], as done in Ekya and RECL. To our knowledge, our work is the first to propose energy-aware, On-Device online continual learning, bridging the gap between algorithmic efficiency and system-level practicality in resource-constrained environments.

9.7.2 DVFS and Energy Management on Embedded Platforms

Dynamic Voltage and Frequency Scaling (DVFS) is a cornerstone of embedded energy management, spanning real-time scheduling [373, 16], heterogeneous architectures [75, 97, 35, 341], and edge neural network inference [259]. Our work diverges from these efforts in two key ways. First, we target the dynamically shifting computational profiles of OCL rather than static inference. Second, because complex SoC power dynamics (e.g., thermal coupling and memory contention) complicate analytical modeling, we employ a data-driven black-box surrogate. This ensures platform-agnostic adaptivity without exhaustive hardware profiling, though integrating analytical priors to accelerate convergence remains a promising direction for future work.

9.7.3 Model-based Optimization

Model-based optimization (MBO), often realized as Bayesian optimization (BO), is a sample-efficient framework for black-box problems utilizing probabilistic surrogates like Gaussian Processes [182, 328]. Toolchains like BoTorch [15] enable extensions to time-varying settings via sliding windows or periodic restarts with modest overhead [274, 408, 306]. While BO is widely successful across diverse domains, including materials [400, 172], chemistry [320], and server-class ML hyperparameter/system optimization [100, 284, 285], these applications largely focus on static and pre-deployment settings. In contrast, our work applies MBO to embedded, On-Device OCL deployment to dynamically minimize run-time energy under strict hardware constraints.

9.8 Conclusion

We presented *EOCL*, a deployability-first runtime that enables energy-aware evaluation and retraining for On-Device OCL. By coupling a recency-aware model-based optimizer with lightweight runtime reconfiguration of DVFS and batch size, *EOCL* adapts to non-stationary conditions while preserving streaming accuracy and keeping latency overhead low. Our concurrent pipeline atop Avalanche provides fine-grained control and unified instrumentation, and we introduced the CER to quantify stability under drift. Experiments across Jetson-class platforms, multiple OCL algorithms, and diverse workloads demonstrate consistent energy reductions and steady control under both sudden and gradual environmental changes. Extending *EOCL* with drift detectors, broader control knobs, adaptive control intervals that dynamically adjust reconfiguration frequency based on detected drift intensity, along with multi-objective scheduling under explicit latency/accuracy guarantees, remains promising future work.

Chapter 10

Conclusions

As safety-critical cyber-physical systems (CPS) increasingly rely on deep neural networks for perception, decision-making, and closed-loop control, the limitations of static, offline-trained models have become starkly apparent. Real deployments are non-stationary: data distributions drift, workload mixes shift, and energy supplies fluctuate over long missions. While deep reinforcement learning and online continual learning offer algorithmic pathways for models to adapt on the fly, this dissertation has demonstrated that algorithmic adaptation alone is insufficient. Deploying online learning on embedded edge devices presents a distinct systems problem: models must be updated concurrently with real-time inference, while strictly respecting end-to-end timing, memory, and energy budgets on heterogeneous, non-preemptive accelerators. Existing machine-learning pipelines that overlook these constraints produce unpredictable latencies, frequent out-of-memory failures, and energy-oblivious behavior that cannot sustain long-duration autonomy.

This dissertation addresses these challenges by developing a predictable and self-adaptive On-Device machine-learning stack, organized as three tightly coupled thrusts that span model structure, runtime scheduling, and deployment-time resource management.

Fitting modern models On-Device under stringent memory budgets. The first thrust makes modern multi-task and learning-enabled workloads runnable on small embedded platforms, both for inference and for On-Device updates. **MIMONet** (Chapter 2) restructures multi-task perception into a single weight-shared multi-input multi-output network, extending the variational information bottleneck to residual architectures and adapting cross-model weight zipping to remove inter-task redundancy, reducing runtime memory by up to 80.7% on Jetson-class devices. **MixTraining** (Chapter 3) consolidates self-supervised and supervised passes through a shared backbone via a unified mixtraining phase, Pareto-dominating SSL+SL in both accuracy and training compute. $\mathbf{R^3}$ (Chapter 4) co-optimizes batch size and replay-buffer memory under a deadline-driven feedback loop, delivering timing-predictable On-Device deep reinforcement learning across Classic Control, Atari, and a DonkeyCar simulator. **Orion** (Chapter 5) generalizes this memory-aware perspective to online continual learning, using a unified runtime health score to autonomously rebalance batch execution, replay storage, and optimization plugins, eliminating out-of-memory failures while preserving plasticity and stability.

Reducing latency and deadline misses in real-time learning. The second thrust turns memory-efficient models into deployable real-time systems on shared, non-preemptive embedded accelerators. **RED** (Chapter 6) is the first real-time scheduler to support weight-shared multi-task DNNs, intermediate-deadline assignment, and on-demand

synchronization across asynchronous perception–control DAGs, reducing deadline-miss rates by 40.5% and response times by 24.7% on average over EDF across four Jetson platforms while integrating cleanly with ROS 2. **AOCL** (Chapter 7) extends real-time scheduling to single-GPU On-Device OCL, where a Unified Adaptation Metric drives a finite-state, sign-based controller that safely overlaps training, evaluation, and inference, accompanied by analytical guarantees on per-axis settling time, steady-state inference feasibility, and a transient deadline-miss bound during safety fallback.

Deployable On-Device learning ecosystems in uncertain, resource-constrained environments. The third thrust closes the loop with the environment and the energy supply. **Genie** (Chapter 8) introduces a distributed ROS-based caching layer that maintains a shared 3D object map across edge servers, cutting perception latency by up to 95% (82% on average, with 31%–67% reusable objects) for Autoware-based connected autonomous robots on Jetson TX2 and AGX Xavier nodes. **EOCL** (Chapter 9) makes On-Device OCL itself energy- and environment-aware, formalizing deployment as a constrained online optimization problem and introducing a recency-aware Gaussian-process surrogate with Expected Improvement that jointly tunes CPU/GPU DVFS and batch size, reducing energy by up to 26% versus the strongest online schedulers and by up to 70% versus RECL while maintaining streaming accuracy within roughly two percentage points under sudden environmental change.

Collectively, this body of work establishes that predictable, energy-aware, and self-adaptive On-Device learning is a distinct systems problem requiring cross-layer coordination among model structure, runtime scheduling, and resource management rather than isolated

algorithmic adjustments. By co-designing multi-task model architectures with memory- and deadline-aware schedulers and by closing the loop with environment- and energy-aware controllers, the stack presented here sustains real-time inference and online adaptation within tight memory and energy budgets across diverse embedded platforms and changing environments.

BIBLIOGRAPHY

[1] Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving, howpublished = https://github.com/apolloauto/apollo, note = Accessed: 2019-02-11.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[4] MICHAEL Aeberhard. Automated driving with ros at bmw.

[5] Mahbod Afarin, Chao Gao, Shafiur Rahman, Nael Abu-Ghazaleh, and Rajiv Gupta. Commongraph: Graph analytics on evolving data. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 133–145, 2023.

[6] Marco Aggravi, Ahmed Alaaeldin Said Elsherif, Paolo Robuffo Giordano, and Claudio Pacchierotti. Haptic-enabled decentralized control of a heterogeneous human-robot team for search and rescue in partially-known environments. *IEEE Robotics and Automation Letters*, 6(3):4843–4850, 2021.

[7] Ankit Agrawal, Renato Mancuso, Rodolfo Pellizzoni, and Gerhard Fohler. Analysis of dynamic memory bandwidth regulation in multi-core real-time systems. *CoRR*, abs/1809.05921, 2018.

[8] Shareef Ahmed and James H Anderson. Exact response-time bounds of periodic dag tasks under server-based global scheduling. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 447–459. IEEE, 2022.

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

[10] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

[11] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. Cloudcache: On-demand flash cache management for cloud computing. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies*, FAST'16, pages 355–369, Berkeley, CA, USA, 2016. USENIX Association.

[12] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[13] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 472–487, 2022.

[14] Ali Ayub and Carter Fendley. Few-shot continual active learning by a robot. In *NeurIPS*, 2022.

