CARIn: Constraint-Aware and Responsive Inference on Heterogeneous Devices for Single- and Multi-DNN Workloads

IOANNIS PANOPOULOS, National Technical University of Athens, Greece
STYLIANOS I. VENIERIS, Samsung AI Center, UK
IAKOVOS S. VENIERIS, National Technical University of Athens, Greece

The relentless expansion of deep learning (DL) applications in recent years has prompted a pivotal shift towards on-device execution, driven by the urgent need for real-time processing, heightened privacy concerns, and reduced latency across diverse domains. This paper addresses the challenges inherent in optimising the execution of deep neural networks (DNNs) on mobile devices, with a focus on device heterogeneity, multi-DNN execution, and dynamic runtime adaptation. We introduce CARIn, a novel framework designed for the optimised deployment of both single- and multi-DNN applications under user-defined service-level objectives (SLOs). Leveraging an expressive multi-objective optimisation (MOO) framework and a runtime-aware sorting and search algorithm (RASS) as the MOO solver, CARIn facilitates efficient adaptation to dynamic conditions while addressing resource contention issues associated with multi-DNN execution. Notably, RASS generates a set of configurations, anticipating subsequent runtime adaptation, ensuring rapid, low-overhead adjustments in response to environmental fluctuations. Extensive evaluation across diverse tasks, including text classification, scene recognition, and face analysis, showcases the versatility of CARIn across various model architectures, such as Convolutional Neural Networks (CNNs) and Transformers, and realistic use cases. We observe a substantial enhancement in the fair treatment of the problem's objectives, reaching 1.92× when compared to single-model designs, and up to 10.69× in contrast to the state-of-the-art OODIn framework. Additionally, we achieve a significant gain of up to 4.06× over hardware-unaware designs in multi-DNN applications. Finally, our framework sustains its performance while effectively eliminating the time overhead associated with identifying the optimal design in response to environmental challenges.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Computer systems organization** → Embedded systems; • **General and reference** → *Performance*; *Evaluation.*

Additional Key Words and Phrases: Deep neural networks, on-device inference, service-level objectives, heterogeneity. runtime adaptation, multi-dnn execution

ACM Reference Format:
Ioannis Panopoulos, Stylianos I. Venieris, and Iakovos S. Venieris. 2024. **CARIn: Constraint-Aware and Responsive Inference on Heterogeneous Devices for Single- and Multi-DNN Workloads**. *ACM Trans. Embedd. Comput. Syst.* 37, 4, Article 111 (August 2024), 31 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In recent years, the pervasive growth of deep learning (DL) applications has catalysed a paradigm shift in the field of artificial intelligence, rendering on-device execution a critical imperative [69]. The burgeoning demand for sophisticated deep neural networks (DNNs) spans a myriad of domains,

Authors' addresses: Ioannis Panopoulos, ioannispanop@mail.ntua.gr, National Technical University of Athens, Athens, Greece; Stylianos I. Venieris, Samsung AI Center, Cambridge, UK, s.venieris@samsung.com; Iakovos S. Venieris, venieris@cs.ece.ntua.gr, National Technical University of Athens, Athens, Greece.

1539-9087/2024/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

from computer vision to natural language processing, necessitating the deployment of these models directly on mobile devices. This shift from centralised to decentralised computation arises from the intrinsic requirements of real-time processing, enhanced privacy concerns, and the need for reduced latency in diverse applications. As a consequence, the optimisation of executing deep neural networks on-device has emerged as a paramount research frontier.

While the shift towards on-device execution of DNNs represents a pivotal advancement, it is not without its formidable challenges. Device heterogeneity, characterised by the diverse array of hardware and computational capabilities across mobile devices, remains a persistent hurdle. Moreover, emerging challenges, such as the simultaneous execution of multiple DNNs on a single device [60] and the need for dynamic runtime adaptation [64] to evolving environmental conditions, add layers of complexity to the optimisation landscape. Multi-DNN execution introduces intricate dependencies and resource contention issues, necessitating sophisticated orchestration strategies. Runtime adaptation, on the other hand, mandates the development of intelligent mechanisms capable of dynamically adjusting model parameters and system configurations to optimise performance in real-time scenarios. Addressing these challenges is paramount to unlocking the full potential of on-device deep learning, as it paves the way for the seamless integration of advanced AI capabilities into the fabric of our interconnected devices.

Enhancing the work presented in [61], namely OODIn, this paper presents `CARIn`, a novel framework for the optimised deployment of both single- and multi-DNN applications on mobile devices. The initial work in [61] focused primarily on presenting a new highly parametrised software architecture for DL mobile apps, optimising single-DNN applications and evaluating solely on the image classification task. In this paper, we build upon the architecture of OODIn that allows us to efficiently modify model and system parameters, and introduce two novel components in order to meet the new demands of model multi-tenancy and efficient runtime adaptability. First, we develop an expressive multi-objective optimisation (MOO) framework that allows us to capture both single- and multi-DNN workloads and to formally model the performance requirements and constraints of DL applications. Second, we present `RASS`, a runtime-aware MOO solver that enables rapid, low-overhead adaptation while sustaining high performance under dynamic conditions. Contrary to existing optimisers that yield a *single* execution plan for a given device [28, 62], our solver generates a *set* of configurations to accommodate potential variations in resource availability. This eliminates the necessity to continually adjust and resolve the MOO problem whenever a runtime issue arises. Additionally, we broaden the scope of targeted tasks and further augment this by conducting a comprehensive evaluation spanning various model architectures, including Convolutional Neural Networks (CNNs) and Transformers, across a spectrum of realistic scenarios characterised by diverse performance demands.

2 BACKGROUND & RELATED WORK

2.1 Problem Statement

The primary aim of a DL application is to consistently uphold its performance goals or service-level objectives (SLOs), often referred to as quality of service (QoS) targets. These SLOs encompass a multifaceted range of critical metrics, including but not confined to accuracy, latency, throughput, memory utilisation and energy consumption. Achieving and sustaining these objectives requires careful consideration of the specific demands inherent to a given application or system. It is crucial to recognise that this challenge is compounded by two primary factors: (a) the inherent heterogeneity and (b) the dynamic nature of mobile and embedded devices. Adding to this complexity is the increasingly prevalent use of multiple models within DL applications, which places additional demands on the already intricate ecosystem of these devices.

2.1.1 Device Heterogeneity. Compact devices involve a wide array of hardware configurations, characteristics, and capabilities, leading to a significant level of diversity [1, 2, 22, 66]. This diversity manifests not only across distinct devices, which is referred to as "inter-device heterogeneity," but also within individual devices, a concept known as "intra-device heterogeneity." Inter-device heterogeneity reflects the variations in size, processing power, memory capacity, energy efficiency, and more, across different devices. For example, a high-end smartphone will have markedly different hardware specifications than a low-cost IoT sensor node, illustrating the extent of diversity that exists across various devices. Intra-device heterogeneity, on the other hand, arises from the presence of multiple hardware components and subsystems within a device, each with its distinct attributes. In a standard smartphone setup, for instance, one may encounter a CPU, a GPU, and an NPU, all with varying clock speeds, energy consumption profiles, memory requirements, and parallelisation capabilities. **Due to device heterogeneity, it is very challenging to design a universal DL model that performs efficiently across devices**. For example, a model meticulously crafted and optimised to run seamlessly on Google's Edge TPU may encounter performance issues and inefficiencies when deployed on a different processor, such as a conventional mobile GPU.

As a result, rather than pursuing a one-size-fits-all approach, where a single model is expected to excel universally, the focus shifts towards creating models that are fine-tuned and optimised for the unique features of each target device. These device-specific models are designed to leverage the strengths of the hardware and maximise performance, thereby addressing the inherent challenges associated with device heterogeneity.

2.1.2 Dynamic Environment. In contemporary mobile and embedded computing ecosystems, the concurrent execution of multiple applications and processes is commonplace. This inherent multi-tasking characteristic introduces notable fluctuations in resource availability and workload demands, thereby rendering the acquisition of sufficient resources for performant task execution a challenging endeavour. **Due to environment dynamicity, it is very challenging for a static execution configuration to consistently satisfy the application's SLOs at any given time**. For instance, if a user runs the application outdoors on a hot day, the device's temperature may rise and thermal throttling mechanisms can be triggered, causing the CPU or GPU to reduce their clock speeds to prevent overheating [54], resulting in reduced throughput or execution slowdown.

Such scenarios necessitate the ability to dynamically adapt to changing conditions and varying resource availability in real time. This adaptive behaviour is vital to ensure that the application consistently maintains satisfactory performance levels despite the variability in its operational environment.

2.1.3 Multiple DNNs. Today's growing demand for more advanced and intelligent systems has given rise to scenarios that mandate the simultaneous utilisation of multiple models, often referred to as "multi-DNN" configurations [60]. This paradigm shift is mainly driven by the need to address specific tasks or solve complex problems that demand a diversified approach, benefiting from the combined expertise of multiple specialised models. Multi-DNN applications showcase a high degree of adaptability in employing multiple models, as they can be harnessed to address a singular, intricate task [24, 73] or several distinct, autonomous tasks [8, 36]. In the former scenario, models typically exhibit interdependence and may require sequential execution, while in the latter scenario, models operate independently, affording them the capability to run in parallel. **The efficient deployment of multi-DNN configurations introduces intricacies that pertain to resource allocation and load distribution.** Particularly, parallel model execution presents a notably more intricate challenge compared to sequential execution, as models compete for the device's finite resources. This concurrent operation, coupled with the simultaneous management of multiple tasks, amplifies the overall workload and poses new challenges to resource allocation and coordination.

The attainment of seamless orchestration and effective collaboration among multiple specialised models while upholding stringent performance and quality benchmarks stands as a substantial and multifaceted challenge within the domain of multi-DNN applications.

2.2 Related Work

The field of on-device deep learning has witnessed significant advancements in addressing the challenges posed by device heterogeneity and environment dynamicity in both single- and multi-DNN use cases. This progress reflects a profound shift in the landscape of deep learning research, where a growing emphasis has been placed on ensuring that DL models not only function effectively but also meet specific SLOs across a spectrum of computational environments, ranging from powerful high-end devices to resource-constrained edge computing platforms. Moreover, recent efforts have explored the integration of MOO techniques to further enhance the adaptability and efficiency of on-device DL solutions.

2.2.1 Service-Level Objectives. Most of the prior works directed towards achieving SLOs have predominantly concentrated on scenarios involving multiple DNNs, primarily investigating the trade-off between accuracy and latency. Within this domain, the majority is focused on system development for edge servers [13, 29, 53, 83, 84], and only a limited number of studies have been devoted to on-device execution, specifically by orchestrating multiple inference requests across heterogeneous processors [24, 51, 73]. In contrast, our proposed framework demonstrates the capability to accommodate a diverse array of SLOs, including but not limited to accuracy, latency, memory footprint, size, and energy consumption, tailored for both single- and multi-DNN on-device applications.

2.2.2 Multi-Objective Optimisation. MOO has been widely employed in conjunction with NAS methodologies, culminating in the development of Multi-Objective Neural Architecture Search (MONAS). This approach is particularly valuable for (a) designing DNNs with the goal of not solely optimising the accuracy but also considering resource consumption [11, 23, 56], and (b) for compressing pretrained models [18, 40, 63]. Notably, our framework represents one of the pioneering efforts to formulate and address device-specific MOO problems to achieve specific SLOs at the system level.

2.2.3 Device-Specific Solutions. The majority of endeavours aimed at addressing device heterogeneity predominantly concentrate on the model level, *i.e.* by identifying the most fitting DL architecture tailored to a specific hardware platform. Among the prominent model-level methodologies, neural architecture search (NAS) and model scaling have a central role.

Hardware-aware NAS (HW-NAS) approaches seek to optimise DNN architectures both for high predictive accuracy and for efficient execution on a target deployment platform. Its most prominent premise is the inclusion of (a) hardware constraints, and (b) latency, energy and other system metrics, as objectives during the search process [3, 4, 65, 80]. HW-NAS usually involves performance prediction in order to guide the search algorithm. Nonetheless, estimating precise latency, memory or energy figures can be challenging, and the method's effectiveness heavily relies on the accuracy of these estimates. Such approaches can also be computationally intensive due to the need to *train* and evaluate a large number of candidate architectures.

Supernet-based NAS, also known as One-shot NAS, is an approach that leverages a supernet along with weight sharing to facilitate efficient architecture search [6, 30, 35, 64]. A supernet is a network containing all possible architectural choices of a given search space and it enables the exploration of diverse neural architectures while significantly reducing computational overhead. While this approach reduces the training-time computational requirements, it may not be as

effective at tailoring architectures to specific hardware constraints and weight sharing may restrict fine-grained control over architectural decisions.

Lastly, model scaling involves adjusting parameters such as the depth, width and input size of a DNN to strike a balance between accuracy and efficiency [10, 58, 68]. This technique is often applied along with NAS or knowledge distillation methods to also accommodate resource constraints. However, model scaling might not fully exploit the unique hardware characteristics of specific devices, potentially leading to sub-optimal performance.

