Energy-efficient use of an embedded heterogeneous SoC for the inference of CNNs

Agathe Archet, Nicolas Ventroux, Nicolas Gac, François Orieux

Energy-efficient use of an embedded heterogeneous SoC for the inference of CNNs

Agathe Archet *† Nicolas Ventroux * Nicolas Gac † François Orieux †

* Thales Research and Technology, Avenue Augustin Fresnel, 91120 Palaiseau, France

† Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des Signaux et Systèmes
3 rue Joliot Curie, 91190 Gif-Sur-Yvette, France

Email: agathe.archet@thalesgroup.com

***Abstract*—Energy efficiency is key in many embedded systems in order to reach the best performance on a limited power budget. In addition, new applications based on neural networks integrate various processing requirements, leading to the use of dedicated hardware functions to optimize energy efficiency. Heterogeneous system-on-chips (SoC) bring together different computing capabilities, such as the Nvidia Jetson AGX Orin. This type of SoC includes a CPU for general-purpose processing, a GPU for intensive data parallelism, and a Deep Learning Accelerator (DLA) dedicated to neural network processing**

Together, these three components enable new latency and energy consumption trade-offs for Deep-Learning-based applications. But finding the right configuration to reach the best energy efficiency is difficult and sometimes counterintuitive. To take this into account, this paper studies deep neural network design and inference options for each accelerator. Altogether, the study forms guidelines to specifically make the best use of the computing and energy-efficiency capabilities published by manufacturers with the default TensorRT mapping.

***Index Terms*—Heterogeneous computing, Convolution Neural Networks, Low-power inference, DLA, Computer vision, Jetson AGX Orin**

I. Introduction

Deep Neural Networks (DNNs) applications are constantly increasing their complexity and integrating new specific layers for better application performance. While GPUs are widely used as hardware computing platforms for the inference of DNNs, they lack the energy efficiency needed to be deployed in embedded systems. Even if GPUs nicely combine high-performance and programmability, their designs are not dedicated to AI computing and the resulting energy efficiency would need to be improved. In particular, some high-performance AI embedded applications require a very high throughput (hundreds of Tera Operations per Second - TOP/s) within a constrained power budget, typically below 50 W.

For this reason, we decided to study a heterogeneous System-on-Chip (SoC) composed of various computing capabilities in order to respond to the different application needs and reach better energy efficiency. Among new AI SoC platforms, the Nvidia Jetson AGX Orin can deliver up to 270 TOP/s within 50 W and meet the needs of most AI-embedded systems. This heterogeneous architecture comprises a CPU for general-purpose processing, a GP-GPU for intensive data parallelism, and two last-generation fixed-function Deep Learning Accelerators (DLA). The DLA is an AI accelerator dedicated to the processing of neural networks. Thus, it is particularly energy efficient for the execution of DNNs, but offers less programmability and does not support as many machine-learning operations as the GPU.

Together, the CPU, the GPU, and the DLA offer new interesting latency and energy consumption trade-offs for the inference of DNNs. This paper identifies these potential trade-offs for different CNN inferences on Jetson AGX Orin depending on the used accelerator, showing that optimizing energy efficiency on such a platform is complex and sometimes counterintuitive.

The contributions of this paper are (1) a benchmark of various CNNs inferences on the Jetson AGX Orin with a focus on the energy consumption and latency performances, and (2) findings on neural network design and inference options to maximize the latency or the energy for inference on a given accelerator with the default TensorRT mapping.

The paper is organized as follows. Section II provides an overview of the leading AI accelerators operating with power consumption below 60W and describes the Jetson AGX Orin architecture. Section III introduces each step of the general neural network deployment workflow, from a high-level language description to the inference stage on accelerators. The measurements and benchmarking methodology is defined in section IV. The results from different tests are detailed in section V. Lastly, a conclusion in section VI summarizes the contributions.

II. AI Embedded accelerators

A. Related AI hardware accelerators

An AI embedded accelerator is an architecture designed to accelerate artificial intelligence and machine learning applications, including DNNs [1], considering SWaP (Size, Weight, and Power) constraints. Since early computing systems, application-specific hardware accelerators have been deployed to complement general-purpose CPUs to perform specialized tasks more efficiently. Notable hardware accelerators include, for instance, Graphics Processing Units or video decompression accelerators. As deep learning and artificial intelligence workloads began to have more impact on global performances, specialized hardware units were developed or adapted from existing products to accelerate them.

2771-2508/23/$31.00 ©2023 IEEE
DOI 10.1109/DSD60849.2023.00015

In the 1990s, Digital Signal Processors (DSPs) [2] and Field-Programmable Gate Arrays (FPGAs) [3] were used as neural network accelerators for inference, and Qualcomm began introducing AI accelerators in smartphones.

CPUs are processors more adapted to thread-level parallelism but can be used for the inference of neural networks. Because of their genericity and memory hierarchy, they cannot reach the best energy efficiency, but they remain a very programmable solution. Some CPU solutions multiply the number of available cores and can even integrate a specific ISA extension or data parallel processing units in order to increase performance. The Kalray MPPA many-core architecture [4] can be used for the inference of DNNs through OpenCL programming, even if the best performances are reached when using their dedicated accelerator.

Embedded GPUs are General-Purpose Graphics Processing Units (GP-GPUs) adapted to the embedded world with lower energy consumption. They are specialized hardware for handling image processing and massive data parallel tasks, such as matrix multiplication. Since their introduction in the 2010s, they have become increasingly used for machine learning and the inference of neural networks. GP-GPUs continue evolving to better execute DNNs by integrating, for instance, more computing units such as Tensor Cores, and low-precision operators. The Jetson Orin Nano is currently the most energy-efficient available Nvidia's embedded GPU [5].

