Multi-Accelerator Neural Network Inference via TensorRT in Heterogeneous Embedded Systems

Yuxiao Zhou
Texas State University
y_z37@txstate.edu

Zhishan Guo
North Carolina State University
zguo32@ncsu.edu

Zheng Dong
Wayne State University
dong@wayne.edu

Kecheng Yang
Texas State University
yangk@txstate.edu

***Abstract*—Neural Network Inference (NNI) has become a critical element in mobile and autonomous systems, particularly for time-sensitive operations like obstacle detection and avoidance. Alongside execution time, energy consumption holds significant importance in such workloads, given that power is a limited resource in these systems. Modern System-on-Chips (SoCs) in mobile and autonomous devices are equipped with a diverse range of accelerators, each characterized by distinct power and performance features. Adapting to dynamically changing physical conditions, the execution flow of these crucial workloads can be optimized to utilize multiple accelerators, allowing for a flexible trade-off between performance and energy consumption.**

In this study, we leverage multiple accelerators within an SoC to execute NNI using NVIDIA TensorRT. Our primary goal is to enable an energy-performance trade-off by intelligently distributing layers of a neural network between accelerators that prioritize performance and those that emphasize power efficiency. Initially, we analyze the execution time and energy characteristics of neural network layer execution on various accelerators. Subsequently, we examine various factors influencing layer execution. Finally, we propose two algorithms to determine the mapping of layers to accelerators, minimizing energy consumption while adhering to a predetermined target NN inference execution time.

We evaluate our approaches on the NVIDIA AGX Orin SoC using the commonly used ResNet50 model. According to the experiment results, we suggest adopting a coarse-grained layer grouping strategy. For applications with stringent real-time requirements, it is recommended to utilize the proposed LTN approach to better achieve the target execution time. Alternatively, in other scenarios, the Knapsack approach may be chosen for potential improvements in energy consumption.

***Index Terms*—deep neural networks, heterogeneous systems, neural networks inference, autonomous systems, SoC, TensorRT, PyTorch, DLA, edge device.**

I. Introduction

Over the last decade, there has been a widespread growth of autonomous systems, driven by tremendous advancements made in high-performance computing (HPC), machine learning (ML), and robotics. These advancements have supplied the necessary technical capabilities for achieving full autonomy. When it comes to safety-critical tasks on autonomous systems, there is a crucial need for high-performance computing to meet stringent deadlines for safe execution. However, the escalating computational requirements lead to increased energy consumption, effectively equating the power needed for computation with that required for mechanical operations in a cyber-physical system (CPS). Therefore, it is crucial to achieve a balance between energy efficiency and performance aligning with the capabilities of the underlying platform.

Despite the remarkable accuracy and precision achieved by large, and perhaps enormous, deep neural networks, their training and inference runtime can be prolonged and sluggish. Moreover, such extensive model architectures may consume significant computing resources, even for inference alone. However, many applications and systems, especially those embedded, require real-time inference while utilizing limited hardware resources due to size, weight, power, and cost (SWaP-C) constraints. For example, autonomous vehicles must promptly process data from diverse sensors, including cameras and lidars, to make proper control decisions on a SoC. Similarly, a video surveillance system must analyze video footage in real-time to detect abnormal activities and trigger timely warnings. On the other hand, due to privacy and reliability concerns, much of such computation must be performed on embedded platforms with limited computing resources.

Contemporary mobile and embedded SoCs, such as Qualcomm's Snapdragon [21], Xilinx Zynq Ultrascale+ MPSoC [7], and NVIDIA's Jetson platforms [18], incorporate a diverse array of specialized accelerators designed for specific domains. These accelerators are instrumental in performing critical tasks with minimal latency and power consumption. For instance, NVIDIA's Orin SoC is equipped with two dedicated programmable accelerators: an NVIDIA Deep Learning Accelerator (DLA) and a Programmable Vision Accelerator (PVA), in addition to a graphical processing unit (GPU) [13].

In some scenarios, certain compute-intensive deep learning operations, such as convolution, can be performed using various types of accelerators, including DLA, GPU, or PVA, on platforms like NVIDIA Jetson. Each of these accelerators exhibits distinct performance and power characteristics. When faced with such a situation, it may be advantageous to collaboratively distribute the workload across different processing units to optimize resource efficiency and utilization. For example, while the GPU on Orin often provides the best average execution times, the DLA offers 3-5x more power efficiency compared to the GPU. Under a certain energy budget, a practical approach could involve finding a balance between assigning layers to the GPU for lower execution time and assigning other layers to the DLA for energy efficiency. Such a hybrid approach and trade-offs could be particularly useful for systems where executing all layers on the GPU would lead to excessive power consumption while executing

This work is supported in part by NSF grants CNS-2104181, CMMI-2246671, CNS-2103604, CNS-2140346, and CNS-2113817.

all layers on the DLA would result in unacceptable end-to-end latency.

In mobile and autonomous systems, performance objectives are not static; rather, they are dynamically influenced by their environments. Moreover, these systems typically operate with limited system resources, such as constrained battery capacity. While the conventional practice involves adjusting processors' power and frequency using techniques like Dynamic Voltage and Frequency Scaling (DVFS), the recent availability of alternative accelerators on heterogeneous platforms presents a new execution paradigm: collaborative Multi-Accelerator Execution (MAE). The goal of collaborative MAE is to explore various options, aligning computing resource usage with the practical physical requirements of the system. However, this approach has received limited attention in the realm of research [5].

For instance, the research [20] focuses on optimizing task distribution among diverse accelerators to improve resource utilization while adhering to specified constraints. Houssam-Eddine et al. [11] introduced a real-time application model for implementing different software component for different processing engines on heterogeneous hardware, with an emphasis on latency-sensitive task scheduling. Dagli et al. [5] propose a metric in order to measure the energy or performance benefits of MAE of a given workload. The team's other work [6] provided an empirical modeling methodology to characterize execution and inter-layer transition times and find an optimal layers-to-accelerator mapping by representing the trade-off as a linear programming optimization constraint. Odema et al. [19] proposes a GNN architectural design space along with potential mapping options on a heterogeneous SoC, aiming to identify model architectures that optimize on-device resource efficiency. Another research [1] from the same team proposes a framework to identify an optimal partitioning scheme of the NN along its 'width' dimension, which facilitates deployment of concurrent NN blocks onto different hardware computing units. To the best of our knowledge, none of the existing studies are able to address the following challenges for MAE: finding layer-to-accelerator mapping that minimizes energy consumption with predetermined target NNI execution time.