[15] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

[16] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):1–34, 2016.

[17] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[18] Sanjoy Baruah. The federated scheduling of systems of conditional sporadic dag tasks. In *2015 International Conference on Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2015.

[19] S. Bateni and C. Liu. Apnet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 67–79, Dec 2018.

[20] Soroush Bateni and Cong Liu. Apnet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 67–79. IEEE, 2018.

[21] Soroush Bateni and Cong Liu. Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 371–385, 2020.

[22] Soroush Bateni, Husheng Zhou, Yuankun Zhu, and Cong Liu. Predjoule: A timing-predictable energy optimization framework for deep neural networks. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 107–118. IEEE, 2018.

[23] George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.

[24] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[25] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[26] Javad Zolfaghari Bengar, Joost van de Weijer, Bartlomiej Twardowski, and Bogdan Raducanu. Reducing label effort: Self-supervised meets active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1631–1639, 2021.

[27] David Berthelot, Rebecca Roelofs, Kihyuk Sohn, Nicholas Carlini, and Alex Kurakin. Adamatch: A unified approach to semi-supervised learning and domain adaptation. *arXiv preprint arXiv:2106.04732*, 2021.

[28] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 119–135, 2022.

[29] Gantavya Bhatt, Yifang Chen, Arnav M Das, Jifan Zhang, Sang T Truong, Stephen Mussmann, Yinglun Zhu, Jeffrey Bilmes, Simon S Du, Kevin Jamieson, et al. An experimental design framework for label-efficient supervised finetuning of large language models. *arXiv preprint arXiv:2401.06692*, 2024.

[30] Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. Energy-efficient parallel real-time scheduling on clustered multi-core. *IEEE Trans. Parallel Distributed Syst.*, 31(9):2097–2111, 2020.

[31] Ran Bi, Qingqiang He, Jinghao Sun, Zhenyu Sun, Zhishan Guo, Nan Guan, and Guozhen Tan. Response time analysis for prioritized dag task with mutually exclusive vertices. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 460–473. IEEE, 2022.

[32] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[33] Tobias Blaß, Daniel Casini, Sergey Bozhko, and Björn B. Brandenburg. A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance. In *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*, pages 41–53. IEEE, 2021.

[34] Tobias Blaß, Arne Hamann, Ralph Lange, Dirk Ziegenbein, and Björn B. Brandenburg. Automatic latency management for ROS 2: Benefits, challenges, and open problems. In *27th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2021, Nashville, TN, USA, May 18-21, 2021*, pages 264–277. IEEE, 2021.

[35] Paul Bogdan. Mathematical modeling and control of multifractal workloads for data-center-on-a-chip optimization. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, pages 1–8, 2015.

[36] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[37] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of ACM International Conference on Machine Learning*, pages 527–536, New York, NY, USA, 2017. ACM.

[38] Karim Botros, Mohammad Alkhatib, David Folio, and Antoine Ferreira. Fully automatic and real-time microrobot detection and tracking based on ultrasound imaging using deep learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9763–9768. IEEE, 2022.

[39] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[40] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[41] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[42] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. Legup: high-level synthesis for fpga-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 33–36, 2011.

[43] Louis-Claude Canon, Emmanuel Jeannot, Rizos Sakellariou, and Wei Zheng. Comparative evaluation of the robustness of dag scheduling heuristics. *Grid Computing: Achievements and Prospects*, pages 73–84, 2008.

[44] Donkey Car. Donkey docs. https://docs.donkeycar.com/guide/build_hardware, 2023.

[45] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*. Springer, 2020.

[46] Antonio Carta, Lorenzo Pellegrini, Andrea Cossu, Hamed Hemati, and Vincenzo Lomonaco. Avalanche: A pytorch library for deep continual learning. *Journal of Machine Learning Research*, 24(363):1–6, 2023.

[47] Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.

[48] Luca Castri, Sariah Mghames, and Nicola Bellotto. From continual learning to causal discovery in robotics. In *AAAI Bridge Program on Continual Causality*, pages 85–91. PMLR, 2023.

[49] Devendra Singh Chaplot, Lisa Lee, Ruslan Salakhutdinov, Devi Parikh, and Dhruv Batra. Embodied multimodal multitask learning. *arXiv preprint arXiv:1902.01385*, 2019.

[50] Athanasios Charisoudis, Simon Ekman von Huth, and Emil Jansson. [re] masked autoencoders are small scale vision learners: A reproduction under resource constraints. In *ML Reproducibility Challenge 2022*, 2023.

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

[52] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, P Dokania, P Torr, and M Ranzato. Continual learning with tiny episodic memories. In *Workshop on Multi-Task and Lifelong Reinforcement Learning*, 2019.

[53] Jagmohan Chauhan, Young D Kwon, Pan Hui, and Cecilia Mascolo. Contauth: Continual learning framework for behavioral-based user authentication. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(4):1–23, 2020.

[54] Kaiyuan Eric Chen, Yafei Liang, Nikhil Jha, Jeffrey Ichnowski, Michael Danielczuk, Joseph Gonzalez, John Kubiatowicz, and Ken Goldberg. Fogros: An adaptive framework for automating fog robotics deployment. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 2035–2042. IEEE, 2021.

[55] Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, et al. Wavlm: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1505–1518, 2022.

[56] Simin Chen, Mirazul Haque, Cong Liu, and Wei Yang. Deepperform: An efficient approach for performance testing of resource-constrained neural networks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

[57] Simin Chen, Hamed Khanpour, Cong Liu, and Wei Yang. Learning to reverse dnns from ai programs automatically. *arXiv preprint arXiv:2205.10364*, 2022.

[58] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15365–15374, 2022.

[59] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. Tvm: An automated end-to-end optimizing compiler for deep learning. *arXiv preprint arXiv:1802.04799*, 2018.

[60] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[61] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[62] Yiming Chen, Simin Chen, Zexin Li, Wei Yang, Cong Liu, Robby T Tan, and Haizhou Li. Dynamic transformers provide a false sense of efficiency. *arXiv preprint arXiv:2305.12228*, 2023.

[63] Yiming Chen, Simin Chen, Zexin Li, Wei Yang, Cong Liu, Robby T. Tan, and Haizhou Li. Dynamic transformers provide a false sense of efficiency. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 7164–7180. Association for Computational Linguistics, 2023.

[64] Yiming Chen, Yan Zhang, Bin Wang, Zuozhu Liu, and Haizhou Li. Generate, discriminate and contrast: A semi-supervised sentence representation learning framework. *arXiv preprint arXiv:2210.16798*, 2022.

[65] Yiming Chen, Yan Zhang, Chen Zhang, Grandee Lee, Ran Cheng, and Haizhou Li. Revisiting self-training for few-shot learning of language model. *arXiv preprint arXiv:2110.01256*, 2021.

[66] Hyunjong Choi, Yecheng Xiang, and Hyoseung Kim. Picas: New design of priority-driven chain-aware scheduling for ROS2. In *27th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2021, Nashville, TN, USA, May 18-21, 2021*, pages 251–263. IEEE, 2021.

[67] Yi-Min Chou, Yi-Ming Chan, Jia-Hong Lee, Chih-Yi Chiu, and Chu-Song Chen. Unifying and merging well-trained deep neural networks for inference stage. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 2049–2056, Burlington, MA, USA, 2018. Morgan Kaufmann.

[68] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

[69] Adrian Cottam, Xiaofeng Li, Xiaobo Ma, and Yao-Jan Wu. Large-scale freeway traffic flow estimation using crowdsourced data: A case study in arizona. *Journal of Transportation Engineering, Part A: Systems*, 150(7):04024030, 2024.

[70] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[71] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 613–627, 2017.

[72] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[73] Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 4. ACM, 2016.

[74] Bin Dai, Chen Zhu, Baining Guo, and David Wipf. Compressing neural networks using the variational information bottleneck. In *International Conference on Machine Learning*, pages 1135–1144. PMLR, 2018.

[75] Radu David, Paul Bogdan, Radu Marculescu, and Umit Ogras. Dynamic power management of voltage-frequency island partitioned networks-on-chip using intel sing-chip cloud computer. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, pages 257–258, 2011.

