Divide and Save: Splitting Workload Among Containers in an Edge Device to Save Energy and Time

Aria Khoshsirat, Giovanni Perin, and Michele Rossi

Department of Information Engineering (DEI)
University of Padova (Padova, Italy)
Emails: aria.khoshsirat@unipd.it, giovanni.perin.1@unipd.it, michele.rossi@unipd.it

Abstract—The increasing demand for edge computing is leading to a rise in energy consumption from edge devices, which can have significant environmental and financial implications. To address this, in this paper we present a novel method to enhance the energy efficiency while speeding up computations by distributing the workload among multiple containers in an edge device. Experiments are conducted on two Nvidia Jetson edge boards, the TX2 and the AGX Orin, exploring how using a different number of containers can affect the energy consumption and the computational time for an inference task. To demonstrate the effectiveness of our splitting approach, a video object detection task is conducted using an embedded version of the state-of-the-art YOLO algorithm, quantifying the energy and the time savings achieved compared to doing the computations on a single container. The proposed method can help mitigate the environmental and economic consequences of high energy consumption in edge computing, by providing a more sustainable approach to managing the workload of edge devices.

Index Terms—Energy Efficiency, Inference Time, Edge Computing, Containers, Object Detection

I. INTRODUCTION

Multi-access edge computing (MEC) is a rapidly growing field that entails performing computational tasks at the network's edge, closer to the data source. This approach, other than inherently lowering the total communication and computation latency, provides other benefits, such as improving security and privacy. However, as the demand for edge computing services increases, it becomes increasingly important to address the issue of the joint optimization of energy efficiency and computational time. In this paper, we refer to energy efficiency as to the ability of executing a computational unit of a task (CPU cycles) using the least amount of energy possible. This is crucial for two main reasons. First, edge computing devices are often powered by batteries or have limited power supplies. Energy efficiency is thus key to ensure their operation for long periods of time and for making their computations feasible, even with the limited amount of energy available. Second, as the number of edge devices increases, the overall energy consumption of the edge computing system increases accordingly. This can lead to a significant environmental footprint and to unsustainable economic costs for the service providers.

In this paper, we investigate how splitting a computing task into multiple containers within the same Nvidia Jetson device impacts the energy drained and the execution time. Instances of the popular deep learning-based algorithm "You only look once" (YOLO) [1] are used, employing YOLOv4-tiny [2], which is suitable for execution on energy and memory constrained hardware. According to a recent survey [3], MEC schedulers decide, other than container placement and migration, *how* incoming tasks should be allocated to containers. This paper provides useful insights on how to optimally allocate *splittable tasks* into multiple containers from an energy and execution time perspective. As such, it is indirectly related to the concept of *split computing* [4], i.e., the division of a neural network (NN) model into a head and a tail, to be executed on different devices (and, thus, on different containers). Notably, however, split-computing introduces time dependencies, as the head has to be executed *before* the tail. The application chosen in this paper, namely, YOLO, does not require keeping temporal dependencies into account since frames are processed *independently* of one another. For now, we leave the problem of time dependencies for future studies. To summarize, the main contributions of this work are:

- We provide a new approach to improve energy efficiency and decrease the computational time of edge computing by splitting the computations among multiple containers.
- Testbed results of our experiments are shown, performed on two commercial edge devices of the series Nvidia Jetson, namely, the TX2 and the AGX Orin. They provide evidence of the effectiveness of our method in reducing both energy consumption and processing time.
- We obtain simple convex models for the energy consumption and inference time of commercial edge devices as a function of the number of containers used to split the task. Such models can be used to effectively schedule the computation of workload in a MEC server.

We believe this paper will be helpful for researchers, practitioners, and policymakers who are interested in the energy and time optimization of MEC platforms.

The remainder of this paper is organized as follows: The related work is briefly reviewed in Section II. In Section III, we

arXiv:2302.06478v2 [cs.DC] 23 Mar 2023

present some background on YOLO and Docker containers. The experimental setup is explained in Section IV. The method is detailed in Section V, while the results are given together with the discussion in Section VI. Finally, conclusions and future research lines are discussed in Section VII.