Contribution. In this research, we focus on diversely heterogeneous systems, aiming to distribute the execution of NNI across different types of accelerators. Our objective is to investigate and understand the trade-off between execution time and energy consumption. We achieve this by exploring the execution of various NNI workloads on a heterogeneous system through strategic partitioning of layers among multiple accelerators. Each layer is allocated to a specific accelerator based on its capabilities to enhance performance for a target execution completion time.

This paper contributes in the following ways: We demonstrate the existence of a trade-off between performance and energy consumption. This trade-off can be effectively managed by distributing and executing layers across diverse accelerators. We examine various factors affecting the mapping of layers to accelerators, such as the layer grouping granularity, the overhead of MAE transition, and the input batch size. These factors have not been previously investigated. We introduce two multi-accelerator execution schemes for diversely heterogeneous SoCs, which identify schedules with near-optimal energy consumption for a given execution completion time. We evaluate two proposed algorithms on the NVIDIA AGX Orin by using its embedded GPU and DLA. We discuss the pros and cons of both algorithms.

Organization. The rest of this paper is organized as follows: Sec. II gives a background overview of TensorRT and NVIDIA DLA. Sec. III presents the motivation, challenges, and considerations of our research. Sec. IV describes a case study, including experiment setup, proposed algorithms, and evaluation of algorithms. Sec. VI provides more related works, while Sec. VII concludes our work.

II. Background and Related Frameworks

NVIDIA Orin SoCs consist of various programmable accelerators, including a GPU, a PVA, and two DLAs, complemented by robust tools for performance and energy measurement [13]. In this research, we focus on MAE for NNI on the DLA and GPU of the Orin platform.

TensorRT. TensorRT is a comprehensive framework comprising an NNI optimizer and runtime, aimed at achieving exceptional performance across a range of platforms. The TensorRT engine builder is a critical component that implements pre-runtime optimizations, including layer fusing, precision fine-tuning, and reduction of memory requirements. [4].

NVIDIA Deep Learning Accelerator (DLA). NVIDIA DLA is a specialized hardware accelerator designed to accelerate NNI tasks. It incorporates a dedicated pipeline with layer-specific engines optimized for convolution, activation, pooling, and reshaping operations, all of which are essential in neural network computations. In the NVIDIA Orin SOC, DLA offers significant performance capabilities, achieving 52.5 Tera Operations Per Second (TOPS) with int8 precision at MAXN power mode[3].

In NVIDIA Orin SoC, the DLA plays a crucial role in the overall deep learning performance, contributing between 38% and 74% of the total DL performance, depending on the power mode. Despite its high performance, the power consumption of DLA is considerably lower than that of the GPU, resulting in an impressive performance-per-watt ratio. On average, the DLA offers 3–5 times better performance per watt compared to the GPU, depending on the power mode and workload [3]. Offloading inference tasks to the DLA allows the main GPU to focus on other computations or remain in a low-power state, contributing significantly to overall energy efficiency.

III. Multi-Accelerator Execution

A. Motivation

Heterogeneous SoCs integrate diverse cores with different performance and energy tradeoffs [2]. In heterogeneous SoCs, an operation within an application may be accelerated through distinct accelerators, each possessing varying performance,

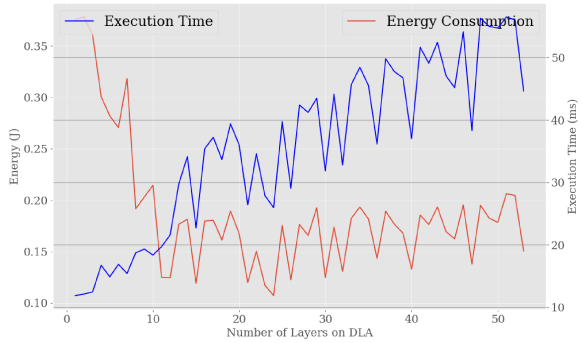


Fig. 1. Tradeoff between energy consumption and execution time.

power efficiency, and execution time characteristics[5]. Optimizing the optimal execution time and energy efficiency for such an operation requires considering both the capabilities of the accelerator and the properties of this operation. To address various system requirements and runtime parameters of the operation, it becomes essential to have the capability to map different operations to diverse accelerators throughout the application's execution. Conversely, the collaborative execution of workloads like NNI on multiple accelerators represents a relatively novel and unexplored approach that holds the potential for delivering unique advantages in budgeted execution scenarios.

To illustrate the viability of such executions, we present preliminary findings from an experiment outlined in Fig. 1. Our results demonstrate that distributing the layer of an NNI across a GPU and a DLA in a collaborative manner could allow for a customizable balance between power consumption and performance on NVIDIA's Orin system. Specifically, by executing more layers of the network on the DLA while running the remainder on the GPU, we can notably enhance power efficiency at the cost of slightly increased execution time and vice versa[5].

B. Challenges

In neural network inference, achieving the desired tradeoff necessitates a meticulous allocation of workloads across accelerators. Nevertheless, employing multiple accelerators to optimize the system's utilization while adhering to resource constraints, presents a set of challenges.

Lack of flexibility in layer-to-accelerator assignment. The lack of flexibility in assigning layers to accelerators poses a significant challenge. Each accelerator has specific limitations regarding the types of operations it can effectively handle. Currently, the DLA in Jetson Orin SoC can only be accessed through the TensorRT library, with some generic and layer-wise limitations. For instance, some less common layers like LeakyReLU and GlobalAveragePool are not supported by TensorRT for DLA execution. Additionally, NVIDIA's DLA imposes restrictions on layer parameters and batch sizes; for instance, it does not support dynamic dimensions, and the maximum allowed batch size is 4096 [17]. While TensorRT treats activation functions as distinct layers within its layer set, it doesn't allow the separate assignment of activation layers and other layers across different accelerators. Furthermore, transitioning from DLA to GPU after specific layers is not allowed in TensorRT [6]. These constraints lead to certain layers fall back to the GPU, even if the intention was for them to execute on the DLA. This situation may require flushing all transient data from the previous layer back to the system memory. As a result, the potential transition points between accelerators are inflexible and depend on both the neural network's architecture and the characteristics of the accelerators.

Profiling. Certain highly specialized accelerators, including DLA, process consecutively assigned layers as a unified entity and do not permit internal profiling of execution times and energy consumption on a layer-by-layer basis. This constraint poses a significant challenge for mapping layers to accelerators, as it hinders detailed performance characterization at a layer level.