[76] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[77] Aleksandr Dekhovich, David MJ Tax, Marcel HF Sluiter, and Miguel A Bessa. Continual prune-and-select: class-incremental learning with specialized subnetworks. *Applied Intelligence*, 53(14):17849–17864, 2023.

[78] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[79] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[80] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: a comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[81] Xinke Deng, Yu Xiang, Arsalan Mousavian, Clemens Eppner, Timothy Bretl, and Dieter Fox. Self-supervised 6d object pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3665–3671. IEEE, 2020.

[82] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[83] Digitaldreamlabs. Vector 2.0 ai robot companion. https://www.digitaldreamlabs.com/products/vector-robot, 2022.

[84] Nikolaos Dimitriadis, Francois Fleuret, and Pascal Frossard. Sequel: A continual learning library in pytorch and jax. *arXiv preprint arXiv:2304.10857*, 2023.

[85] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature machine intelligence*, 5(3):220–235, 2023.

[86] Anuj Diwan, Ching-Feng Yeh, Wei-Ning Hsu, Paden Tomasello, Eunsol Choi, David Harwath, and Abdelrahman Mohamed. Continual learning for on-device speech recognition using disentangled conformers. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[87] Guimin Dong. *DEEP GRAPH LEARNING IN MOBILE HEALTH*. PhD thesis, University of Virginia, 2022.

[88] Guimin Dong, Yonghyeon Kweon, B Brian Park, and Mehdi Boukhechba. Utility-based route choice behavior modeling using deep sequential models. *Journal of big data analytics in transportation*, 4(2):119–133, 2022.

[89] Guimin Dong, Mingyue Tang, Zhiyuan Wang, Jiechao Gao, Sikun Guo, Lihua Cai, Robert Gutierrez, Bradford Campbel, Laura E Barnes, and Mehdi Boukhechba. Graph neural networks in iot: A survey. *ACM Transactions on Sensor Networks*, 19(2):1–50, 2023.

[90] Guimin Dong, Mingyue Tang, Runze Yan, Zeyu Mu, Lihua Cai, and B Brian Park. Deep learning for autonomous vehicles and systems. In *Autonomous Vehicles and Systems*, pages 9–47. River Publishers, 2023.

[91] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017.

[92] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[93] Arthur Douillard and Timothée Lesort. Continuum: Simple management of complex continual learning scenarios. *arXiv preprint arXiv:2102.06253*, 2021.

[94] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan. Cachier: Edge-caching for recognition applications. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 276–286, June 2017.

[95] Utsav Drolia, Katherine Guo, and Priya Narasimhan. Precog: prefetching for image recognition applications at the edge. In *SEC*, 2017.

[96] Xianzhi Du, Barret Zoph, Wei-Chih Hung, and Tsung-Yi Lin. Simple training strategies and model scaling for object detection. *arXiv preprint arXiv:2107.00057*, 2021.

[97] Shukai Duan, Heng Ping, Nikos Kanakaris, Xiongye Xiao, Panagiotis Kyriakis, Nesreen K Ahmed, Peiyu Zhang, Guixiang Ma, Mihai Capotă, Shahin Nazarian, et al. A structure-aware framework for learning device placements on computation graphs. *Advances in Neural Information Processing Systems*, 37:81748–81772, 2024.

[98] 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*, pages 33–44, Dec 2013.

[99] Glenn A. Elliott, Bryan C. Ward, and James H. Anderson. GPUSync: A framework for real-time GPU management. In *2013 IEEE 34th Real-Time Systems Symposium (RTSS)*, pages 33–44. IEEE, 2013.

[100] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1436–1445. PMLR, 2018.

[101] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of ACM Annual International Conference on Mobile Computing and Networking*, pages 115–127, 2018.

[102] Francesca Favarò, Sky Eurich, and Nazanin Nader. Autonomous vehicles' disengagements: Trends, triggers, and regulatory limitations. *Accident Analysis & Prevention*, 110:136–148, 2018.

[103] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pages 3061–3071. PMLR, 2020.

[104] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[105] Mengyin Fu, Wenjie Song, Yang Yi, and Meiling Wang. Path planning and decision making for autonomous vehicle in urban environment. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 686–692. IEEE, 2015.

[106] Chao Gao, Mahbod Afarin, Shafiur Rahman, Nael Abu-Ghazaleh, and Rajiv Gupta. Mega evolving graph accelerator. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023.

[107] Chao Gao and Sai Qian Zhang. Dlora: Distributed parameter-efficient fine-tuning solution for large language model. *arXiv preprint arXiv:2404.05182*, 2024.

[108] Longsen Gao, Kevin Aubert, David Saldana, Claus Danielson, and Rafael Fierro. Decentralized adaptive aerospace transportation of unknown loads using a team of robots. *arXiv preprint arXiv:2407.08084*, 2024.

[109] Longsen Gao, Giovanni Cordova, Claus Danielson, and Rafael Fierro. Autonomous multi-robot servicing for spacecraft operation extension. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10729–10735. IEEE, 2023.

[110] Longsen Gao, Claus Danielson, and Rafael Fierro. Adaptive robot detumbling of a non-rigid satellite. *arXiv preprint arXiv:2407.17617*, 2024.

[111] Xiaoxue Gao, Chitralekha Gupta, and Haizhou Li. Automatic lyrics transcription of polyphonic music with lyrics-chord multi-task learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:2280–2294, 2022.

[112] Xiaoxue Gao, Chitralekha Gupta, and Haizhou Li. Polyscriber: Integrated fine-tuning of extractor and lyrics transcriber for polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.

[113] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[114] Andreas Geiger, P Lenz, Christoph Stiller, and Raquel Urtasun. The kitti vision benchmark suite. *URL http://www. cvlibs. net/datasets/kitti*, 2015.

[115] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[116] Binzong Geng, Fajie Yuan, Qiancheng Xu, Ying Shen, Ruifeng Xu, and Min Yang. Continual learning for task-oriented dialogue system with iterative network pruning, expanding and masking. *arXiv preprint arXiv:2107.08173*, 2021.

[117] Yasir Ghunaim, Adel Bibi, Kumail Alhamoud, Motasem Alfarra, Hasan Abed Al Kader Hammoud, Ameya Prabhu, Philip HS Torr, and Bernard Ghanem. Real-time evaluation in online continual learning: A new hope. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11888–11897, 2023.

[118] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Joseph E Gonzalez, and Ion Stoica. D3: a dynamic deadline-driven approach for building autonomous vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 453–471, 2022.

[119] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Joseph E Gonzalez, and Ion Stoica. D3: a dynamic deadline-driven approach for building autonomous vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 453–471, 2022.

[120] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Matthew A Wright, Joseph E Gonzalez, and Ion Stoica. Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8806–8813. IEEE, 2021.

[121] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[122] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[123] Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

[124] Philipp M Grulich and Faisal Nawab. Collaborative edge and cloud neural networks for real-time video processing. *Proceedings of the VLDB Endowment*, 11(12):2046–2049, 2018.

[125] Junfeng Guo, Ang Li, and Cong Liu. Backdoor detection and mitigation in competitive reinforcement learning. *arXiv preprint arXiv:2202.03609*, 2022.

[126] Junfeng Guo, Ang Li, and Cong Liu. Backdoor detection and mitigation in competitive reinforcement learning, 2023.

[127] Peizhen Guo, Bo Hu, and Wenjun Hu. Mistify: Automating {DNN} model porting for {On-Device} inference at the edge. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 705–719, 2021.

[128] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. Foggycache: Cross-device approximate computation reuse. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 19–34. ACM, 2018.

[129] Peizhen Guo and Wenjun Hu. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18. ACM, 2018.

[130] RUCHI GUPTA. Alphabet's waymo is planning massive expansion of taxi service, Jun 2019.

[131] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[132] John K Haas. A history of the unity game engine. *Diss. Worcester Polytechnic Institute*, 483(2014):484, 2014.

[133] Elvin Hajizada, Balachandran Swaminathan, and Yulia Sandamirskaya. Continual learning for autonomous robots: A prototype-based approach. *arXiv preprint arXiv:2404.00418*, 2024.

[134] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.

[135] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 539–558, 2022.

[136] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16. ACM, 2016.