FPGAs allow fine-grained spatial parallelism and the implementation of dedicated operators. Many computing architectures can be implemented on a FPGA [6], such as 1D or 2D arrays [7], SIMD processors [8], or systolic arrays [9]. Direct mapping of CNN topologies into FPGAs is also possible through dedicated frameworks such as fpgaConvNets [10]. Depending on the implemented architecture, it will remain a trade-off between programmability, scaling to complex DNNs, operators' accuracy, and frequency performance. FPGAs can reach high performances in a reduced power budget (inferior to 30W), leading to better energy efficiency than GP-GPUs.

Neural Processing Units (NPUs) are IP accelerators for AI that can be found in a hardware component (e.g., Intel Movidius VPU) or an IP integrated into a SoC (e.g. Apple Neural Engine). While GPUs and FPGAs perform better than CPUs for AI processing, dedicated hardware functions can be 10x more energy efficient [11]. Thanks to the integration of specialized operators, it offers the lowest possible latency and energy consumption for the inference of a given model. Their only drawback remains in their programmability and the few DNN models they can support.

Heterogeneous SoCs composed of a CPU, a GPU, and a NPU are then interesting to find a better trade-off between performance/programmability/models support and maximizing energy efficiency. Nvidia's Jetson modules family offers many GPUs combined with more or fewer accelerators to cover low-powered application use cases. The Jetson AGX Orin module is described in the next section.

B. Nvidia's Jetson AGX Orin

Before the Orin series, Jetson Xavier modules were the first generation of embedded Nvidia boards to work with both GPU and other AI accelerators. All planned Jetson Orin modules are now equipped with the latest generations of Nvidia's Ampere GPU architecture and AI accelerators. The Jetson Orin has two versions: with 64 and 32 GB of LPDDR5 RAM at a bandwidth of 204 GB/s. The 64 GB variant is the one referred to in the rest of this paper.

The Jetson Orin comprises a 12-core A78 Arm Arch v8.2 CPU at a maximum frequency of 2.2 GHz. It also integrates one Ampere GP-GPU organized in two Graphic Processing Clusters (GPCs), each with 16 Streaming Multiprocessors (SMs). There are 128 CUDA cores and 4 Tensor cores per SM, for a total of 2048 CUDA cores and 64 Tensor cores with up to 170 Sparse TOP/s of INT8 Tensor compute, and up to 5.3 FP32 TFLOP/s of CUDA compute. The CUDA cores allow Multiply-And-Accumulate (MAC) operations for general parallel computing tasks and the Tensor cores single precision MAC operations that can concurrently run with the CUDA cores for accelerating Deep Learning workload. The GPU can be programmed through the CUDA library or directly with CUDA programming language [12].

Moreover, the Jetson Orin integrates two Deep Learning Accelerators (DLAs), which are fixed-function accelerators optimized for deep learning operations. The Jetson board's 2 DLAs are configurable SIMD (Single Instruction, Multiple Data) fixed-point functions capable of 2.5 more power-efficiency (TOPs/Watt) than the GPU despite a lower compute capability (105 INT8 Dense TOP/s) [13]. Each DLA 2.0 brings six computing blocks to support specific operations: convolution, single data point (activation), planar data (pooling), multi-plane (normalization), and data memory and reshape operations. Compared to the GPU, the DLA only supports INT8 and FP16 operations for inference [14].

The Orin consumes between 15 and 60 W and supports different operating modes. As presented in Table I, it is possible to adapt the computing power to the maximum energy constraint of a system. Depending on the activated mode, the user can vary the number of cores of the CPU and GPU, or the maximum frequency. This freedom allows more trade-offs to optimize the energy efficiency for a given application but makes it more complex to find the right configuration.

TABLE I: Power modes of Jetson AGX Orin

Modes		**15W**	**30W**	**50W**	**MAXN***
cores	CPU	4	8	12	12
	GPU	3	4	8	8
	DLA	2	2	2	2
Maximum frequency (MHz)	CPU	1113.6	1728	1497.6	2201.6
	GPU	420.75	624.75	828.75	1301
	DLA	614.4	1369.6	1369.6	1600

* Default mode

III. Neural Networks deployment workflow

The following section describes the tools and strategies used to infer CNNs on the Jetson AGX Orin. For Nvidia products, TensorRT framework ensures this deployment to the final hardware target.

A. TensorRT principles

TensorRT is a Deep Learning inference optimizer and runtime framework for deep learning applications on all Nvidia products [15], [16]. To deploy a CNN on the Jetson AGX Orin, the network goes through successive high-level to low-level representations with the tools depicted in Figure 1. The TensorRT tool is involved at the end of the inference preparation process. A binary engine is built for the inference invocation with the TensorRT Runtime.

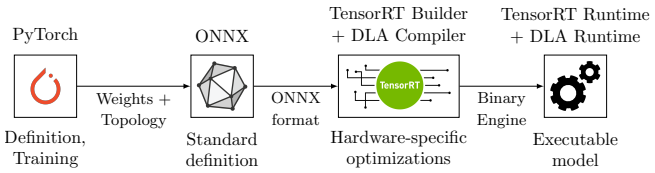

Fig. 1: Inference workflow steps to the Jetson AGX Orin.

During the building phase, TensorRT performs multiple automatic optimizations depending on the final hardware targets (GPU or DLA). Table II summarizes the available optimizations specific to the GPU and DLA with TensorRT. These optimizations lead to lower inference latency and energy consumption, more throughput, less memory storage, and fewer data movements [17]. By default, both the CUDA and Tensor cores of the GPU are used for the inference. If the DLA is enabled, TensorRT calls the DLA Compiler Library, which provides a DLA *loadable* that will be encapsulated in the inference engine. In this case, TensorRT makes all DLA-supported layers run on one DLA and the remaining unsupported layers fall back on the GPU. Therefore, and because TensorRT plans executions on only one processing element at a time, TensorRT builder produces an optimized inference engine running either: (1) entirely on the DLA (fully DLA-compatible), (2) sequentially on one DLA and the GPU (partially DLA-compatible), or (3) on the GPU alone (no DLA-compatibility or DLA disabled).

