Convergo: Multi-SLO-Aware Scheduling for Heterogeneous AI Accelerators on Edge Devices

Ting Jiang*, Jianwei Hao†, Sushruth Harsha*, Rakandhiya D. Rachmanto*,
Arief Setyanto‡, Lakshmish Ramaswamy*, and In Kee Kim*
*University of Georgia, {ting.jiang1, sushruth.harsha, rakandhiya.rachmanto, laksmr, inkee.kim}@uga.edu
†Governors State University, jhao@govst.edu
‡Universitas Amikom Yogyakarta, arief_s@amikom.ac.id

***Abstract*—With the growing prevalence of edge AI, systems are increasingly required to meet stringent and diverse service level objectives (SLOs), such as maintaining specific accuracy levels, ensuring sufficient inference throughput, and meeting deadlines, often simultaneously. However, concurrently achieving these varied and complex SLOs is particularly challenging due to the resource constraints of edge devices and the heterogeneity of AI accelerators. To address this gap, we present a novel AI scheduling framework, Convergo, which uniquely integrates heterogeneous accelerator management, multi-tenancy, and multi-SLO prioritization into one scheduling solution. Convergo not only leverages heterogeneous AI accelerators and supports AI multi-tenancy, but also integrates scheduling heuristics to meet multiple SLOs concurrently. Convergo enables the simultaneous satisfaction of multiple/complex SLO requirements (*e.g.*, accuracy, throughput, and deadline constraints). The scheduling algorithm prioritizes inference requests, imposes critical constraints, and selects the best model combinations for current inferencing. We evaluated Convergo on the Jetson Xavier platform with portable TPU accelerators across various AI workloads, demonstrating its effectiveness. The evaluation results show that Convergo outperforms state-of-the-art baselines, achieving over 90% satisfaction of all three distinct SLO requirements simultaneously while maintaining approximately 95% satisfaction for individual SLOs. Furthermore, Convergo achieves these results with negligible overhead, making it a promising solution for edge AI systems.**

***Index Terms*—Edge Computing; Edge AI; On-Device AI Scheduler; Edge Accelerators;**

I. Introduction

Edge AI and edge inference systems are gaining significant attention due to the increasing deployment of AI services and the rapid development of pre-trained models, devices, and AI accelerators [9], [19], [21], [28]. The primary benefit of edge AI is its ability to offer low-latency inference services. By placing small yet capable resources close to end-users and data sources, edge computing facilitates AI computations directly on the device [21], [22], [31]–[33]. This approach significantly reduces reliance on cloud resources in remote data centers, thereby minimizing data transfer overheads.

However, edge AI has an intrinsic challenge: *it must operate on resource-constrained devices while meeting specific (often multiple) service-level objectives (SLOs)*, such as accuracy targets, inference deadline goals, or throughput requirements. To address this challenge and enhance edge AI's performance, various approaches have been developed [11], [17], [23], [30], [41]. One approach is focused on enhancing AI inference on edge accelerators using AI multi-tenancy [21], [22], [33], which involves running multiple deep learning (DL) models concurrently. Another commonly used approach relies on model compression, which reduces the size of DL models, making them suitable for execution on resource-constrained edge devices [20], [38].

Nevertheless, these methods often address only a subset of the challenges – *optimizing either a single SLO or a single accelerator type* – without providing an integrated solution that can handle multiple SLOs across heterogeneous accelerators. As a result, they remain *insufficient* to tackle the core challenges of edge AI, particularly when multiple SLOs (*e.g.*, accuracy, deadline, and throughput) must be met concurrently. For example, with AI multi-tenancy, it is difficult to schedule AI tasks to meet their SLOs due to resource heterogeneity, as different edge devices and AI accelerators exhibit distinct performance characteristics and variations. A scheduler on edge devices should have accurate knowledge of the DL models' performance, inference latency, and resource consumption when running with other DL models. However, having an accurate understanding of such characteristics is a complex task.

Moreover, while on-device model compression *may* generate a model that meets multiple SLOs, it faces two main obstacles. The first obstacle is the time and resource-intensive nature of on-device compression as the compression process demands substantial computing resources [27]. Given the constrained edge resources, this process often results in considerable scheduling delays. The second obstacle is the inference accuracy of compressed models. Due to the complexity of determining the right compression options [43], it is challenging to create a compressed model that effectively meets specific/complex SLOs.

In this work, we present Convergo, a novel AI scheduler for resource-constrained edge devices that combines AI multi-tenancy with heterogeneous accelerator management to meet multiple and complex SLO requirements simultaneously. Unlike previous studies [9], [11], [17], [21], [23], [30], [41] that typically address one or two constraints, Convergo integrates three critical capabilities – *multi-tenancy, heterogeneous accelerator management, and multi-SLO prioritization* – into one cohesive solution. Convergo leverages a set of pre-trained DL models for various real-world edge AI applications optimized

for different accelerators. It determines the best combinations of pre-trained models considering accelerator heterogeneity and resource constraints, facilitating the concurrent execution of multiple DL models to meet various SLO requirements. In particular, Convergo's scheduling algorithm prioritizes AI inference requests, imposes critical constraints based on SLO requirements, and selects the best combinations of models to ensure SLO satisfaction. Moreover, we further enhance the adaptability of Convergo to address more challenging edge AI use cases when newly pre-trained models are added and required for AI inferences. To this end, Convergo utilizes slack-aware minimal profiling and lightweight performance prediction.

We implemented Convergo on the Jetson Xavier platform, equipped with a Volta GPU (384 CUDA cores, 48 Tensor cores) and multiple portable TPU accelerators. To assess its performance, we conducted extensive evaluations across three widely-used AI application categories: image classification, object detection, and pose estimation. We generated realistic AI inference workloads, with each request having three distinct SLO requirements: *accuracy*, *throughput*, and *inference deadline*. Evaluating Convergo against state-of-the-art baselines [11], [30], our results show that Convergo not only outperforms the baselines but also substantially improves the ability to simultaneously meet multiple and complex SLO requirements by leveraging heterogeneous AI accelerators and enabling AI multi-tenancy. Specifically, Convergo achieves a satisfaction rate of over 90% for meeting all three distinct SLO requirements concurrently, which is up to 37% higher than the baselines, while maintaining approximately 95% satisfaction for individual SLOs. Moreover, the results show that Convergo not only maintains high SLO satisfaction but also makes the best use of any remaining device and accelerator capacity to maximize inference accuracy when device and accelerator resources are available. We also evaluated the overhead of Convergo, considering its operation on resource-constrained edge devices. Our evaluation revealed that Convergo requires just an additional 5% to 6% of memory and CPU resources and consumes approximately 530mW of additional power, which are acceptable overheads for an edge AI scheduler.

In summary, Convergo's primary contribution is its holistic approach to multi-tenancy, heterogeneous acceleration, and multi-SLO coordination within the strict constraints of an edge environment, thereby bridging a gap left by existing approaches. Specifically, we make the following contributions:

1. **Design and implementation of Convergo.** We developed a lightweight AI scheduler on NVidia's Jetson Xavier platform, incorporating portable TPU accelerators. Convergo effectively manages diverse AI workloads while adhering to various SLO constraints.

2. **Novel AI Scheduling Algorithm.** We designed a novel scheduling algorithm that orchestrates various AI applications on heterogeneous AI accelerators (e.g., applications on both GPUs and TPU accelerators). By enabling AI multi-tenancy and utilizing different accelerators, this scheduling algorithm allows Convergo to simultaneously meet complex and diverse SLO requirements.