2.2.4 Runtime Adaptation. In the context of runtime adaptation, research efforts span both the model and system levels. On the model level, the primary focus revolves around the development of techniques designed to dynamically adjust the model's architecture in response to fluctuations in resource availability. These adaptive models possess the capability to modify their architecture and parameters in real time during inference, effectively responding to the evolving constraints of the computing environment. Prominent examples of such models comprise adaptive supernets [7, 16, 43, 64], adaptive model scaling [20, 72, 77], multi-branch networks [17], early-exit models [4, 34], and a variety of other innovative approaches. However, crafting adaptive mechanisms that seamlessly function across a diverse range of devices can pose significant technical challenges and the adaptability of these networks may introduce certain computational overhead, potentially impacting the performance of real-time applications. At the system level, complementary methods come into play, including dynamic compression [38], adaptive model selection [31], and efficient scheduling on available hardware [70], among others.

2.2.5 Multi-DNN Inference. To facilitate multi-DNN inference, researchers have also explored solutions at both the model and system levels [60]. From the model perspective, the execution of multiple DNNs aligns closely with the principles of multi-task learning [21, 49, 78, 82], a technique that trains a single model to perform multiple related tasks simultaneously. Consequently, using a single multi-task model for inference can replace the need for concurrent inferences from multiple models. At the system level, most research efforts leverage the heterogeneous processors available on devices and aim to identify the highest-performing mapping strategy. This is typically achieved through approaches that partition the model at the layer level [24, 26, 27, 67] or by introducing task-level priorities [37]. Additionally, there exists a body of research focused on multi-tenant inference systems [12, 75], albeit predominantly concentrating on server-based configurations [71, 74] rather than on-device implementations [60].

3 OVERVIEW OF CARIn

3.1 Proposed Solution

CARIn addresses the main challenges of on-device DL inference (Section 2.1) in two ways. First, we introduce a novel approach of modelling DL applications, utilising a multi-objective optimisation (MOO) framework to encapsulate their characteristics (Section 4). Given the rising number and diversity of DL applications, CARIn is able to analytically represent their various performance requirements and constraints, with the required expressivity to support both single- and multi-DNN scenarios. Second, to enable runtime adaptation, we introduce RASS, a runtime-aware MOO solver that allows for low-overhead and effective dynamic adjustment of the execution. The key principle behind RASS's design is to explicitly consider during the MOO solution stage that adaptation may subsequently be required at deployment time. As such, RASS operates in two steps: *i)* it generates a set of alternative execution configurations with diverse trade-offs prior to deployment, and *ii)* configures the inference engine with a policy of switching among them.

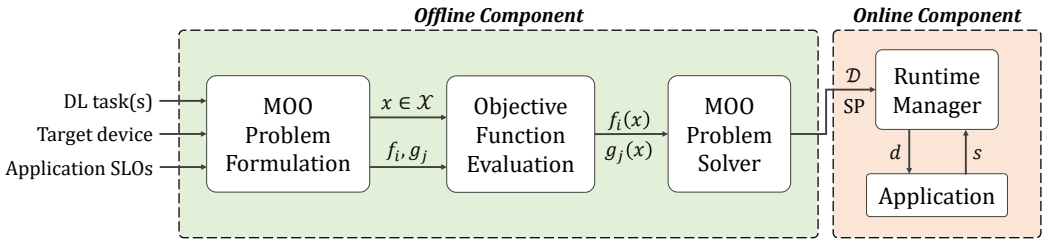

Fig. 1. High-level workflow of CARIn.

Towards alleviating the impact of device heterogeneity and resource fluctuation, CARIn operates exclusively at the system level, bypassing the need to produce an optimal model for each target device. Model-level solutions typically include the design, exploration, training, and adaptation of a DNN's architecture to specific target devices and resource availability changes. These procedures can be cumbersome, time-consuming, and lead to complex pipelines. Instead, our framework employs a repository of pre-trained models with varying architectures and complexities. The singular requisite action in relation to the models entails the application of post-training quantisation (Section 6.1).

The design of our framework was driven by the fact that satisfying SLOs depends not only on the target model but also on the specific target device, especially the processor in use. Consequently, CARIn's primary objective is to determine, at any given time, the most suitable model-processor[1] pair (or pairs) for a specified device. Internally, our MOO framework expresses this as a DL-based, device-centric problem, to effectively capture both the application's SLOs and the unique characteristics of the target device. Given the device-specific nature of our MOO formulation, a distinct optimisation problem is formed for each given device, effectively circumventing the challenge posed by device heterogeneity. Additionally, in order to facilitate real-time adaptation, CARIn leverages the device's intra-level heterogeneity, specifically the array of available processors, as well as the range of solutions offered by the RASS solver, which allow the adoption of a swift and efficient switching mechanism between execution plans.

3.2 Workflow

Figure 1 depicts CARIn's operational flow, which is divided into the sequential *offline* and *online* phases. The offline component is responsible for constructing and resolving the device-specific MOO problem. Then, at runtime, the online component's Runtime Manager (RM) constantly monitors the application's dynamic behaviour, ensuring real-time adaptation to emergent changes. Algorithm 1 presents a comprehensive top-level overview of our framework, delineating its primary components and illustrating their main operations. These operations will be thoroughly elucidated in Section 4. The input parameters of our framework are: (a) the designated DL task(s) associated with the application, (b) the stipulated SLOs, and (c) the target device's characteristics, while the outputs consist of: (a) the set of solutions (designs $\mathcal{D}$), and (b) the switching policy SP.

The specified DL task or tasks dictate the set of models to be considered during the optimisation process. A model in CARIn is represented by the following tuple:

$$m = (arch, params, s_{\text{in}}, task, ds, pr)$$

where $arch$ is the model's architecture (*i.e.* layers and connections), $params$ are the model's trained parameters, s_{in} is the input size, $task$ is the target DL problem, ds is the name of the corresponding DL testing dataset, and pr is the numerical precision to account for quantised models.

[1]Processor here refers also to the exact configuration of a given processor, *e.g.* threads in a CPU or precision in a GPU.

Algorithm 1: CARIn's end-to-end operation.

Input: DL task(s)
Target device
Application SLOs
Output: Designs, $\mathcal{D}$
Switching Policy, SP
/* MOO Problem Formulation */
1 $\mathcal{HW} \leftarrow$ ObtainHardwareCharacteristics(Target device)
2 **if** *app type == single-DNN* **then**
3 $\quad \mathcal{X} \leftarrow$ ConstructDecisionSpace(DL task, $\mathcal{HW}$)
4 **else**
5 $\quad \mathcal{X} \leftarrow$ ConstructDecisionSpace(DL tasks, $\mathcal{HW}$)
6 **end**
7 $f_i, g_j \leftarrow$ ExtractFunctions(Application SLOs)
/* Objective Function Evaluation */
8 $f_i(x), g_j(x) \leftarrow$ EvaluateFunctions($\mathcal{X}, f_i, g_j$)
/* MOO Problem Solver */
9 $\mathcal{X}' \leftarrow \{x \mid g_j(x) \leq 0\}$ // apply constraints
10 $opt(x) \leftarrow$ CalculateOptimality($\mathcal{X}'$)
11 $\mathcal{X}'_s \leftarrow$ Sort($\mathcal{X}', opt$)
12 $\mathcal{D}$, SP $\leftarrow$ Search($\mathcal{X}'_s$)
/* Runtime */
13 **while** *true* **do**
14 $\quad c \leftarrow$ Analyse(s) // c indicates a change in resource availability
15 $\quad$ **if** *c == True* **then**
16 $\quad\quad d_{\text{new}} \leftarrow$ RM($\mathcal{D}$, SP, c) // select new design
17 $\quad$ **end**
18 **end**

The target device defines the hardware resources at the system's disposal, which are represented by the tuple:

$$hw = (ce, op(ce))$$

where $ce \in \mathcal{CE}$ is the compute engine (*i.e.* processor) performing the inference computations and $op(ce)$ is a set of options tied to the given processor, *e.g.* the number of CPU threads or the GPU's numerical precision. The tuple of tunable system parameters can be extended to capture a more detailed space, *e.g.* by including the DVFS governor selection which determines the dynamic voltage and frequency scaling policy of the device [61].

An individual model m running under the selected system parameters hw represents a single execution configuration:

$$e = \langle m, hw \rangle \in \mathcal{E} \tag{1}$$

During the *MOO Problem Formulation* stage (lines 1-7), CARIn considers every generated space of execution configurations, $\mathcal{E}_i$, in order to form the problem's decision space, $\mathcal{X}$, depending on whether the application requires single- or multi-DNN execution. At the same time, the application's SLOs delineate the MOO problem's objective functions and constraints, denoted as f_i and g_j

respectively. Once the problem is formulated for the target device, the *Objective Function Evaluation* stage evaluates each function for every $x \in \mathcal{X}$ (line 8). Following this, CARIn's *MOO Problem Solver* is poised to solve the MOO problem (lines 9-12). The functions CalculateOptimality, Sort, and Search shown in Algorithm 1, which constitute the three stages of the solver, are discussed in detail in Section 4.3.

In order for CARIn to accommodate runtime adaptation, it is important to establish a robust system for perpetually monitoring the dynamic aspects of the executing application and the state of the device itself. This ongoing vigilance enables timely recognition of abrupt alterations in operational conditions, thereby facilitating immediate corrective measures. We call this subsystem the *Runtime Manager* (RM). The output of CARIn's solving algorithm consists of a set $\mathcal{D}$ of highest-performing solutions, called designs, which are passed to RM along with the appropriate switching policy (SP). Leveraging a collection of periodically captured statistics s from the *Application*'s runtime, the RM module has the ability to discern dynamic changes in resource allocation (c in Algorithm 1) and rapidly switch to an alternative design d_{new} to effectively and robustly meet the application-level SLOs (lines 13-18).

4 MULTI-OBJECTIVE OPTIMISATION FRAMEWORK

Multi-objective optimisation (MOO) constitutes a mathematical and computational approach employed to find the best solutions or trade-offs in scenarios that involve multiple interrelated and, at times, antagonistic objectives [15, 44]. The appropriateness of a MOO framework for our problem is underscored by (a) the inherent nature of DL application SLOs, which typically comprise objectives that exhibit conflicts, and (b) the inherent attribute of MOO to yield a solution space rich in diversity, which, in turn, can enable dynamic adaptation.

4.1 MOO Problem Formulation

For CARIn's DL-based MOO formulation, we adopt the following mathematical description:

$$\begin{aligned} &\text{min/max} && f_i(x) && 1 \leq i \leq N \\ &\text{subject to} && g_j(x) = g_j(h_j(x)) \leq 0, && 1 \leq j \leq P \end{aligned}$$

where x denotes the decision variable, N is the number of objective functions, $f_i(x)$ is the i-th objective function, P is the number of inequality constraints, and $g_j(x)$ is the j-th inequality constraint, which is always a composite function of a given inner function $h_j(x)$. Note that when there is only a single objective function ($N = 1$), then the problem is reduced to single-objective optimisation (SOO). The problem's objective functions and constraints are extracted from the application's SLOs, which can be split into two categories:

- Broad SLOs: Such objectives define the problem's objective functions and come in the form of $\langle min/max, p \rangle$, where p is a DL-related performance metric. For instance, $\langle max, mIoU \rangle$ means that the mean Intersection-over-Union (mIoU) accuracy metric should be maximised for an image segmentation task. For CARIn, this objective translates to the maximisation of the objective function $f(x) = A(x) = mIoU(x)$.
- Narrow SLOs: These objectives define the problem's constraints and come in the form of $\langle min/max/avg/std/n^{\text{th}}, p, v \rangle$, which means that the minimum, maximum, average, standard deviation or n^{th} percentile value of p is bounded by a target value v. For instance, $\langle avg, L, 15 \rangle$ means that the average latency needs to be less than 15 ms, which translates to the constraint $g(x) \leq 0$, where $g(x) = g(h(x)) = g(L(x)) = \overline{L}(x) - 15$.

Given that both types of objectives concern the same set of performance metrics, it follows that both the objective and inner functions, $f_i(x)$ and $h_j(x)$, share a common function space,

denoted by $\mathcal{F}$, which encompasses the entirety of available functions associated with various DL performance metrics. For this reason, in cases where the application defines constraints without explicitly specifying objective functions, CARIn can duly regard all specified inner functions $h_j(x)$ as objective functions as well.

4.1.1 Single-DNN Setting. When there is only one DL task to optimise, the decision variable x is a single execution configuration e, as defined in Equation 1. Therefore, the execution configuration space $\mathcal{E}$ effectively transforms into the decision space $\mathcal{X}$:

$$x_{\text{single}} = e = \langle m, hw \rangle \in \mathcal{X}_{\text{single}} = \mathcal{E}$$

For the objective functions, CARIn leverages the following DNN-specific performance metrics:

- Size (S): Size is conventionally represented by either the total count of parameters within the neural network or the physical file size of the model stored in memory.
- Workload (W): This metric is typically measured in terms of numerical operations, such as floating-point operations (FLOPs) or multiply-accumulate operations (MACs).
- Accuracy (A): Accuracy is contingent upon the specific DL task in question, *e.g.* top-1 accuracy for classification tasks or exact match for question answering tasks.
- Latency (L): Latency delineates the temporal lag between the transmission of input data to the DNN model and the reception of the corresponding output. It is quantified in units of milliseconds or seconds.
- Throughput (TP): Throughput provides an indication of the model's real-time processing capabilities and is computed as the total number of input samples (batch size), divided by the total inference latency. This metric is denominated in samples per second (*e.g.* images per second when images constitute the inputs).
- Energy Consumption (E): This metric is of paramount importance for the evaluation of the energy efficiency of DNN applications in resource-constrained environments and is measured in energy units, such as watt-hours or joules.
- Memory Footprint (MF): Memory footprint encapsulates the extent of random-access memory (RAM) required for the loading and execution of a DNN. It is traditionally assessed in terms of memory size units, such as megabytes (MB) or gigabytes (GB).