DLA 2.0 integrates new capabilities such as softmax activation, depth-wise convolution, and a hardware scheduler. The following CNN layers are supported:

- Convolutions (basic, dilated, depth-wise, deconvolution),
- Fully connected layers,
- Pooling layers (max, average),
- Activation layers (ReLUs, Tanh, Sigmoïd, Softmax),
- Scale (Batchnorm) and resize layers,
- And other layers such as concatenation, mult, add, sub, max, shuffle.

This allows the execution of full CNNs as long as it remains compatible with supported layers and configurations.

B. Existing strategies to leverage TensorRT

Compared to basic inferences with Pytorch, TensorRT optimizations bring a significant gain in terms of throughput, latency, and energy consumption [15]. But optimizing an application over a very heterogeneous hardware architecture remains challenging for an automatic workflow. Based on previous Jetson architectures, the literature shows that custom strategies and additional optimizations can better meet application constraints. The proposed solutions are manual or algorithmic, work on only one or several CNN models, and imply sequential or parallel executions.

For instance, in [18], the authors manually optimize CNN inferences on a Jetson Nano's GPU and CPU. Scenarios on four hardware settings are analyzed: the number of CPU cores, GPU and CPU frequencies, and synchronization modes (parallel or sequential inferences). From the results, they summarize the parameters' optimal settings to reach the lowest energy consumption, power consumption, or latency. This manual approach gives insight into each parameter's influence on the criteria. However, due to the lack of automation, they remain difficult to scale to other CNNs or hardware platforms.

In addition, the authors of [19] implement an algorithm-based framework to maximize the inference throughput of a unique neural network on Jetson AGX Xavier's GPU and DLAs. Their JEDI TensorRT-based framework relies on applying diverse parallelization techniques (multi-threading, pipelining, buffer assignment, and network duplication on the DLAs) to find a low-latency allocation of subnetwork parts to the different processing elements. Through global and local search heuristics, pipelined cutting points inside the model are searched for a given hardware mapping. Then, parallelization strategies parameters are fined-tuned on the best-found mapping. Depending on the neural network models, some of found DLA-based solutions were more energy-intensive and slower than the CPU-GPU-based solutions but still more interesting than the basic TensorRT GPU implementation ones.

Kim and Ha [20] developed an energy-aware algorithm-based optimization methodology to map and schedule different Deep Leaning applications on Jetson AGX Xavier's CPU, GPU, and DLAs. They notably highlight the importance of preserving specific layer sequences that could match layer-

TABLE II: HW-specific TensorRT optimizations [13][17]

TensorRT optimizations	**GPU**	**DLA 2.0**
Accuracy	fp32, fp16, int8 (implicit[a] or explicit[b])	fp16, int8 (implicit[a])
Layers and tensors fusion	Mathematical equivalents, Fused operations kernels	Fused operations
Operation instructions	Auto-tuning from kernel libraries	CuDLA Compiler Library
Memory accesses/reuse	Dynamic tensor memory	Contiguous subgraphs without DRAM access
Sparsity	Supported	Supported
Custom layers	Only with CUDA	Unsupported

[a] Model-wise rescaling [b] Layer-wise rescaling

fusion patterns offered by the GPU and the DLA. Their methodology employs a first genetic algorithm to identify the Pareto-optimal mapping of a given network through network pipelined cutting points and a second genetic algorithm to find a working scheduled mapping of all neural networks at an optimal frequency. The found mappings and scheduling led to a 31% reduction in energy consumption and a 40% margin for additional latency objectives.

In [21], Dagli et al. characterize execution and inter-layer transition times between the Jetson Xavier AGX's GPU and DLA. This fined-grained empirical modeling allows the authors to solve the GPU-DLA layer-mapping problem as a linear programming optimization constraint. Best mappings under energy constraints use both the GPU and the DLA.

Finally, Bouzidi et al. propose Map-and-Conquer [22], an execution scheme for mapping dynamic multi-exit networks among the CPU, the GPU, and the DLA of the Jetson AGX Xavier. The mapping occurs at the layers' channels level for each CNN or Transformer to leverage the underlying processing concurrency. Through an evolutionary algorithm, the author obtained DNNs with high accuracy and reduced latency and energy consumption by using all available accelerators.

In this paper, we only consider the default TensorRT mapping as it is available to all users, but we are aware that optimization-based method could provide better mapping.

C. Current limitations of the DLA

Since introducing of the DLA in Nvidia SoCs, several limitations have made it difficult to leverage all its computing potential. The Jetson AGX Orin's DLA 2.0 brings many new interesting features compared to the previous version, but again it comes with some restrictions [17].

First, Nvidia does not allow an easy and efficient way to communicate with the DLA. Contrary to communications between the CPU and the GPU that use a physically unified memory, communications with the DLAs are performed through explicit TensorRT data transfers [19]. This adds a consequential execution overhead that directly impacts global performances. With DLA 2.0, this effect has been reduced by the increase of its internal SRAM memory. It allows more data fusion into one subgraph and then entire CNN subgraphs can be accelerated by staying in the SRAM without accessing the DRAM.

Besides, each DLA has only 1 MB of dedicated SRAM used as cache memory, which is smaller than all combined caches of GPU cores. If this SRAM is unavailable, a DLA can still run by falling back to the local DRAM, but it will be slower. Therefore, sub-networks larger than 1 MB cannot be executed in one run with the SRAM and benefit from its high-speed processing. DNNs with numerous layers or cumbersome operations are not well suited for the DLA design.

In addition, the supported layers come with a rather complex parameter acceptance range. For example, a convolution operation is valid for a limited range of kernel sizes, padding sizes, and group numbers, and the whole combination should respect the buffer limitations. Consequently, computationally-heavy convolutions, convolutions on enormous images, or convolutions with padding higher than the kernel are not compatible with the DLA compute units. Other incompatibilities could come from a batch size higher than 4096 or some layer combinations causing a DLA internal state overhead. In this case, TensorRT makes the layer fall back to the GPU.