[137] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[138] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[139] Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

[140] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 164–171, 1993.

[141] Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M Dai, and Dustin Tran. Training independent subnetworks for robust prediction. *arXiv preprint arXiv:2010.06610*, 2020.

[142] Andrew J. Hawkins. Uber's self-driving cars return to public roads for the first time since fatal crash, Dec 2018.

[143] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[144] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[145] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[146] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[147] Qingqiang He, Nan Guan, Mingsong Lv, Xu Jiang, and Wanli Chang. Bounding the response time of dag tasks using long paths. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 474–486. IEEE, 2022.

[148] Sihong He, Shuo Han, and Fei Miao. Robust electric vehicle balancing of autonomous mobility-on-demand system: A multi-agent reinforcement learning approach. *arXiv preprint arXiv:2307.16228*, 2023.

[149] Sihong He, Songyang Han, Sanbao Su, Shuo Han, Shaofeng Zou, and Fei Miao. Robust multi-agent reinforcement learning with state uncertainty. *Transactions on Machine Learning Research*, 2023.

[150] Sihong He, Yue Wang, Shuo Han, Shaofeng Zou, and Fei Miao. A robust and constrained multi-agent reinforcement learning framework for electric vehicle amod systems. *arXiv preprint arXiv:2209.08230*, 2022.

[151] Tianxing He, Jun Liu, Kyunghyun Cho, Myle Ott, Bing Liu, James Glass, and Fuchun Peng. Analyzing the forgetting problem in pretrain-finetuning of open-domain dialogue response models. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1121–1133, 2021.

[152] Xiaoxi He. *Towards On-Device Intelligence*. PhD thesis, ETH Zurich, 2022.

[153] Xiaoxi He, Dawei Gao, Zimu Zhou, Yongxin Tong, and Lothar Thiele. Pruning-aware merging for efficient multitask inference. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 585–595, New York, NY, USA, 2021. ACM.

[154] Xiaoxi He, Xu Wang, Zimu Zhou, Jiahang Wu, Zheng Yang, and Lothar Thiele. On-device deep multi-task inference via multi-task zipping. *IEEE Transactions on Mobile Computing*, 2023.

[155] Xiaoxi He, Xu Wang, Zimu Zhou, Jiahang Wu, Zheng Yang, and Lothar Thiele. On-device deep multi-task inference via multi-task zipping. *IEEE Transactions on Mobile Computing*, pages 1–1, 2023.

[156] Xiaoxi He, Zimu Zhou, and Lothar Thiele. Multi-task zipping via layer-wise neuron sharing. In *Advances in Neural Information Processing Systems*, pages 6019–6029, 2018.

[157] Xiaoxi He, Zimu Zhou, and Lothar Thiele. Multi-task zipping via layer-wise neuron sharing. In *Advances in Neural Information Processing Systems*, pages 6019–6029, 2018.

[158] Larkin Heintzman, Amanda Hashimoto, Nicole Abaid, and Ryan K Williams. Anticipatory planning and dynamic lost person models for human-robot search and rescue. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8252–8258. IEEE, 2021.

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

[160] Timm Hess, Martin Mundt, Iuliia Pliushch, and Visvanathan Ramesh. A procedural world generation framework for systematic evaluation of continual learning. *arXiv preprint arXiv:2106.02585*, 2021.

[161] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[162] Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems*, 6, 1993.

[163] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[164] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM transactions on audio, speech, and language processing*, 29:3451–3460, 2021.

[165] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[166] Ronghang Hu and Amanpreet Singh. Unit: Multimodal multitask learning with a unified transformer. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 1419–1429. IEEE, 2021.

[167] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[168] Zhijian Huang, Sihao Lin, Guiyu Liu, Mukun Luo, Chaoqiang Ye, Hang Xu, Xiaojun Chang, and Xiaodan Liang. Fuller: Unified multi-modality multi-task 3d perception via multi-level gradient calibration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3502–3511, 2023.

[169] Jeffrey Ichnowski, Kaiyuan Chen, Karthik Dharmarajan, Simeon Adebola, Michael Danielczuk, Vıctor Mayoral-Vilches, Hugo Zhan, Derek Xu, Ramtin Ghassemi, John Kubiatowicz, et al. Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2. In *Proceedings IEEE International Conference on Robotics and Automation*, 2023.

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

[171] NVIDIA AI IOT. Ros2 real-time classification and detection. https://github.com/NVIDIA-AI-IOT/ros2_torch_trt, 2020.

[172] Akimitsu Ishii, Shinjiro Kikuchi, Akinori Yamanaka, and Akiyasu Yamamoto. Application of bayesian optimization to the synthesis process of bafe2 (as, p) 2 polycrystalline bulk superconducting materials. *Journal of Alloys and Compounds*, 966:171613, 2023.

[173] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[174] Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. Band: coordinated multi-dnn inference on heterogeneous mobile processors. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 235–247, 2022.

[175] Mingoo Ji, Saehanseul Yi, Changjin Koo, Sol Ahn, Dongjoo Seo, Nikil D. Dutt, and Jong-Chan Kim. Demand layering for real-time DNN inference with minimized memory usage. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*, pages 291–304. IEEE, 2022.

[176] Jingyan Jiang, Ziyue Luo, Chenghao Hu, Zhaoliang He, Zhi Wang, Shutao Xia, and Chuan Wu. Joint model and data adaptation for cloud inference serving. In *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*, pages 279–289. IEEE, 2021.

[177] Xu Jiang, Nan Guan, Xiang Long, Yue Tang, and Qingqiang He. Real-time scheduling of parallel tasks with tight deadlines. *Journal of Systems Architecture*, 108:101742, 2020.

[178] Xu Jiang, Nan Guan, Xiang Long, and Wang Yi. Semi-federated scheduling of parallel real-time tasks on multiprocessors. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 80–91. IEEE, 2017.

[179] Xu Jiang, Dong Ji, Nan Guan, Ruoxiang Li, Yue Tang, and Yi Wang. Real-time scheduling and analysis of processing chains on multi-threaded executor in ROS 2. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*, pages 27–39. IEEE, 2022.

[180] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479, 2020.

[181] Zhong-Ping Jiang and Yuan Wang. Input-to-state stability for discrete-time nonlinear systems. *Automatica*, 37(6):857–869, 2001.

[182] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[183] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[184] Woosung Kang, Kilho Lee, Jinkyu Lee, Insik Shin, and Hoon Sung Chwa. Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 329–341. IEEE, 2021.

[185] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[186] Vedant Karia, Abdullah Zyarah, and Dhireesha Kudithipudi. Positcl: Compact continual learning with posit aware quantization. In *Proceedings of the Great Lakes Symposium on VLSI 2024*, pages 645–650, 2024.

[187] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296. IEEE, 2018.

[188] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. {RECL}: Responsive {Resource-Efficient} continuous learning for video analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 917–932, 2023.

[189] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.

[190] Joohyung Kim, Janghun Hyeon, Hyunga Choi, Bumchul Jang, Bokyeon Jeong, and Nakju Doh. Cloc: Confident initial estimation of long-term visual localization using a few sequential images in large-scale spaces. *IEEE Sensors Journal*, 23(8):8613–8629, 2023.

[191] Youngjun Kim, Taewan Kim, Suhyun Kim, Seongjae Lee, and Taehyoun Kim. Design and implementation of a lightweight on-device ai-based real-time fault diagnosis system using continual learning. *IEMEK Journal of Embedded Systems and Applications*, 19(3):151–158, 2024.

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

[193] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

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

[195] Branislav Kisačanin. Deep learning for autonomous vehicles. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 142–142. IEEE, 2017.

[196] Zelun Kong, Junfeng Guo, Ang Li, and Cong Liu. Physgan: Generating physical-world-resilient adversarial examples for autonomous driving, 2019.

[197] Tawn Kramer. Openai gym environments for donkey car, 2023.

[198] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[199] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[200] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. *arXiv preprint arXiv:2202.10054*, 2022.

[201] Hyoukjun Kwon, Krishnakumar Nair, Jamin Seo, Jason Yik, Debabrata Mohapatra, Dongyuan Zhan, Jinook Song, Peter Capak, Peizhao Zhang, Peter Vajda, et al. Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse. *arXiv preprint arXiv:2211.08675*, 2022.