C. Our Considerations

Developing a comprehensive schema for multi-accelerator neural network inference on diversely heterogeneous SoCs execution requires careful consideration of the following aspects.

Characterizing execution time and power consumption. Processing units such as CPU, GPU, and DLA have distinct architectural designs and constraints, influencing the workloads they can handle To enable heterogeneity-aware assignment of layers, it is crucial to measure or predict characteristic features of layers. Recent performance analysis on neural networks in embedded environments demonstrates computation complexity and memory communication dependent on numerous parameters. These parameters can be challenging to detect and optimize during compile time. Key factors such as kernel size, operation type, input matrix dimensions, activation size, and activation function significantly impact the execution time and power consumption of operations in neural networks. Understanding and considering these parameters are critical for effective workload distribution and optimization across different accelerators [5].

Load balancing to maximize system efficiency. Achieving maximum efficiency in the system requires careful consideration of load balance, particularly because PUs possess varying computing power capabilities, and tasks consist of multiple subtasks with differing complexities. Different types of PUs exhibit diverse capabilities in running kernels based on the operation type and data size. To optimize performance utilizing the available PUs, it is essential to distribute the workload among PUs, considering estimated or expected execution time and energy consumption. Load distribution is a well-studied mapping problem, known to be NP-complete [23]. Existing solutions, often utilize heuristics or dynamic scheduling techniques to effectively manage load distribution across PUs. Failing to achieve a balanced distribution of workloads across PUs may lead to an overall slowdown of the system, underscoring the importance of effective load-balancing strategies for maximizing system efficiency [5].

IV. Methodology

In this section, we provide an in-depth explanation of our methodology, focusing on the selection of MAE transition points (MTPs) and the establishment of layer mappings in collaborative MAE, using a case study as a framework. Initially, we examine various factors impacting MTP selection including layer grouping granularity, characterization of layer groups across different accelerators, overhead of MAE transition, and input batch size. Following this, we introduce two algorithms specifically designed to identify near-optimal locations for the MTPs.

A. Setup

In this research, we utilized NVIDIA's Jetson AGX Orin SoCs. This system is chosen for its incorporation of one high-performance GPU and two DLAs. Our experimentation was restricted to utilizing a single DLA due to TensorRT's limitation in simultaneously employing multiple DLAs for executing a given neural network. NVIDIA provides a comprehensive solution, JetPack, for hardware-accelerated AI-at-the-edge development on Nvidia Jetson modules. The most up-to-date JetPack version facilitated the necessary configuration for our experiment. This setup encompassed TensorRT 8.5.2, cuDNN 8.6.0 and CUDA 11.4.19 [15].

We leveraged the TensorRT engine to optimize pre-trained PyTorch models from the PyTorch TorchVision library repository. Specifically, we conducted experiments using the ResNet50 [10], a classic neural network used as a backbone for many computer vision tasks. The other reason we focus on ResNet50 is that all layers in ResNet50 can be scheduled on both the GPU and the DLA. This allows us to flexibly explore all possible layer-to-accelerator assignments, without the TensorRT engine falling back to GPUs.

For transitioning a specific layer from the GPU to DLA, manual programming is necessary. We first check whether a layer can run on DLA by using canRunOnDLA TensorRT API calls, and then set the device that this layer must execute on through the setDeviceType TensorRT API call. Conversely, transitioning back to GPU from DLA occurred when a planned DLA execution fell back to GPU or setDeviceType was not specified for a particular layer. Additionally, all layers within a network could be globally configured to execute on DLA by setting the useDLACore parameter for the TensorRT runtime executable, trtexec [17].

B. Factors Affecting the Selection of MAE Transition Point

We examine the trade-off related to MAE by exploring the selection of the MTPs, shown in Fig. 2, which determines the transition of execution flow between accelerators. We make three assumptions: (1) execution starts in either the GPU or DLA, (2) there exists multiple MTPs during the execution of an NNI, and (3) the DLA and GPU are mutually exclusive and not utilized simultaneously.

Layer grouping granularity. In convolutional neural networks (CNNs), fundamental layers like Convolution layers are commonly succeeded by specific operations like ReLU and Pooling. These sequences of layers are often consolidated into larger layer groups to reduce memory transfers, decrease computation overhead, and enhance overall inference speed. This optimization, called fusion, is carried out automatically during the TensorRT engine building phase. Fusions are normally handled by creating a new layer with a name containing the names of both of the layers that were fused. For example, in ResNet50, a Convolution layer named `/conv1/Conv` is fused with a ReLU Activation layer named `relu/Relu`; to create a new layer named `/conv1/Conv + relu/Relu` [17]. We refer to these layer groups based on TensorRT fusion as fine-grained groups.

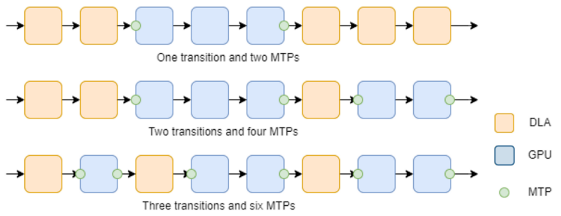


Fig. 2. Diagram to illustrate various numbers of MTPs.

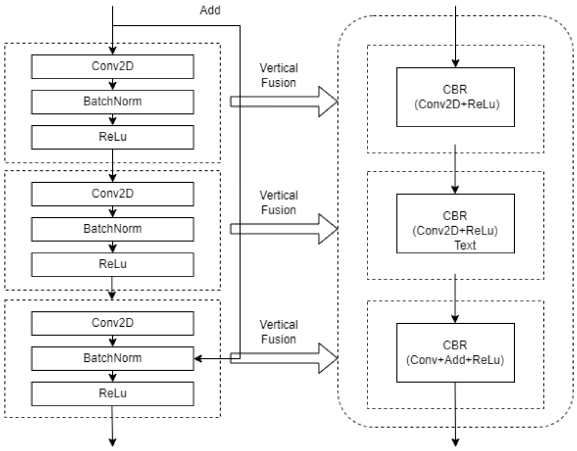


Fig. 3. Layer grouping based on bottleneck.

In the ResNet50, the bottleneck Residual Block is proposed, as a combination of a sequence of three convolution layers utilizing filters with sizes of 1×1, 3×3, and 1×1 respectively. These bottleneck blocks are often replicated multiple times, allowing for variations in matrix and convolutional filter sizes [25]. To optimize memory usage and reduce access overheads between layers within the bottleneck, we group 53 fused layers into 17 groups for group-to-accelerator mapping. We refer to these bottleneck-based layer groups as coarse-grained groups. Both grouping strategies are employed in this research to analyze how the granularity of layer grouping affects the overall network performance.