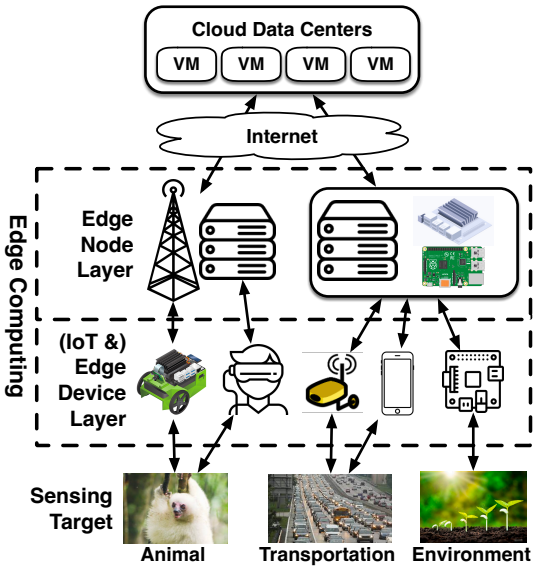

Fig. 1. **Edge Computing and Edge AI Architecture.**

3. **Thorough performance evaluation of Convergo.** We conducted an in-depth performance evaluation of Convergo on a real-world edge platform. Specifically, we analyzed its performance across three key AI application requests, each with distinct SLO requirements. Our evaluation encompassed resource utilization, overhead, and adaptability to model uncertainty (*e.g.*, using *unprofiled* or *unknown* DL models for AI inference processing).

The rest of this paper is organized as follows: §II describes the background and motivation of this work. §III provides the design of Convergo and detailed descriptions of its components. §IV reports the performance evaluation results of Convergo against state-of-the-art baselines. §V describes related work, and finally, §VI concludes the paper.

II. Background and Motivation

Edge Computing and Edge AI. Edge computing [24] is a decentralized computing model that places small-scale computing resources closer to data sources and end-users. Its primary goal is to achieve low-latency data processing by leveraging geographical proximity and protecting end-users' privacy via minimized data transmissions to cloud data centers.

Fig. 1 illustrates the edge computing architecture, typically composed of two layers: *edge node* and *(IoT &) edge device* layers. As stated in §I, advances in small edge devices, AI accelerators, and pre-trained models have enabled active deployment of AI inference services (edge AI) directly on edge devices (in the device layer) [9], [21]. Specifically, edge AI enables the execution of AI models directly on edge devices to provide low-latency AI inference capabilities (*e.g.*, object detection [18], [37], image/video processing [39], [40], [42]) to nearby users and to facilitate various operations related to target sensing (the bottom of Fig. 1).

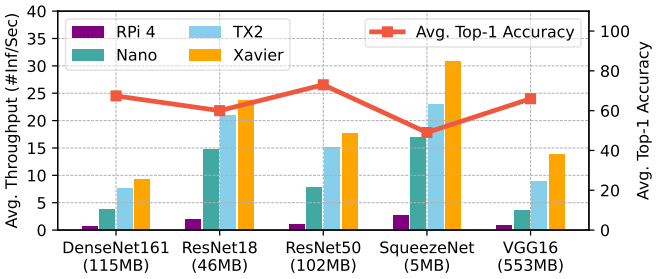

Fig. 2. **Inference Throughput vs. Accuracy of DL Models on Edge Devices.** *The bar graphs indicate the inference throughput on four edge devices (RPI4: Raspberry PI4, Nano: Jetson Nano, TX2: Jetson TX2, and Xavier: Jetson Xavier NX), and the line shows the top-1 accuracy of the models. The numbers in parentheses (on the x-axis) indicate the model size.*

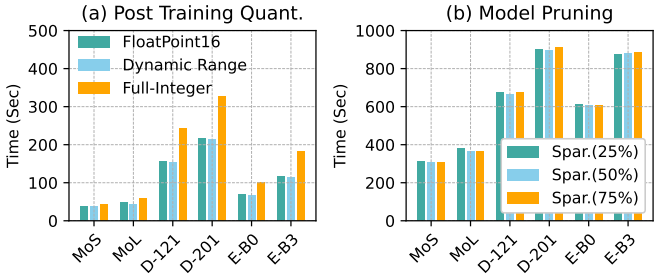

Fig. 3. **Compression Overhead (Time) on Jetson Xavier.** *(a) Compression overhead of three post-training quantization methods: Float16, Dynamic Range, and Full Integer; and (b) overhead of the pruning method at different sparsity levels (25%, 50%, and 75%). "MoS": MobileNetV3Small, "MoL": MobileNetV3Large, "D-xxx": DenseNet-xxx, "E-Bx": EfficientNet-Bx.*

For the successful operation of edge AI, it is important to satisfy multiple requirements simultaneously, including latency (or deadline), throughput, and accuracy. Specifically, inference latency is a critical aspect of edge AI, indicating that edge devices should complete AI computations within a specified time frame. However, this is challenging due to the resource heterogeneity and constrained resources commonly found in edge devices, often leading to varying processing times and inference latencies. For example, Fig. 2 (bar graphs) shows the inference throughput of DL models, which exhibits significant variability across various edge devices, hardware configurations, and accelerators. Moreover, given the large number of existing pre-trained models available (*e.g.*, models for image classification and object detection), each model has a unique combination of model size, accuracy, resource demand, and latency characteristics, as also shown in Fig. 2.

An effective way to meet multiple SLOs is to develop an on-device scheduler that can efficiently manage and prioritize computing resources despite resource constraints and heterogeneity. Furthermore, the scheduler's capabilities need to extend beyond resource management. For example, the on-device scheduler should intelligently select the most appropriate models for AI inference, balancing desired latency with other constraints such as model size, resource availability, and acceptable accuracy.

Limitations of Model Compression. Model compression techniques *may* be considered a viable approach to facilitate the execution of DL models on resource-constrained edge devices. These techniques can reduce the size of DL models, requiring fewer computing resources and offering shorter inference latency. Still, model compression *alone* is insufficient for edge AI and online DL model scheduling.

In edge AI, the diversity of SLOs, *e.g.*, varying accuracy and latency requirements, demands a scheduler that can create a variety of compressed models to meet these specific SLOs. This, however, poses additional challenges for AI scheduling due to the time-consuming nature of online model compression, which causes significant delays [27]. Fig. 3 shows our measurements of the compression overhead using two widely-used techniques, Post-Training Quantization (PTQ) and model pruning, on various DL models with the Jetson Xavier device. For PTQ shown in Fig. 3(a), we observed that compression times vary with the technique and model size. For example, small models like MobileNetV3Small required 36s – 43s with PTQ, whereas larger models like DenseNet201 took more than 200s. Moreover, model pruning (Fig. 3(b)) consistently resulted in longer compression times compared to PTQ.

While there are expected benefits of model compression, *e.g.*, reduced model size, lower resource demands, and faster inferences, its high compression overhead is often *unacceptable* in on-device AI scheduling. Additionally, accuracy degradation (the accuracy results are omitted due to page limits) is another drawback. Our observations confirmed that accuracy can vary significantly with different compression techniques, with top-1 accuracy reductions ranging from 20% to 30% or even more. For example, pruning EfficientNet-B0 at a 75% sparsity level led to an about 30% drop in accuracy, and using full integer PTQ on MobileNetV3Large resulted in an over 20% decrease in accuracy, consistent with prior work [27].

Motivation and `Convergo` Approach. Our main motivation for this work is to develop a novel scheduler for resource-constrained edge devices that can satisfy multiple SLOs for AI inference tasks. We aim to create a scheduler capable of simultaneously meeting various SLOs, *i.e.*, accuracy and throughput requirements. To achieve this, we adopt the following design policies for our new scheduler, `Convergo`.