[202] Young D Kwon, Jagmohan Chauhan, Hong Jia, Stylianos I Venieris, and Cecilia Mascolo. Lifelearner: Hardware-aware meta continual learning system for embedded computing platforms. *arXiv preprint arXiv:2311.11420*, 2023.

[203] Mohammed Yazid Lachachi, Mohamed Ouslim, Smail Niar, and Abdelmalik Taleb-Ahmed. Trueview: A lidar only perception system for autonomous vehicle (interactive presentation). In *Workshop on Autonomous Systems Design (ASD 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[204] Michael Larabel. Nvidia's jetson agx xavier carmel performance vs. low-power x86 processors.

[205] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[206] Bhoram Lee, Jonathan Brookshire, Rhys Yahata, and Supun Samarasekera. Towards safe, realistic testbed for robotic systems with human interaction. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11280–11287. IEEE, 2022.

[207] Daehee Lee, Minjong Yoo, Woo Kyung Kim, Wonje Choi, and Honguk Woo. Incremental learning of retrievable skills for efficient continual task adaptation. *Advances in Neural Information Processing Systems*, 37:17286–17312, 2024.

[208] Seulki Lee and Shahriar Nirjon. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *Proceedings of ACM Annual International Conference on Mobile Systems, Applications, and Services*, pages 175–190, 2020.

[209] Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Outstanding paper award: Analysis of global edf for parallel tasks. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 3–13. IEEE, 2013.

[210] Jing Li, Jian Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 85–96. IEEE, 2014.

[211] Ruoxiang Li, Nan Guan, Xu Jiang, Zhishan Guo, Zheng Dong, and Mingsong Lv. Worst-case time disparity analysis of message synchronization in ROS. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*, pages 40–52. IEEE, 2022.

[212] Yufei Li, Zexin Li, Yingfan Gao, and Cong Liu. White-box multi-objective adversarial attack on dialogue generation. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1778–1792. Association for Computational Linguistics, 2023.

[213] Yufei Li, Zexin Li, Wei Yang, and Cong Liu. Rt-lm: Uncertainty-aware resource management for real-time inference of language models. *arXiv preprint arXiv:2309.06619*, 2023.

[214] Zexin Li, Xiaoxi He, Yufei Li, Shahab Nikkhoo, Wei Yang, Lothar Thiele, and Cong Liu. Mimonet: Multi-input multi-output on-device deep learning. *arXiv preprint arXiv:2307.11962*, 2023.

[215] Zexin Li, Tao Ren, Xiaoxi He, and Cong Liu. Red: A systematic real-time scheduling approach for robotic environmental dynamics. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 210–223. IEEE, 2023.

[216] Zexin Li, Aritra Samanta, Yufei Li, Andrea Soltoggio, Hyoseung Kim, and Cong Liu. $\mathrm{R}ˆ{3}$: On-device real-time deep reinforcement learning for autonomous robotics. In *IEEE Real-Time Systems Symposium, RTSS 2023, Taipei, Taiwan, December 5-8, 2023*, pages 131–144. IEEE, 2023.

[217] Zexin Li, Aritra Samanta, Yufei Li, Andrea Soltoggio, Hyoseung Kim, and Cong Liu. R^3: On-device real-time deep reinforcement learning for autonomous robotics. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 131–144. IEEE, 2023.

[218] Zexin Li, Bangjie Yin, Taiping Yao, Junfeng Guo, Shouhong Ding, Simin Chen, and Cong Liu. Sibling-attack: Rethinking transferable adversarial attacks against face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24626–24637, 2023.

[219] Zexin Li, Jiancheng Zhang, Yufei Li, Yinglun Zhu, and Cong Liu. Mixtraining: A better trade-off between compute and performance. *arXiv preprint arXiv:2502.19513*, 2025.

[220] Zexin Li, Yuqun Zhang, Ao Ding, Husheng Zhou, and Cong Liu. Efficient algorithms for task mapping on heterogeneous cpu/gpu platforms for fast completion time. *Journal of Systems Architecture*, 114:101936, 2021.

[221] Zexin Li, Yuqun Zhang, Ao Ding, Husheng Zhou, and Cong Liu. Efficient algorithms for task mapping on heterogeneous cpu/gpu platforms for fast completion time. *Journal of Systems Architecture*, 114:101936, 2021.

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

[223] Zhuo Li, Hengyi Li, and Lin Meng. Model compression for deep neural networks: A survey. *Computers*, 12(3):60, 2023.

[224] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[225] Andy T Liu, Yi-Cheng Lin, Haibin Wu, Stefan Winkler, and Hung-yi Lee. Efficient training of self-supervised speech foundation models on a compute budget. In *2024 IEEE Spoken Language Technology Workshop (SLT)*, pages 961–968. IEEE, 2024.

[226] Jane WSW Liu. *Real-time systems*. Prentice Hall PTR, 2000.

[227] Sicong Liu, Junzhao Du, Kaiming Nan, Zimu Zhou, Hui Liu, Zhangyang Wang, and Yingyan Lin. Adadeep: A usage-driven, automated deep model compression framework for enabling ubiquitous intelligent mobiles. *IEEE Transactions on Mobile Computing*, pages 1–1, 2020.

[228] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation. *arXiv preprint arXiv:2205.13542*, 2022.

[229] Zicheng Liu, Siyuan Li, Di Wu, Zhiyuan Chen, Lirong Wu, Jianzhu Guo, and Stan Z. Li. Automix: Unveiling the power of mixup for stronger classifiers. In *European Conference on Computer Vision*, pages 441–458, 2022.

[230] Steven R Livingstone and Frank A Russo. The ryerson audio-visual database of emotional speech and song (ravdess): A dynamic, multimodal set of facial and vocal expressions in north american english. *PloS one*, 13(5):e0196391, 2018.

[231] V Lomanco and Davide Maltoni. Core50: a new dataset and benchmark for continual object recognition. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 17–26, 2017.

[232] Vincenzo Lomonaco, Davide Maltoni, Lorenzo Pellegrini, et al. Fine-grained continual learning. *arXiv preprint arXiv:1907.03799*, 1, 2019.

[233] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolias, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni. Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop, 2021.

[234] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[235] Sidi Lu, Yongtao Yao, and Weisong Shi. Collaborative learning on the edges: A case study on connected vehicles. In *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[236] Weimin Lyu, Zexin Bi, Fusheng Wang, and Chao Chen. Badclm: Backdoor attack in clinical language models for electronic health records. *arXiv preprint arXiv:2407.05213*, 2024.

[237] Weimin Lyu, Xinyu Dong, Rachel Wong, Songzhu Zheng, Kayley Abell-Hart, Fusheng Wang, and Chao Chen. A multimodal transformer: Fusing clinical notes with structured ehr data for interpretable in-hospital mortality prediction. In *AMIA Annual Symposium Proceedings*, volume 2022, page 719. American Medical Informatics Association, 2022.

[238] Haixu Ma, Donglin Zeng, and Yufeng Liu. Learning individualized treatment rules with many treatments: A supervised clustering approach using adaptive fusion. *Advances in Neural Information Processing Systems*, 35:15956–15969, 2022.

[239] Haixu Ma, Donglin Zeng, and Yufeng Liu. Learning optimal group-structured individualized treatment rules with many treatments. *Journal of Machine Learning Research*, 24(102):1–48, 2023.

[240] Xiaobo Ma, Abolfazl Karimpour, and Yao-Jan Wu. Eliminating the impacts of traffic volume variation on before and after studies: a causal inference approach. *Journal of Intelligent Transportation Systems*, pages 1–15, 2023.

[241] Xiaobo Ma, Abolfazl Karimpour, and Yao-Jan Wu. Data-driven transfer learning framework for estimating on-ramp and off-ramp traffic flows. *Journal of Intelligent Transportation Systems*, pages 1–14, 2024.

[242] Xinyue Ma, Suyeon Jeong, Minjia Zhang, Di Wang, Jonghyun Choi, and Myeongjae Jeon. Cost-effective on-device continual learning over memory hierarchy with miro. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2023.