Characterization of layer groups. Once we establish the granularity for layer grouping, we proceed to analyze accelerator performance during distinct group executions. We assess how each accelerator performs when handling these grouped layers and measure the neural network's resource utilization on a group-by-group basis. This evaluation allows us to understand how efficiently the resources are utilized within each group.

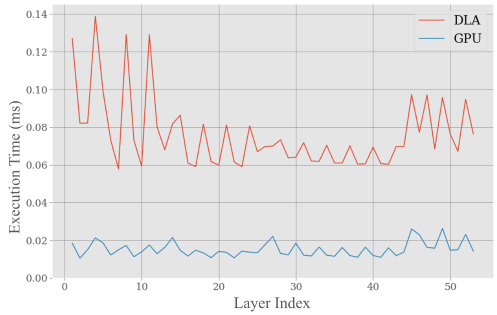


Fig. 4. Execution time of fine-grained Group on DLA and GPU.

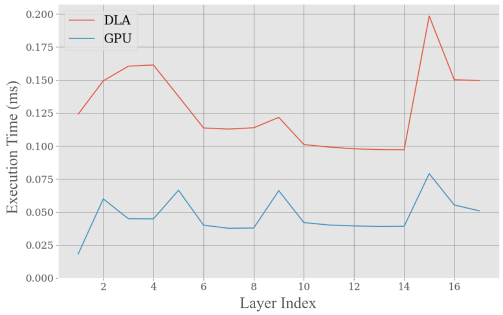


Fig. 5. Execution time of coarse-grained Group on DLA and GPU.

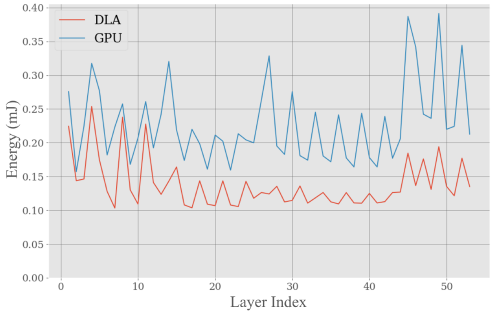


Fig. 6. Energy consumption of fine-grained group on DLA and GPU.

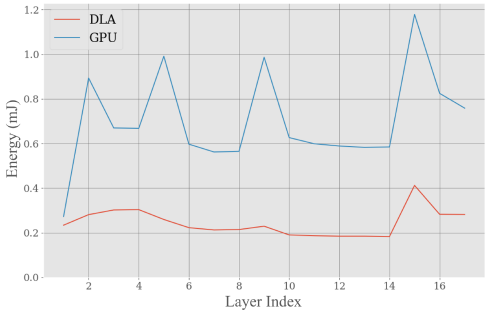


Fig. 7. Energy consumption of coarse-grained group on DLA and GPU.

(1) Execution time on DLA and GPU. In the experimentation process, we initially allocate all layers to the GPU and profile the inference time of each layer on the GPU using `trtexec`. Subsequently, we iterate through the experiment by transitioning from GPU to DLA for each layer group, measuring the execution time of these layers on DLA.

The results are illustrated in Fig. 4 and Fig. 5, demonstrating the impact of transitioning each layer group on execution time. It's observed that depending on the data size, operations within fine-grained groups on the GPU exhibit a speed advantage ranging from approximately 3.13x to 7.03x over the DLA. On the other hand, within coarse-grained groups, the GPU's speed advantage over DLA fluctuates between 1.84x to 6.78x. The DLA demonstrates improved performance when processing layers towards the earlier stages of ResNet50. This is attributed to the usage of smaller kernel sizes in those layers. Smaller kernels require less computation and are more efficient to execute. The GPU excels at leveraging greater data parallelism when using larger kernels, while the compact buffers in the DLA prove highly effective with smaller matrix sizes.

(2) Energy consumption on DLA and GPU. We obtain the energy consumption by integrating the PU-specific power consumption values reported by tegrastats, a utility that provides information on memory usage and processor usage for Tegra-based devices. The reported power consumption values are for three power rails: `VDD_GPU_SOC` (GPU and SOC Combined power rail), `VDD_CPU_CV` (CPU and CV Combined power rail) and `VIN_SYS_5V0` (System 5V power rail) [16]. From these values, we estimate the GPU and DLA power consumption. The energy consumption is then computed by multiplying the estimated average power consumption by the execution time. We show the comparison of energy consumption of each layer group on GPU and DLA in Fig. 6 and Fig. 7. For energy consumption, operations in fine-grained groups on the GPU utilize 1.34x to 3.01x more energy compared to the DLA. In the coarse-grained group, the GPU's energy consumption exceeds that of the DLA by a factor ranging from 1.06x to 3.92x.

Overheads of MAE transition. Transferring a layer from one accelerator to another involves relocating the input/output data of the layer across the memory subsystems of both units. In Jetson Orin, particularly within the DLA, there is a local buffer for efficiency and reduced DRAM width. When executing a group of layers in the DLA, if the execution needs to transition to the GPU, this switch preserves the state of the local buffer in the DLA by transferring it to the shared memory of the SoC.

Once the last DLA layer's output becomes visible to other PUs within the SoCs, the GPU proceeds with executing the remaining layers using the CuDNN kernel call integrated by the TensorRT engine. Additionally, after the transition, the initial memory instructions executed by the GPU lead to cold cache misses. Consequently, this triggers a warm-up phase, resulting in a slowdown in layer execution for an unspecified duration. The duration of this slowdown depends on factors such as cache size, the number of ports, and the available memory bandwidth [6].

We estimate the overhead of MAE transition by calculating the difference between the inference time of the combined layer and the cumulative execution time of separate layers. The calculated transition overhead is illustrated in both Fig. 8 and Fig. 9 for both groups. Fig. 8 suggests that there is a relatively higher MAE transition overhead between each bottleneck in ResNet50.

Impact of batch size. It is noteworthy the preceding exper-

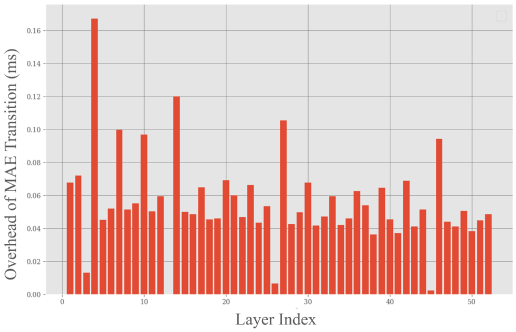


Fig. 8. Overhead of MAE transition on fine-grained Group.

