Evaluating the energy impact of device and workload parameters for DNN inference on edge

RPE Report

Anurag Dutt

Department of Computer Science
Stony Brook University

This report is submitted for the RPE exam requirement of
Doctor of Philosophy

November 2023

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this report are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This report is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text.

Anurag Dutt
November 2023

Abstract

DNN inference is crucially employed in applications ranging from real-time speech recognition and language translation to autonomous vehicle navigation and personalized content recommendation, enabling applications to make quick, data-driven decisions. Consequently, these models are increasingly being deployed on edge devices, facilitating ubiquitous computing and enabling closer interaction with end users. More often than not, edge devices are constrained in computation resources and often run in battery-operated environments, which makes sustainable deployment of deep learning models essential.

This work presents an evaluation of the energy impact of device and workload parameters for DNN inference on edge devices. Our work strives to study the impact of energy and power for inference on 6 different deep-learning workloads for two different Jetson Class edge devices. We identified various device and workload parameters that affect the energy consumption of DNN workload execution during inference and showed how tuning these parameters can lead to energy savings without compromising accuracy. This work provides valuable insights for researchers and practitioners working on sustainable edge deployments of DNN workloads.

The experiments were conducted on Jetson Nanos and Orins which are commonly used System-on-Chip edge devices with onboard GPUs for DNN inference. Both the devices offer a large number of operating configurations which complicates the problem of finding the optimal operating configuration. We ran our tests on device parameters such as CPU and GPU frequency and workload parameters such as batch size, number of layers, model initialization, and quantization. We also compared against Dynamic Voltage and Frequency Scaling (DVFS) governors, the default power management technique used in computing that adjusts the voltage and frequency of a processor's operations dynamically based on workload demand to conserve energy. Our results indicate that tuning the CPU and GPU frequency can lead to significant energy savings to the tune of 19% over DVFS. We also found that workload parameters such as batch size and model settings such as graph initialization can impact inference energy.

Table of contents

Chapter 1

Introduction

Deep learning inference on edge devices like the NVIDIA Jetson Nano and Xavier series represents a significant shift in the deployment of AI applications, moving from centralized, cloud-based computing to localized, edge-based processing. This transition is driven by the need for real-time processing, reduced latency, increased privacy, and lower bandwidth requirements, particularly critical in applications like autonomous vehicles, smart cameras, and IoT devices. Edge devices like the Jetson Nano and Xavier are specifically designed to handle the computational demands of deep learning models while operating within the power and space constraints typical of edge environments. They are equipped with powerful GPUs, capable CPUs, and specialized hardware accelerators to efficiently execute complex neural network computations.

The deployment process typically involves training the deep learning models on high-performance computing systems, and then optimizing and compressing these models for edge deployment. Techniques like quantization, pruning, and model distillation are employed to reduce the size of the model and the computational load, ensuring that the model can run efficiently on the edge device without significant loss in accuracy. Once deployed, these models can perform tasks like image and speech recognition, object detection, and various real-time analytics directly on the edge device. NVIDIA's ecosystem, including CUDA, cuDNN, and TensorRT, provides a comprehensive set of software tools that streamline the process of optimizing and deploying deep learning models on Jetson devices. This ecosystem, coupled with the powerful hardware of Jetson Nano and Xavier, enables developers to bring sophisticated AI capabilities to a wide range of edge applications, leveraging the benefits of localized, low-latency processing.

Over the past several years, the field of artificial intelligence has witnessed remarkable advancements, especially in the realm of Deep Neural Networks (DNNs). These advancements have led to the creation and widespread use of various DNN applications, with Large

Language Models (LLMs) being particularly notable for their complexity and capabilities. LLMs, like GPT-3 and its successors, represent a significant leap in machine learning.

1.1 Challenges with efficient deployment of DNN models

Despite the widespread acclaim and adoption of these models, much of the focus has been on their development and training. Training a DNN, particularly LLMs, involves feeding vast amounts of data into the network, allowing it to learn and make predictions or decisions based on that data. This process requires substantial computational resources and expertise, and as a result, has been the primary focus of many research and development teams in both academic and commercial settings.

However, an equally important but less discussed aspect of these AI models is the strategy for deploying them, specifically for running *inference*. Inference refers to the process of using a trained model to make predictions or decisions. Deploying these models effectively for inference poses unique challenges, as it requires the models to not only be accurate but also efficient and scalable.

Scalability is another critical factor in the deployment of AI models for inference. As the volume of data and the complexity of tasks increase, the models must be able to scale without a significant drop in performance or efficiency. This involves optimizing the models to handle varying workloads and adapting to different hardware architectures, from powerful cloud servers to compact edge devices. The trade-offs between model complexity, computational demands, and prediction latency need to be carefully considered to ensure that the deployed model meets the desired performance metrics.