Overall, the set of potential objective functions in single-DNN cases is denoted as:

$$\mathcal{F}_{\text{single}} = \{S, W, A, L, TP, E, MF\}$$

collectively empowering a multifaceted assessment of DNN models and providing a holistic understanding of their performance across diverse dimensions.

It is important to recognise that the latency and energy consumption metrics are subject to inherent fluctuations when executing DNNs on mobile devices. These fluctuations can arise due to various factors, including device load, temperature, input values, and other environmental variables (see Section 4.3.2). As a result, relying on a single, instantaneous value may not provide a robust and representative assessment of system performance. To account for these fluctuations, CARIn considers statistical measures, such as the average or maximum energy consumption or the variance of the latency, as objective functions.

4.1.2 Multi-DNN Setting. When there are M independent DNNs to optimise jointly, the decision variable x comprises of M distinct execution configurations e_i, $1 \leq i \leq M$. Hence, the decision space $\mathcal{X}$ is an M-dimensional space, where each component of the decision variable can separately take values in the corresponding execution configuration space $\mathcal{E}_i$:

$$x_{\text{multi}} = \{e_1, \ldots, e_M\} = \{\langle m, hw \rangle_1, \ldots, \langle m, hw \rangle_M\} \in \mathcal{X}_{\text{multi}} = \mathcal{E}_1 \times \ldots \times \mathcal{E}_M$$

The array of potential objective functions is expansively broadened to encompass an additional triad of performance metrics pertaining to parallel execution [12, 60]:

- Normalised Turnaround Time (NTT): NTT serves as a quantifier of the perceived execution slowdown during multi-DNN execution. The NTT for the i-th DNN is computed as:

$$NTT_i = \frac{L_i^{\mathrm{M}}}{L_i^{\mathrm{S}}}$$

  where L_i^{S} and L_i^{M} are the average latencies of the i-th DNN under the single- and multi-DNN modes. NTT_i is a value greater than or equal to 1, with lower values indicating superior performance. For the sake of standardisation across models, it is common practice to calculate the average or maximum NTT.
- System Throughput (STP): STP quantifies the accumulated single-DNN progress under multi-DNN execution and is computed as:

$$STP = \sum_{i=1}^{M} NP_i = \sum_{i=1}^{M} \frac{1}{NTT_i} = \sum_{i=1}^{M} \frac{L_i^{\mathrm{S}}}{L_i^{\mathrm{M}}}$$

  where NP_i is the normalised progress of the i-th DNN. Its maximum magnitude is M, with higher values signifying enhanced performance.
- Fairness (F): The concept of fairness in a multi-DNN execution environment is contingent upon the equitable relative progress experienced by co-executing DNNs, in comparison to their single-DNN execution counterparts. Fairness, as denoted herein, is quantified as the minimum ratio of relative normalised progress rates observed among any two DNNs concurrently operating within the system:

$$F = \min_{i,j} \frac{NP_i}{NP_j}$$

  This metric adheres to a higher-is-better paradigm with values within the range $[0, 1]$, where 0 signifies an absence of fairness and 1 perfect fairness.

As a consequence, we augment `CARIn`'s objective function set to encompass both single-DNN metrics, which pertain to individual tasks or DNNs, and multi-DNN metrics, which characterise the collective performance of the entire system during concurrent execution:

$$\mathcal{F}_{\text{multi}} = \{S_i, W_i, A_i, L_i, TP_i, E_i, MF_i\} \cup \{STP, NTT, F\}, 1 \leq i \leq M$$

4.2 Objective Function Evaluation

Upon the formulation of the device-specific MOO problem, it becomes necessary to assess each objective function across the entire set of decision variables $x \in \mathcal{X}$. Assessing these functions is straightforward for certain objectives; however, it presents challenges for device-dependent functions like E and MF, and those relying on latency, including L, TP, STP, NTT, and F. The approach adopted by `CARIn` for this evaluation involves the profiling of functions on individual target devices. In practical terms, this entails the deployment of all candidate models on each target device, followed by the measurement of each device-reliant objective function for all feasible model-processor combinations. We acknowledge that this procedure, albeit comprehensive, is inherently time-consuming and, in many instances, such as in multi-DNN cases, infeasible for seamless integration into real-world scenarios and practical applications. However, the optimisation of the evaluation process itself does not constitute a primary objective of this work. We extensively discuss potential enhancements of this aspect in Section 8.

4.3 MOO Problem Solver

Following the formulation of the problem and the evaluation of objective functions, the conclusive stage involves resolving the optimisation problem. The initial step of the optimisation process is to apply the problem's constraints. Consequently, the decision variables are bound to the constrained decision space $\mathcal{X}'$, defined as:

$$\mathcal{X}' = \{x \mid g_j(x) \leq 0, \forall j\}$$

MOO problems are frequently addressed using evolutionary algorithms, such as NSGA-II, SMS-EMOA, and MOEAD, or swarm-based algorithms, such as Ant Colony Optimisation (ACO) and Particle Swarm Optimisation (PSO) [52]. These algorithms systematically explore the decision variable space to discover the Pareto frontier, which represents the optimal trade-offs among conflicting objectives. While these algorithms excel in identifying the optimal execution configuration, we acknowledge that potential runtime issues may either alter the solution space, consequently affecting the Pareto frontier of a MOO problem, or introduce new constraints that were not considered during the problem's formulation. Consequently, to address the potential decline in performance, it becomes imperative to rerun these algorithms whenever a runtime issue arises, however, such repetitive executions are impractical for real-life applications and systems.

To address this challenge, we introduce a runtime-aware sorting and search algorithm, denoted as `RASS`, whose primary goal is to solve a device-specific MOO problem once, while concurrently addressing potential future runtime challenges. To achieve this, `RASS` considers both non-dominated and dominated solutions in a predictive manner, estimating the impact of possible runtime issues. In addition to providing the initial solution d_0, `RASS` also yields a set of supplementary runtime designs d_i, which serve as a proactive measure for runtime adaptation, *i.e.* in instances where the currently employed design encounters performance issues. This approach alleviates the need for repetitive executions of optimisation algorithms.

The operation of `RASS` involves a sorting stage followed by a search stage. To accommodate both non-dominated and dominated solutions, our solving algorithm initially sorts candidate solutions according to their optimality (Section 4.3.1), a metric quantifying the distance from the problem's utopia point. Subsequently, based on this sorting, `RASS` identifies a set of solutions (Section 4.3.4) representing the various execution plans of the application which correspond to possible runtime issues (Section 4.3.2), along with a switching policy facilitating prompt transitioning between them (Section 4.3.3) for the RM module.

4.3.1 Optimality. To quantify optimality for a given candidate solution $x \in \mathcal{X}'$, we first calculate the weighted Mahalanobis distance between the solution's objective vector, which is defined as $f(x) = [f_1(x), f_2(x), ..., f_n(x)]$, and the utopia point, represented as $up = [up_1, up_2, ..., up_n]$:

$$d(x) = \sqrt{\sum_{i=1}^{n} w_i^2 \frac{[f_i(x) - up_i]^2}{s_i^2}}$$

where w_i is the user-supplied weight for the i-th objective, s_i^2 is the calculated variance of the i-th objective, and each component of the utopia point depends on the corresponding objective function:

$$up_i = \begin{cases} \max f_i, & \text{if } f_i \in \{A, TP, STP, F\} \\ \min f_i, & \text{if } f_i \in \{S, W, L, E, MF, NTT\} \end{cases}$$

By utilising the Mahalanobis distance, we effectively accommodate the disparate scales of the diverse objectives. Consequently, optimality could also be regarded as a metric of fairness for the problem's objective functions. However, it is important to acknowledge that these functions may

carry distinct significance for the problem, hence, we afford users the opportunity to define weights, thereby introducing a formal mechanism for enabling tailored optimisation strategies. Notably, the calculated distances range within the interval $[0, d_{\max}]$, where the maximum distance is:

$$d_{\max} = \sqrt{\sum_{i=1}^{n} w_i^2 \frac{(\max f_i - \min f_i)^2}{s_i^2}}$$

This factor necessitates the use of normalisation, which results in the distance being confined to the $[0, 1]$ range:

$$d_s(x) = \frac{d(x)}{d_{\max}}$$

The optimality metric for each $x \in \mathcal{X}'$ can then formally be defined as the reciprocal of the scaled weighted Mahalanobis distance, thus, its range extends from $[1, +\infty)$:

$$opt(x) = \frac{1}{d_s(x)}$$

Utilising these values, the candidate solutions are sorted in descending order, resulting in the creation of the sorted decision space $\mathcal{X}_s$.

4.3.2 Runtime Challenges. During the runtime of the application, a multitude of dynamic alterations in the device's resource availability may occur. These fluctuations impact our problem formulation in different ways, thus necessitating targeted approaches for their management. `CARIn` focuses on addressing two main challenges, regarding the processors and memory of the target device:

- **Processor Overload or Overheating:** Processor-related concerns manifest when the processor at use is continuously subjected to sustained processing demands exceeding its peak processing capacity, primarily due to resource-intensive computational tasks. The protracted imposition of such an overload condition may subsequently lead to overheating, which means the escalation of the SoC's temperature to a critical and potentially harmful level. Overheating may also result from insufficient cooling mechanisms or other impediments hindering the effective dissipation of heat by the SoC. As a protective measure against potential harm, mobile SoCs are equipped with thermal throttling capabilities, which are activated when temperatures exceed predefined thresholds. Thermal throttling encompasses the deliberate reduction of the processor's clock speed and performance to mitigate heat generation and maintain a safe temperature range. The intricate interplay between processor overload and overheating significantly impacts performance and power consumption, underscoring the significance of diligent management and effective mitigation strategies.
- **Variability in RAM Utilisation:** Owing to the multifaceted nature of mobile devices, the utilisation of Random Access Memory (RAM) is also characterised by dynamic fluctuations. Within the execution scope of an application, numerous ancillary applications, processes, or services continually initiate and terminate in the background, potentially culminating in an unforeseen saturation of RAM capacity. Consequently, this phenomenon may precipitate performance-related challenges, encompassing lag, application crashes, and an overall deceleration of device functionality. Furthermore, the perpetual management of excessive RAM consumption may also entail elevated power consumption, thereby engendering consequential ramifications.

4.3.3 Model/Processor Switching. In response to runtime fluctuations, `CARIn`'s RM adopts a strategic approach that involves alterations to either the model (change model, CM), processor (CP), or both (CB) within the current execution plan. These three fundamental adjustments serve

as effective measures for mitigating the challenges encountered during runtime. To this end, we introduce a prioritisation scheme. In the case of processor-related phenomena, CARIn prioritises transferring DNN execution from the currently used processors to inactive ones (CP or CB). This transition allows the overloaded or overheated processor to dissipate excess heat and gradually restore its performance. In cases where migration is not a viable option, such as in devices limited solely to CPU usage or multi-DNN scenarios where all processors are occupied, CARIn employs an alternative approach which involves replacing the current model with one of reduced computational workload (CM). Conversely, addressing the memory-related issue involves transitioning to a more compact model either on the same (CM) or a different processor (CB).

4.3.4 Design Selection & Switching Policy. A primary principle guiding the design of RASS is to ensure low complexity in order to facilitate rapid switching. This objective manifests in the generation of a relatively small number of designs, which in turn offers two additional distinct benefits: firstly, minimise storage requirements for the models; and secondly, maintain a concise switching policy comprising only a limited number of transition rules. For RM to determine the appropriate timing to transition to a new execution plan, several system parameters, related to (a) the workload distribution across processors and (b) the aggregate memory utilisation, need to be continuously monitored. These parameters are represented by the boolean variables c_{ce} and c_{m}, indicating the presence of issues pertaining to a processor ce and the memory, respectively.

The first step in identifying the solutions to the problem involves identifying the sets of different model-to-processor mappings viable for processor switching, *i.e.* for reallocating DL execution to idle processors. Symbolising the number of these sets as T, in consideration of RASS's need for simplicity, if $T > 3$, we retain only the top three sets, corresponding to the highest attained optimality scores. Next, we partition our sorted decision space $\mathcal{X}_s$ into T distinct subspaces $\mathcal{X}_i$, each corresponding to specific model-to-processor mappings and arranged in descending order of observed optimality. Regarding processor-related phenomena, we select designs associated with the highest optimality score within each set:

$$d_i = \mathcal{X}_i[0], i = 0, \ldots, T-1$$

For the memory-related issue, we extract the solution with the smallest memory footprint:

$$d_{\mathrm{m}} = \underset{x}{\operatorname{argmin}}\, MF(x),\ \ x \in \mathcal{X}_i,\ \ i = 0, \ldots, T-1$$

Lastly, we extract complementary designs for two extreme (highly improbable) scenarios. The first one arises when all related processors present an issue, while the memory does not, prompting the extraction of the solution with the lightest workload:

$$d_{\mathrm{w}} = \underset{x}{\operatorname{argmin}}\, W(x),\ \ x \in \mathcal{X}_i,\ \ i = 0, \ldots, T-1$$

and the second scenario surfaces when both the processors and memory encounter issues simultaneously, necessitating the identification of the solution that strikes the optimal balance between memory usage and workload among d_m and d_w:

$$d_{\mathrm{wm}} = \begin{cases} d_{\mathrm{w}}, & \text{if } C\left(MF(d_{\mathrm{w}}), W(d_{\mathrm{w}})\right) < C\left(MF(d_{\mathrm{m}}), W(d_{\mathrm{m}})\right) \\ d_{\mathrm{m}}, & \text{else} \end{cases}$$

where we use the normalised sum to compute the cost function C. Collectively, the set of designs is denoted as:

$$\mathcal{D} = \{d_i, d_{\mathrm{m}}, d_{\mathrm{w}}\}, i = 0, \ldots, T-1$$

therefore RASS can generate a maximum of five designs for a MOO problem, since $T \leq 3$.