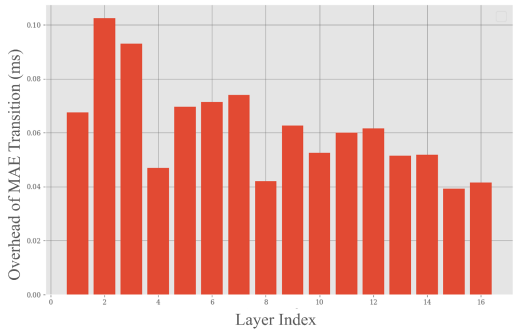


Fig. 9. Overhead of MAE transition on coarse-grained Group.

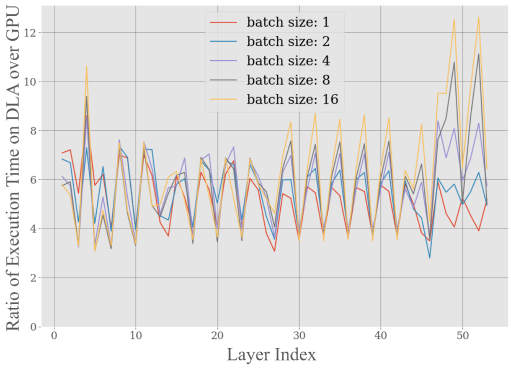


Fig. 10. Impact of batch size on execution Time of fine-grained group on DLA and GPU.

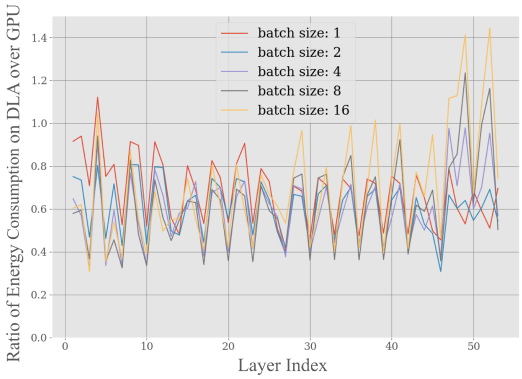


Fig. 11. Impact of batch size on energy consumption of fine-grained group on DLA and GPU.

iments were conducted with a batch size set to one. This specific configuration provides a baseline understanding of the system's performance under minimal processing load. Given that increasing the batch size often improves total throughput, we want to further investigate how variations in batch size can impact overall execution time and energy consumption on both DLA and GPU.

Illustrated in Fig. 10 through Fig. 13 are the relative energy and power consumption ratios of DLA, with GPU values serving as the baseline, for both fine-grained group and coarse-grained group. The trend in the result underscores that DLA exhibits optimal performance-to-energy efficiency when operating with lower batch values.

C. MAE Transition Point Selection

Having examined the factors influencing the choice of the MTP, we propose two distinct approaches targeted at selecting the MTP while adhering to specified execution time constraints. The core idea underlying these approaches is to minimize energy consumption while imposing an upper limit on the execution time of NNI. By integrating execution time constraints into the selection process for the MTP, we aim to ensure that the system operates within defined time boundaries while striving to achieve the most energy-efficient configuration.

Given that DLA demonstrates optimal performance and energy efficiency at lower batch sizes, our experiments are carried out within the batch size range of 1 to 4.

Limited Number of Transitions Approach. The Limited Number of Transition (LNT) approach involves exploring the optimal set of a restricted number of MTPs, all aimed at minimizing energy consumption.

Initially, the method generates combinations of n MTPs from the entire set while satisfying predefined execution time constraints. Consistent with previous studies, as noted in [6], we impose a limitation on the value of n, restricting it to six. Subsequently, the algorithm identifies the near-optimal combination that not only satisfies the execution time constraints but also achieves the maximum reduction in energy consumption. This iterative process ensures a thorough search for the most energy-efficient configuration considering both execution time requirements and the need for energy resource conservation.

Knapsack Approach. In this approach, the MTP selection is modeled as a Knapsack problem. The objective is to navigate a set of CNN layer groups, each linked with an energy gain (profit) and an execution time (weight). The challenge is to identify a subset of these layer groups that yield the maximum total energy gain while adhering to an execution time requirement that should not exceed the given capacity, similar to a container (knapsack).

The energy gain is calculated as the relative energy consumption ratios of GPU, with DLA values serving as the baseline. This approach utilizes the knapsack to solve the problem of MTP selection, ensuring a trade-off between energy consumption and execution time constraints.

V. Experiment Results and Evaluation

In our experiment evaluating the the feasibility of proposed algorithms, we provide the profiling results of the layer group

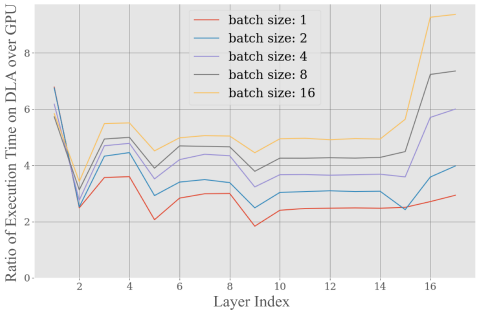


Fig. 12. Impact of batch Size on execution Time of coarse-grained group on DLA and GPU.

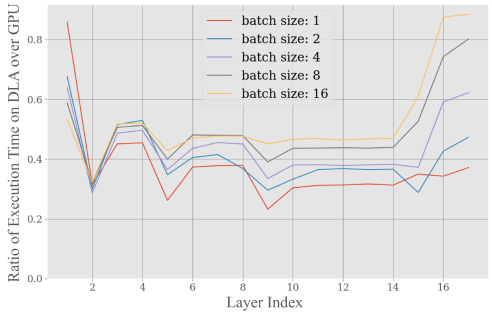


Fig. 13. Impact of batch size on energy consumption of coarse-grained group on DLA and GPU.

including execution time, and energy, as inputs to the algorithms. The algorithms then compute near-optimal transition points as output based on the provided data. Subsequently, we use these transition points to generate TensorRT engines, enabling efficient optimization of the NNI for MAE. It's noteworthy that valid MTPs cannot be found under certain execution time constraints in some cases.

We present our results on execution time from both approaches in Fig. 14 and Fig. 15. The X-axis represents the target execution completion time. The Y-axis illustrates the actual measured execution times, corresponding to each specific upper limit on execution time. In Table I and Table II, we list the number of executions completed within time constraints on both groups.