II. RELATED WORK

Recent work has modeled the energy consumption of neural network models over Nvidia Jetson hardware [5]–[8]. Specifically, in [5], the authors develop a framework to measure the energy consumption of specific layers of a convolutional neural network (CNN) on a Jetson TX1. Papers [6] and [7] instead profile the energy consumption of Jetsons TX2 and Nano, providing an optimized set of parameters to increase energy efficiency. In our previous work [8], we profiled Jetsons TX2 and Xavier, also providing models to estimate their energy consumptions based on the neural network layers features. In [9], the authors compare containers, virtual machines and multi-thread computing on a parallel benchmark. They show that the overhead of using Docker containers is very small compared to virtual machines.

To the best of our knowledge, our paper is the first providing an energy and latency profiling of task splitting on Jetson devices. Notably, this is of great interest for a number of applications concerning container allocation and migration [3] and energy efficiency [10] at the network's edge. In [11], the authors develop a Kubernetes-based [12] container scheduler that takes into account the problem of energy minimization. A similar problem involving container migration in an urban vehicular context is treated in [13], with the objective of minimizing the MEC carbon footprint. However, both these papers consider the tasks as unsplittable monolithic entities. In the present work, we show that splitting tasks among multiple containers, when possible, is beneficial both in terms of energy consumption and computational time.

III. APPLICATION BACKGROUND

A. YOLO

YOLO [1] is a state-of-the-art object detection algorithm that is widely used in computer vision applications. This algorithm is based on a deep CNN architecture and can detect objects within an image or a video frame in real-time. One of the key advantages of YOLO is its ability to detect multiple objects within a single forward pass of the network, as opposed to traditional object detection methods that require multiple runs. This allows the model to achieve a faster detection rate and a higher accuracy compared to previous algorithms. Additionally, YOLO's architecture allows for easy integration with other computer vision tasks such as object tracking and semantic segmentation. YOLO has shown good performance in various applications such as self-driving cars, surveillance and augmented reality (AR) systems. However, its performance is sensitive to the quality of the training dataset and the architecture design. Therefore, researchers have proposed improvements and variants to the original algorithm to improve it. In this paper, as a case study, we consider an

TABLE I: Device hardware specifications

	Jetson TX2	Jetson AGX Orin
CPU	Quad-core ARM Cortex-A57 + Dual-core Denver 2*	12-core Arm Cortex-A78
GPU	256-core NVIDIA Pascal	2048-core NVIDIA Ampere
Memory	8 GB 128-bit LPDDR4	32GB 256-bit LPDDR5
Performance	1.33 TFLOPs	200 TOPS

*Denver cores are turned off by default for consistency

object detection task on video data using the YOLOv4-Tiny [2] algorithm. YOLOv4-tiny is based on YOLOv4 [14] with the objective of making the YOLO neural network structure simpler (fewer parameters), which makes it suitable for mobile and embedded devices with power and memory constraints.

B. Docker Containers

To create containers we have used Docker [15]. A Docker container is a lightweight executable package including everything that is needed to run a piece of software, including the code, a runtime, system tools, libraries, and settings. Containers provide a consistent way to package and distribute software, making it easier to deploy and run applications on different environments. Containers are isolated from one another and from the host system, so they can run without conflicts, even when multiple containers run on the same host. Containers are based on Docker images, which are snapshots of a container's file system at a specific point in time. It is possible to limit the CPU resources of a Docker container, which is usually done to control the amount of resources the host uses to execute it. This may be used to enforce a fair distribution of computing resources among the different running containers and processes. The "--cpus" runtime option allows us to specify the number of CPU cores that the container can use. For instance, the command "docker run --cpus=2 Yolo-Container" limits the created container to use only 2 CPU cores.