Moreover, the Orin DLA can significantly increase latency when using FP16 convolution operations compared to the 1.0 version. DLA 2.0 has been much more optimized for INT8 operations, and using FP16 can increase the DLA loadable size. Graph optimization may unintentionally trigger this behavior by changing the type of a layer.

Therefore, to improve the deployment of a CNN on the DLA, the model should avoid computation-intensive operations, its layers should respect the restrictions imposed by Nvidia, its weights should be preferably represented in INT8, and continuous sequences of compatible layers should be prioritized to reduce expensive back and forths communications between the GPU and the DLA.

According to Nvidia, typical models supported on the DLA are backbones (classification CNN) such as MobileNet-EdgeTPU, EfficientNet-Lite, EfficientNet-EdgeTPU, or Inception, ResNets, and YOLO (object detection) family models. Finally, Nvidia suggests that models with high arithmetic intensity are best suited to maximize the DLA's resource utilization. Therefore, a CNN topology may somehow impact the energy-efficiency of the computing platform.

IV. Methodology

In order to identify what are the best configurations of the Jetson AGX Orin leading to optimized energy efficiency for a given application with TensorRT default mapping, many experiments have been carried out to characterize and verify claims and behaviors of the DLA and the GPU. Multiple parameters are evaluated, such as the power modes, the CPU/GPU/DLA frequencies, the batch and image sizes, and the precision format (INT8, FP16, FP32). We decided to focus on two main criteria since we targeted embedded SoC: the latency and the average energy consumption during the inference. The Jetson AGX Orin 64 GB development kit is used in our experimentations.

A. Application benchmarking

TABLE III: Benchmarked CNNs characteristics

Name	Task	Input size	GMAC	# Params (M)	# Conv layers
MobileNetV2	Classification	3x224x224	0.39	3.4	52
MobileNet -SSD	Object detection	3x300x300	1.3	86.8	35
Inception	Classification	3x32x32	1.5	6.2	64
ResNet50	Classificaiton	3x224x224	4.2	25.5	48
UNet	Semantic segmentation	3x128x128	10.6	7.8	15
ResNet34 -SSD	Object detection	3x1200x1200	216.8	20.1	51
DeepLabV3	Semantic segmentation	3x512x512	241.8	58.5	112

In order to carry out a representative study of typical AI applications, we considered different well-known CNNs for classification, object detection, and semantic segmentation. The first set of CNN applications selected for our study gathers 3 DLA-compatible CNNs implemented on the DLA by NVidia: ResNet50 for classification, ResNet34-SSD, and MobileNet-SSD for object detection. The second set puts together 3 DLA-compatible CNNs that we manually modified to be supported by the DLA: UNet for semantic segmentation, MobileNet-V2, and Inception for classification. The last group is only composed of 1 model incompatible with the DLA due to unsupported Atrous Spatial Pyramid Pooling (ASPP) layers. The DeepLabV3 model for semantic segmentation belongs to this group. This makes 7 different CNN applications for our study, as described in Table III.

B. CNN adaptations for the DLA

The deployment of a CNN with TensorRT often requires extra modifications of its topology to ensure compatibility all over the workflow described in Figure 1. For the DLA, frequent sources of unsupported layers may come at three levels: either during the Pytorch definition step, the ONNX transcription phase, or during the inference engine creation with TensorRT.

At the Pytorch network definition step, obvious DLA-unsupported layers of a model can be replaced with algorithmic-equivalent functions. Thus, a flatten operation followed by a fully-connected layer can be fused into a 1x1 convolution without bias. Besides, default available parameters options or layer functions evolve with versions to offer optimized state-of-the-art last findings to the user. The DLA-incompatibilities can occur here from implicit layer definitions and options or dynamic shape dependencies.

These implicit mechanisms are unveiled with the ONNX transcription step. Possible solutions are redefining the Pytorch model or modifying the ONNX representation with ONNX's Graph Surgeon and TensorRT's Polygraphy tools. For example, in MobileNetV2 definition provided in the Torchvision library, the original Average-Pooling layer was replaced with Adaptative-Average-Pooling, which the DLA does not support.

Finally, when using TensorRT builder to create the engine, the building reports highlight potential memory overheads from big image sizes and input/output DLA impossible configurations options.

C. Energy measurement methodology

The latency can be easily retrieved with good precision from the performance summary report printed by TensorRT at the end of each inference experiment. The energy consumption, however, is not directly provided.

Several software-based power consumption modelings are described in the literature. In [23], authors implement a CPU-GPU energy consumption model based solely on MAC counts. The authors of [18] and [24] use the *tegrastats* utility, or a derived version, and estimate the average energy consumption on multiple end-to-end executions. Finally, the authors of [25] read CPU and GPU instant powers directly from the power monitor unit through the sysfs filesystem and can then correlate the energy consumption to a specific layer with timestamps markers. Then, the power monitor filesystem is a faster solution than the *tegrastats/Jtop* tools and can reflect the general power activity despite not being as exact as employing board-level meters (e.g., 1 W difference measured in [26]).

We suggest to update this last existing work by considering the DLA energy consumption during several inferences. In this paper, we estimate the mean power consumption with the instant voltage and current provided through the reading of the following I2C power rails [27] :

- VDD_GPU_SOC: for the GPU and the SOC cores,
- VDD_CPU_CV: for the CPU, the DLA, and the PVA,
- VIN_SYS_5V0: for the system 5V rail.

We define the mean power as the sum of the instant powers of the three previous power rails. A comparison with the *tegrastat* tool's estimated power shows similar estimated values for the two methods. Finally, one should note that the power consumptions of the CPU, the GPU, and the DLA cannot be retrieved separately.

The main measurement workflow is the following. Each neural network is first defined with Pytorch 1.13.1, then its weights and topology are standardized with ONNX 1.13.1. If the network has DLA-incompatible elements, they are replaced with equivalent allowed ones. The inference engines are created on the embedded target under TensorRT 8.4.0.1 through the *trtexec* command line, with a memory pooling size set to 1 MB for the DLA, and I/O format sets to INT8/FP16 when using the DLA (to prevent reformatting overhead). During runtime, latency and energy measurements are performed on 500 consecutive inferences after a warmup of 10 seconds to stabilize the performances.