In summary, our experiment results indicate engines acquired via both approaches can complete the majority of inferences within predefined time constraints on fine-grained groups. However, the measured times fall significantly below the specified limits for the LNT approach with one transition. These differences imply that some layers still remain on the GPU, potentially leading to additional energy consumption. The cause might be that the layer execution time reported by the TensorRT profiler may include the transition overhead, and having only one transition point may not result in significant overhead.

In coarse-grained groups, it is worth noting engines obtained through the Knapsack approach incur less execution time than all engines from the LNT approach when the input batch size is set to one. However, with an increased input batch size of four, the Knapsack approach can only finish one inference out of four within the specified time limits.

MTP Selection Approach	LNT: 1 Transition	LNT: 2 Transitions	LNT: 3 Transitions	Knapsack
Batch Size = 1	4	3	3	2
Batch Size = 2	4	3	3	3
Batch Size = 4	4	4	4	3

TABLE I
NUMBER OF EXECUTIONS COMPLETED IN TARGET TIMES ON FINE-GRAINED GROUPS.

MTP Selection Approach	LNT: 1 Transition	LNT: 2 Transitions	LNT: 3 Transitions	Knapsack
Batch Size = 1	4	3	1	3
Batch Size = 2	4	3	2	2
Batch Size = 4	4	3	2	1

TABLE II
NUMBER OF EXECUTIONS COMPLETED IN TARGET TIMES ON COARSE-GRAINED GROUPS.

From Fig. 16 to Fig. 17, we present an extensive comparison of the energy consumption for the TensorRT engines derived from both the LTN and the Knapsack Approach.

Based on the comparison results, the advantage of a coarse-grained layer grouping strategy over a fine-grained one is obvious. For the fine-grained groups, engines produced through both approaches might have a higher energy consumption compared to an engine where all layers are allocated to the GPU especially when the target time is limited. This phenomenon is not observed when employing coarse-grained grouping.

It is noted that engines created using the Knapsack approach demonstrate lower energy consumption than those generated by the LTN Approach within the coarse-grained group when the input batch size exceeds one. However, this comes at the expense of a bit higher execution time costs, surpassing the target time.

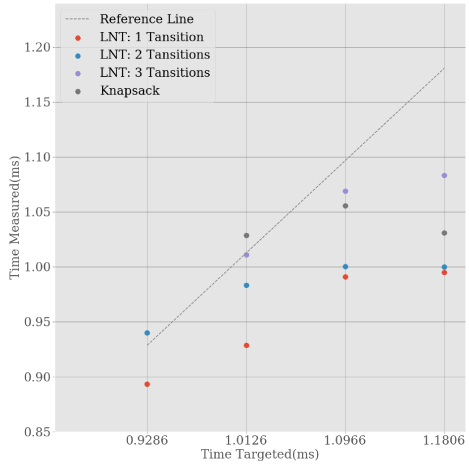
While increasing the number of transitions in the LTN approach doesn't significantly impact energy consumption, there is a consistent rise in execution time with an increased number of transitions.

Based on prior observations, we recommend using a coarse-grained layer grouping strategy for layer mapping on accelerators. For applications with strict real-time requirements, we suggest adopting the LTN approach and setting the number of transitions to one. Conversely, in scenarios where this is not the case, we recommend utilizing the Knapsack approach.

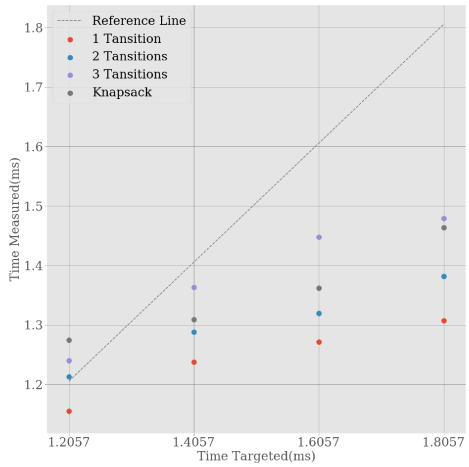
VI. RELATED WORK

Yang et al. [24] proposed a hierarchical DL tasks distribution framework, where the unmanned aerial vehicles (UAV) are embedded with lower layers of the pre-trained CNN model, while the mobile edge computing (MEC) server with rich computing resources will handle the higher layers of the CNN model. This research work mainly focuses on the trade-off between the weighted-sum cost and inference error rate in the proposed framework.

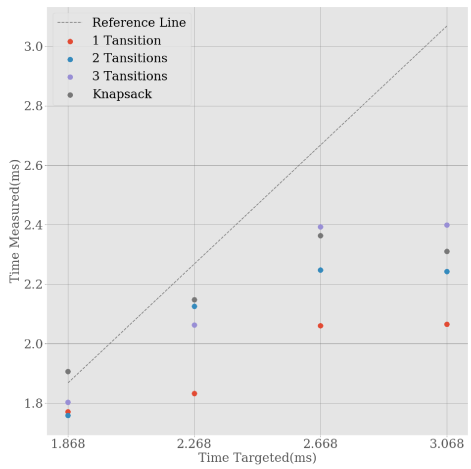
There are also research works focusing on the knowledge of the energy consumption behaviors of different CNNs and training frameworks. For example. Li et al. [14] performed an extensive assessment of the energy efficiency of training frameworks for deep CNN. Martin et al. [9] offers an overview