First, considering the diversity of edge AI accelerators, the scheduler must be capable of leveraging **heterogeneous accelerators simultaneously**. Second, by utilizing these heterogeneous accelerators, the scheduler should **maximize AI multi-tenancy** to enhance the likelihood of meeting multiple, distinct SLO requirements. Third, because our target platform is resource-constrained edge devices, the new scheduler should **be lightweight**. Specifically, it should avoid computation-heavy methods (*e.g.*, on-device model compressions) and efficiently determine suitable combinations of multiple model executions to meet various SLOs. Finally, given the rapid pace of development in new pre-trained AI models, the scheduler should **be adaptable enough** to handle performance uncertainties associated with unknown models without relying on extensive and heavy profiling.

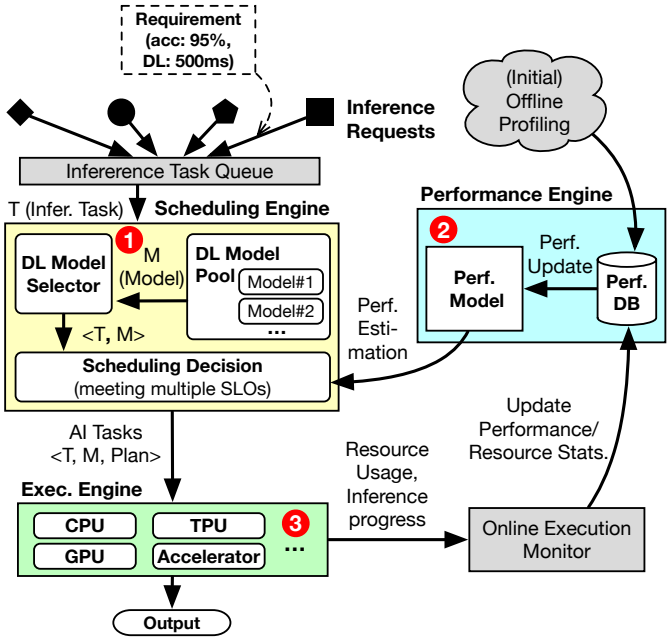

Fig. 4. **Overview of Convergo.**

III. Design of Convergo

This section begins with an overview of Convergo, followed by a detailed description of its components.

A. Convergo Overview

We design Convergo with four key principles: leveraging heterogeneous AI accelerators, maximizing AI multi-tenancy, maintaining a lightweight framework, and ensuring adaptability, as discussed in §II. These policies guide AI task scheduling to meet multiple SLOs for edge AI.

Fig. 4 illustrates the overall architecture of Convergo, including three main components executed on edge devices: ❶ *Performance Engine*, ❷ *Scheduling Engine*, and ❸ *Execution Engine*. Please note that while all components are meticulously designed following the four key policies, each has a distinct primary function: the performance engine is mainly for maximizing AI multi-tenancy and partially for its adaptability, the scheduling engine ensures lightweight operation and adaptability, and the execution engine enables Convergo to leverage heterogeneous accelerators, with a focus on GPUs on edge devices and TPU accelerators.

❶ **Performance Engine.** The performance engine contains a *performance DB* and a *performance model*. The performance database (DB) holds various system- and model-level statistics related to DL model executions, including inference time and resource utilization. This DB is initially built with offline profiling but is constantly updated with runtime datasets as actual DL models are executed on edge devices.

The performance model provides information about the latency and throughput of specific models when they are run on edge devices under various conditions (*e.g.*, execution with other models). Moreover, Convergo is designed to perform scheduling operations with unknown models, which are models not yet profiled, by employing two approaches: 1) slack-aware minimal profiling, and 2) lightweight performance estimation, which are further discussed in §III-E.

❷ **Scheduling Engine.** This engine aims to optimize the scheduling plan for inference requests, ensuring that multiple SLOs can be met simultaneously. This component incorporates a *DL model pool*, a *DL model selector*, and a *scheduling decision module*.

The DL model pool includes various pre-trained DL models and their static attributes (*e.g.*, size and accuracy). The DL model selector then identifies models that are likely to meet designated SLOs. Subsequently, the model execution plan is developed by the scheduling component, detailed in §III-B. Convergo employs various performance metrics and forecasts (*e.g.*, accuracy, throughput, resource usage) to create this plan, determining the types of models, their start times, and the AI accelerators for the executions.

❸ **Execution Engine.** The execution engine performs AI inference tasks by executing the DL models on heterogeneous edge accelerators. The model executions follow the plan created by the scheduling engine, which determines where and when DL models should be deployed, including resources like GPUs and TPU accelerators, to ensure efficient task execution.

Moreover, while the models are running, an online execution monitor (bottom right component in Fig. 4) collects runtime statistics on resource usage and model progress, and the monitor continually updates this information to the performance DB within the performance engine.

B. Convergo Scheduling

We discuss Convergo's scheduling algorithm in detail.

User Requests for AI Inference. In our design, we assume that multiple users submit various requests to Convergo. A user request refers to an AI inference task submitted by users, with each request having its own multiple SLOs that need to be satisfied simultaneously. For example, each request (R_i) contains three specific sub-requirements, denoted as $< Th_i, Acc_i, D_i >$, where Th_i represents the throughput goal, Acc_i is the accuracy target, and D_i is the deadline of the request (R_i).

Priority Decision for User Requests. Requests submitted to Convergo are placed in a priority queue, represented as $\mathcal{R} = \{R_1, R_2, R_3, ...R_n\}$, where n is the number of requests in the priority queue. Each request (R) includes its arrival time (t_{arr}) and its associated SLOs, such as a deadline (D). Users can also specify a weight (w) of each request when submitting multiple requests to our scheduler. The default value for w is 1, but users have the option to assign weights of up to 10 for requests that require urgent processing. Convergo assigns priority to a request (w_i) based on the following equation.

$$Priority_i = \frac{D_i - (t_{curr} - t_{arr,R_i})}{w_i}, \quad (1)$$

where D_i represents the deadline for request R_i, t_{curr} denotes the current time, t_{arr,R_i} is R_i's arrival time, and w_i is the user-

Algorithm 1 `Convergo` Scheduling

Input: $\mathcal{R} = \{R_1, R_2, ..., R_n\}$: User Request Arrivals. Each R_i has three specific goals: Throughput, Accuracy, and Deadline.

Output: Scheduling results

1: $\mathcal{R} = \{R_0, R_1, ...R_n\}$ arrive in `Convergo` queue
2: **function** CONVERGO _SCHEDULING($\mathcal{R}$)
3: **while** R_i in `Convergo` queue **do**
4: $Priority_i \leftarrow$ Equation-(1)
5: PRIORITYQUEUE.PUT($Priority_i$)
6: **end while**
7: PRIORITYQUEUE.REARRANGE()
8: **while** $R_k \leftarrow$ PRIORITYQUEUE.GET() **do**
9: $\mathcal{C}1$: Equation-(2)
10: $\mathcal{C}2$: Equation-(3)
11: $\mathcal{C}3$: Equation-(4)
12: //$\mathcal{M}_C$: A set of candidate models for R_k
13: **if** $\mathcal{M}_C \leftarrow$ min(SOLVE($\mathcal{M}_{all}$, $\mathcal{C}1$, $\mathcal{C}2$, $\mathcal{C}3$)) **then**
14: // Create model execution plan via Algo. 2
15: $\mathcal{P} \leftarrow$ MODELEXECUTIONPLAN($\mathcal{M}_{candi}$)
16: **end if**
17: **if** $\mathcal{P}$ **then**
18: // Execute $\mathcal{M}_C$ according to $\mathcal{P}$
19: INVOKE($\mathcal{P}$)
20: **else**
21: // No feasible scheduling plan for R_k
22: SEND MESSAGE("infeasible request", R_k))
23: **end if**
24: **end while**
25: **end function**

assigned weight. Moreover, within the priority queue, requests are organized by their priority, which updates dynamically with new request arrivals.