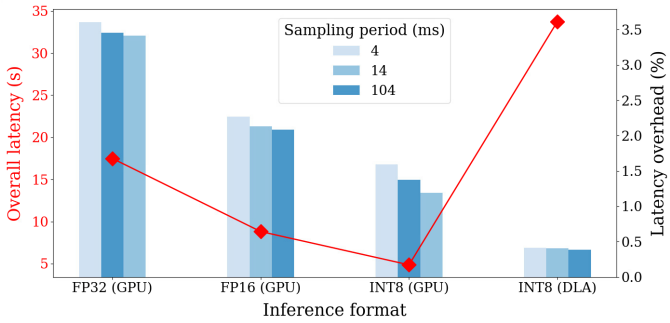

Fig. 2: Overall execution latencies and corresponding overheads on 500 inferences of DeepLabV3 when varying the sampling period (for MAXN power mode).

We implemented a bash script to collect the power monitor's values directly at I2C drivers' addresses. Our new method can retrieve the needed modules' internal power rails (GPU and logic rails, CPU and DLA, Data handling, Input and Output ports) at a maximum sampling rate of 4 ms, closer to some inference duration times compared to the 13 ms of *tegrastats*. Despite being less precise than measuring the board's external

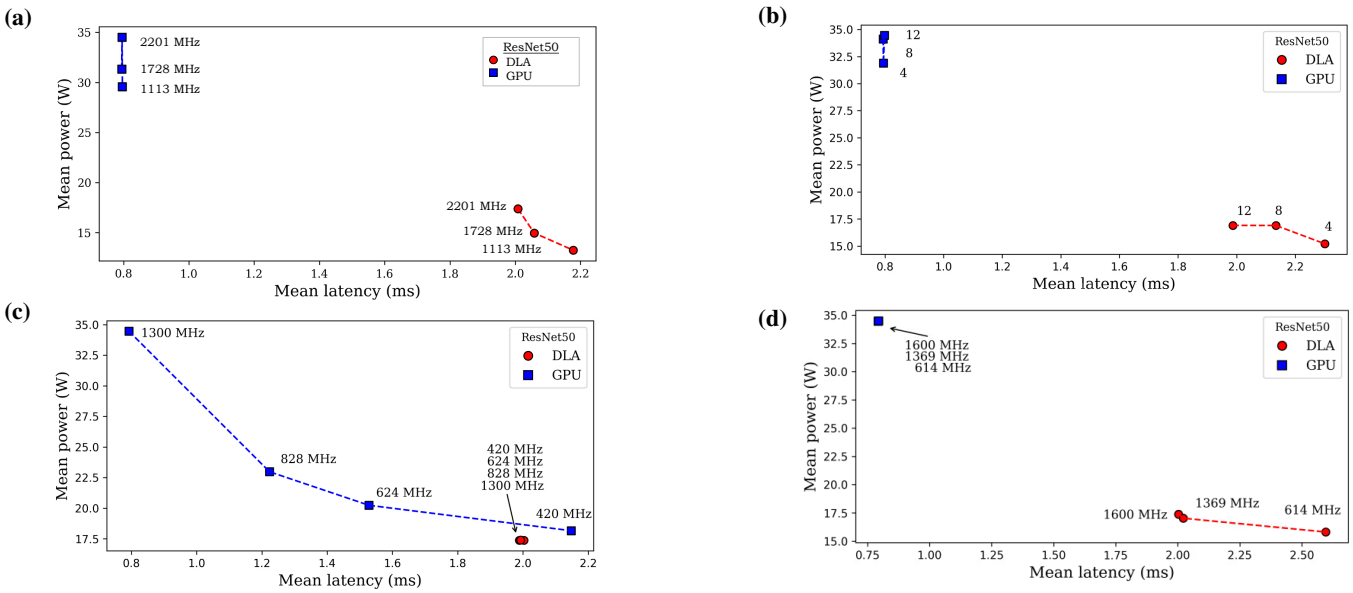

Fig. 3: Influence of the the CPU frequency (a), the CPU cores number (b), the GPU frequency (c) and the DLA frequency (d) on the power consumption and the mean latency for 500 ResNet-50 INT8 inferences (all other parameters are set to their MAXN mode configuration).

power supply value, the method still gives an insight into general power activity.

To quantify the impact of these reading accesses on the latency, inferences have been run with and without energy consumption measurements for many configurations, including different CNN applications, power modes, precision, accelerators (GPU/DLA), and sampling periods. As reported in Figure 2 as an example, the maximal latency overhead we could obtain remains beyond 4 ms compared to the same inference without energy measurements. It is acceptable for our experiments, and the minimum sampling rate of 4 ms will be kept in all our experimentations.

V. Experiments and Analysis

The next section compares the impact of accelerators' settings and CNN topologies on the energy consumption and latency of inferences. The observations are valid for all the benchmarked CNNs.

A. Impact of the CPU cores number and frequency

This subsection studies the impact of CPU frequencies and cores number on the energy consumption and the latency of inferences. In addition to the multiple available power modes, Nvidia allows further configuration of each computing unit with the *nvpmodel* configuration file to explore other settings, thus enabling the evaluation of one parameter at a time. All the fixed parameters are based on the MAXN power mode values and only INT8 accuracy is considered.

With the DLA, the CPU makes some data parallel processing before and/or after exchanging data. Thus, increasing the number of CPU cores or their frequency improves the inference time on the DLA, contrary to the GPU. This behavior is illustrated in Figures 3a and 3b for the ResNet50 model. It can be noticed that the DLA latency slightly decreases with the CPU frequency and the number of cores. The mean power logically increases when rising the CPU cores number or frequency.

Finding 1: Increasing the frequency of the CPU or its number of cores generally improves the DLA inference latency, whereas it has no impact on the GPU one.