(a) batch size = 1

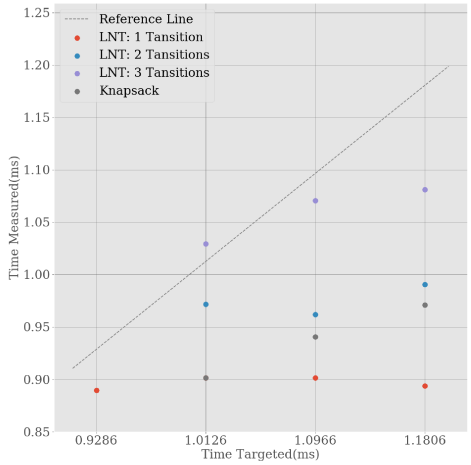


(b) batch size = 2

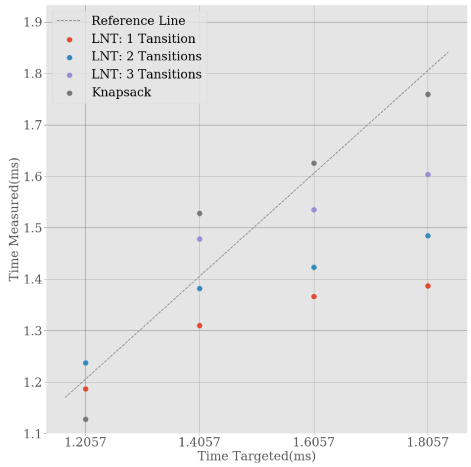


(c) batch size = 4

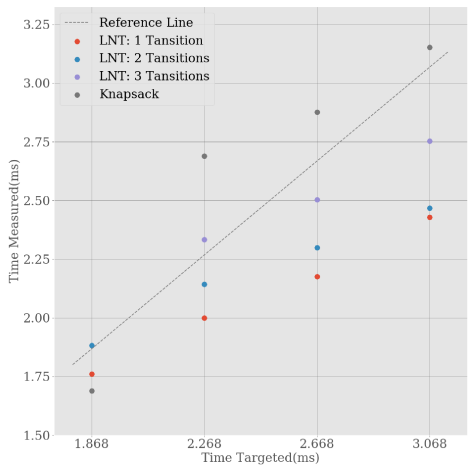
Fig. 14. Times measured vs target times on fine-grained groups.



(a) batch size = 1



(b) batch size = 2

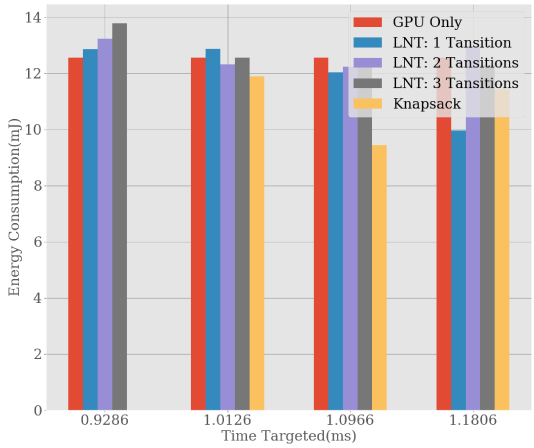


(c) batch size = 4

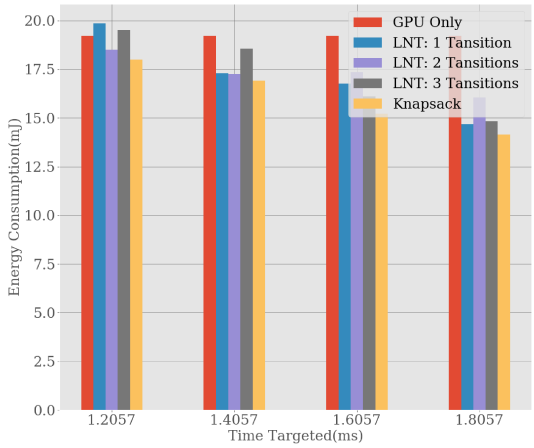
Fig. 15. Times measured vs target times on coarse-grained groups.

of various methods for estimating energy consumption in both general computational contexts and, more specifically, within machine learning applications. They also introduce software tools that provide energy estimation metrics. However, all these studies do not consider account the trade-off between time and energy as a metric in multi-accelerator systems.

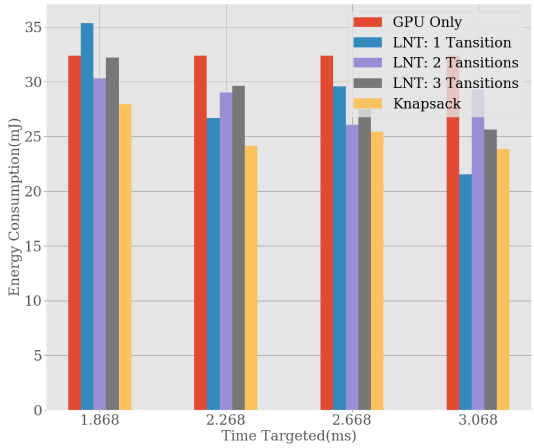
To increase neural network performance, some researchers emphasize data parallelism, effectively splitting the data be-



(a) batch size = 1

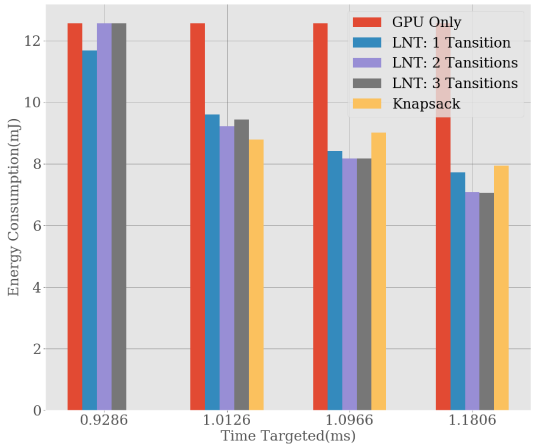


(b) batch size = 2

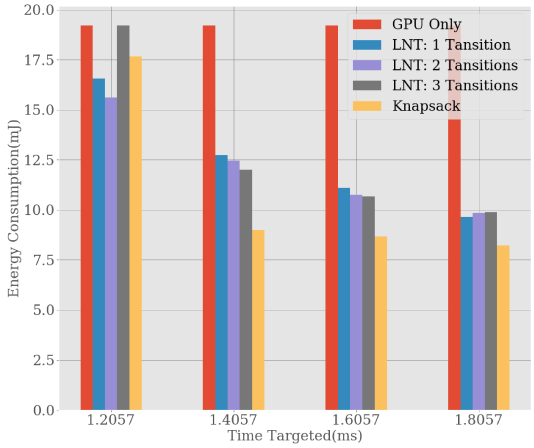


(c) batch size = 4

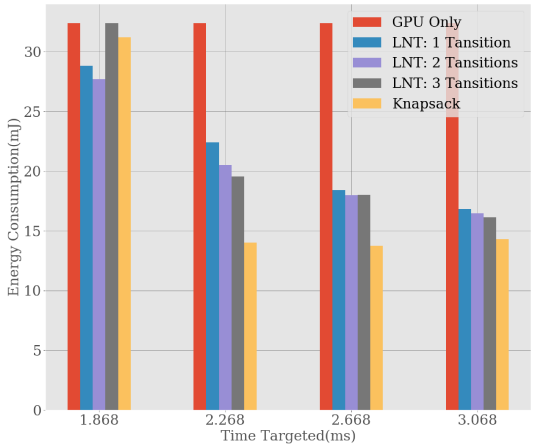
Fig. 16. Energy consumption on fine-grained groups.