[243] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[244] Pullarao Maddu, Wayne Doherty, Ganesh Sistu, Isabelle Leang, Michal Uricar, Sumanth Chennupati, Hazem Rashed, Jonathan Horgan, Ciaran Hughes, and Senthil Yogamani. Fisheyemultinet: Real-time multi-task learning architecture for surround-view automated parking system. *arXiv preprint arXiv:1912.11066*, 2019.

[245] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, 2022.

[246] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4), 2017.

[247] Akhil Mathur, Nicholas D Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proceedings of ACM Annual International Conference on Mobile Systems, Applications, and Services*, pages 68–81, 2017.

[248] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[249] Sanket Vaibhav Mehta, Darshan Patil, Sarath Chandar, and Emma Strubell. An empirical investigation of the role of pre-training in lifelong learning. *Journal of Machine Learning Research*, 24(214):1–50, 2023.

[250] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C Buttazzo. Response-time analysis of conditional dag tasks in multiprocessor systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 211–221. IEEE, 2015.

[251] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Neural autonomous navigation with riemannian motion policy. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8860–8866. IEEE, 2019.

[252] Umberto Michelucci. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898*, 2022.

[253] Emiliano Miluzzo, Cory T. Cornelius, Ashwin Ramaswamy, Tanzeem Choudhury, Zhigang Liu, and Andrew T. Campbell. Darwin phones: The evolution of sensing and inference on mobile phones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 5–20, New York, NY, USA, 2010. ACM.

[254] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Comput. Surv.*, 56(2), sep 2023.

[255] Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, pages 15630–15649. PMLR, 2022.

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

[257] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[258] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*, 2020.

[259] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. Batchsizer: Power-performance trade-off for DNN inference. In *ASPDAC: 26th Asia and South Pacific Design Automation Conference*, pages 819–824. ACM, 2021.

[260] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[261] Fatemeh Nazari, Mohamadhossein Noruzoliaee, and Abolfazl Kouros Mohammadian. Shared versus private mobility: Modeling public interest in autonomous vehicles accounting for latent attitudes. *Transportation Research Part C: Emerging Technologies*, 97:456–477, 2018.

[262] Xiangli Nie, Zhiguang Deng, Mingdong He, Mingyu Fan, and Zheng Tang. Online active continual learning for robotic lifelong object recognition. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[263] Vinod Nigade, Pablo Bauszat, Henri E. Bal, and Lin Wang. Jellyfish: Timely inference serving for dynamic edge networks. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*, pages 277–290. IEEE, 2022.

[264] Shahab Nikkhoo, Zexin Li, Aritra Samanta, Yufei Li, and Cong Liu. Pimbot: Policy and incentive manipulation for multi-robot reinforcement learning in social dilemmas. *arXiv preprint arXiv:2307.15944*, 2023.

[265] Alexander Nikulin, Vladislav Kurenkov, Denis Tarasov, Dmitry Akimov, and Sergey Kolesnikov. Q-ensemble for offline rl: Don't scale the ensemble, scale the batch size. *arXiv preprint arXiv:2211.11092*, 2022.

[266] Fabrice Normandin, Florian Golemo, Oleksiy Ostapenko, Pau Rodriguez, Matthew D Riemer, Julio Hurtado, Khimya Khetarpal, Ryan Lindeborg, Lucas Cecchi, Timothée Lesort, et al. Sequoia: A software framework to unify continual learning research. *arXiv preprint arXiv:2108.01005*, 2021.

[267] Chris Nota. The autonomous learning library. https://github.com/cpnota/autonomous-learning-library, 2020.

[268] NVIDIA. Jetson agx xavier. https://developer.nvidia.com/embedded/jetson-agx-xavier, 2020.

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

[270] NVIDIA. Jetson agx orin. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2022.

[271] NVIDIA. Sparkfun jetbot ai kit. https://www.sparkfun.com/products/18486, 2022.

[272] 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.

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

[274] Favour M. Nyikosa. *Adaptive Bayesian optimization for dynamic problems*. PhD thesis, University of Oxford, UK, 2018.

[275] Eugene Pleasants Odum, Gary W Barrett, et al. *Fundamentals of ecology*, volume 3. Saunders Philadelphia, 1971.

[276] OpenAI. Gpt-4 technical report, 2023.

[277] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba,

Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023.

[278] Kazuki Osawa, Shigang Li, and Torsten Hoefler. Pipefisher: Efficient training of large language models using pipelining and fisher information matrices. *Proceedings of Machine Learning and Systems*, 5, 2023.

[279] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[280] Arthi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. Gemel: Model merging for {Memory-Efficient},{Real-Time} video analytics at the edge. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 973–994, 2023.

[281] Fei Pan, Inkyu Shin, Francois Rameau, Seokju Lee, and In So Kweon. Unsupervised intra-domain adaptation for semantic segmentation through self-supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3764–3773, 2020.

[282] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[283] Mark Panahi, Weiran Nie, and Kwei-Jay Lin. A framework for real-time service-oriented architecture. In *2009 IEEE Conference on Commerce and Enterprise Computing*, pages 460–467. IEEE, 2009.

[284] Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. PABO: pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD*, pages 1–8. ACM, 2019.

[285] Maryam Parsa, J. Parker Mitchell, Catherine D. Schuman, Robert M. Patton, Thomas E. Potok, and Kaushik Roy. Bayesian-based hyperparameter optimization for spiking neuromorphic systems. In *IEEE International Conference on Big Data (IEEE BigData)*, pages 4472–4478. IEEE, 2019.

[286] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

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

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

[289] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10203–10209. IEEE, 2020.

[290] Lorenzo Pellegrini, Vincenzo Lomonaco, Gabriele Graffieti, and Davide Maltoni. Continual learning at the edge: Real-time training on smartphone devices. *arXiv preprint arXiv:2105.13127*, 2021.

[291] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143, 2018.

[292] Matthew E Peters, Sebastian Ruder, and Noah A Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*, 2019.

[293] Alexander Popov, Patrik Gebhardt, Ke Chen, Ryan Oldja, Heeseok Lee, Shane Murray, Ruchi Bhargava, and Nikolai Smolyanskiy. Nvradarnet: Real-time radar obstacle and free space detection for autonomous driving. *arXiv preprint arXiv:2209.14499*, 2022.

[294] Ameya Prabhu, Hasan Abed Al Kader Hammoud, Puneet K Dokania, Philip HS Torr, Ser-Nam Lim, Bernard Ghanem, and Adel Bibi. Computationally budgeted continual learning: What does matter? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3698–3707, 2023.

[295] Hang Qiu, Pohan Huang, Namo Asavisanu, Xiaochen Liu, Konstantinos Psounis, and Ramesh Govindan. Autocast: Scalable infrastructure-less cooperative perception for distributed collaborative driving. *arXiv preprint arXiv:2112.14947*, 2021.

[296] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system, 2009.

[297] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[298] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[299] Simon Ramstedt and Chris Pal. Real-time reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32:*

Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 3067–3076, 2019.

[300] Leonardo Ravaglia, Manuele Rusci, Davide Nadalini, Alessandro Capotondi, Francesco Conti, and Luca Benini. A tinyml platform for on-device continual learning with quantized latent replays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4):789–802, 2021.

[301] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[302] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[303] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[304] Scott E. Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent. *CoRR*, abs/2205.06175, 2022.

[305] Weijieying Ren and Vasant G Honavar. Esacl: Efficient continual learning of sparse models. *arXiv preprint arXiv:2401.05667*, 2024.

[306] Jakob Richter, Junjie Shi, Jian-Jia Chen, Jörg Rahnenführer, and Michel Lang. Model-based optimization with concept drifts. In *GECCO '20: Genetic and Evolutionary Computation Conference*, pages 877–885. ACM, 2020.

[307] Matthew Riemer, Sharath Chandra Raparthy, Ignacio Cases, Gopeshh Subbaraj, Maximilian Puelma Touzel, and Irina Rish. Continual learning in environments with polynomial mixing times. In *NeurIPS*, 2022.

[308] robocarstore. Donkey car s1. https://www.robocarstore.com/products/donkey-car-starter-kit, 2023.

[309] ROBOTIS. *TurtleBot3 Overview*. ROBOTIS, 2024. Accessed: 15 April 2024.