Efficient deployment is crucial for real-world applications, where response time and resource utilization are critical factors as the ability to quickly and accurately process requests directly impacts the effectiveness and usability of the application. Research in this area is ongoing, aiming to address various aspects such as optimizing model architectures for faster inference, reducing the computational load, ensuring the models can operate effectively on different hardware platforms, and making them more energy-efficient. This research is vital for the practical application of DNNs, ensuring that these powerful tools can be used effectively in a wide range of settings, from cloud servers to mobile devices [18].

1.2 Challenges with efficient deployment of DNN models on edge

Edge computing has emerged as a transformative paradigm that shifts the processing of data closer to where it's generated, at the edge of the internet. This approach positions data and computational resources near end-users, enhancing the efficiency and responsiveness of applications by reducing the need to transmit large volumes of data to centralized data centers [68].

The limited availability of resources and energy in edge nodes necessitates a more in-depth examination of how energy-efficiently Deep Neural Network (DNN) inferences can be deployed in these settings. This is especially critical considering that edge computing often operates in environments with constrained power supply, such as those reliant on solar energy, making the optimization of energy use a key factor in the success and sustainability of edge-based applications [39].

Deploying Deep Neural Network (DNN) inference tasks at the edge of networks in an energy-efficient manner presents multiple challenges. This complexity arises from several factors that must be carefully considered to ensure effective implementation. First among these challenges is the increasing complexity of DNN models. As these models grow in sophistication, they require larger sets of parameters, which in turn demand more memory.

The issue of memory becomes significant in the context of edge computing, where devices and accelerators often have limited memory capacity. This limitation means that these devices struggle to accommodate the large models typical of advanced DNNs. Consequently, practitioners face the critical task of managing various *workload parameters*, such as batch size in a DNN model, when running inference tasks on edge nodes. Proper management of these parameters is essential to prevent issues such as Out-Of-Memory (OOM) errors, which can disrupt the inference process.

However, adjusting workload parameters is not a straightforward task. These parameters play a significant role in determining the latency of the DNN inference, which is crucial for the DNN inference. Workload parameters also have a direct impact on the energy consumption of the inference process. As a result, selecting the right set of parameters for deployment on edge nodes becomes a complex task. This complexity is due to the need to balance between maintaining high inference accuracy and ensuring low energy consumption, both of which are critical for the successful deployment of DNNs in energy-constrained edge environments.

The energy consumption of Deep Neural Network (DNN) inference on edge devices is significantly influenced by various hardware parameters, such as the configuration of CPUs

and GPUs and the type of device being used. These hardware parameters are crucial because they determine how efficiently a device can process the complex computations required for DNN inference. For instance, the frequency at which a GPU operates can greatly affect both the speed and energy usage of DNN inference tasks.

However, the relationship between these hardware parameters and the energy efficiency of DNN inference is often non-monotonic. This means that changes in parameters like GPU frequency do not always result in predictable changes in energy consumption or inference time. For example, increasing the frequency could speed up the inference but also significantly increase energy consumption, and the optimal balance between the two is not always clear.

1.3 Reliance on DVFS for energy optimization

In practice, finding the optimal settings for these hardware parameters is a challenging task. Many practitioners, instead of manually optimizing these parameters, resort to using existing heuristics or tools. One such tool is Dynamic Voltage and Frequency Scaling (DVFS), a widely used technique in power management. DVFS allows the adjustment of the voltage and frequency of a processor dynamically, based on the current workload [76, 70]. By reducing the frequency and voltage during less demanding tasks, DVFS can effectively lower energy consumption, but it may also lead to increased inference time.

DVFS operates on the principle that a processor's power consumption is proportional to its operating frequency and the operating voltage. By scaling these parameters down during periods of low computational demand, it tries to reduce power usage, thus enhancing energy efficiency and extending the lifespan of the device. In the context of running AI models, this adaptability is crucial, especially in edge computing environments where energy resources are often limited. While DVFS is a valuable tool for power management, its effectiveness in adapting to the needs of Deep Learning models can be limited. One of the primary reasons for this limitation is the inherent nature of Deep Learning workloads, which are typically characterized by high computational intensity and relatively stable operational patterns. Unlike traditional applications where workload varies significantly, the consistent and heavy processing demands of Deep Learning models mean that there are fewer opportunities for DVFS to effectively reduce voltage and frequency without impacting performance.

Although DVFS provides a method to control energy usage, relying solely on this technique without fine-tuning other hardware parameters can result in missed opportunities for further energy savings. The challenge lies in balancing the trade-offs between energy consumption, inference time, and processing power, to achieve the most energy-efficient

configuration for DNN inference on edge devices. This balancing act requires a nuanced understanding of both the hardware capabilities and the specific demands of the DNN tasks being performed.