(a) batch size = 1



(b) batch size = 2



(c) batch size = 4

Fig. 17. Energy consumption on coarse-grained groups.

tween available devices. Collaboratively utilizing multiple accelerators can outperform a single PU. Achieving a well-distributed workload between them, depending on the data size, can lead to a higher speed-up. In this research [8], a framework is introduced that seamlessly integrates FPGAs, enabling effective cooperation between CPU, GPU, and FPGA. Song et al. [22] propose a solution named HYPAR to determine layer-wise parallelism for deep neural network training with an array of DNN accelerators. They use a hierarchical layer-wise dynamic programming method to search for the

partition for each layer. However, these studies do not consider energy as a metric when distributing workloads among PUs.

The improvement of neural network effectiveness, considering latency per energy, is typically achieved by automating hardware allocation for Deep Neural Networks (DNNs). Kao et al. propose a specialized genetic algorithm-based technique called GAMMA, outlined in their work [12]. GAMMA is tailored to address the per-layer DL accelerator mapping problem. Their approach involves constructing a flexible mapping space and demonstrates GAMMA's ability to efficiently navigate this space and determine an optimized mapping with a high degree of sample efficiency. The optimization objective of GAMMA encompasses both latency and energy minimization. It's important to note that GAMMA primarily focuses on optimizing the tiling strategy, computation order, and parallelization strategy, distinguishing its scope from the objectives of our work.

VII. Conclusion

This study investigates the trade-off between energy consumption and execution time within deep neural network models on SoCs with diverse heterogeneity. The research highlights how different accelerators within the system could benefit specific layers within a neural network differently in terms of execution time and energy consumption. We discuss the motivation, challenges, and consideration of MAE, exploring factors affecting the selection of MTPs in MAE. We proposed two algorithms, the LTN approach and the Knapsack approach, for MAE transition point selection which minimize the energy consumption while achieving target neural network inference execution times. We evaluate our methodology using ResNet50 on NVIDIA Jetson Orin SoCs. Based on the experiment results, we recommend using a coarse-grained layer grouping strategy. For applications with strict real-time requirements, consider employing the LTN approach, while in other scenarios, opt for the Knapsack approach for potentially better energy consumption.

References

[1] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous mpsocs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[2] Xing Chen, Anish Krishnakumar, Umit Ogras, and Chaitali Chakrabarti. Ped: Probabilistic energy-efficient deadline-aware scheduler for heterogeneous socs. *Journal of Systems Architecture*, 147:103051, 2024.

[3] Ram Cherukuri and Oliver Knieps. Maximizing deep learning performance on nvidia jetson orin with dla. Online at https://developer.nvidia.com/blog/maximizing-deep-learning-performance-on-nvidia-jetson-orin-with-dla/.

[4] NVIDIA Corporation. Nvidia tensorrt. Online at https://developer.nvidia.com/tensorrt.

[5] Ismet Dagli and Mehmet E Belviranli. Multi-accelerator neural network inference in diversely heterogeneous embedded systems. In *2021 IEEE/ACM Redefining Scalability for Diversely Heterogeneous Architectures Workshop (RSDHA)*, pages 1–7. IEEE, 2021.

[6] Ismet Dagli, Alexander Cieslewicz, Jedidiah McClurg, and Mehmet E Belviranli. Axonn: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1069–1074, 2022.

[7] Advanced Micro Devices. Zynq™ ultrascale+™ mpsoc. Online at https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

[8] María Angélica Dávila Guzmán, Raúl Nozal, Rubénand Gran Tejero, María Villarroya-Gaudó, Darío Suárez Gracia, and Jose Luis Bosque. Cooperative cpu, gpu, and fpga heterogeneous execution with enginecl. *The Journal of Supercomputing*, 75:1732–1746, 2019.

[9] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Zahaf Houssam-Eddine, Nicola Capodieci, Roberto Cavicchioli, Giuseppe Lipari, and Marko Bertogna. The hpc-dag task model for heterogeneous real-time systems. *IEEE Transactions on Computers*, 70(10):1747–1761, 2021.

[12] Sheng-Chun Kao and Tushar Krishna. Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[13] Leela S. Karumbunathan. Nvidia jetson agx orin series. Online at https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf.

[14] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 477–484. IEEE, 2016.

[15] NVIDIA. Jetpack sdk 5.1. Online at https://developer.nvidia.com/embedded/jetpack-sdk-51.

[16] NVIDIA. Jetson orin nx series and jetson agx orin series. Online at https://docs.nvidia.com/jetson/archives/r35.2.1/DeveloperGuide/text/SD/PlatformPowerAndPerformance/JetsonOrinNxSeriesAndJetsonAgxOrinSeries.html.

[17] NVIDIA. Nvidia deep learning tensorrt documentation. Online at https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#dla_topic.

[18] NVIDIA. Nvidia jetson. Online at https://www.nvidia.com/en-sg/autonomous-machines/embedded-systems/.

[19] Mohanad Odema, Halima Bouzidi, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. Magnas: A mapping-aware graph neural architecture search framework for heterogeneous mpsoc deployment. *ACM Transactions on Embedded Computing Systems*, 22(5s):1–26, 2023.

[20] Roger Pujol, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. Generating and Exploiting Deep Learning Variants to Increase Heterogeneous Resource Utilization in the NVIDIA Xavier. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133, pages 23:1–23:23, 2019.

[21] Qualcomm. Deliver next-generation mobile experiences with snapdragon mobile platforms and processor technologies. Online at https://www.qualcomm.com/products/technology/processors/mobile-processors.

[22] Linghao Song, Jiachen Mao, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. Hypar: Towards hybrid parallelism for deep learning accelerator array. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 56–68, 2019.

[23] Wenhong Tian, GuoZhong Li, Xinyang Wang, Qin Xiong, and Yaqiu Jiang. Transforming np to p: An approach to solve np complete problems. *arXiv preprint arXiv:1505.00058*, 2015.

[24] Bo Yang, Xuelin Cao, Chau Yuen, and Lijun Qian. Offloading optimization in edge computing for deep-learning-enabled target tracking by internet of uavs. *IEEE Internet of Things Journal*, 8(12):9878–9893, 2020.

[25] Qi Zhao, Jiahui Liu, Boxue Zhang, Shuchang Lyu, Nauman Raoof, and Wenquan Feng. Interpretable relative squeezing bottleneck design for compact convolutional neural networks model. *Image and vision computing*, 89:276–288, 2019.