`Convergo` Scheduling Algorithm. Algo. 1 describes the `Convergo`'s scheduling mechanism. As discussed in §III-A, `Convergo` leverages multiple pre-trained models (in "DL model pool" in Fig. 4) and must select candidate models that fullfill all the requirements. Specifically, `Convergo` creates three constraints ($\mathcal{C}1$, $\mathcal{C}2$, and $\mathcal{C}3$), shown in lines 9 to 10 of Algo. 1.

The first constraint ($\mathcal{C}1$) is the throughput requirement of R_i, and candidate models will be chosen based on this equation.

$$\sum_{j=1}^{n} \mathcal{M}_j(Th) \times t_j \geq Th_i, \tag{2}$$

where $\mathcal{M}_j(Th)$ represents the throughput statistics of j^{th} model ($\mathcal{M}_j$), t_j denotes the execution time of $\mathcal{M}_j$.

Given the models that can meet the throughput requirement, `Convergo` uses the second constraint ($\mathcal{C}2$) to select models that meet the accuracy requirement. Equation-(3) is used to calculate the overall model accuracy, and only models with an overall accuracy higher than A_j will be selected. In equation-(3), $\mathcal{M}_j(Acc)$ represents the accuracy of $\mathcal{M}_j$.

$$\frac{\sum_{j=1}^{n} \mathcal{M}_j(Th) \times t_j \times \mathcal{M}_j(Acc)}{\sum_{j=1}^{n} \mathcal{M}_j(Th)} \geq Acc_i, \tag{3}$$

Given the model selected for throughput and accuracy requirements, `Convergo` now needs to apply the third constraint ($\mathcal{C}3$) to determine the models that can satisfy all the requirements. This step tests whether the models can meet the deadline (D_i) requirements by calculating

$$\sum_{j=1}^{n} t_j \leq D_i. \tag{4}$$

By using these three constraints, `Convergo` identifies the candidate models ($\mathcal{M}_C$) that can meet R_i's SLOs. In scenarios where multiple DL models, potentially leading to various execution plans, are available for processing R_i, `Convergo` prioritizes the models with the shortest execution time as shown in line 13. This approach enables `Convergo` to reveal slack-time for managing *unknown* models via slack-aware minimal profiling, a method detailed further in §III-E.

Once $\mathcal{M}_C$ is identified, `Convergo` creates a model execution plan ($\mathcal{P}$), composed of multiple small DL execution batches. The procedure for creating a model execution plan is explained in Algo. 2, further detailed in §III-C. After creating the model execution plan for R_i, the system will invoke the execution of DL models based on $\mathcal{P}$ (a series of small execution batches in §III-D).

C. Creating Model Execution Plan

`Convergo` develops a model execution plan ($\mathcal{P}$), leveraging $\mathcal{M}_C$ (from Algo. 1), to enable the concurrent use of heterogeneous accelerators like GPUs and TPUs. Therefore, it is important to determine which models should be executed on which AI accelerators. To this end, `Convergo` has two policies for making this decision:

1) The total CPU and memory usage of all enabled DL models *must not exceed* the device's available resources.
2) Given that an edge device can support two different types of AI accelerators (*e.g.*, GPUs and TPUs), *TPU accelerators have higher priority over GPUs* for model executions.

These policies are designed to optimize simultaneous model executions based on our observation that running DL models on TPUs, rather than GPUs, typically results in lower memory and CPU consumption. The high GPU consumption is due to their design, particularly on Jetson devices, where GPUs share an integrated memory subsystem with the host device, leading to increased memory usage during DL model executions.

We make specific assumptions to develop model execution plans on AI accelerators. For instance, we assume only one model can be deployed on a TPU, unlike multiple models on GPUs, due to the limited benefits of running multiple DL models on an edge TPU [14]. Additionally, edge devices can support 2 to 4 portable TPU accelerators, and their resources are sufficient to run all deployed models.

Algorithm 2 Creating Model Execution Plan

Input: $\mathcal{M}_C = \{m_1, m_2, ..., m_l\}$: Candidate DL models ($m$) for processing an user request (R)

Output: $\mathcal{P}$: Execution plan: A set of execution batches, each containing DL models on edge TPUs and edge GPUs, along with each batch's start and end times.

```
1:  function ModelExecutionPlan(M_C)
2:      P ← {} // initialize execution batches
3:      while |M_C| do
4:          // models on edge GPU and edge TPUs w/i a batch
5:          M_TPU ← {}, M_GPU ← {}
6:          // place models first edge TPUs, n: # of edge TPUs
7:          for i ∈ [1, min{n, |M_C|}] do
8:              m ← M_C.pop(0)
9:              M_TPU.add(m)
10:             CPU_tot += m(CPU)
11:             MEM_tot += m(MEM)
12:         end for
13:         // place next models on edge GPU
14:         for i ∈ [0, |M_C|] do
15:             CPU_tot += M_C[0](CPU)
16:             MEM_tot += M_C[0](MEM)
17:             if CPU_tot ≤ CPU_dev and
18:                MEM_tot ≤ MEM_dev then
19:                M_GPU.add(M_C.pop(0))
20:             else
21:                 // cannot run more models in this batch
22:                 // due to resource constraints
23:                 break
24:             end if
25:         end for
26:         // calculate duration of this execution batch
27:         t_end ← t_start + max(duration(M_GPU, M_TPU))
28:         // single batch execution plan is completed
29:         P.add(t_start, t_end, M_TPU, M_GPU)
30:         t_start ← t_end
31:     end while
32:     return P
33: end function
```

Algo. 2 illustrates how Convergo creates model execution plans. Specifically, a $\mathcal{P}$ to process a R can be composed of several small execution batches. Each execution batch includes the start (t_{start}) and end (t_{end}) times, along with the models to be deployed on both accelerators. The input for this algorithm comprises the selected DL models ($\mathcal{M}_C$) for processing a R.

Given that TPU accelerators have higher priority than GPUs, the first n models (n is the number of TPU accelerators) are initially assigned to the TPUs, as shown from lines #7 to #12. Convergo also calculates the total resource consumption (*e.g.*, CPU_{tot}, MEM_{tot}) for running models on TPUs. Subsequently, Convergo assigns DL models to GPUs based on the availability of resources. As shown from lines #14 to #25, DL models are sequentially assigned to GPUs until the cumulative resource consumption reaches the device's maximum capacity of CPU and/or memory. The models assigned (whether on TPUs or GPUs) constitute the small batch, for which Convergo calculates the start time and total duration, determined by the longest-running model among those selected. This batch is then incorporated into the $\mathcal{P}$ in line #29. Convergo continues to generate batches for $\mathcal{P}$ as long as the DL model in $\mathcal{M}_C$ exists. Once all models have been assigned with defined start and end times, the final $\mathcal{P}$ is created and returned to Algo. 1.