IV. EXPERIMENTAL SETUP

For the experiments of this work, we employed two Nvidia edge devices: Jetson TX2 and Jetson AGX Orin. The Jetson TX2 is a previous generation cost-effective edge device with a powerful GPU and CPU, which can run complex algorithms and process large amounts of data at the edge. The Jetson AGX Orin, on the other hand, is a newer and more powerful platform with a higher number of CPU cores and more memory, making it ideal for running more demanding tasks. Both platforms support a wide range of neural networks and machine learning frameworks, making them versatile tools for research and industry applications alike. Table I shows the hardware specifications of these devices. Note that for our experiments we have only used 4 ARM CPU cores on the Jetson TX2 since the Denver cores are disabled by default in the device due to incompatible performance.

Although these devices have powerful GPUs, we have used only the CPU for our experiments. This is because
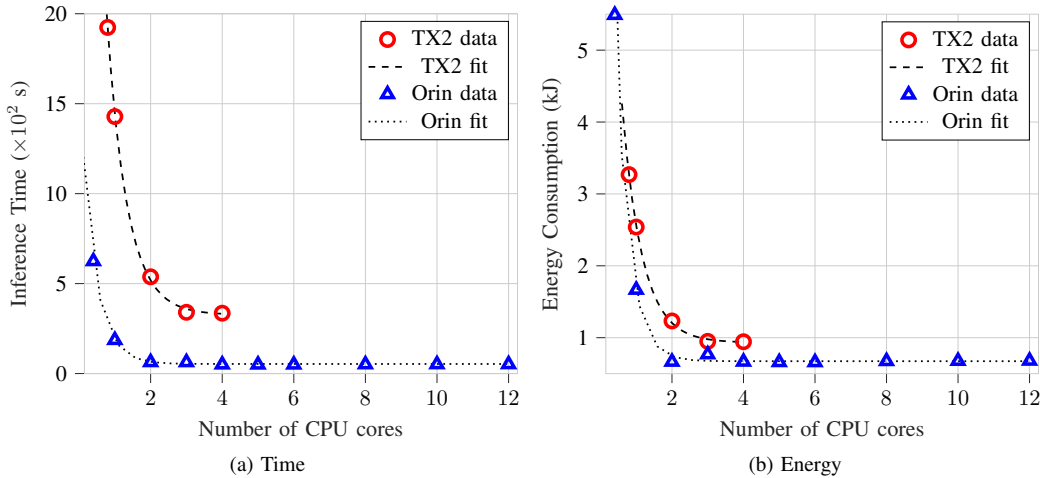
Fig. 1: Plots showing the variation in inference time and energy consumption as the number of CPU cores allocated to a single container is increased, on Jetson TX2 and Jetson AGX Orin edge devices.

CPU cores can be readily divided among containers, whereas a container's allocation in terms of GPU resources cannot be easily enforced. In fact, without any restriction on GPU usage, containers end up competing for the majority of GPU resources, making it challenging for the GPU scheduler to operate efficiently. For more than two containers on the TX2 and four containers on the AGX Orin, this leads to irregular and slow computation. It is worth noting that in an edge device that has multiple GPUs, the method outlined in this paper can still be applied by assigning each GPU to a specific (different) container, with no conflicts.

For measuring the power of these devices, we have used the built-in power monitoring sensors included in both Nvidia boards. Such sensor can be read with a sampling time of about 10 milliseconds, which is accurate enough for our experiments. The energy consumption is then calculated by taking the sum of the power readings multiplied by the time period between subsequent power samples.

The base experiment that is evaluated in this paper consists of the application of object detection on a 30-second-long video, using YOLOv4-Tiny. Notably, through experimentation with object detection on videos with differing formats, we found that the *number of frames* in a video has the greatest impact on the energy and time needed for YOLO inference. Other characteristics of a video, such as the frame size, the bitrate, or even the number of objects per frame, have minimal effect on the time and energy, and thus can be neglected.