B. Impact of GPU and DLA frequencies

This subsection evaluates the influence of the GPU and the DLA frequencies over the inference performances. Again, we will consider an INT8 accuracy and all other settings to their MAXN power mode values.

Increasing the frequency of the two accelerators separately should decrease the latency and raises the general energy consumption. Also, increasing the frequency of one accelerator should not disrupt the inferences performed on the other one. These expectations are verified for all the benchmark models. It is what Figures 3c and 3d show for the ResNet50, for instance. However, we can notice that the GPU inference energy consumption is more impacted by the GPU frequency changes, than it is for the DLA (approximately 45 % vs 10 %). In addition, above a certain DLA frequency (1,369 MHz for the ResNet50), the performance of some CNNs is not improved despite an increase in frequency, possibly explained by a lack of memory bandwidth.

Finding 2: Contrary to the DLA, reducing the GPU frequency has an important impact on the GPU inference energy consumption and latency.

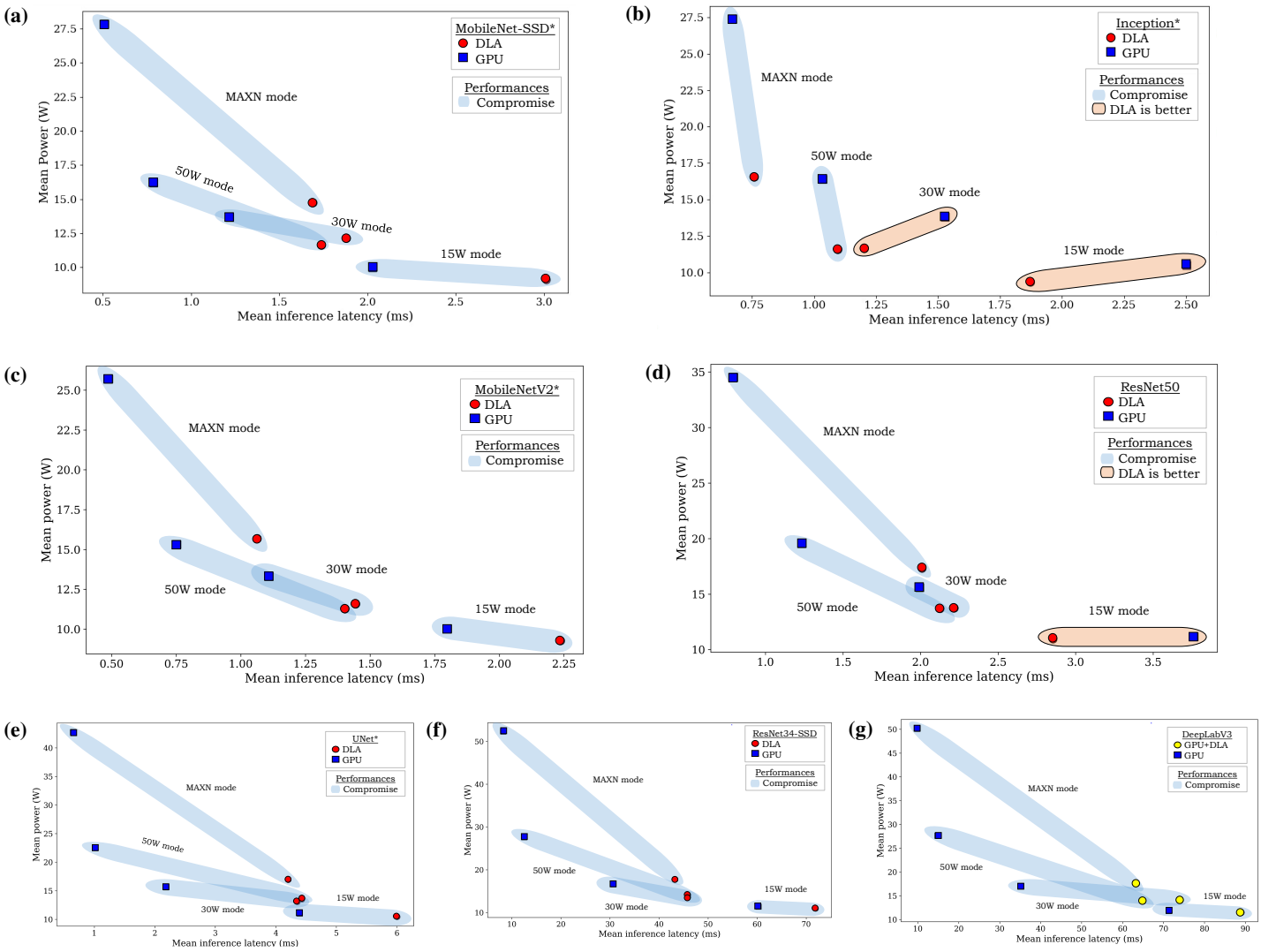

Fig. 4: Study of the different power modes on the latency and the energy consumption of 500 INT8 inferences on MobileNet-SSD (a), Inception (b), MobileNetV2 (c), ResNet50 (d), UNet (e), ResNet34-SSD (f), and DeepLabV3 (f) DNNs.

C. Impact of models' topology

The selected DNNs have different architectures that mainly differ in the number of layers, the type of operations (presence of application-specific layers), and the computation complexity. The present section analyzes their impact on the inference latency and energy. To this end, the 7 previously adapted CNN models of section IV.A are deployed for different precision formats and all power modes configurations on the GPU and, when possible, on the DLA. For clarity, the FP32 and FP16 formats results on the GPU are not studied since they always lead to less energy-efficient inferences than INT8 on GPU.

1) DLA-incompatible CNN

Inferences of partially DLA-incompatible CNNs on the DLA are possible if they rely on both the GPU and the DLA. This kind of setup may be actually problematic as the TensorRT framework may not be able to optimize inference as much as for a single accelerator. In the end, extra communications and missed TensorRT optimizations may lead to longer and energy-intensive inferences compared to the ones on the GPU alone. This case is partially observed with DeepLabV3 model for example.