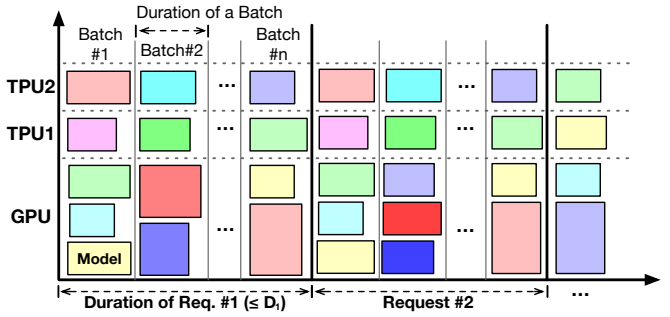

Fig. 5. **Model Executions with Multi-Batches and Multi-Tenancy in Convergo's scheduling.**

D. Concurrent DL Model Executions

Convergo executes batches within $\mathcal{P}$ sequentially. Fig. 5 shows an example of model executions within $\mathcal{P}$ when an edge device has a GPU and two TPU accelerators. Each TPU runs one model per batch, while the GPU can execute multiple DL models within the same batch. Please note that the number of models the GPU runs per batch varies based on the models' resource requirements and sizes. For example, in the first batch of R_1 in Fig. 5, the GPU runs three models, while in the second batch of R_1, it executes two models. Moreover, the execution duration for each batch may vary since it is determined by the longest-running model within the batch. Furthermore, Convergo must ensure that the total duration of executing all batches in $\mathcal{P}$ (or processing R) remains shorter than the R's deadline (D).

E. Supporting Unknown DL Models

Convergo relies on pre-profiled models in the performance engine for scheduling. However, Convergo should also be able to make a scheduling plan when the DL model pool contains *unknown (or unprofiled)* models. For example, new pre-trained models can be added to the model pool, or users may want to use specific models to process their requests. To this end, Convergo employs two approaches to obtaining the models' performance data: slack-aware minimal profiling and performance estimation.

Slack-aware Minimal Profiling. Convergo can perform minimal profiling during slack (idle) times, which occur between the end of the previous request and the start of the next request. We observed that slack times often exist due to the dynamics of request arrivals and Convergo's scheduling mechanism (specifically model selection in §III-B). For example, Convergo always chooses models with the shortest

Table 1. **AI Applications and DL Models Used For Evaluation.**

AI Application	Edge Acceler.	Pre-trained DL Models	Size (MB)	Num. Layers
Image Classification	GPU	EfficientNet [36]	5.4	5
		Inception-v1 [34]	6.4	22
		MobileNet-v2 [29]	3.4	20
		SqueezeNet [16]	1.3	15
	TPU	EfficientNet	6.8	7
		Inception-v1	7.0	22
		Inception-v3 [35]	12.0	48
		MobileNet-v1 [15]	1.6	28
Object Detection	GPU	EfficientNet	4.5	5
		MobileNet-v2	6.5	20
		YOLOX [12]	5.4	11
	TPU	EfficientNet	7.6	18
		MobileNet-v1	7.0	28
		MobileNet-v2	6.6	20
Pose Estimation	GPU	MobileNet-v1 [15]	8.2	28
		MoveNet [3]	5.6	25
		PoseNet [25]	4.3	23
	TPU	MoveNet	7.5	25
		PoseNet-MobileNet [25]	1.5	23
		PoseNet-ResNet [25]	24.4	18

execution time. This approach potentially enables Convergo to complete user requests before their deadlines, leveraging the slack time to perform minimal profiling of unknown models. This profiling collects basic performance data, including resource consumption, accuracy, throughput, and latency. The duration or iterations of executing the unknown model vary depending on the availability of slack times.

Performance Estimation. We developed a performance estimator for unknown DL models using Random Forest (RF). This performance estimator is useful when slack time is *unavailable*. To build this model, we utilize static information from existing profiled models in the performance engine (in Fig. 4). Features include model size, memory and CPU requirements, accuracy, latency, and throughput.

While the above two approaches allow Convergo to create scheduling plans for unknown models, these plans may initially be inaccurate, potentially leading to SLO violations. However, as illustrated in Fig 4, the online execution monitor captures all execution results, which are then updated in the performance engine. As more executions are performed, more accurate performance data is stored in the performance DB, enabling the creation of future scheduling plans with greater accuracy. In §IV, we conduct a detailed evaluation of Convergo's performance when using unknown models to validate the effectiveness of these approaches.

F. Implementation of Convergo

The prototype of Convergo has been implemented on Nvidia's Jetson Xavier [4], which is equipped with multiple Coral TPU accelerators [1]. Jetson Xavier is an edge device equipped with a 6-core Carmel CPU, 8GB of RAM, and a Volta GPU with 386 Tensor Cores for AI operations. Coral TPU accelerator is a portable AI accelerator, delivering 4 TOPS at 2W of power consumption.

Convergo is developed using approximately 2.5K lines of Python code and incorporates various libraries, including PuLP [5] for solving linear programming problems. Convergo supports various AI applications by leveraging two DL frameworks: Tensorflow [8] for GPU-based inference and TFLite [2] for TPU-based inference. Because TFLite is the only framework that supports inference tasks on TPUs on Jetson Xavier or other edge devices [14], we must use two different frameworks to enable support for both GPUs and TPUs. Convergo also supports other frameworks like PyTorch [26] and TensorRT [7]. Lastly, Convergo makes use of Redis [6] to manage user requests effectively, ensuring their seamless integration into the scheduling process.

Table 2. **Accuracy, Throughput, and Latency (Deadline) Requirements for Evaluated Workloads.** *"Accuracy": "Top-1 Accuracy," "Throughput": "Number of Inferences per Second," "Latency (DL)": "Latency (Deadline) in Second."*

AI App.	Num. of Requests	Range of SLOs		
		Accuracy	Throughput	Latency (DL)
Img. Class.	500	75% – 90%	500 – 2500	0.1s – 1.5s
Obj. Detect.		73% – 89%	500 – 2500	0.1s – 1.5s
Pose Est.		70% – 83%	500 – 2500	0.1s – 2s

IV. Evaluation Results of Convergo

A. Evaluation Setup

Edge Device and AI Accelerators. We conducted our evaluation using Jetson Xavier as the primary platform and augmented it with two TPU accelerators to assess the performance of Convergo and other baselines. These two TPUs were connected to the Jetson Xavier via the USB interface. Our decision to use two TPUs aligns with recommendations from prior research [14], which showed that two TPUs often provide better performance gains than three or four.

Evaluated AI Applications. To evaluate Convergo's performance, we employed three edge AI applications: *image classification*, *object detection*, and *pose estimation*. These applications use a variety of pre-trained models listed in Table 1. Specifically, image classification involves eight models, with four tailored for GPU and four for TPU inferences. Even when models share the same architecture (*e.g.*, Inception-v1 [34] for both GPU and TPU in the image classification category), they are individually optimized for each type of accelerator. For object detection and pose estimation, we use three GPU models and three TPU models for each, respectively.

Workload Generation. Our evaluation workloads combine all three above AI applications, with each request being randomly generated to assess Convergo under more realistic use cases. The evaluation workloads consist of a set of 500 requests generated using the Poisson distribution. Each request within these workloads has specific accuracy, throughput, and latency (deadline) requirements, which are randomly selected from the ranges detailed in Table 2. Moreover, we conducted the evaluation with multiple iterations with 10 different workloads to mitigate the impact of outlier results.