As an initial step to demonstrate a baseline, we performed the task on each device using one container with a varying number of CPU cores available to it. Fig. 1 shows the time and energy of performing the object detection task for the full 30-second video, as a function of the number of CPUs allotted to the container. This is a real number and in our experiment is varied from 0.1 (ten percent of one CPU core) to the number of cores available on the device. On the Jetson TX2, we observe that using four CPU cores only results in

a slight improvement in time and energy efficiency when compared to using three cores. Doing the same experiment on the AGX Orin device exhibits a similar behavior; employing more than two CPU cores minimizes efficiency. This led us to conjecture that increasing the utilization of the CPU cores for computations and parallelizing the workload, could enhance the efficiency of the computations on edge devices.

V. METHODOLOGY

This study aims to propose a method to decrease the energy consumption and computational time of an inference task, namely, the object detection task introduced in the previous section, on edge devices. To achieve this, the following methods are used:

1) **Data splitting:** The test data, in our case the whole input video, is split into equal size segments. The splitting is done along the time dimension of the video, resulting in the same number of frames for each segment, to be used as input for the object detection task.
2) **Creating containers:** We subsequently generate a number of containers matching the number of data segments, with each container running an instance of the YOLO model to perform the inference.
3) **Dividing computational resources:** The processing units, i.e., the CPU cores, are evenly split among the containers. Each container receives a share of the maximum processing capacity of the device, depending on the number of containers that are created.
4) **Parallelization:** The inference is carried out on all the containers *simultaneously*, each accessing its designated segment of input data and using the available processing units. The results from all the containers are then combined and presented to the user.

To show the impact of utilizing multiple containers in parallel, we conducted experiments by varying the number of segmented sections and corresponding containers. Then,
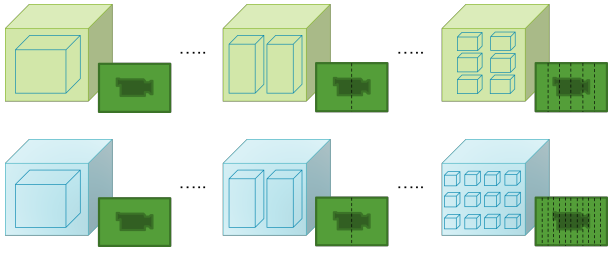
Fig. 2: Different experiment scenarios for Jetson TX2 (green) and Jetson AGX Orin (blue) devices. The processing units are equally split between the containers in each device. The input video is also split into equal segments along the temporal dimension, resulting in each container processing an equal number of frames.

the inference time and the energy drained in each scenario were recorded. We evaluated the performance of the proposed method on both Jetson TX2 and AGX Orin devices and we repeated the experiments multiple times to ensure the accuracy of the results. As the confidence interval for the measured data of the multiple runs is less than 1%, we have opted not to display it in the plots. The number of containers and split segments used in our experiments was limited by the memory capacity and processing power of each edge device, with a maximum of six containers on the Jetson TX2 and twelve containers on the AGX Orin. Fig. 2 depicts the process.

VI. EMPIRICAL RESULTS AND ANALYSIS

In the following plots, as a *benchmark*, we consider the case of a single container running YOLO, i.e., a single container, with no data splitting, and all CPU cores (four CPU cores for TX2 and twelve for Orin). All performance metrics (average power, energy and processing time) are normalized with respect to those in the benchmark scenario.

Fig. 3a and Fig. 3b respectively show the normalized *inference time* and *energy consumed* for an increasing number of containers used to execute the object detection task, as explained in Section V. For the Jetson TX2, when running the task on two containers, each using two of the four cores and processing half of the video frames in parallel, we observe a 19% reduction in the inference time (Fig.3a) and a 10% reduction in the energy consumption (Fig. 3b) compared to the benchmark. Increasing the number of containers to four reduced the time further by 25% and the energy by 15%. As the number of containers increases beyond four, the system performance degrades in terms of both time and energy. We believe that when the number of containers is increased beyond the number of available CPU cores on the Jetson TX2, it becomes challenging for the CPU scheduler to allocate the CPU cores effectively, worsening the performance.