The DeepLabV3 CNN is a particularly incompatible model for the DLA because of the location of its DLA-unsupported layers. When deployed on the DLA-GPU, these incompatibilities create different subgraphs, each made of a continuous sequence of DLA-supported operations, generating six extra data movements between the GPU and the DLA. With the lowest power mode and precision, the GPU-DLA inference of DeepLabV3 brings little gains in power consumption (-3%) but is significantly longer (+28%). Here, the GPU solution is more interesting for all criteria.

However, voluntarily adding incompatible layers to an initially DLA-compatible network shows different results from DeepLabV3 ones. For example, adding an incompatible option to the 3 upsample layers of the UNet model leads to a power reduction on the inference compared to the GPU

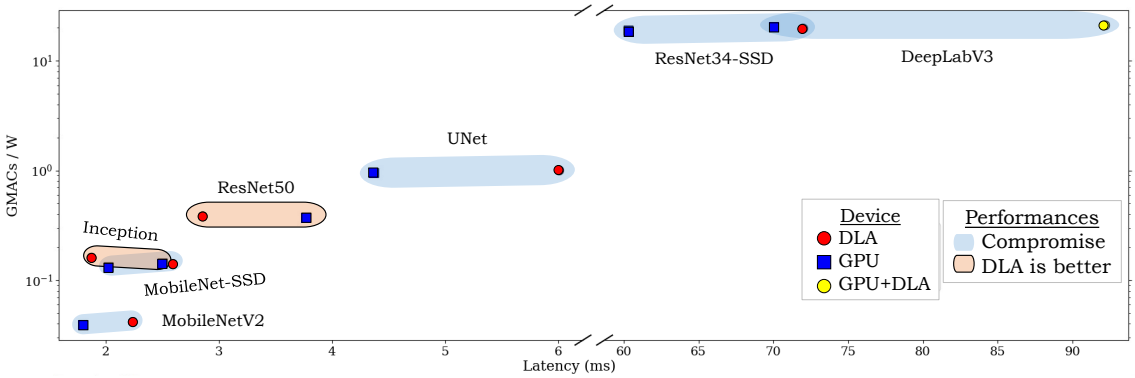

Fig. 5: Benchmark analysis of all considered DNNs in terms of latency and energy efficiency (15W mode and INT8 accuracy)

(-60%) despite a higher latency (+300%). Moreover, when compared to the DLA solution, the GPU-DLA inference is a better choice since it is much faster for equivalent energy consumption (-28% latency, +1% power). So, in this case, adding incompatibilities brings benefits.

Finding 3: Depending on the topology and the nature of incompatibilities, GPU-DLA inferences CNNs may be an interesting compromise to GPU-only inferences.

2) DLA-compatible CNN

When using a DLA-compatible neural network, the DLA is expected to offer better energy efficiency when the GPU reaches the best latency. Indeed, the DLA is a dedicated hardware for the inference of neural networks. On the other hand, the GPU may allow very fast inferences thanks to its numerous computing cores, but at the cost of more energy consumed.

Such a trend is indeed observed and is illustrated with MobileNet-SDD inferences shown in Figure 4a. For example, with the 15W power mode, if there is a need to optimize latency first, the inference on the GPU should be selected. Otherwise, if the energy efficiency must be optimized, the inference on the DLA is the best choice.

Still, some exceptions occur for which only one accelerator configuration is better. Sometimes, either the DLA or the GPU can provide both the fastest and most energy-efficient inference. This is the case for the manually modified Inception model, in Figure 4b. Here, the inference on the DLA for the 30W power mode is both faster and more energy-efficient than the GPU inference. Then, there is no reason to use the GPU in this configuration for topologies with a low-memory need.

Finding 4: Whatever the power mode, inferring a CNN model fully on the GPU is not necessarily much faster than inferring on the DLA.

Changing the power mode is also expected to control the power and the latency ranges. Choosing a low power mode should reduce the energy and increase the latency compared to a higher power mode. Indeed, as shown with MobileNet-SSD in Figure 4a, lowering the power mode from MAXN to 15W makes the GPU inferences longer and less energy-intensive. Yet, in some cases lowering the power mode may bring irregularities for the DLA inferences. For example, for MobileNet-SSD (Figure 4a), the 30W mode DLA inference is not more energy-efficient than the 50W one. The 50W mode DLA inference is a preferable solution to the 30W inference because it uses less power and has less latency These irregularities partly come from the power modes characteristics of the CPU, where the 30 W and 50 W configurations make the number of cores and the frequency vary oppositely. No clear link could be established between these irregularities and the CNN architecture characteristics.

Finding 5: Lowering the power mode configuration of the DLA may not decrease the energy consumption. Notably, 50W mode can be preferred to 30W to save energy and time.

Finally, for all the models, the DLA and GPU energy disparities are narrowed for the lowest power mode (15 W). In the case of the MobileNet-SSD (Figure 4a), the DLA inference at 15 W brings little power saving (-10 %) but a much longer latency (+50 %) compared to the GPU. Then choosing an inference on the GPU here gives more improvements for the two criteria compared to the DLA inference.

Finding 6: At the lowest power mode (15W), a low-complex CNN in INT8 using a GPU can be nearly as energy-efficient as the DLA.

3) Global analysis of the benchmark

To better summarize the inferences trends of the benchmark models, Figure 5 compares all the CNNs on their latency and energy efficiency at the lowest power mode (15W) and format precision (INT8). Blue-colored zones indicate existing power consumption and energy trade-offs when the inference of a model is more energy-efficient on the DLA and faster on the GPU. Almost all CNNs of the benchmark show such a compromise for this configuration, except for the ResNet50 and the Inception models for which inferences on the DLA are more performant for both criteria.