Baselines. To evaluate the performance of Convergo, we employed two state-of-the-art baselines: SLO-MAEL [30] and Kalmia [11]. SLO-MAEL is built upon a minimum-average-expected-latency (MAEL) scheduling algorithm by integrating

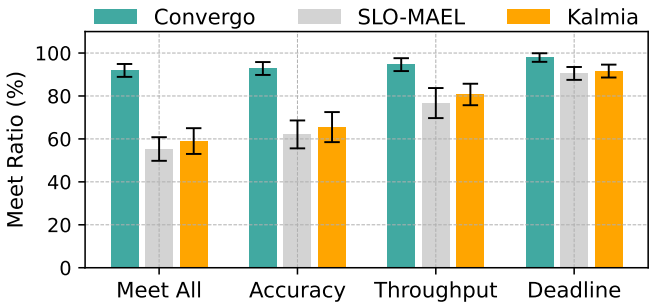

Fig. 6. **SLO Satisfaction Results.** *"Meet All" indicates the results when all three requirements are met. The other three – "Accuracy," "Throughput," and "Deadline" – represent the satisfaction rate for each individual requirement.*

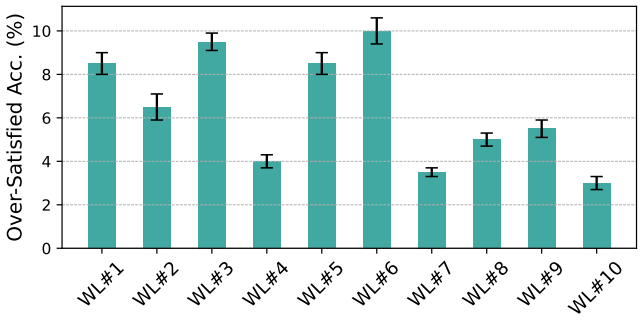

Fig. 7. **Over-Satisfied Inference Accuracy (%) for All 10 Evaluated Workloads.** *For example, 8.3% of WL #1 means that* `Convergo`*'s accuracy achievement is, on average, 8.3% higher than the requirements*

SLO awareness into its operations. SLO-MAEL uses a dual-scoring mechanism where tasks are evaluated based on both expected latency and their adherence to SLOs. The scheduler prioritizes tasks that are most likely to violate their SLOs, trying to ensure that critical tasks are processed within their required deadlines. Additionally, SLO-MAEL incorporates preemptive scheduling, allowing it to interrupt and reprioritize ongoing tasks based on the degree of SLO violation.

Kalmia [11] is a QoS-aware scheduler designed to manage heterogeneous DNN inference tasks on edge servers via a mix of preemptive and context-aware scheduling strategies to optimize both task timeliness and system throughput. AI tasks are categorized into urgent and non-urgent types. For urgent tasks, it enables preemptive scheduling to give them higher processing priority, while non-urgent tasks are used to increase resource utilization by leveraging slack time.

B. Evaluation Results

We now report the performance evaluation results of `Convergo` and the baselines.

SLO Satisfaction. Fig. 6 shows the SLO satisfaction results of `Convergo` and the baselines. Each request has distinct accuracy, throughput, and deadline requirements. We measured: (1) "Meet All", which is the success rate when all requirements were met; and (2) the satisfaction rate for each individual requirement (*e.g.*, "Accuracy," "Throughput," and "Deadline", which are also shown in the graph).

`Convergo` achieved over 91% SLO satisfaction by meeting all three requirements simultaneously, which is significantly higher than the other baselines. Additionally, `Convergo` outperformed the other baselines by achieving high satisfaction rates for each individual requirement: 93% for accuracy, 95% for throughput, and 98% for deadlines met. Such a high SLO satisfaction rate is primarily due to the efficacy of `Convergo`'s scheduling algorithm, which enables the simultaneous execution of multiple AI models on different AI accelerators (*e.g.*, models on both GPUs and TPU accelerators). `Convergo`'s multi-constraint design highlights its novelty relative to simpler solutions that focus on a single metric. By contrast, the baselines are primarily designed to meet either deadline or throughput requirements, missing many multi-SLO considerations. Hense, the baselines' satisfaction rate for "Meet All" is less than 60% in Fig. 6. Additionally, they showed a satisfaction rate of less than 70% for the accuracy requirement and less than 80% for throughput requirements.

Over-Satisfied Inference Accuracy. We also measured the over-satisfied accuracy (defined as how much the actual accuracy surpassed the requirement) to demonstrate `Convergo`'s ability to maximize accuracy while meeting throughput and deadline requirements. As shown in Fig. 7, on average, `Convergo` provided 7% higher accuracy than required across the 10 evaluated workloads. We observed that `Convergo` provided over 8% over-satisfied accuracy for 4 of the evaluated workloads. These results confirm that `Convergo` not only achieves a high SLO satisfaction rate but also exceeds accuracy requirements in many cases, highlighting its additional ability to provide better quality of service (QoS).

Analysis of SLO-missed Cases. We analyzed how close the system came to meeting SLOs when they were missed. For the accuracy miss cases, `Convergo`'s results were only 1.3% below target accuracy; in contrast, the baselines were about 8% lower, confirming that `Convergo`'s accuracy achievements are much closer to its target accuracy. For the deadline miss cases, `Convergo` processed the missed jobs using over 58% of the request deadlines. However, SLO-MAEL processed the requests with 97% extra time over the deadline, and Kalmia used 134% extra time over the deadlines. These results showed that `Convergo`'s QoS degradation due to the missed accuracy requirement is negligible, and `Convergo` was able to complete the requests much earlier than the other baselines (even when the deadlines were not met).

However, the results measuring the gap between the achieved throughput and the target throughput (when the throughput requirements were not met) showed similar results in all three approaches. All three approaches showed 18% to 20% of missed throughput for the error cases.

Resource Utilization of Edge Devices. Fig. 8 reports the resource utilization and power consumption of the Jetson Xavier device for `Convergo` versus the baselines. In particular, `Convergo` increased CPU utilization by up to 12% and memory utilization by up to 10% over the baselines. This increased utilization is mainly due to `Convergo`'s sophisticated

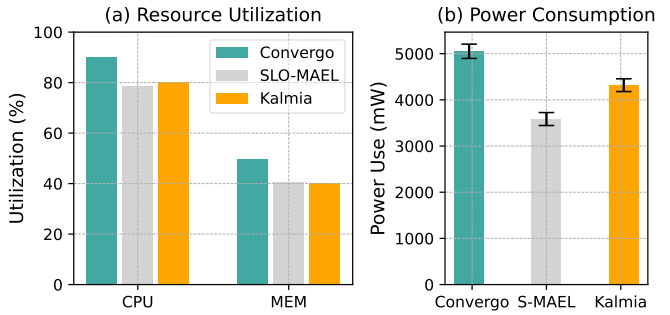

Fig. 8. **Resource Utilization and Power Consumption.**

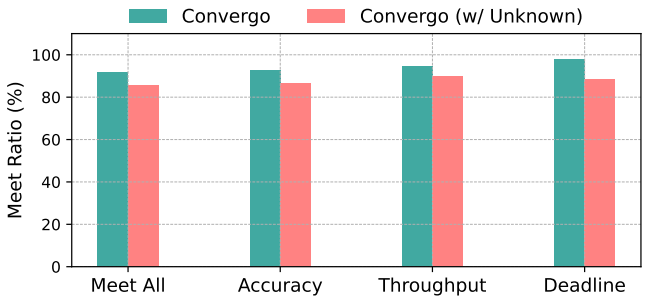

Fig. 9. **SLO Satisfaction Results with 'Unknown' Models.** *Green bars (left): SLO satisfaction rate with all profiled models (also in Fig. 6). Red bars (right): SLO satisfaction rate with 30% of unknown models.*

Table 3. `Convergo` **Scheduler Overhead.** *The scheduler overhead was calculated from extra CPU (%), MEM (%), and power (mW) consumption.*

	Average	Std Dev.
Extra CPU Consumption	6.3%	1.6
Extra Memory Consumption	4.9%	1.1
Extra Power Consumption	527.2mW	45

scheduling operations, which improved throughput and met SLO requirements. However, with respect to power consumption, `Convergo` consumed 700mW to 1500mW more power compared to the baselines. Despite `Convergo`'s significant improvements in other metrics (*e.g.*, multi-SLO satisfactions), its scheduling algorithm is more complicated and requires extra optimizations and operations that can consume more power. Therefore, improving power efficiency will be our next step in enhancing `Convergo`.