On the AGX Orin, which is a more powerful and energy-efficient edge device, splitting the computations between two and four containers respectively results in reductions of 43% and 62% in inference time, and in 25% and 40% reductions in energy consumption, as compared to utilizing a single container. Additionally, increasing the number of containers to twelve on the Orin results in the most efficient scenario for the considered video object detection task. This leads to reductions of about 70% in the inference time and 43% in the energy consumption. However, time and energy curves flatten beyond four containers: since memory resources are used to open new containers, limiting to four can be a good choice.

By analyzing the average power of the devices in each scenario, we get a deeper understanding of how the processing resources are employed in each case. Fig. 3c shows the average power for an increasing number of containers. We see that splitting the resources and the data among multiple containers leads to a power increase. For the TX2, from one to four containers, we measured a 13% increase in the average power, while, for the AGX Orin, the increase is about 84% for twelve containers. This increased average power reflects a better utilization of the available processing resources (CPU cores) on the edge device, which translates into a higher energy efficiency.

Model fitting is also performed for each of the performance metrics (time, power, and energy). The inferred formulae are given in Table II. These fitted models (Fig. 3) can be beneficial for estimating the savings when applying our method, based on a given reference value ("Ref." in Tab. II). In our case the reference value corresponds to the metrics for the benchmark scenario.

We believe that presenting energy and time metrics, in addition to the average power for the different scenarios, provides valuable insights for understanding fundamental trade-offs in edge computing systems. These trade-offs are particularly useful in cases where there are constraints on power or energy usage for the devices. It is also worth noting that the approach outlined in this paper can be extended to other similar tasks and models, not just the video object detection tasks using YOLO-v4Tiny as used in our experiments. We also applied the proposed splitting method to a simple CNN inference task. Splitting the input data (images) between containers led to similar improvements.

VII. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed a simple methodology to distribute the workload among multiple containers within an edge device. Experiments were conducted on Nvidia Jetson TX2 and AGX Orin boards to analyze the effects on energy consumption and computational time of splitting data and assigning them to multiple containers, while always using all the CPU cores available. Our experimental results show that this procedure yields significant reductions in energy consumption and computational time for an inference task on both devices. Notably, this approach is simple and easy to implement, making it a practical solution for edge computing systems employing embedded limited-memory devices.

We remark that the data for the object detection task that was considered in this paper could easily be split into a number of segments by neither negatively impacting the performance nor the accuracy of the model's inference. This descends from
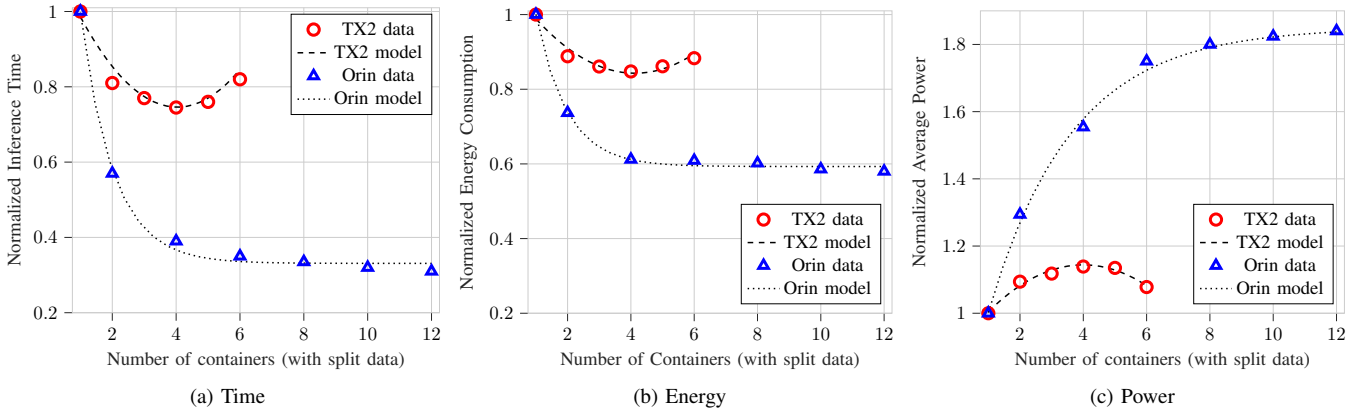
Fig. 3: Normalized energy consumption, computation time and average power to execute the object detection task on Jetson TX2 and AGX Orin devices in different scenarios. In every scenario, we use a specific number of containers, each having an equal amount of divided data and processing units.