After establishing the set of designs, RASS's final step involves crafting the rule-based switching policy, which serves as a reference for the RM module, guiding its decision-making process each time the boolean variables c_{ce} or c_{m} undergo a change in value. With the aim of ensuring simplicity and conciseness in the rule set, we ensure that the selection of a new design is contingent solely upon the state of the environmental variables and independent of the presently employed design. The rationale behind the construction of the rules is deliberately straightforward, as demonstrated in Section 7.2, where two representative use cases are presented and analysed.

5 IMPLEMENTATION

CARIn is implemented in Java for the Android operating system. Its primary integration leverages the TensorFlow Lite (TFLite) package in its nightly build to facilitate on-device DNN execution, as well as its delegates to access mobile accelerators. The concurrent execution of multiple DNNs is achieved through the utilisation of the `java.util.concurrent` Java package.

In order to create the model suite used for our framework's evaluation, *i.e.* for model retrieval, training, and preparation, TensorFlow (v2.12.0) was employed in Python. Our framework seamlessly interfaces with TensorFlow Hub and Hugging Face model repositories, which allows researchers to easily access and experiment with a wide range of pre-trained models on publicly available datasets for their specific use cases. Additionally, the TFLite Converter's optimisation module was utilised to apply post-training quantisation to the models in order to enhance inference speed, efficiency and accelerator compatibility.

Regarding the objective function evaluation process, a diverse set of tools and libraries is employed. For accuracy assessments, we use custom evaluation scripts, as well as TFLite's image classification evaluation tool for the ImageNet ILSVRC 2012 task. Furthermore, to comprehensively capture the model's computational complexity and resource requirements, we use the `tflite` Python package to count model parameters and FLOPs. Lastly, to assess the on-device performance of the models, we employ the C++-based TFLite benchmark tool[2]. This tool offers a comprehensive suite of measurements, encompassing execution time, memory utilisation, and other pertinent metrics, thereby providing a robust evaluation of the models' real-world performance characteristics.

6 EXPERIMENTAL METHODOLOGY

In this section, we present the experimental methodology that underpins our research, offering insight into the comprehensive approach we have undertaken to investigate our study's core objectives. We have structured this methodology into several key subsections, each addressing a crucial aspect of our experimental design. Figure 2 depicts the toolflow used to conduct our experiments.

6.1 Quantisation

CARIn embraces *post-training quantisation* as one of the most simple and mobile-friendly compression methods presently available, with benefits not only in model size, but also in latency and memory requirements. Additionally, quantisation becomes indispensable for the execution of DNNs within DSPs or NPUs designed to primarily support integer models [22], thus unlocking complete compatibility with mobile accelerators. Notably, additional methods which also introduce trade-offs between accuracy and complexity, such as weight pruning or clustering, are orthogonal to our framework and amenable to integration. The potential synergy resulting from the combined application of various compression techniques merits further investigation.

[2]https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmark

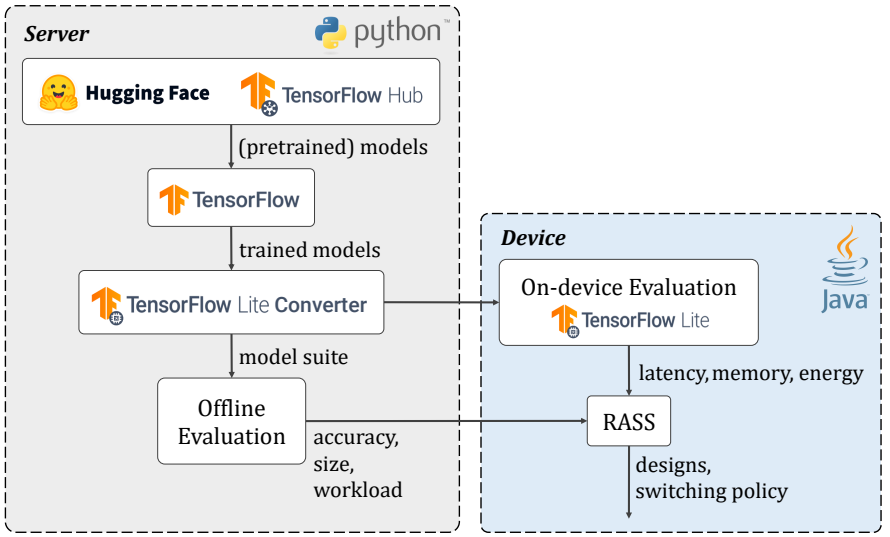

Fig. 2. Toolflow for the evaluation of CARIn.

Driven by the capabilities of the TFLite Converter, CARIn currently incorporates four distinct quantisation techniques, namely half-precision floating-point (FP16), 8-bit dynamic range (DR8), 8-bit fixed-point with float fallback (FX8) and full 8-bit fixed-point (FFX8). Table 1 enumerates the numerical types associated with inputs, outputs, weights, and activations for both the original (FP32) and the quantised models. It is important to note that the data type of the weights defines the storage requirements of the model. Specifically, FP16 quantisation leads to a 2× reduction in model size, while the remaining schemes (DR8, FX8, and FFX8) yield a 4× reduction in size. The operational procedures of these quantisation schemes are elucidated as follows:

- FP16, by default, employs 16-bit floating-point computations, yet it possesses the flexibility to revert (fall back) to 32-bit floating-point (FP32) calculations when the hardware lacks support for 16-bit arithmetic. In such instances, the weights undergo a dequantisation process to 32-bit before the first inference. Concurrently, the activations are stored in 32-bit format. The most common processors with native support for FP16 operations are mobile GPUs.
- In the case of DR8, weights are represented with 8 bits, while activations persistently remain in FP32. Nevertheless, certain activations may undergo dynamic quantisation during inference, utilising quantised kernels for faster execution. The utilisation of fixed-point arithmetic, whenever feasible, may result in reduced computation times compared to relying solely on floating-point arithmetic, contingent on the specific model's characteristics.
- FX8, analogous to FP16, represents an 8-bit equivalent and operates with integer kernels as the default mode of execution. However, it retains the ability to utilise 32-bit operators when integer implementations are unavailable on the given hardware (floating-point fallback). Importantly, in this scheme, the converted model maintains inputs and outputs in floating-point format, allowing the model itself to determine the quantisation parameters to minimise accuracy loss.
- FFX8 enforces full integer quantization for all components of the model, encompassing weights, activations, operations, inputs, and outputs. This stringent quantisation scheme guarantees compatibility with integer-only devices and accelerators, such as microcontrollers, DSPs, and NPUs.

Table 1. Quantisation Schemes

Scheme	Inputs & Outputs	Weights	Activations
FP32	fp32/int32/int64	fp32	fp32
FP16	fp32/int32/int64	fp16	fp16/fp32
DR8	fp32/int32/int64	int8	fp32
FX8	fp32/int32/int64	int8	int8/fp32
FFX8	int8/int32	int8	int8

6.2 Application Scenarios, Models and Tasks

In the following part, we outline four discrete application use cases, which form the bedrock of our experimental evaluation. These application scenarios represent diverse real-world settings in which our research findings will be tested and validated, providing valuable insights into the effectiveness of our proposed formulation and methodology.

Notably, the first two scenarios pertain to the execution of a single DNN, while the later two involve the execution of multiple DNNs in parallel, affording us the opportunity to assess performance outcomes in instances where dependencies among multiple DNNs exist. This dichotomy is instrumental in affording a comprehensive evaluation of our methodology's versatility, applicability, and scalability. We define specific SLOs for each use case and showcase the list of models to be considered during evaluation, along with their device-independent evaluation, which includes (a) the accuracy of both the original and quantised variants, (b) the computational workload in FLOPs, and (c) the model size in terms of parameters.

6.2.1 Use Case #1 (UC1). In our first single-DNN scenario, we examine the practical application of **real-time image classification**. In this setting, the camera of a mobile device continuously captures frames that require prompt and accurate recognition. The term "real-time" is qualified by a temporal restriction mandating that the maximum permissible latency is 41.67 ms, underscoring the necessity to uphold a recognition rate of no less than 24 frames per second (FPS). The principal objectives of this use case encompass the joint maximisation of accuracy and throughput. Mathematically, this MOO problem comprises two objective functions and a single constraint:

$$\begin{aligned} \max \quad & A(x), TP(x) \\ \text{subject to} \quad & \max L(x) \leq 41.67 \text{ ms} \end{aligned}$$

For UC1 we used the ImageNet-1k dataset [47]. Table 2 lists the eight models under consideration, which are drawn from four distinct families: MobileNets [48], EfficientNets [57], RegNets [46], and MobileViTs [39]. The rationale behind this extensive model selection is to ensure a well-rounded exploration of compact and mobile-friendly architectures that span a broad spectrum, encompassing both conventional CNNs and emerging Transformer-based models. Each of these architectural paradigms exhibits unique characteristics and design principles.

It is worth noting that we also contemplated the inclusion of higher-accuracy models, such as NASNet and ConvNeXt, in our analysis. However, our assessment revealed that these models failed to meet the stipulated latency constraint. Consequently, they were excluded from our study to maintain adherence to the predefined performance criteria.

6.2.2 Use Case #2 (UC2). In our second single-DNN scenario, we study the task of **text classification**, with a particular emphasis on the memory requirements of the models. To this end, we impose a memory constraint, stipulating that the executing DNN's maximum memory footprint must not

Table 2. UC1 Models

DL Task	Architecture	Image Size	FLOPs	#Params	Top-1 Accuracy (%)				
					FP32	FP16	DR8	FX8	FFX8
Image Classification on ImageNet-1k	MobileNet V2 1.0	224x224	0.60 G	3.49 M	71.92	71.96	71.65	71.28	71.26
	RegNetY 008	224x224	1.60 G	6.25 M	74.28	74.28	74.18	74.45	74.47
	MobileViT XS	256x256	2.10 G	2.31 M	74.61	74.61	-	-	-
	EfficientNet Lite0	224x224	0.77 G	4.63 M	75.19	75.23	75.14	75.09	75.11
	MobileNet V2 1.4	224x224	1.16 G	6.09 M	75.66	75.68	75.47	75.41	75.45
	RegNetY 016	224x224	3.23 G	11.18 M	76.76	76.76	76.62	76.92	76.84
	MobileViT S	256x256	4.06 G	5.57 M	78.31	78.30	-	-	-
	EfficientNet Lite4	300x300	5.11 G	12.95 M	80.81	80.80	80.78	80.69	80.71

Table 3. UC2 Models

DL Task	Architecture	Sequence Length	FLOPs	#Params	Top-1 Accuracy (%)				
					FP32	FP16	DR8	FX8	FFX8
Text Classification on Emotions	BERT-L2-H128	64	0.05 G	4.31 M	92.10	92.10	91.90	91.75	91.75
	XtremeDistil-L6-H256	64	0.63 G	12.57 M	93.30	93.30	93.20	93.15	93.20
	MobileBERT-L24-H512	64	2.66 G	24.33 M	93.80	93.80	93.80	93.65	94.10

exceed 90 MB. The objectives of this use case revolve around three critical factors: minimising the average latency, reducing the model size, and maximising accuracy. Mathematically, this MOO problem encompasses three objective functions and a singular constraint:

$$\begin{aligned} \min \quad & \overline{L}(x), S(x) \\ \max \quad & A(x) \\ \text{subject to} \quad & MF(x) \leq 90 \text{ MB} \end{aligned}$$

For UC2, we obtained three pre-trained Transformer models on various large datasets, including Reddit comments and 2ORC citation pairs, and subsequently fine-tuned them on Emotions [50], a dataset comprising of English Twitter messages which is employed for the task of classifying input sequences into six distinct emotions. We adopted the dataset's split configuration, which allocated 16k samples for training, 2k for validation, and 2k for testing. The reported top-1 accuracy corresponds to the dataset's test set. The selected models, detailed in Table 3, encompass the traditional BERT architecture in a lightweight version, alongside two mobile-grade models: XtremeDistil [41] and MobileBERT [55]. The letter "L" in each model's name stands for the number of Transformer layers and "H" stands for the hidden dimension. In preparation for training, we further optimised BERT and XtremeDistil to enhance mobile-friendliness by replacing the GELU activation function with ReLU and substituting Layer Normalisation with Batch Normalisation [42].