Overhead of `Convergo`. Table 3 details the extra resource consumption (*e.g.*, scheduler's resource consumption) of `Convergo` beyond normal inference. On average, `Convergo` required 4.9% and 6.3% CPU and memory for scheduling operattions, respectively. Additionally, it consumes about 527mW extra power for scheduling operations, which is less than 10% of the total power consumption of the Jetson Xavier device. We consider this overhead acceptable given `Convergo`'s performance gains and unique multi-SLO scheduling approach.

Evaluation with *'Unknown'* AI Models. The evaluations we performed so far relied on 'fully' profiled models for AI processing. However, `Convergo` is also designed to process AI applications with unknown (unprofiled) models, as discussed in §III-E. Therefore, we evaluated `Convergo`'s performance with unknown AI models. In this experiment, we removed two models from each AI application in Table 1 in §IV-A (*e.g.*, one model for GPU and another one for TPU). In total, we used 6 out of 21 models (about 30%) as unknown models. Additionally, this evaluation was performed with three randomly generated workloads, each with 500 requests.

SLO-satisfaction results of `Convergo` with the unknown models are shown in Fig. 9. The green bars represent results with all-profiled models (the same results as in Fig. 6 earlier in this section), and the red bars represent results with unknown models. As expected, we observed performance degradation with unknown models. For example, meeting all requirements ("Meet All") with unknown models had a 6.4% lower satisfaction rate. Other metrics, such as accuracy, throughput, and deadline miss rate, showed 6.3%, 3.5%, and 9.2% lower satisfaction rates, respectively. While there is performance degradation, we believe this is acceptable given that 30% of the models were unprofiled.

We further analyzed the frequency of SLO misses when processing the evaluated workloads. Fig. 10 shows the SLO miss frequencies for one evaluated workload. The blue dots indicate "accuracy miss" cases, the red dots indicate "throughput miss" cases, and the yellow dots indicate "deadline miss" cases. As shown in the graph, SLO miss cases often occurred in earlier requests (from the 1st to the 100th request), with fewer violations in later requests. These results suggest that early violations help improve the performance DB/model in `Convergo` for more accurate scheduling and demonstrate the efficacy of `Convergo`'s minimal slack-aware scheduling and performance estimation. As `Convergo` processed more requests, it was able to reduce violation cases and maintain a high overall SLO satisfaction rate. It is worth noting that `Convergo`'s performance with unknown models still showed a higher SLO satisfaction rate compared to the two baselines with all-profiled models, highlighting its adaptability.

V. Related Work

AI on Resource-Constrained Edge Devices. On-device AI inference serving often struggles to meet SLOs due to resource constraints, and multiple approaches have been proposed to facilitate on-device AI inference [13], [20], [23], [24], [38], [42]. Han *et al.* [13] and Lee *et al.* [20] explored efficient model compression techniques to reduce the computational load on edge devices. While these methods shrink model size and inference time, they often degrade performance under strict SLOs (*e.g.*, meeting both accuracy and latency) [27]. In contrast, `Convergo` optimizes model selection based on multiple SLO constraints, ensuring accuracy while meeting other SLOs like throughput and deadlines. Xie *et al.* [23] and Zhang *et al.* [42] proposed on-device inferencing frameworks that distribute heavy inferencing workloads across multiple devices. However, these approaches face high variability in model accuracy and latency due to heterogeneous edge hardware. `Convergo`

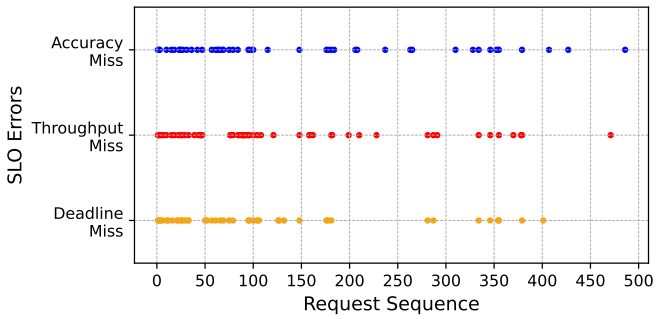

Fig. 10. **Frequency of SLO-misses Occurred in an Evaluated Workload with *'Unknown'* Models.**

tackles this by using a heterogeneous scheduling engine to dynamically select the best models and accelerators for each task, balancing accuracy, latency, and throughput. Wang *et al.* [38] introduced an adaptive inferencing that dynamically compress models based on resource availability. While this system reduces latency, they struggle to consistently meet multiple SLOs, such as latency and accuracy together, due to compression-induced accuracy drops. In contrast, `Convergo` uses a multi-SLO-aware scheduling algorithm to prioritize inference tasks and adapt to changing conditions, ensuring high SLO satisfaction.

Edge AI Schedulers. Several edge AI schedulers are designed to meet SLOs [9]–[11], [21], [30], but they often fail to address multiple SLOs simultaneously. For instance, SLOMAEL [30] prioritizes tasks based on expected latency and SLO violation risks but lacks mechanisms for managing heterogeneous resources and multi-SLO requirements, leading to suboptimal performance under complex SLOs (as shown in §IV). `Convergo` addresses these gaps by leveraging heterogeneous AI accelerators to meet multiple SLOs concurrently. Kalmia [11] and Dělen [21] optimize inference throughput and resource utilization but often at the expense of other critical SLOs like accuracy and deadlines. However, `Convergo` balances these conflicting requirements, ensuring all SLOs are met without compromising overall performance. Clipper [10] and Masa [9] focus on low-latency prediction serving and multi-DNN inference but lack flexibility for multi-SLO requirements in heterogeneous edge environments. `Convergo`'s multi-tenancy and model selection capabilities allow it to optimize multiple SLOs simultaneously, even in resource-constrained settings.

VI. Conclusion

In this paper, we presented `Convergo`, a new edge AI scheduler designed to meet multiple and complex SLOs concurrently. Unlike prior solutions that focus on a narrower scope, `Convergo`'s unique approach fuses heterogeneous device management, AI multi-tenancy, and multi-SLO prioritization into one framework. The scheduling algorithm prioritizes inference requests, applies critical constraints, and selects the best model combinations for concurrent executions. Implemented on the Jetson Xavier platform with TPU accelerators, `Convergo` was evaluated using realistic workloads spanning across various AI applications. Our evaluation, compared to state-of-the-art edge AI schedulers, shows that `Convergo` achieved over 90% satisfaction rate for concurrent SLOs, significantly outperforming the baselines. Additionally, we assessed its overhead and resource utilization, confirming that `Convergo` has an acceptable overhead for deployment on resource-constrained edge devices, demonstrating its effectiveness for edge AI.

Acknowledgment

The authors thank Jacob Stein for his contributions during the early phase of this work. This research was partially supported by the U.S. Department of Agriculture (USDA) under Grant No. 2021-67019-34342 and the National Science Foundation (NSF) under Grant No. CNS-2416214. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the USDA or NSF. This work was also partially supported by the Ministry of Education, Culture, Research, and Technology, Directorate General of Higher Education, under Grant No. 107/E5/PG.02.00.PL/2024.