[310] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[311] Olivier Roustant, David Ginsbourger, and Yves Deville. Dicekriging, diceoptim: Two r packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of statistical software*, 51:1–55, 2012.

[312] Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 111. IEEE, 2004.

[313] Peter Schafhalter, Sukrit Kalra, Le Xu, Joseph E Gonzalez, and Ion Stoica. Leveraging cloud computing to make autonomous vehicles safer. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5559–5566. IEEE, 2023.

[314] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[315] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *4th International Conference on Learning Representations, ICLR 2016*, 2016.

[316] Martin Schiemer, Lei Fang, Simon Dobson, and Juan Ye. Online continual learning for human activity recognition. *Pervasive and Mobile Computing*, 93:101817, 2023.

[317] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. *International conference on machine learning*, pages 1889–1897, 2015.

[318] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[319] Sefik Ilkin Serengil and Alper Ozpinar. Lightface: A hybrid deep face recognition framework. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 23–27. IEEE, 2020.

[320] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE*, 104(1):148–175, 2016.

[321] Yujun Shi, Li Yuan, Yunpeng Chen, and Jiashi Feng. Continual learning via bit-level information preserving. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 16674–16683, 2021.

[322] Soheil Shirvani, Aritra Samanta, Zexin Li, and Cong Liu. Duojoule: Accurate on-device deep reinforcement learning for energy and timeliness. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, pages 109–122. IEEE, 2024.

[323] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[324] Sudipta Saha Shubha and Haiying 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*, pages 473–485, 2023.

[325] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[326] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[327] Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh, and Jiasi Chen. Going through the motions:{AR/VR} keylogging from user head motions. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[328] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *26th Annual Conference on Neural Information Processing Systems*, pages 2960–2968, 2012.

[329] Han Song, Cong Liu, and Huafeng Dai. Bundledslam: An accurate visual slam system using multiple cameras. In *2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 7, pages 106–111. IEEE, 2024.

[330] Han Song, Zhongche Qu, Zhi Zhang, Zihan Ye, and Cong Liu. Eta-init: Enhancing the translation accuracy for stereo visual-inertial slam initialization. *arXiv preprint arXiv:2405.15082*, 2024.

[331] Eduardo D Sontag. Smooth stabilization implies coprime factorization. *IEEE transactions on automatic control*, 34(4):435–443, 1989.

[332] Amelia Sorrenti, Giovanni Bellitto, Federica Proietto Salanitri, Matteo Pennisi, Concetto Spampinato, and Simone Palazzo. Selective freezing for efficient continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3550–3559, 2023.

[333] Albin Soutif-Cormerais, Antonio Carta, Andrea Cossu, Julio Hurtado, Vincenzo Lomonaco, Joost Van de Weijer, and Hamed Hemati. A comprehensive empirical evaluation on online continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3518–3528, 2023.

[334] Matteo Spezialetti, Giuseppe Placidi, and Silvia Rossi. Emotion recognition for human-robot interaction: Recent advances and future perspectives. *Frontiers in Robotics and AI*, 7, 2020.

[335] Peter Stone, Rodney Brooks, Erik Brynjolfsson, Ryan Calo, Oren Etzioni, Greg Hager, Julia Hirschberg, Shivaram Kalyanakrishnan, Ece Kamar, Sarit Kraus, et al. Artificial intelligence and life in 2030: the one hundred year study on artificial intelligence. *arXiv preprint arXiv:2211.06318*, 2022.

[336] Sanbao Su, Yiming Li, Sihong He, Songyang Han, Chen Feng, Caiwen Ding, and Fei Miao. Uncertainty quantification of collaborative detection for self-driving. *arXiv preprint arXiv:2209.08162*, 2022.

[337] Yue Tang, Zhiwei Feng, Nan Guan, Xu Jiang, Mingsong Lv, Qingxu Deng, and Wang Yi. Response time analysis and priority assignment of processing chains on ROS2 executors. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 231–243. IEEE, 2020.

[338] Harun Teper, Mario Günzel, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. End-to-end timing analysis in ROS2. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*, pages 53–65. IEEE, 2022.

[339] Pierre Thodoroff, Wenyu Li, and Neil D Lawrence. Benchmarking real-time reinforcement learning. In *NeurIPS 2021 Workshop on Pre-registration in Machine Learning*, pages 26–41. PMLR, 2022.

[340] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[341] Ehsan Totoni, Babak Behzad, Swapnil Ghike, and Josep Torrellas. Comparing the power and performance of intel's scc to state-of-the-art cpus and gpus. In *2012 IEEE International Symposium on Performance Analysis of Systems & Software*, pages 78–87. IEEE, 2012.

[342] Niklas Ueter, Georg Von Der Brüggen, Jian-Jia Chen, Jing Li, and Kunal Agrawal. Reservation-based federated scheduling for parallel real-time tasks. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 482–494. IEEE, 2018.

[343] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[344] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[345] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[346] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International conference on machine learning*, pages 6438–6447. PMLR, 2019.

[347] Niclas Vödisch, Daniele Cattaneo, Wolfram Burgard, and Abhinav Valada. Covio: Online continual learning for visual-inertial odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2464–2473, 2023.

[348] Niclas Vödisch, Kürsat Petek, Wolfram Burgard, and Abhinav Valada. Codeps: Online continual learning for depth estimation and panoptic segmentation. *arXiv preprint arXiv:2303.10147*, 2023.

[349] Mingyang Wang, Heike Adel, Lukas Lange, Jannik Strötgen, and Hinrich Schütze. Learn it or leave it: module composition and pruning for continual learning. *arXiv preprint arXiv:2406.18708*, 2024.

[350] Peihao Wang, Rameswar Panda, and Zhangyang Wang. Data efficient neural scaling law via model reusing. In *International Conference on Machine Learning*, pages 36193–36204. PMLR, 2023.

[351] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. Melon: Breaking the memory wall for resource-efficient on-device machine learning. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 450–463, 2022.

[352] Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, and Furu Wei. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. *CoRR*, abs/2208.10442, 2022.

[353] Xiao Wang, Guangyao Chen, Guangwu Qian, Pengcheng Gao, Xiao-Yong Wei, Yaowei Wang, Yonghong Tian, and Wen Gao. Large-scale multi-modal pre-trained models: A comprehensive survey. *Machine Intelligence Research*, pages 1–36, 2023.

[354] Yidi Wang, Mohsen Karimi, and Hyoseung Kim. Towards energy-efficient real-time scheduling of heterogeneous multi-gpu systems. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 409–421. IEEE, 2022.

[355] Yidi Wang, Cong Liu, Daniel Wong, and Hyoseung Kim. Gcaps: Gpu context-aware preemptive priority-based scheduling for real-time tasks. *arXiv preprint arXiv:2406.05221*, 2024.

[356] Yiding Wang, Decang Sun, Kai Chen, Fan Lai, and Mosharaf Chowdhury. Egeria: Efficient dnn training with knowledge-guided layer freezing. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 851–866, 2023.

[357] Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-train: Training state-of-the-art cnns with over 80% energy savings. *Advances in Neural Information Processing Systems*, 32, 2019.

[358] Zifeng Wang, Zheng Zhan, Yifan Gong, Geng Yuan, Wei Niu, Tong Jian, Bin Ren, Stratis Ioannidis, Yanzhi Wang, and Jennifer Dy. Sparcl: Sparse continual learning on the edge. *Advances in Neural Information Processing Systems*, 35:20366–20380, 2022.

[359] Kyle Wiggers. Tier iv raises over $100 million to develop open source software for driverless cars, Jul 2019.

[360] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision, 2020.

[361] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers. In *European conference on computer vision*, pages 68–85. Springer, 2022.

[362] Wei Wu, Aurelien Bouteiller, George Bosilca, Mathieu Faverge, and Jack Dongarra. Hierarchical dag scheduling for hybrid distributed systems. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 156–165. IEEE, 2015.

[363] Yecheng Xiang and Hyoseung 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)*, pages 392–405. IEEE, 2019.

[364] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 392–405, Piscataway, NJ, USA, 2019. IEEE Press.

[365] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 392–405. IEEE, 2019.

[366] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 595–610, 2018.