6.2.3 Use Case #3 (UC3). In our first multi-DNN scenario, we employ two DNNs for the purpose of **scene recognition**. One DNN is dedicated to processing and classifying images, while the other can process audio data in order to identify sounds from the device's surroundings. These models operate concurrently, running in parallel, and their outputs are collectively utilised to determine the specific scene within which the mobile device is situated.

In this scenario, we seek to minimise both the average latency and its standard deviation, while simultaneously maximising the attained accuracy. We impose two latency constraints for both tasks, mandating that (a) the average latency remains consistently below 100 ms to ensure near-real-time responsiveness, and (b) the standard deviation of latency stays below 10 ms for minimal fluctuations. The inclusion of the latency's standard deviation aims to minimise performance variability, which

Table 4. UC3 Models

DL Task	Architecture	Input Size	FLOPs	#Params	Accuracy				
					FP32	FP16	DR8	FX8	FFX8
Scene Classification on MIT Indoor Scenes	EfficientNet Lite0	224x224	0.59 G	3.44 M	69.78	69.70	68.96	69.18	69.18
	EfficientNet Lite2	260x260	1.51 G	4.87 M	76.72	76.72	77.16	77.69	77.54
	EfficientNet Lite4	300x300	4.57 G	11.76 M	79.33	79.33	79.18	79.78	79.48
Audio Classification on AudioSet	YAMNet	15600	0.14 G	3.75 M	0.3756	0.3757	0.3620	-	-

still constitutes a withstanding challenge for on-device inference [66]. Mathematically, this MOO problem is formulated as:

$$\begin{aligned} \min \quad & \overline{L}_i(x), \sigma_{L_i}(x) \\ \max \quad & A_i(x), \qquad\qquad i = 1, 2 \\ \text{subject to} \quad & \overline{L}_i(x) \leq 100 \text{ ms}, \sigma_{L_i}(x) \leq 10 \text{ ms} \end{aligned}$$

Table 4 presents the models for each task. For the vision task, we fine-tuned three EfficientNet Lite models on the MIT Indoor Scenes dataset [45], which includes 67 classes and 100 images per class (80 for training and 20 for testing). We report the top-1 accuracy on the test set. For the audio task, we use YAMNet, which is trained on the AudioSet dataset [14] for multi-label classification. The dataset consists of 521 sound events (classes) and 18k samples. We report the mean average precision (mAP) on the validation set. YAMNet's input waveform can vary in length. In our experiments, we use the model's minimum possible length of 975 ms, which corresponds to 15600 input samples and a total workload of 0.14 GFLOPs.

6.2.4 Use Case #4 (UC4). In our second multi-DNN scenario, we deploy three distinct models designed for **facial attribute prediction** tasks, namely gender, age and ethnicity estimation. These models are conceptualised as the second stage of a face detection and attribute prediction pipeline, wherein they operate concurrently on the same set of input images. As such, it is imperative for these models to adhere to stringent latency constraints to ensure minimal impact on the overall pipeline. UC4's objectives revolve around the collective optimisation of five key metrics for each model, specifically average latency, standard deviation of latency, size, memory footprint and accuracy, all while adhering to a maximum latency threshold of 10 ms. Formally:

$$\begin{aligned} \min \quad & \overline{L}_i(x), \sigma_{L_i}(x), S_i(x), MF_i(x) \\ \max \quad & A_i(x), \qquad\qquad i = 1, 2, 3 \\ \text{subject to} \quad & \max L_i(x) \leq 10 \text{ ms} \end{aligned}$$

In UC4, the training data are sourced from the UTKFace dataset [81]. To ensure relevance to real-time applications, the dataset is filtered to retain samples corresponding to the age range of 18-75. Consequently, the utilised dataset comprises 18.6k facial images, partitioned into training, validation, and testing sets with a ratio of 72/8/20, respectively. The employed models leverage MobileNetV2 as the backbone architecture, extracting 576 features of size 4×4, which are used for predicting the outcomes across the three distinct facial attribute prediction tasks. Notably, UC4 stands as the sole task within our study that incorporates batching during inference. Specifically, the models are configured with a batch size of 4, a choice motivated by the common case where the preceding face detection component identifies multiple faces within a single image. Table 5 details the attained accuracy metrics for each task on the filtered dataset's test set: binary accuracy

Table 5. UC4 Models

DL Task	Architecture	Image Size	FLOPs	#Params	Accuracy				
					FP32	FP16	DR8	FX8	FFX8
Facial Attribute Prediction on UTKFace	GenderNet-MNV2	62x62	0.04 G	0.66 M	95.12	94.95	94.90	94.79	94.90
	AgeNet-MNV2	62x62	0.04 G	0.66 M	5.976	5.974	5.964	5.947	5.923
	EthniNet-MNV2	62x62	0.04 G	0.66 M	78.17	78.04	78.55	79.30	79.14

Table 6. Target Devices

Device	Google Pixel 7	Samsung Galaxy S20 FE	Samsung Galaxy A71
Launch	2022, October	2020, October	2020, January
SoC	Tensor G2	Exynos 990	Snapdragon 730
CPU	2×2.85 GHz Cortex-X1 2×2.35 GHz Cortex-A76 4×1.80 GHz Cortex-A55	2×2.73 GHz Exynos M5 2×2.50 GHz Cortex-A76 4×2.00 GHz Cortex-A55	2×2.20 GHz Kryo 470 Gold 6×1.80 GHz Kryo 470 Silver
GPU	Mali-G710 MP7 @850 MHz	Mali-G77 MP11 @800 MHz	Adreno 618 @700 MHz
NPU	Tensor Processing Unit	✓	Hexagon Tensor Accelerator
RAM	8 GB @3200 MHz	6 GB @2750 MHz	6 GB @1866 MHz
TDP	7 W	9 W	5 W

for gender recognition, mean absolute error for age recognition, and top-1 accuracy for ethnicity recognition across 5 output classes.

6.3 Mobile Devices

In our study, we have selected three smartphones for our evaluation: Google Pixel 7, Samsung Galaxy S20 FE, and Samsung Galaxy A71. These devices have been deliberately chosen to represent distinct categories within the modern mobile phone landscape. A71 serves as an archetype of a mid-tier device, while S20 and P7 exemplify the high-end category, showcasing state-of-the-art features and cutting-edge technology. A detailed overview of the specifications and processing capabilities of these smartphones is shown in Table 6.

Each of the three devices is equipped with its own NPU. Concretely, P7 incorporates a custom mobile-oriented Tensor Processing Unit (TPU), S20 features the EDEN API, which grants access to the Exynos NPU for fixed-point models and specialised GPU kernels for floating-point models, and lastly, A71 hosts the Hexagon Tensor Accelerator (HTA), a dedicated compute engine for fixed-point CNNs. Additionally, it is noteworthy that among these three devices, only A71 offers access to the device's DSP for DNN inference. Therefore, we result in the following compute engine sets for each device:

$$\mathcal{CE}_{\text{P7}} = \mathcal{CE}_{\text{S20}} = \{\text{CPU}, \text{GPU}, \text{NPU}\}$$
$$\mathcal{CE}_{\text{A71}} = \{\text{CPU}, \text{GPU}, \text{NPU}, \text{DSP}\}$$

6.4 Profiling Details

In this section, we present the available configuration options for each compute engine, $op(ce)$, within the context of an execution plan's tunable hardware parameters, which are employed by `CARIn` during the profiling phase of the device-specific objective functions. In the case of CPUs, we have the capability to tune the number of threads employed for multithreading and utilise the XNNPACK delegate, which serves as a back-end for the CPU, leveraging the XNNPACK library to provide highly optimised implementations for 32- and 16-bit floating-point computations, as

well as symmetrically quantised DNN operations. Since all the devices under consideration are equipped with 8 CPU cores, the set of tunable options can be defined as follows:

$$op(\text{CPU}) = \{N_{\text{threads}}, \text{XNNPACK}\}$$

where $N_{\text{threads}} = \{1, 2, 4, 8\}$ and XNNPACK = {TRUE, FALSE}, resulting in 8 distinct CPU execution combinations. On the other hand, for GPUs and NPUs, `CARIn` exclusively employs fp16 arithmetic when feasible, as it offers reduced latency without compromising accuracy:

$$op(\text{GPU}) = op(\text{NPU}) = \{\text{precision} = \text{fp16}\}$$

Lastly, it should be noted that the DSP does not expose any configurable parameters, and thus its set of options can be defined as an empty set:

$$op(\text{DSP}) = \{\}$$

In terms of the profiling process, we initiate each execution configuration with 5 warm-up runs to stabilise the target processor's performance and reduce variability. Subsequently, to gather statistically significant latency and energy consumption values, we execute each experiment 100 times. Lastly, to maintain consistent device temperatures and mitigate the risk of overheating, we incorporate a device idle period of 2 minutes prior to commencing the next set of runs.

7 RESULTS

This section presents the outcomes of our comprehensive evaluation of `CARIn`. Our findings provide valuable insights into the effectiveness of our framework in mitigating the challenges stemming from device heterogeneity and runtime fluctuations across both single- and multi-DNN scenarios, while concurrently meeting predetermined SLOs.

7.1 Designs

Our initial assessment focuses on evaluating the performance of `CARIn`'s designs within each available state, *i.e.* single processor or combinations of processors for single- and multi-DNN applications respectively.

7.1.1 Comparison Methods. To comprehensively evaluate `CARIn`'s performance against existing methodologies, we employ three simple and empirical baselines and additionally compare against our earlier work, OODIn. Our baseline methods are formulated upon empirical observations to offer a fundamental level of performance and aid in setting a minimum performance expectation in real-world applications.

- *Single-architecture baseline:* The effectiveness of `CARIn` is contrasted with the traditional approach of considering a single model architecture, even if it is also accompanied by its quantised versions. This paradigm typically revolves around the selection of the model with the highest accuracy, optimal memory efficiency, compact size, and other relevant criteria.
- *Transferred baseline:* To assess the extent to which `CARIn` addresses device heterogeneity, we utilise the transferred baseline, where the MOO problem is solved on a specific device, and the resultant designs are then applied to different devices. This baseline, being device-agnostic, overlooks the inherent characteristics and limitations of individual devices.
- *Multi-DNN-unaware baseline:* The third baseline assesses the efficacy of our framework in handling concurrent model executions, particularly its capability to generate optimal model-to-processor mappings for multi-DNN workloads. The multi-DNN-unaware baseline dissects a multi-DNN MOO problem into M single-DNN uncorrelated problems, solves each one independently and then combines the solutions.

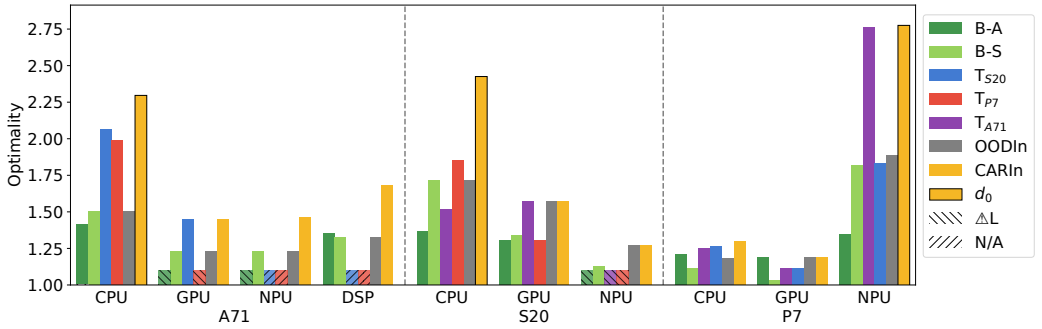

Fig. 3. UC1 evaluation.

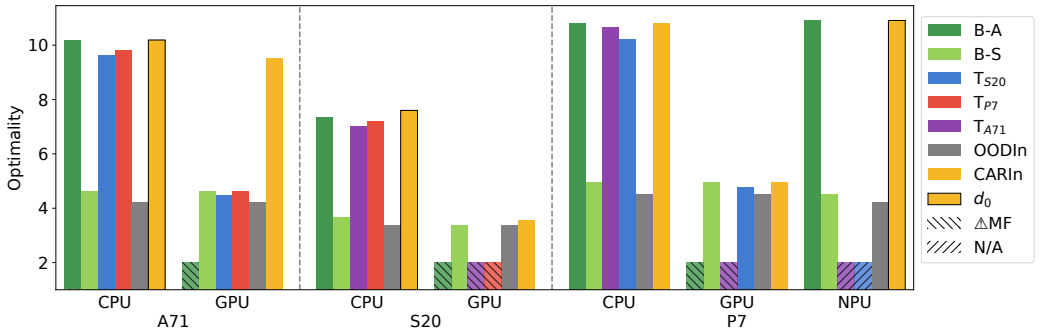

Fig. 4. UC2 evaluation.

- *OODIn [61]:* In our prior research, we utilised the weighted sum method as a means to address MOO problems. More precisely, OODIn aims to maximise the weighted sum obtained from the normalised objective functions. This approach fails to account for the inherent scale discrepancies among the diverse objective functions, particularly evident in DL metrics. While the utilisation of assigned weights may potentially mitigate this limitation, it necessitates prior knowledge of the statistical characteristics of the functions involved. When dealing with multi-DNN configurations, OODIn would operate as the multi-DNN-unaware baseline presented above, differing only in its utilisation of the weighted sum method instead of computing optimalities.