References

[1] Coral TPU USB Accelerator datasheet. https://coral.ai/docs/accelerator/datasheet/.
[2] ML for Mobile and Edge Devices - TensorFlow Lite. https://www.tensorflow.org/lite.
[3] MoveNet: Ultra fast and accurate pose detection model. https://www.tensorflow.org/hub/tutorials/movenet.
[4] NVIDIA Jetson Xavier. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.
[5] Optimization with PuLP. https://coin-or.github.io/pulp/.
[6] Redis – The Real-time Data Platform. https://redis.io/.
[7] TensorRT Command-Line Wrapper: trtexec. https://github.com/NVIDIA/TensorRT/tree/main/samples/trtexec.
[8] Martín Abadi and et al. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Savannah, GA, USA, November, 2016.
[9] Bart Cox, Jeroen Galjaard, Amirmasoud Ghiassi, Robert Birke, and Lydia Y. Chen. Masa: Responsive Multi-DNN Inference on the Edge. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Kassel, Germany, March, 2021.
[10] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, March, 2017.
[11] Ziyan Fu, Ju Ren, Deyu Zhang, Yuezhi Zhou, and Yaoxue Zhang. Kalmia: A Heterogeneous QoS-aware Scheduling Framework for DNN Tasks on Edge Servers. In *IEEE Conference on Computer Communications (INFOCOM)*, London, United Kingdom, May, 2022.
[12] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO Series in 2021. *CoRR*, abs/2107.08430, 2021.
[13] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, Seoul, South Korea, June, 2016.
[14] Jianwei Hao, Piyush Subedi, Lakshmish Ramaswamy, and In Kee Kim. Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices with AI Multi-Tenancy. *ACM Transactions on Internet Technology (TOIT)*, 23(1):2:1–2:33, 2023.
[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861, 2017.

[16] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, abs/1602.07360, 2016.

[17] Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. Band: Coordinated Multi-DNN Inference on Heterogeneous Mobile Processors. In *The Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, Portland, Oregon, USA, June, 2022.

[18] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. Flexible High-resolution Object Detection on Edge Devices with Tunable Latency. In *International Conference on Mobile Computing and Networking (MobiCom)*, New Orleans, Louisiana, USA, October, 2021.

[19] Achintya Kundu, Laura Wynter, Rhui Dih Lee, and Luis Angel D. Bathen. Transfer-Once-For-All: AI Model Optimization for Edge. In *IEEE International Conference on Edge Computing and Communications (EDGE)*, Chicago, IL, USA, July, 2023.

[20] Hyunseung Lee, Jihoon Hong, Soosung Kim, Seung Yul Lee, and Jae W. Lee. A Memory-Efficient Edge Inference Accelerator with XOR-based Model Compression. In *ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, July, 2023.

[21] Qianlin Liang, Walid A. Hanafy, Noman Bashir, Ahmed Ali-Eldin, David E. Irwin, and Prashant J. Shenoy. Dělen: Enabling Flexible and Adaptive Model-serving for Multi-tenant Edge AI. In *ACM/IEEE Conference on Internet of Things Design and Implementation (IoTDI)*, San Antonio, TX, USA, May, 2023.

[22] Qianlin Liang, Prashant J. Shenoy, and David E. Irwin. AI on the Edge: Characterizing AI-based IoT Applications Using Specialized Edge Architectures. In *IEEE International Symposium on Workload Characterization (IISWC)*, Beijing, China, October, 2020.

[23] Neiwen Ling, Kai Wang, Yuze He, Guoliang Xing, and Daqi Xie. RT-mDL: Supporting Real-Time Mixed Deep Learning Tasks on Edge Platforms. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Coimbra, Portugal, November, 2021.

[24] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proceedings of IEEE*, 107(8):1697–1716, 2019.

[25] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin Murphy. PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model. In *European Conference on Computer Vision (ECCV)*, Munich, Germany, September, 2018.

[26] Adam Paszke and et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, BC, Canada, December, 2019.

[27] Rakandhiya D. Rachmanto, Zaki Sukma, Ahmad N. L. Nabhaan, Arief Setyanto, Ting Jiang, and In Kee Kim. Characterizing Deep Learning Model Compression with Post-Training Quantization on Accelerated Edge Devices. In *IEEE International Conference on Edge Computing and Communications (EDGE)*, Shenzhen, China, July, 2024.

[28] Dheeraj Ramchandani, Bahar Asgari, and Hyesoon Kim. Spica: Exploring FPGA Optimizations to Enable an Efficient SpMV Implementation for Computations at Edge. In *IEEE International Conference on Edge Computing and Communications (EDGE)*, Chicago, IL, USA, July, 2023.

[29] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, June, 2018.

[30] Wonik Seo, Sanghoon Cha, Yeonjae Kim, Jaehyuk Huh, and Jongse Park. SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms. *ACM Transactions on Architecture and Code Optimization (TACO)*, 18(4):1–26, 2021.

[31] Arief Setyanto, Theopilus Bayu Sasongko, Muhammad Ainul Fikri, Dhani Ariatmanto, I Made Artha Agastya, Rakandhiya Daanii Rachmanto, Affan Ardana, and In Kee Kim. Knowledge Distillation in Object Detection for Resource-Constrained Edge Computing. *IEEE Access*, 13:18200–18214, 2025.

[32] Arief Setyanto, Theopilus Bayu Sasongko, Muhammad Ainul Fikri, and In Kee Kim. Near-Edge Computing Aware Object Detection: A Review. *IEEE Access*, 12:2989–3011, 2024.

[33] Piyush Subedi, Jianwei Hao, In Kee Kim, and Lakshmish Ramaswamy. AI Multi-Tenancy on Edge: Concurrent Deep Learning Model Executions and Dynamic Model Placements on Edge Devices. In *IEEE International Conference on Cloud Computing (CLOUD)*, Sep., 2021.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper With Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, June, 2015.

[35] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, June, 2016.

[36] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *The International Conference on Machine Learning (ICML)*, Long Beach, CA, USA, 2019.

[37] Guanchu Wang, Zaid Pervaiz Bhat, Zhimeng Jiang, Yi-Wei Chen, Daochen Zha, Alfredo Costilla-Reyes, Afshin Niktash, Mehmet Görkem Ulkar, Osman Erman Okman, Xuanting Cai, and Xia Ben Hu. BED: A Real-Time Object Detection System for Edge Devices. In *ACM International Conference on Information & Knowledge Management (CIKM)*, Atlanta, GA, USA.

[38] Lingdong Wang, Liyao Xiang, Jiayu Xu, Jiaju Chen, Xing Zhao, Dixi Yao, Xinbing Wang, and Baochun Li. Context-Aware Deep Model Compression for Edge Cloud Computing. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Singapore, Nov., 2020.

[39] Ning Wang, Jiangrong Xie, Hang Luo, Qinglin Cheng, Jihao Wu, Mingbo Jia, and Linlin Li. Efficient Image Captioning for Edge Devices. In *AAAI Conference on Artificial Intelligence (AAAI)*, Washington, DC, USA, February, 2023.

[40] Zhujun Xiao, Zhengxu Xia, Haitao Zheng, Ben Y. Zhao, and Junchen Jiang. Towards Performance Clarity of Edge Video Analytics. In *IEEE/ACM Symposium on Edge Computing (SEC)*, San Jose, CA, USA, December, 2021.

[41] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices. *IEEE/ACM Transactions on Networking*, 29(2):595–608, 2020.

[42] Xiaojie Zhang, Houchao Gan, Amitangshu Pal, Soumyabrata Dey, and Saptarshi Debroy. On Balancing Latency and Quality of Edge-native Multi-view 3D Reconstruction. In *IEEE/ACM Symposium on Edge Computing (SEC)*, Wilmington, DE, USA, December, 2023.

[43] Yanfu Zhang, Shangqian Gao, and Heng Huang. Exploration and Estimation for Model Compression. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada, October, 2021.