1.4 Motivation

There has been some related work recently that looks at the energy efficiency of DNN inference. Holly *et al.* [33] profile the Mobilenet-V2 as a function of hardware parameters (*e.g.*, number of cores); however, the authors do not consider workload parameters or the impact on accuracy. DeepEdgeBench [6] analyzes the power consumption of running DNN models on different edge devices, but the authors do not investigate workload and hardware parameter changes or their impact on energy. There are several prior works that focus on the energy impact of fine-grained power state and CPU frequency changes [81, 20, 21, 61, 7, 13], but such works are limited to servers and workstations (see Chapter 3).

As part of this Research Proficiency Examination (RPE), we study the impact of workload parameters and hardware knobs on the energy consumption of DNN *inference*, specifically for *edge* devices. We consider three edge devices—an entry-level Jetson Nano, a mid-tier Jetson Xavier NX, and a Jetson Orin NX, a top-tier SoC board—and six different DNN models for various inference tasks ranging from objection detection to language modeling. For workload parameters, we investigate different batch sizes, and consider other factors such as lazy loading and static graph execution which might impact inference. For hardware-related knobs, we explore different CPU and GPU frequency settings for each of three devices.

Our experimental results show that we can reduce inference energy by as much as 22% compared to DVFS by optimally tuning the CPU and GPU frequencies. We also find that batch size can impact energy consumption significantly (to the tune of 40–45%), even within the limited range of batch sizes supported by resource-scarce edge devices. Further, the above factors are independent and can thus be *combined* for even greater energy savings.

1.5 Roadmap

The rest of the RPE report is structured as follows:

1. **Background (Chapter II)**: This chapter will delve into the fundamentals of edge computing and its significance in the context of Deep Neural Networks (DNN) inference. It will explore the characteristics of edge devices, focusing on the Jetson Nano, Xavier

NX and, Orin NX, and highlight the challenges of energy-efficient DNN deployment on these devices.

2. **Related Work (Chapter III)**: This chapter will provide a comprehensive review of existing literature and research in the field. It will cover previous studies on DNN inference and training frameworks on multiple classes of devices including edge devices and servers. We focus on existing works studying energy consumption and performance of DNN deployments. This chapter will also highlight the gaps in the current research that our study aims to address.

3. **Experimental Design (Chapter IV)**: In this chapter, we discuss the experimental setup for evaluating the energy consumption of DNN workloads on the three aforementioned three edge devices. It will describe the chosen DNN models, hardware configurations, and the methodology for used for our study.

4. **Evaluation Results (Chapter V)**: This chapter will present the outcomes of the experiments. It will analyze the impact of various hardware settings (like CPU and GPU frequencies) and workload parameters (such as batch size and model initialization) on the energy efficiency of DNN inference on edge devices, along with the balance between energy efficiency.

5. **Conclusion (Chapter VI)**: The concluding chapter will summarize the main findings of the research, emphasizing the criticality of optimizing hardware and workload parameters for energy-efficient DNN inference on edge devices. It will also propose future research directions in this domain.

Chapter 2

Background

This background chapter provides an in-depth exploration of edge computing, emphasizing its critical role in the context of Deep Neural Network (DNN) inference. It delves into the attributes and challenges of edge devices, focusing on the Jetson Nano, Xavier NX, and Orin NX, and discusses the intricacies of deploying DNN models efficiently on these platforms.

2.1 DNN inference on smart edge devices

Edge computing is the layer closest to the users in the computing and IoT stack and is used to process incoming data in real-time. Edge computing encompasses a diverse array of devices, each playing a unique role in processing and managing data at the network's edge. This range includes everything from IoT sensors, which collect data from the environment to actuators that interact with it, and even extends to on-premises data centers that handle more substantial computational tasks. However, our focus is primarily on a specific subset of these devices known as *smart edge devices*. These devices are characterized by their ability to perform more complex computing tasks, often involving data processing and analysis directly at the source of data generation.

The increasing deployment of smart edge devices, such as Jetson Nanos, marks a significant shift in the domain of Artificial intelligence. These compact yet powerful devices are ideally suited for executing Deep Learning models at the edge of the network, closer to where data is generated and consumed. This shift not only reduces the latency associated with sending data back and forth to the cloud but also addresses privacy concerns by processing sensitive information locally. Additionally, the use of Jetson Nanos and similar devices enables real-time analytics and decision-making, essential for applications like autonomous vehicles, smart surveillance, and IoT-based systems. By harnessing the power of Deep

(a) Jetson Nano (b) Jetson Xavier NX (c) Jetson Orin NX

Fig. 2.1 Comparative Display of NVIDIA's Jetson Series: Showcasing the Jetson Nano, Jetson Xavier NX, and Jetson Orin, each exemplifying progressive leaps in AI and machine learning capabilities for edge computing, from entry-level to high-end applications.