[367] Guoqi Xie, Renfa Li, and Keqin Li. Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems. *Journal of Parallel and Distributed Computing*, 83:1–12, 2015.

[368] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[369] J. Xu and D.L. Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, 1990.

[370] Zhefan Xu, Di Deng, Yiping Dong, and Kenji Shimada. Dpmpc-planner: A real-time uav trajectory planning framework for complex static environments with dynamic obstacles. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 250–256. IEEE, 2022.

[371] Yu-Qi Yang, Yu-Xiao Guo, Jian-Yu Xiong, Yang Liu, Hao Pan, Peng-Shuai Wang, Xin Tong, and Baining Guo. Swin3d: A pretrained transformer backbone for 3d indoor scene understanding. *arXiv preprint arXiv:2304.06906*, 2023.

[372] Zeyu Yang, Angus B Clark, Digby Chappell, and Nicolas Rojas. Instinctive real-time semg-based control of prosthetic hand with reduced data acquisition and embedded deep learning training. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5666–5672. IEEE, 2022.

[373] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th annual foundations of computer science*, pages 374–382. IEEE, 1995.

[374] Xingcheng Yao, Yanan Zheng, Xiaocong Yang, and Zhilin Yang. Nlp from scratch without large-scale pretraining: A simple and efficient framework. In *International Conference on Machine Learning*, pages 25438–25451. PMLR, 2022.

[375] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[376] Fei Ye and Adrian G Bors. Continual variational autoencoder via continual generative knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 10918–10926, 2023.

[377] Hanjing Ye, Jieting Zhao, Yu Zhan, Weinan Chen, Li He, and Hong Zhang. Person re-identification for robot person following with online continual learning. *IEEE Robotics and Automation Letters*, 2024.

[378] Se-Wook Yoo and Seung-Woo Seo. Learning multi-task transferable rewards via variational inverse reinforcement learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 434–440. IEEE, 2022.

[379] Bum-Jae You, Myung Hwangbo, Sung-On Lee, Sang-Rok Oh, Young Do Kwon, and San Lim. Development of a home service robot'issac'. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 3, pages 2630–2635. IEEE, 2003.

[380] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation NSDI*, pages 119–139. USENIX Association, 2023.

[381] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *Proceedings of International Conference on Learning Representations*, 2019.

[382] Liqiang Yu, Chen Li, Longsen Gao, Bo Liu, and Chang Che. Stochastic analysis of touch-tone frequency recognition in two-way radio systems for dialed telephone number identification. In *2024 7th International Conference on Advanced Algorithms and Control Engineering (ICAACE)*, pages 1565–1572. IEEE, 2024.

[383] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[384] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pages 2528–2535. IEEE, 2010.

[385] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1476–1485, 2019.

[386] Chen Zhang, Luis Fernando D'Haro, Yiming Chen, Thomas Friedrichs, and Haizhou Li. Investigating the impact of pre-trained language models on dialog evaluation. In *Conversational AI for Natural Human-Centric Interaction: 12th International Workshop on Spoken Dialogue System Technology, IWSDS 2021, Singapore*, pages 291–306. Springer, 2022.

[387] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[388] Jifan Zhang, Yifang Chen, Gregory Canal, Arnav Mohanty Das, Gantavya Bhatt, Stephen Mussmann, Yinglun Zhu, Jeff Bilmes, Simon Shaolei Du, Kevin Jamieson, et al. Labelbench: A comprehensive framework for benchmarking adaptive label-efficient learning. *Journal of Data-centric Machine Learning Research*, 2024.

[389] Jingyao Zhang, Huaxi Gu, Grace Li Zhang, Bing Li, and Ulf Schlichtmann. Hardware-software codesign of weight reshaping and systolic array multiplexing for efficient cnns. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 667–672. IEEE, 2021.

[390] Jingyao Zhang, Mohsen Imani, and Elaheh Sadredini. Bp-ntt: Fast and compact in-sram number theoretic transform with bit-parallel modular multiplication. *arXiv preprint arXiv:2303.00173*, 2023.

[391] Lin Zhang, Robert Merrifield, Anton Deguet, and Guang-Zhong Yang. Powering the world's robots—10 years of ros. *Science Robotics*, 2(11):eaar1868, 2017.

[392] Shiqing Zhang, Shiliang Zhang, Tiejun Huang, and Wen Gao. Multimodal deep convolutional neural network for audio-visual emotion recognition. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 281–284, 2016.

[393] Shiqing Zhang, Shiliang Zhang, Tiejun Huang, and Wen Gao. Multimodal deep convolutional neural network for audio-visual emotion recognition. In *Proceedings of ACM on International Conference on Multimedia Retrieval*, pages 281–284, 2016.

[394] Tianyi Zhang and Matthew Johnson-Roberson. Learning cross-scale visual representations for real-time image geo-localization. *IEEE Robotics and Automation Letters*, 7(2):5087–5094, 2022.

[395] Wenxuan Zhang, Youssef Mohamed, Bernard Ghanem, Philip HS Torr, Adel Bibi, and Mohamed Elhoseiny. Continual learning on a diet: Learning from sparsely labeled streams under constrained computation. *arXiv preprint arXiv:2404.12766*, 2024.

[396] Xueyang Zhang, Hang Li, Xi Chen, and Xue Liu. Impact patterns of combining model pruning and continual learning on model performance. In *2021 IEEE Third International Conference on Cognitive Machine Intelligence (CogMI)*, pages 27–33. IEEE, 2021.

[397] Xumiao Zhang, Anlan Zhang, Jiachen Sun, Xiao Zhu, Y Ethan Guo, Feng Qian, and Z Morley Mao. Emp: Edge-assisted multi-vehicle perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 545–558, 2021.

[398] Yicheng Zhang, Dhroov Pandey, Di Wu, Turja Kundu, Ruopu Li, and Tong Shu. Accuracy-constrained efficiency optimization and gpu profiling of cnn inference for detecting drainage crossing locations. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 1780–1788, 2023.

[399] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. Stealing neural network structure through remote fpga side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 16:4377–4388, 2021.

[400] Yichi Zhang, Daniel W. Apley, and Wei Chen. Bayesian optimization for materials design with mixed quantitative and qualitative variables. *CoRR*, abs/1910.01688, 2019.

[401] Yufeng Zhang, Xue Wang, Longsen Gao, and Zongbao Liu. Manipulator control system based on machine vision. In *International Conference on Applications and Techniques in Cyber Intelligence ATCI 2019: Applications and Techniques in Cyber Intelligence 7*, pages 906–916. Springer, 2020.

[402] Zijian Zhang, Yujie Sun, Zepu Wang, Yuqi Nie, Xiaobo Ma, Peng Sun, and Ruolin Li. Large language models for mobility in transportation systems: A survey on forecasting tasks. *arXiv preprint arXiv:2405.02357*, 2024.

[403] Jing Zhao, Xijiong Xie, Xin Xu, and Shiliang Sun. Multi-view learning overview: Recent progress and new challenges. *Information Fusion*, 38:43–54, 2017.

[404] Yuqing Zhao, Divya Saxena, and Jiannong Cao. Memory-efficient domain incremental learning for internet of things. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 1175–1181, 2022.

[405] Baoding Zhou, Zhiqian Wu, Zhipeng Chen, Xu Liu, and Qingquan Li. Wi-fi rtt/encoder/ins-based robot indoor localization using smartphones. *IEEE Transactions on Vehicular Technology*, 72(5):6683–6694, 2023.

[406] Husheng Zhou, Soroush Bateni, and Cong Liu. Sˆ 3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 190–201. IEEE, 2018.

[407] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, pages 347–358, 2020.

[408] Xingyu Zhou and Ness B. Shroff. No-regret algorithms for time-varying bayesian optimization. In *55th Annual Conference on Information Sciences and Systems, CISS*, pages 1–6. IEEE, 2021.

[409] Yingtian Zou, Vikas Verma, Sarthak Mittal, Wai Hoh Tang, Hieu Pham, Juho Kannala, Yoshua Bengio, Arno Solin, and Kenji Kawaguchi. Mixupe: Understanding and improving mixup from directional derivative perspective. In *Uncertainty in Artificial Intelligence*, pages 2597–2607. PMLR, 2023.