7.1.2 Single-DNN Execution. Figures 3 and 4 delineate the benefits of `CARIn` in relation to the optimality metric for the two single-DNN use cases. We compare against two single-architecture baselines, specifically using the model with the highest accuracy (best accuracy, B-A) and the model with the smallest size (best size, B-S), the transferred baselines from the other two devices, collectively designated as T_{A71}, T_{S20} and T_{P7}, and OODIn. The initial designs d_0 for each device are prominently indicated, affirming the presence of device heterogeneity. Patterned bars in the figures highlight instances where certain baselines fail to yield a solution due to non-compliance with the problem's constraints (denoted by `!`) or inapplicability to different devices (denoted by `N/A`).

Takeaways: *Our framework achieves a substantial improvement, with an average gain of 1.19× and 1.57× (up to 1.46× and 1.92×) over the B-A and B-S baselines, respectively. It is noteworthy that these baselines, primarily designed for SOO problems, prove inadequate in capturing the multi-objective*

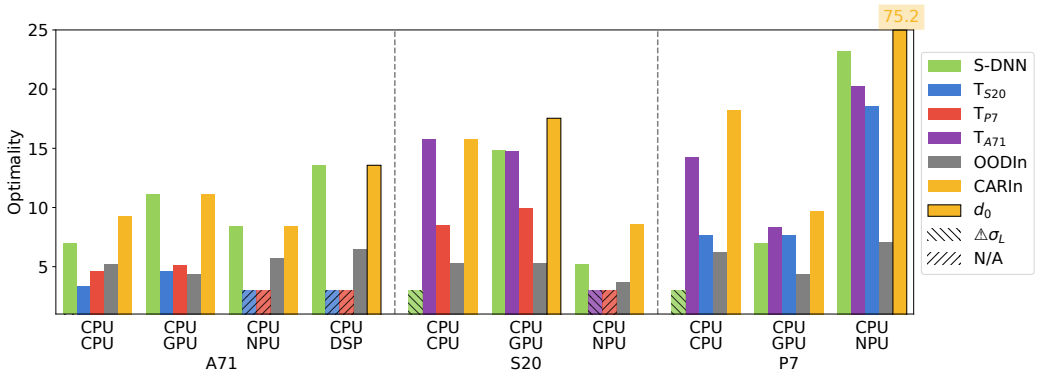

Fig. 5. UC3 evaluation.

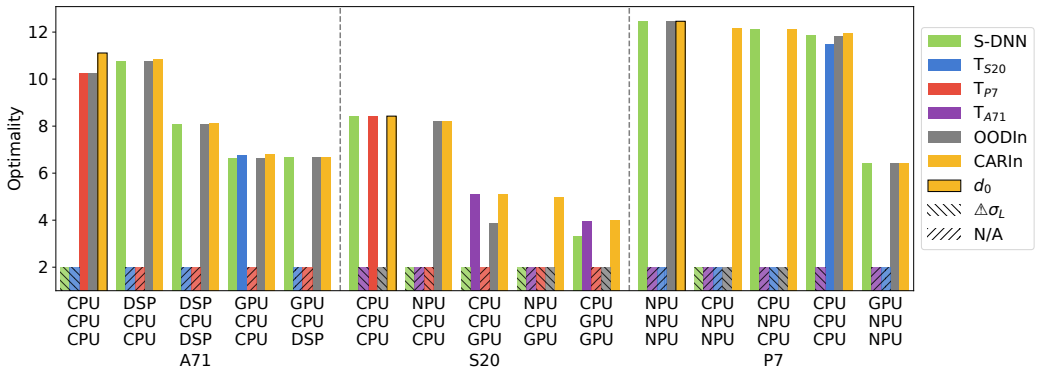

Fig. 6. UC4 evaluation.

nature inherent in DL applications. Regarding the transferred baselines, `CARIn` *achieves an average improvement of 1.17× in optimality (up to 1.84×). Importantly, it not only enhances overall optimality, but also exhibits improvements across all considered objective functions. Specifically, for UC1, we observe an average increase of 0.156 units in accuracy and a 32.7% boost in throughput, and for UC2, observable improvements include an average reduction of 2.8 MB in model size and a notable 19.9% latency speedup at the same accuracy level. Compared to OODIn, an optimality increase of 1.5× is achieved in average (up to 1.99×).*

7.1.3 Multi-DNN Execution. Figures 5 and 6 show the benefits of the `CARIn` framework concerning the optimality metric in the context of the two multi-DNN use cases. In these scenarios, we compare against the multi-DNN-unaware baseline, the transferred baselines from other devices, and OODIn. The horizontal axis illustrates combinations of processors for each device. In the case of UC3, all possible combinations are presented, while for UC4, due to the considerable number of combinations, we organise and display them based on optimality, showcasing the top 5 for each device.

Takeaways: *In the context of UC3,* `CARIn` *delivers a significant average optimality improvement of 1.47× across devices (up to 3.24×) over the multi-DNN-unaware baseline and an even more substantial gain of 1.87× (up to 4.06×) over the transferred baselines. Notably, these enhancements extend across all specified objectives. Compared to OODIn, we observe a 2.83× improvement in optimality (up to*

Table 7. Selected designs and switching policy for the single-DNN UC1 scenario on S20.

c_{CPU}	c_{GPU}	c_{NPU}	c_{m}	d_{new}
F	-	-	F	$d_0 = \langle \text{EfficientNet Lite0 FFX8, CPU}_{4,\text{T}} \rangle$
T	F	-	F	$d_1 = \langle \text{EfficientNet Lite0 FP16, GPU} \rangle$
T	T	F	F	$d_2 = \langle \text{MobileNet V2 1.4 FP16, NPU} \rangle$
T	T	T	F	$d_{\text{w}} = \langle \text{MobileNet V2 1.0 FX8, CPU}_{4,\text{T}} \rangle$
T	T	T	T	$d_{\text{wm}} \equiv d_{\text{w}}$
-	-	-	T	$d_{\text{m}} = \langle \text{EfficientNet Lite0 FX8, CPU}_{8,\text{F}} \rangle$

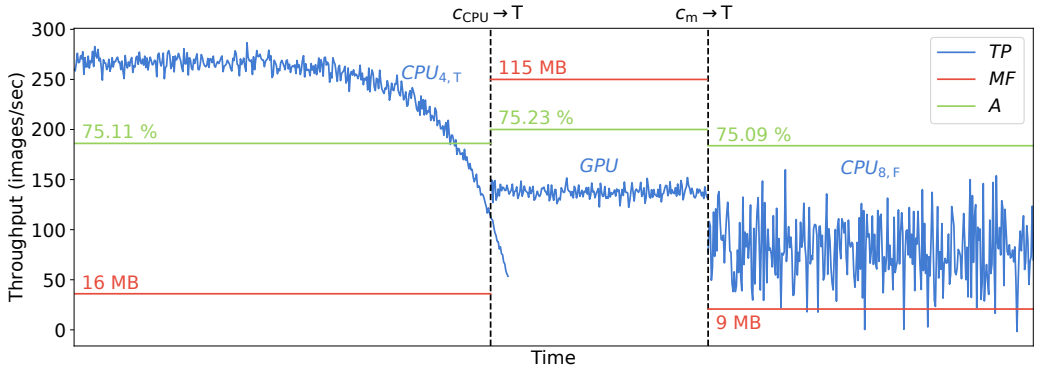

Fig. 7. CARIn's runtime behaviour targeting the single-DNN UC1 scenario on S20.

10.69×). Meanwhile, UC4 poses a distinctive challenge, where the majority of baselines struggle to produce a viable solution, primarily due to their inability to satisfy the stringent latency constraints inherent in this use case, underscoring the intricacy of UC4. It is noteworthy that, given the utilisation of a singular model per task, instances where baselines do not fail result in performance parity with our framework, emphasising the importance of accommodating a diverse array of models for each task.

7.2 Runtime Adaptation

In this section, we assess the responsiveness of the Runtime Manager (RM) and its adept utilisation of designs generated by RASS to dynamically adapt to a series of runtime fluctuations. For our evaluation, we target the UC1 single-DNN scenario on S20 and the UC3 multi-DNN scenario on A71. Through this experiment, we aim to demonstrate the efficacy of the RM module to seamlessly respond to dynamic runtime conditions, thereby validating its role in enhancing the adaptability and performance of CARIn across diverse use cases and devices.

7.2.1 Single-DNN Execution. Table 7 presents the selected designs and switching policy, while Figure 7 depicts the behaviour of RM in the single-DNN scenario. The initial design for UC1 on S20, d_0, involves the utilisation of EfficientNet Lite0 FFX8 on the CPU with 4 threads and the enabled XNNPACK library, resulting in 75.11% accuracy and a 16 MB memory footprint. As the CPU gradually becomes overloaded, the throughput experiences a decline until RM identifies an alternative design as the current highest-performing solution. The new configuration, d_1, entails the use of EfficientNet Lite0 FP16 on the GPU. Following further inferences, RM triggers another switch due to an impending memory issue. In this instance, RASS has identified the memory-efficient design, d_{m}, to involve the device's CPU.

Table 8. Selected designs and switching policy for the multi-DNN UC3 scenario on A71.

c_{DSP}	c_{GPU}	c_{CPU}	c_{m}	d_{new}
F	-	-	F	$d_0 = \{\langle \text{YAMNet FP16, CPU}_{2,\text{F}} \rangle, \langle \text{EfficientNet Lite2 FFX8, DSP} \rangle\}$
T	F	-	F	$d_1 = \{\langle \text{YAMNet FP16, CPU}_{2,\text{F}} \rangle, \langle \text{EfficientNet Lite2 FX8, GPU} \rangle\}$
T	T	F	F	$d_2 = \{\langle \text{YAMNet FP16, CPU}_{4,\text{F}} \rangle, \langle \text{EfficientNet Lite2 FFX8, CPU}_{1,\text{T}} \rangle\}$
T	T	T	F	$d_{\text{w}} = \{\langle \text{YAMNet DR8, CPU}_{2,\text{F}} \rangle, \langle \text{EfficientNet Lite0 FFX8, CPU}_{4,\text{F}} \rangle\}$
T	T	T	T	$d_{\text{wm}} \equiv d_{\text{w}}$
-	-	-	T	$d_{\text{m}} \equiv d_{\text{w}}$

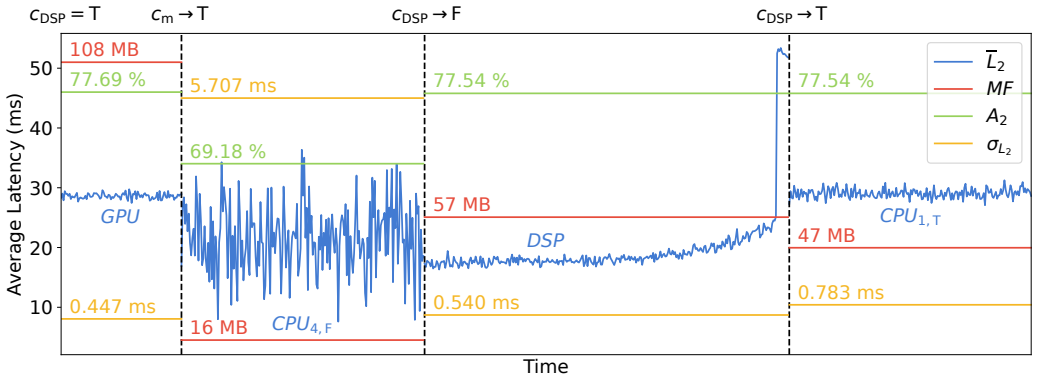

Fig. 8. CARIn's runtime behaviour targeting the multi-DNN UC3 scenario on A71.

Takeaways: *It is worth highlighting that despite modifications in the execution plan, our framework consistently upholds accuracy levels, even when employing the memory-efficient design. This steadfast commitment to preserving user Quality of Experience (QoE) underscores CARIn's resilience in the face of dynamic alterations.*

7.2.2 Multi-DNN Execution. Table 8 and Figure 8 correspond to the multi-DNN scenario. In the context of UC3, where two models with distinct workloads are employed, CARIn recognises the heavier workload associated with the second task and acknowledges that this specific task is primarily responsible for triggering the switching mechanisms. The figure illustrates the average latency, standard deviation of latency, and accuracy for the second task, as well as the combined memory footprint of both models. UC3 involves the processing of audio data, introducing the potential use of the device's DSP for data capture and processing. Given the likelihood of DSP overload during DNN inference, suppose that the highest-performing GPU-based design, d_1, is currently employed with EfficientNet Lite2 FX8. However, due to the impending threat of a memory issue arising from this design's memory footprint, RM opts to switch to the memory-efficient design, d_{m}, resulting in a saving of 92 MB of RAM. Subsequently, as RM observes a reduction in DSP overload, it triggers a switch to the highest-performing design, d_0, characterised by lower latency and reduced memory requirements. In the event of a potential DSP overload resurgence, RM strategically avoids reverting to the GPU-based design to mitigate previous concerns of excessive memory usage. Instead, it selects the next design in line, transferring the second model to the CPU while maintaining accuracy levels.

Takeaways: *It is important to acknowledge that CARIn may not always maintain predefined metric levels. As demonstrated in this instance, transitioning to the memory-efficient design resulted in an*

Table 9. OODIn's solving time in milliseconds. Given that this time is inevitable whenever a runtime issue occurs, it has the potential to become a bottleneck, thus impeding the seamless execution of a DL application.