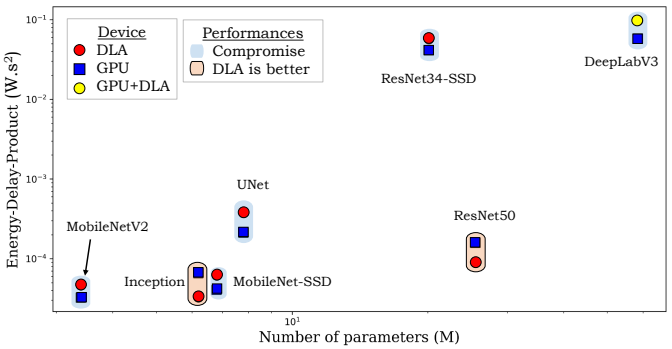

Fig. 6: Benchmark analysis of all considered DNNs in terms of memory and EDP (15W mode and INT8 accuracy)

These embedded performances on the benchmark can be verifies with another metric such as the Energy-Delay-Product (EDP). The EDP also reflects energy efficiency but combines both the power consumption and the application latency to consider trading increased delay for lower energy/operation. As observed in Figure 5, Figure 6 shows that the inferences on the DLA are more energy-efficient, in a EDP meaning, than on the GPU for the Inception and ResNet50 models.

Here, the GPU and the DLA inferences have energy consumptions of the same order of magnitude, but it is less verified when increasing the image size (e.g, 3x500x500). Energy consumption differences are effectively more pronounced for bigger images. No clear link could be established between the topology of a CNN and its energy consumption, but a correlation may exist between a model's complexity and latency.

VI. Conclusion

In this paper, we explored the energy consumption and latency trade-offs offered by the Jetson AGX Orin's accelerators during the inferences of a CNN. This heterogeneous SoC is designed with a GP-GPU specialized in massive data parallel tasks, as well as a second accelerator, the Deep Learning Accelerator (DLA) chip dedicated to energy-efficient neural network processing. Based on these characteristics, one could think there may exist a general latency and energy-consumption trade-off between the inferences executed exclusively either on the GPU or on one DLA.

To verify this idea, a benchmark of CNN models from various applications was used to inspect the inferences performances. Then, using TensorRT framework, neural networks inferences were performed for different configuration scenarios on format precisions, frequencies, number of cores, input sizes, power modes, and neural network topologies. The results showed that configurations changes lead to different responses from the DLA and the GPU by order of magnitude. Inferences on different network topologies do not always bring the initially expected latency-energy trade-off. Because of extra communications overhead and specific TensorRT optimizations related to the topology, the inferences on the DLA may be faster and more energy-efficient, and the GPU may align with the DLA regarding energy efficiency.

Our work shows that energy optimization on the Jetson AGX Orin appears to be complex and sometimes counter-intuitive when using the default TensorRT mapping. Energy consumption impact must therefore be taken into account when designing a neural network for the DLA through design space exploration.

References

[1] A. Reuther et al., "AI accelerator survey and trends," in *HPEC*, 2021.
[2] Y. Lecun et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.
[3] M. Gschwind et al., "Space efficient neural net implementation," in *Proc. of the Second International ACM/SIGDA Workshop on Field Programmable Gate Arrays.*, 1994.
[4] B. de Dinechin et al., "A clustered manycore processor architecture for embedded and accelerated applications," in *HPEC*, 2013.
[5] Nvidia, "Jetson nano," URL: https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-nano.
[6] M. Lebedev et al., "A survey of open-source tools for fpga-based inference of artificial neural networks," in *IVMEM*, 2021.
[7] A. Carbon et al., "Pneuro: A scalable energy-efficient programmable hardware accelerator for neural networks," in *DATE*, 2018.
[8] Y. Ma et al., "Optimizing the convolution operation to accelerate deep neural networks on fpga," *VLSI Systems*, 2018.
[9] J. Zhang et al., "Frequency improvement of systolic array-based cnns on fpgas," in *ISCAS*, 2019.
[10] S. Venieris et al., "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *FCCM*, 2016.
[11] N. Jouppi et al., "Motivation for and evaluation of the first tensor processing unit," *Micro*, 2018.
[12] L. Karumbunathan, "NVIDIA Jetson AGX Orin Series Technical Brief v1.2," Nvidia, Tech. Rep., 2022.
[13] R. Cherukuri, "Leveraging deep learning accelerators on nvidia agx platforms," 2022, nvidia GTC Digital Spring. [Online]. Available: https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41670/
[14] "Jetson orin modules and developer kit," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin//\#orion-prod-module-dev-kit-specs, accessed: 2023-03-15.
[15] Nvidia, "Tensorrt," URL: https://developer.nvidia.com/tensorrt.
[16] O. Shafi et al., "Demystifying tensorrt: Characterizing neural network inference engine on nvidia edge devices," in *IISWC*, 2021.
[17] Nvidia, "Tensorrt 8.5 developer guide," https://docs.nvidia.com/deeplearning/tensorrt/developer-guide, 2023.
[18] S. Holly et al., "Profiling energy consumption of deep neural networks on nvidia jetson nano," in *IGSC*, 2020.
[19] E. Jeong et al., "Tensorrt-based framework and optimization methodology for deep learning inference on jetson boards," *ACM Trans. Embed. Comput. Syst.*, 2022.
[20] J. Kim et al., "Energy-aware scenario-based mapping of deep learning applications onto heterogeneous processors under real-time constraints," *IEEE Trans. Comput.*, 2022.
[21] I. Dagli et al., "Axonn: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs," in *DAC*, 2022.
[22] H. Bouzidi et al., "Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous mpsocs," in *DAC*, 2023.
[23] C. Rodrigues et al., "Synergy: An energy measurement and prediction framework for convolutional neural networks on jetson tx1," in *PDPTA*, 2018.
[24] A. A. Süzen et al., "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *HORA*, 2020.
[25] C. Rodrigues et al., "Fine-grained energy profiling for deep convolutional neural networks on the jetson tx1," in *IISWC*, 2017.
[26] S. Holly et al., "Profiling energy consumption of deep neural networks on nvidia jetson nano," in *IGSCW*, 2020.
[27] Nvidia, "Jetson linux version 35.3.1 ga, developer guide, software-based power consumption modeling."