TABLE II: Reference values and fitted models (x means number of containers).

| | **TX2** | | **AGX Orin** | |
	Ref.	**Model**	**Ref.**	**Model**
Time	325 s	$0.026x^2 - 0.21x + 1.17$	54 s	$0.33 + 1.77e^{-0.98x}$
Energy	942 J	$0.015x^2 - 0.12x + 1.10$	700 J	$0.59 + 1.14e^{-1.03x}$
Power	2.9 W	$-0.016x^2 + 0.12x + 0.90$	13 W	$1.85 - 1.24e^{-0.38x}$

the way in which YOLO works, i.e., processing video frames *independently* of one another. Our presented approach is thus only applicable when the task can be split into independent subtasks. One possible direction for future work could be to investigate the applicability of our method to other types of tasks and models, such as tasks involving data with time correlation or that can only be split into *dependent* subtasks. Additionally, it would be interesting to explore the use of our splitting approach in a distributed edge computing setting, where multiple devices collaborate to perform a task. Finally, our method, as well as the results presented in this paper, can be used in the design of energy-efficient job schedulers that split input data, obtaining the optimal number of containers in an online fashion in order to enhance the energy efficiency and reduce the processing time of the edge system.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[2] Z. Jiang, L. Zhao, S. Li, and Y. Jia, "Real-time object detection method based on improved yolov4-tiny," *CoRR*, vol. abs/2011.04244, 2020. [Online]. Available: https://arxiv.org/abs/2011.04244

[3] O. Oleghe, "Container placement and migration in edge computing: Concept and scheduling models," *IEEE Access*, vol. 9, pp. 68 028–68 043, 2021.

[4] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, vol. 55, no. 5, dec 2022. [Online]. Available: https://doi.org/10.1145/3527155

[5] C. F. Rodrigues, G. Riley, and M. Luján, "Fine-grained energy profiling for deep convolutional neural networks on the jetson tx1," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, 2017, pp. 114–115.

[6] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2020, pp. 1–5.

[7] S. Holly, A. Wendt, and M. Lechner, "Profiling energy consumption of deep neural networks on nvidia jetson nano," in *2020 11th International Green and Sustainable Computing Workshops (IGSC)*, 2020, pp. 1–6.

[8] S. Lahmer, A. Khoshsirat, M. Rossi, and A. Zanella, "Energy consumption of neural networks on nvidia edge boards: an empirical model," in *2022 20th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*, 2022, pp. 365–371.

[9] B. Wang, J. Xie, S. Li, Y. Wan, S. Fu, and K. Lu, "Enabling high-performance onboard computing with virtualization for unmanned aerial systems," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 202–211.

[10] N. Shalavi, G. Perin, A. Zanella, and M. Rossi, "Energy efficient deployment and orchestration of computing resources at the network edge: a survey on algorithms, trends and open challenges," *arXiv preprint arXiv:2209.14141*, 2022.

[11] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4228–4237, 2020.

[12] Google. Kubernetes. [Online]. Available: https://kubernetes.io/

[13] G. Perin, F. Meneghello, R. Carli, L. Schenato, and M. Rossi, "EASE: Energy-Aware Job Scheduling for Vehicular Edge Networks With Renewable Energy Resources," *IEEE Transactions on Green Communications and Networking*, pp. 1–1, 2022.

[14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[15] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.