Decision Space Dimension	A71		S20		P7	
	Average	*Maximum*	*Average*	*Maximum*	*Average*	*Maximum*
500	1.45	2.12	0.55	1.55	3.64	7.99
2000	2.80	5.94	1.70	3.04	4.94	9.38
5000	6.56	10.46	4.98	15.97	7.06	10.09
10000	12.14	16.07	11.09	34.25	10.41	13.38

Table 10. Storage requirements of CARIn and OODIn in MB.

	A71			S20			P7		
	CARIn	**OODIn**	***Reduction***	**CARIn**	**OODIn**	***Reduction***	**CARIn**	**OODIn**	***Reduction***
UC1	13.83	276.36	19.98×	34.37	443.10	12.89×	34.19	443.10	12.96×
UC2	48.64	311.45	6.40×	40.98	311.45	7.60×	52.96	311.45	5.88×
UC3	25.74	205.22	7.97×	58.70	205.22	3.50×	52.81	205.22	3.89×
UC4	2.65	6.56	2.48×	3.95	6.56	1.66×	3.95	6.56	1.66×

8.5% decrease in accuracy and an increase in jitter. However, such occurrences are considered temporary states of urgency, with a firm expectation that they will be swiftly rectified, thereby minimising impact on user QoE. Notably, the rise in average latency or the standard deviation of latency does not significantly affect user QoE, as these metrics already meet the specified latency constraints, which precede the optimisation of the objectives.

7.2.3 Comparison with OODIn. In our previous work, we introduced the model/processor switching technique to mitigate runtime fluctuations. However, OODIn lacks the ability to predict forthcoming changes in resource availability, so upon detecting such events, the MOO problem necessitates readjustment to the new conditions and subsequent resolution to determine the new highest-performing solution. CARIn offers the advantage of solving the specified MOO problem once, prior to application initiation, thus switching to a new execution plan during runtime happens instantaneously and is based on the predetermined designs and switching policy. Table 9 presents the average and maximum observed solution times of OODIn across diverse applications and devices. The solution time primarily hinges on the number of objectives and the dimensionality of the decision space $\mathcal{X}$, contingent upon the number of DL tasks, utilised models per task, compression techniques, and adjustable system parameters. Given that the time required for the TFLite interpreter to load on the CPU is typically around 3-4 ms, it becomes evident that revisiting the MOO problem can potentially become a bottleneck for the application, impacting the user's QoE.

Aside from the time overhead incurred by repeated problem solving, OODIn also requires constant access to the entire array of considered models, necessitating their storage on the user's device, which can impose limitations on the assortment of models and compression techniques initially considered. Our framework obviates the necessity to store all model variants, requiring only those selected by RASS. Table 10 elucidates this contrast in terms of model storage requirements for every examined use case.

8 LIMITATIONS & FUTURE DIRECTIONS

In spite of the challenges mitigated by CARIn, our system exhibits limitations that impede its performance when deployed in practical scenarios. First, as mentioned in Section 4.2, the computation of device-dependent metrics associated with objective functions or constraints across all candidate solutions is unsuitable for realistic mobile applications due to its substantial time requirements and the necessity of deploying entire models onto target devices, particularly within expansive decision spaces. Within the broader landscape of related studies, numerous works have harnessed performance prediction methodologies to estimate such metrics when executing DNNs on specific hardware platforms, without resorting to direct measurements [9, 19, 25, 79]. These models consider a range of inputs, encompassing (a) architectural characteristics of the DNN model such as network topology, layer configurations, and overall complexity, (b) hardware specifications including compute architecture, memory hierarchy, interconnectivity, and support for parallelism, and (c) environmental parameters like batch size, input data characteristics, runtime conditions, and temperature/power conditions. Such approaches are orthogonal to our framework and can be integrated within CARIn to provide a more expedient alternative to exhaustive profiling. In the future, the exploration of such methods is envisioned to furnish a comprehensive assessment of our framework's performance and suitability for real-world scenarios.

An additional limitation arises from the selection of models for evaluation. In the contemporary landscape of generative artificial intelligence (AI) [5, 59], the inclusion of generative models, such as autoregressive language models, becomes paramount. These models, characterised by their ability to generate outputs sequentially based on previously generated tokens, impose heightened demands [32, 76], particularly within the context of mobile environments [33]. Therefore, it is imperative to account for such intricacies when assessing the efficacy of AI frameworks intended for deployment in resource-constrained settings.

9 CONCLUSION

This research underscores the paramount significance of optimising the on-device execution of DNNs to meet the evolving demands of artificial intelligence applications. Building upon the foundational work of [61], the presented framework, CARIn, aims to spearhead progress in this direction. While the challenges of device heterogeneity, runtime adaptation, and multi-DNN execution persist, CARIn provides a novel and comprehensive solution towards alleviating them. The integration of an expressive multi-objective optimisation (MOO) framework and the introduction of RASS as a runtime-aware MOO solver manage to enable efficient adaptation to dynamic conditions while adhering to user-specified SLOs. RASS stands out for its ability to foresee upcoming runtime issues and generate a set of configurations which enable rapid, low-overhead adjustments in response to environmental fluctuations.

ACKNOWLEDGMENTS

This research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5578).

REFERENCES

[1] Mario Almeida, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. 2019. EmBench: Quantifying Performance Variations of Deep Neural Networks Across Modern Commodity Devices. In *3rd International Workshop on Deep Learning for Mobile Systems and Applications (EMDL)*.

[2] Mario Almeida, Stefanos Laskaridis, Abhinav Mehrotra, Lukasz Dudziak, Ilias Leontiadis, and Nicholas D. Lane. 2021. Smart at What Cost? Characterising Mobile Deep Neural Networks in the Wild. In *ACM Internet Measurement Conference (IMC)*. 658–672.

[3] Maxim Berman, Leonid Pishchulin, Ning Xu, Matthew B. Blaschko, and Gérard G. Medioni. 2020. AOWS: Adaptive and Optimal Network Width Search With Latency Constraints. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 11214–11223. https://doi.org/10.1109/CVPR42600.2020.01123

[4] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Mohammad Abdullah Al Faruque, and Smaïl Niar. 2023. HADAS: Hardware-Aware Dynamic Neural Architecture Search for Edge Performance Scaling. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*. IEEE, 1–6. https://doi.org/10.23919/DATE56975.2023.10137095

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), 1877–1901.

[6] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations (ICLR)*.

[7] Bohong Chen, Mingbao Lin, Rongrong Ji, and Liujuan Cao. 2023. Prioritized Subnet Sampling for Resource-Adaptive Supernet Training. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 9 (2023), 11108–11119. https://doi.org/10.1109/TPAMI.2023.3265198

[8] Bart Cox, Jeroen Galjaard, Amirmasoud Ghiassi, Robert Birke, and Lydia Y Chen. 2021. Masa: Responsive Multi-DNN Inference on the Edge. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

[9] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. 2019. ChamNet: Towards Efficient Network Design Through Platform-Aware Model Adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 11398–11407. https://doi.org/10.1109/CVPR.2019.01166

[10] Piotr Dollár, Mannat Singh, and Ross B. Girshick. 2021. Fast and Accurate Model Scaling. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 924–932. https://doi.org/10.1109/CVPR46437.2021.00098

[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ByME42AqK7

[12] Stijn Eyerman and Lieven Eeckhout. 2008. System-Level Performance Metrics for Multiprogram Workloads. *IEEE Micro* 28, 3 (2008), 42–53. https://doi.org/10.1109/MM.2008.44

[13] Hongxiang Fan, , Stylianos I. Venieris, Alexandros Kouris, and Nicholas D. Lane. 2023. Sparse-DySta: Sparsity-Aware Dynamic and Static Scheduling for Sparse Multi-DNN Workloads. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[14] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. 2017. Audio Set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*. IEEE, 776–780. https://doi.org/10.1109/ICASSP.2017.7952261

[15] Nyoman Gunantara. 2018. A review of multi-objective optimization: Methods and its applications. *Cogent Engineering* 5, 1 (2018), 1502242. https://doi.org/10.1080/23311916.2018.1502242 arXiv:https://doi.org/10.1080/23311916.2018.1502242

[16] Junpeng Guo, Shengqing Xia, and Chunyi Peng. 2023. OPA: One-Predict-All For Efficient Deployment. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications, New York City, NY, USA, May 17-20, 2023*. IEEE, 1–10. https://doi.org/10.1109/INFOCOM53939.2023.10228928

[17] Peizhen Guo, Bo Hu, and Wenjun Hu. 2021. Mistify: Automating DNN Model Porting for On-Device Inference at the Edge. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, James Mickens and Renata Teixeira (Eds.). USENIX Association, 705–719. https://www.usenix.org/conference/nsdi21/presentation/guo

[18] Hai Victor Habi, Roy H. Jennings, and Arnon Netzer. 2020. HMQ: Hardware Friendly Mixed Precision Quantization Block for CNNs. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVI (Lecture Notes in Computer Science, Vol. 12371)*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer, 448–463. https://doi.org/10.1007/978-3-030-58574-7_27

[19] Myeonggyun Han and Woongki Baek. 2021. HERTI: A Reinforcement Learning-Augmented System for Efficient Real-Time Inference on Heterogeneous Embedded Systems. In *30th International Conference on Parallel Architectures and Compilation Techniques, PACT 2021, Atlanta, GA, USA, September 26-29, 2021*, Jaejin Lee and Albert Cohen (Eds.). IEEE, 90–102. https://doi.org/10.1109/PACT52795.2021.00014

[20] Rui Han, Qinglong Zhang, Chi Harold Liu, Guoren Wang, Jian Tang, and Lydia Y. Chen. 2021. LegoDNN: block-grained scaling of deep neural networks for mobile vision. In *ACM MobiCom '21: The 27th Annual International*

Conference on Mobile Computing and Networking, New Orleans, Louisiana, USA, October 25-29, 2021. ACM, 406–419. https://doi.org/10.1145/3447993.3483249

[21] Xiaoxi He, Xu Wang, Zimu Zhou, Jiahang Wu, Zheng Yang, and Lothar Thiele. 2023. On-Device Deep Multi-Task Inference via Multi-Task Zipping. *IEEE Trans. Mob. Comput.* 22, 5 (2023), 2878–2891. https://doi.org/10.1109/TMC.2021.3124306

[22] Andrey Ignatov et al. 2019. AI Benchmark: All About Deep Learning on Smartphones in 2019. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*.

[23] Md Shahriar Iqbal, Jianhai Su, Lars Kotthoff, and Pooyan Jamshidi. 2023. FlexiBO: A Decoupled Cost-Aware Multi-Objective Optimization Approach for Deep Neural Networks. *J. Artif. Intell. Res.* 77 (2023), 645–682. https://doi.org/10.1613/JAIR.1.14139

[24] Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. 2022. Band: coordinated multi-DNN inference on heterogeneous mobile processors. In *MobiSys '22: The 20th Annual International Conference on Mobile Systems, Applications and Services, Portland, Oregon, 27 June 2022 - 1 July 2022*, Nirupama Bulusu, Ehsan Aryafar, Aruna Balasubramanian, and Junehwa Song (Eds.). ACM, 235–247. https://doi.org/10.1145/3498361.3538948

[25] Fucheng Jia, Deyu Zhang, Ting Cao, Shiqi Jiang, Yunxin Liu, Ju Ren, and Yaoxue Zhang. 2022. CoDL: efficient CPU-GPU co-execution for deep learning inference on mobile devices. In *MobiSys '22: The 20th Annual International Conference on Mobile Systems, Applications and Services, Portland, Oregon, 27 June 2022 - 1 July 2022*, Nirupama Bulusu, Ehsan Aryafar, Aruna Balasubramanian, and Junehwa Song (Eds.). ACM, 209–221. https://doi.org/10.1145/3498361.3538932

[26] Andreas Karatzas and Iraklis Anagnostopoulos. 2023. OmniBoost: Boosting Throughput of Heterogeneous Embedded Devices under Multi-DNN Workload. *CoRR* abs/2307.03290 (2023). https://doi.org/10.48550/arXiv.2307.03290 arXiv:2307.03290

[27] Jangryul Kim and Soonhoi Ha. 2023. Energy-Aware Scenario-Based Mapping of Deep Learning Applications Onto Heterogeneous Processors Under Real-Time Constraints. *IEEE Trans. Computers* 72, 6 (2023), 1666–1680. https://doi.org/10.1109/TC.2022.3218991

[28] Youngsok Kim, Joonsung Kim, Dongju Chae, Daehyun Kim, and Jangwoo Kim. 2019. μlayer: Low Latency On-Device Inference using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization. In *Proceedings of the Fourteenth EuroSys Conference (EuroSys)*.

[29] Alexandros Kouris, Stylianos I. Venieris, Stefanos Laskaridis, , and Nicholas D. Lane. 2023. Fluid Batching: Exit-Aware Preemptive Serving of Early-Exit Neural Networks on Edge NPUs. In *International Conference on Computer-Aided Design (ICCAD)*.

[30] Achintya Kundu, Laura Wynter, Rhui Dih Lee, and Luis Angel D. Bathen. 2023. Transfer-Once-For-All: AI Model Optimization for Edge. In *IEEE International Conference on Edge Computing and Communications, EDGE 2023, Chicago, IL, USA, July 2-8, 2023*, Claudio A. Ardagna, Feras M. Awaysheh, Hongyi Bian, Carl K. Chang, Rong N. Chang, Flávia Coimbra Delicato, Nirmit Desai, Jing Fan, Geoffrey C. Fox, Andrzej Goscinski, Zhi Jin, Anna Kobusinska, and Omer F. Rana (Eds.). IEEE, 26–35. https://doi.org/10.1109/EDGE60047.2023.00017

[31] Basar Kütükçü, Sabur Baidya, Anand Raghunathan, and Sujit Dey. 2022. Contention Grading and Adaptive Model Selection for Machine Vision in Embedded Systems. *ACM Trans. Embed. Comput. Syst.* 21, 5 (2022), 55:1–55:29. https://doi.org/10.1145/3520134

[32] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. 611–626.

[33] Stefanos Laskaridis, Kleomenis Kateveas, Lorenzo Minto, and Hamed Haddadi. 2024. MELTing point: Mobile Evaluation of Language Transformers. *arXiv preprint arXiv:2403.12844* (2024).

[34] Stefanos Laskaridis, Stylianos I. Venieris, Hyeji Kim, and Nicholas D. Lane. 2020. HAPI: Hardware-Aware Progressive Inference. In *International Conference on Computer-Aided Design (ICCAD)*.

[35] Jaeseong Lee, Jungsub Rhim, Duseok Kang, and Soonhoi Ha. 2022. SNAS: Fast Hardware-Aware Neural Architecture Search Methodology. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41, 11 (2022), 4826–4836. https://doi.org/10.1109/TCAD.2021.3134843

[36] Seulki Lee and Shahriar Nirjon. 2020. Fast and Scalable In-Memory Deep Multitask Learning via Neural Weight Virtualization. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*.

[37] Neiwen Ling, Kai Wang, Yuze He, Guoliang Xing, and Daqi Xie. 2021. RT-mDL: Supporting Real-Time Mixed Deep Learning Tasks on Edge Platforms. In *SenSys '21: The 19th ACM Conference on Embedded Networked Sensor Systems, Coimbra, Portugal, November 15 - 17, 2021*, Jorge Sá Silva, Fernando Boavida, André Rodrigues, Andrew Markham, and Rong Zheng (Eds.). ACM, 1–14. https://doi.org/10.1145/3485730.3485938

[38] Sicong Liu, Bin Guo, Ke Ma, Zhiwen Yu, and Junzhao Du. 2021. AdaSpring: Context-adaptive and Runtime-evolutionary Deep Model Compression for Mobile Applications. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 1 (2021),

24:1–24:22. https://doi.org/10.1145/3448125

[39] Sachin Mehta and Mohammad Rastegari. 2022. MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=vh-0sUt8HlG

[40] Santiago Miret, Vui Seng Chua, Mattias Marder, Mariano Phiellip, Nilesh Jain, and Somdeb Majumdar. 2022. Neuroevolution-enhanced multi-objective optimization for mixed-precision quantization. In *GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022*, Jonathan E. Fieldsend and Markus Wagner (Eds.). ACM, 1057–1065. https://doi.org/10.1145/3512290.3528692

[41] Subhabrata Mukherjee, Ahmed Hassan Awadallah, and Jianfeng Gao. 2021. XtremeDistilTransformers: Task Transfer for Task-agnostic Distillation. arXiv:2106.04563 [cs.CL]

[42] Ioannis Panopoulos, Sokratis Nikolaidis, Stylianos I. Venieris, and Iakovos S. Venieris. 2023. Exploring the Performance and Efficiency of Transformer Models for NLP on Mobile Devices. In *IEEE Symposium on Computers and Communications (ISCC)*.

[43] Hishan Parry, Lei Xun, Amin Sabet, Jia Bi, Jonathon S. Hare, and Geoff V. Merrett. 2021. Dynamic Transformer for Efficient Machine Translation on Embedded Devices. In *3rd ACM/IEEE Workshop on Machine Learning for CAD, MLCAD 2021, Raleigh, NC, USA, August 30 - Sept. 3, 2021*. IEEE, 1–6. https://doi.org/10.1109/MLCAD52597.2021.9531281

[44] João Luiz Junho Pereira, Guilherme Antônio Oliver, Matheus Brendon Francisco, Sebastião Simões Cunha, and Guilherme Ferreira Gomes. 2022. A Review of Multi-objective Optimization: Methods and Algorithms in Mechanical Engineering Problems. *Archives of Computational Methods in Engineering* 29, 4 (01 Jun 2022), 2285–2308. https://doi.org/10.1007/s11831-021-09663-x

[45] Ariadna Quattoni and Antonio Torralba. 2009. Recognizing indoor scenes. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 413–420. https://doi.org/10.1109/CVPR.2009.5206537

[46] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2020. Designing Network Design Spaces. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 10425–10433. https://doi.org/10.1109/CVPR42600.2020.01044

[47] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.

[48] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 4510–4520.

[49] Victor Sanh, Thomas Wolf, and Sebastian Ruder. 2019. A Hierarchical Multi-Task Approach for Learning Embeddings from Semantic Tasks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 6949–6956. https://doi.org/10.1609/aaai.v33i01.33016949

[50] Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. 2018. CARER: Contextualized Affect Representations for Emotion Recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 3687–3697.

[51] Wonik Seo, Sanghoon Cha, Yeonjae Kim, Jaehyuk Huh, and Jongse Park. 2021. SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms. *ACM Trans. Archit. Code Optim.* 18, 4 (2021), 43:1–43:26. https://doi.org/10.1145/3460352

[52] Shubhkirti Sharma and Vijay Kumar. 2022. A Comprehensive Review on Multi-objective Optimization Techniques: Past, Present and Future. *Archives of Computational Methods in Engineering* 29, 7 (01 Nov 2022), 5605–5633. https://doi.org/10.1007/s11831-022-09778-9

[53] Yechao She, Minming Li, Yang Jin, Meng Xu, Jianping Wang, and Bin Liu. 2023. On-demand Edge Inference Scheduling with Accuracy and Deadline Guarantee. In *31st IEEE/ACM International Symposium on Quality of Service, IWQoS 2023, Orlando, FL, USA, June 19-21, 2023*. IEEE, 1–10. https://doi.org/10.1109/IWQOS57198.2023.10188769

[54] Amit Kumar Singh, Somdip Dey, Klaus McDonald-Maier, Karunakar Reddy Basireddy, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2020. Dynamic Energy and Thermal Management of Multi-core Mobile Platforms: A Survey. *IEEE Design Test* 37, 5 (2020), 25–33.

[55] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 2158–2170. https://doi.org/10.18653/V1/2020.ACL-MAIN.195

[56] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2820–2828. https://doi.org/10.1109/CVPR.2019.00293

[57] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6105–6114.

[58] Mingxing Tan and Quoc V. Le. 2021. EfficientNetV2: Smaller Models and Faster Training. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 10096–10106. http://proceedings.mlr.press/v139/tan21a.html

[59] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMa: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023).

[60] Stylianos I. Venieris, Christos-Savvas Bouganis, and Nicholas D. Lane. 2023. Multiple-Deep Neural Network Accelerators for Next-Generation Artificial Intelligence Systems. *Computer* 56, 3 (2023), 70–79.

[61] Stylianos I. Venieris, Ioannis Panopoulos, and Iakovos S. Venieris. 2021. OODIn: An Optimised On-Device Inference Framework for Heterogeneous Mobile Devices. In *IEEE International Conference on Smart Computing (SMARTCOMP)*.

[62] Siqi Wang, Gayathri Ananthanarayanan, Yifan Zeng, Neeraj Goel, Anuj Pathania, and Tulika Mitra. 2019. High-Throughput CNN Inference on Embedded ARM big.LITTLE Multi-Core Processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 10 (2019), 2254–2267.

[63] Zhehui Wang, Tao Luo, Miqing Li, Joey Tianyi Zhou, Rick Siow Mong Goh, and Liangli Zhen. 2021. Evolutionary Multi-Objective Model Compression for Deep Neural Networks. *IEEE Comput. Intell. Mag.* 16, 3 (2021), 10–21. https://doi.org/10.1109/MCI.2021.3084393

[64] Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Yunxin Liu. 2023. AdaptiveNet: Post-deployment Neural Architecture Adaptation for Diverse Edge Environments. *CoRR* abs/2303.07129 (2023). https://doi.org/10.48550/arXiv.2303.07129 arXiv:2303.07129

[65] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 10734–10742. https://doi.org/10.1109/CVPR.2019.01099

[66] Carole-Jean Wu et al. 2019. Machine Learning at Facebook: Understanding Inference at the Edge. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 331–344.

[67] Xiaofeng Wu, Jia Rao, Wei Chen, Hang Huang, Chris H. Q. Ding, and Heng Huang. 2021. SwitchFlow: preemptive multitasking for deep learning. In *Middleware '21: 22nd International Middleware Conference, Québec City, Canada, December 6 - 10, 2021*, Kaiwen Zhang, Abdelouahed Gherbi, Nalini Venkatasubramanian, and Luís Veiga (Eds.). ACM, 146–158. https://doi.org/10.1145/3464298.3493391

[68] Jiyang Xie, Xiu Su, Shan You, Zhanyu Ma, Fei Wang, and Chen Qian. 2022. ScaleNet: Searching for the Model to Scale. In *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXI (Lecture Notes in Computer Science, Vol. 13681)*, Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer, 104–120. https://doi.org/10.1007/978-3-031-19803-8_7

[69] Mengwei Xu et al. 2019. A First Look at Deep Learning Apps on Smartphones. In *WWW*.

[70] Zhiyuan Xu, Dejun Yang, Chengxiang Yin, Jian Tang, Yanzhi Wang, and Guoliang Xue. 2023. A Co-Scheduling Framework for DNN Models on Mobile and Edge Devices With Heterogeneous Hardware. *IEEE Trans. Mob. Comput.* 22, 3 (2023), 1275–1288. https://doi.org/10.1109/TMC.2021.3107424

[71] Qizheng Yang, Tianyi Yang, Mingcan Xiang, Lijun Zhang, Haoliang Wang, Marco Serafini, and Hui Guan. 2024. GMorph: Accelerating Multi-DNN Inference via Model Fusion. In *Conference on Computer Systems (EuroSys)*.

[72] Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew R. Willis. 2020. MutualNet: Adaptive ConvNet via Mutual Learning from Network Width and Resolution. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12346)*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer, 299–315. https://doi.org/10.1007/978-3-030-58452-8_18

[73] Juheon Yi and Youngki Lee. 2020. Heimdall: Mobile GPU Coordination Platform for Augmented Reality Applications. In *Annual International Conference on Mobile Computing and Networking (MobiCom)*.

[74] Fuxun Yu, Shawn Bray, Di Wang, Longfei Shangguan, Xulong Tang, Chenchen Liu, and Xiang Chen. 2021. Automated Runtime-Aware Scheduling for Multi-Tenant DNN Inference on GPU. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*.

[75] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Chenchen Liu, and Xiang Chen. 2022. A Survey of Multi-Tenant Deep Learning Inference on GPU. *arXiv preprint arXiv:2203.09040* (2022).

[76] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. ORCA: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 521–538.

[77] Jiahui Yu and Thomas S. Huang. 2019. Universally Slimmable Networks and Improved Training Techniques. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 1803–1811. https://doi.org/10.1109/ICCV.2019.00189

[78] Mu Yuan, Lan Zhang, Zimu Zheng, Yi-Nan Zhang, and Xiang-Yang Li. 2023. MLink: Linking Black-Box Models From Multiple Domains for Collaborative Inference. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 10 (2023), 12085–12097. https://doi.org/10.1109/TPAMI.2023.3283780

[79] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *MobiSys '21: The 19th Annual International Conference on Mobile Systems, Applications, and Services, Virtual Event, Wisconsin, USA, 24 June - 2 July, 2021*, Suman Banerjee, Luca Mottola, and Xia Zhou (Eds.). ACM, 81–93. https://doi.org/10.1145/3458864.3467882

[80] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. 2020. Fast Hardware-Aware Neural Architecture Search. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. Computer Vision Foundation / IEEE, 2959–2967. https://doi.org/10.1109/CVPRW50498.2020.00354

[81] Song Yang Zhang, Zhifei and Hairong Qi. 2017. Age Progression/Regression by Conditional Adversarial Autoencoder. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[82] Yu Zhang and Qiang Yang. 2022. A Survey on Multi-Task Learning. *IEEE Trans. Knowl. Data Eng.* 34, 12 (2022), 5586–5609. https://doi.org/10.1109/TKDE.2021.3070203

[83] Ziyang Zhang, Huan Li, Yang Zhao, Changyao Lin, and Jie Liu. 2023. BCEdge: SLO-Aware DNN Inference Services with Adaptive Batching on Edge Platforms. *CoRR* abs/2305.01519 (2023). https://doi.org/10.48550/arXiv.2305.01519 arXiv:2305.01519

[84] Ziyang Zhang, Yang Zhao, and Jie Liu. 2023. Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster. In *International Conference on Service-Oriented Computing (ICSOC)*, Flavia Monti, Stefanie Rinderle-Ma, Antonio Ruiz Cortés, Zibin Zheng, and Massimo Mecella (Eds.).

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009