Learning on these smart edge devices, we can unlock a new realm of possibilities, making technology more responsive, efficient, and secure in a variety of innovative applications.

Among the notable examples of smart edge devices are the Jetson Nano, Jetson Xavier NXs, and Jetson Orins. All three devices are a part of NVIDIA's Jetson family of system-on-chip boards with on-board GPUs, and are specially designed for edge AI applications.

2.2 Jetson Nano Specifications

The Jetson Nano, a product of NVIDIA's Jetson family, is a small, powerful computer specifically designed for the edge. It stands out for its unique blend of performance and power efficiency, making it an ideal platform for developing and running AI models in edge-based applications. The Jetson Nano is equipped with a quad-core ARM Cortex-A57 processor and a 128-core NVIDIA Maxwell GPU, which provides substantial processing power for Deep Learning inference tasks. This device supports high-resolution sensors and can process multiple concurrent neural networks on high-resolution videos, making it suitable for advanced AI applications such as image recognition and real-time video analysis in compact, power-constrained environments. Its ability to perform such tasks in a compact, power-constrained environment opens up numerous possibilities in fields like smart cities, healthcare, and industrial automation.

Apart from its robust processing capabilities, the Jetson Nano is distinguished by its ease of use and accessibility. It supports a range of standard hardware interfaces and has extensive software support through NVIDIA's JetPack SDK, which includes AI libraries, drivers, an operating system, and even a development environment. This level of support simplifies the

development process for AI applications, making it more accessible to developers of various skill levels. The Jetson Nano also stands out for its energy efficiency, operating at as low as 5 watts, which is a fraction of the power consumed by typical desktop computers. This low power consumption, coupled with its small form factor, makes the Jetson Nano an ideal choice for deploying AI at the edge, particularly in scenarios where space and power are at a premium.

2.3 Jetson Xavier NX Specifications

The NVIDIA Jetson Xavier NX stands as a significant step up from the Jetson Nano, targeting more demanding edge AI applications that require higher computational power. This advanced module is powered by a six-core NVIDIA Carmel ARM v8.2 64-bit CPU and a 384-core NVIDIA Volta GPU with 48 Tensor Cores. This powerful combination of CPU and GPU capabilities makes the Xavier NX exceptionally well-suited for high-performance AI applications, capable of delivering up to 21 TOPS (Tera Operations Per Second) of performance. This is a marked increase from the Jetson Nano, which offers up to 0.5 TOPS, showcasing the Xavier NX's suitability for more complex and demanding AI tasks, such as those found in robotics, drones, and intelligent video analytics.

In terms of memory and storage, the Jetson Xavier NX is equipped with 8 GB of 128-bit LPDDR4x RAM, significantly more than the 4 GB found in the Jetson Nano. This increased memory capacity, coupled with its 16 GB of eMMC 5.1 storage, enables the Xavier NX to handle larger and more complex neural networks and datasets, an essential factor for advanced machine learning tasks. The Xavier NX also offers superior I/O capabilities, including support for PCIe Gen 3, USB 3.1, and Gigabit Ethernet, which allow for faster data transfer and connectivity options, a critical aspect for integrating the module into complex systems.

Software-wise, the Jetson Xavier NX is supported by NVIDIA's JetPack SDK, which includes CUDA, cuDNN, and TensorRT software libraries. These tools are crucial for optimizing AI and machine learning workflows, offering accelerated computing power, which is vital for real-time processing tasks. The Xavier NX also supports a broad range of AI frameworks and models, making it a versatile platform for developers. This software support, combined with the powerful hardware, positions the Xavier NX as a more suitable option for professional developers and enterprises that require high-performance edge AI solutions.

In comparison to the Jetson Nano, the Jetson Xavier NX is positioned as a more robust and capable platform, designed for edge AI applications where performance and speed

are paramount. While the Jetson Nano is an excellent choice for entry-level and power-constrained applications, the Xavier NX caters to a more professional segment, offering the necessary power to handle complex AI models and high-throughput tasks. Its enhanced capabilities come at a higher cost and power consumption, but for applications that demand such high levels of performance, the Jetson Xavier NX is an ideal choice. The distinction between the Nano and the Xavier NX ultimately lies in their targeted use cases and performance requirements, with the Xavier NX serving as the go-to option for more intensive edge AI applications [5, 62].

2.4 Jetson Orin NX Specifications

The NVIDIA Jetson Orin NX represents the latest advancement in NVIDIA's Jetson series, designed to push the boundaries of AI and machine learning at the edge. As a successor to the earlier models like the Jetson Nano and the Xavier series, the Orin NX brings a significant leap in performance and capabilities. It is powered by an NVIDIA Ampere architecture GPU with up to 1024 CUDA cores and a 12-core Arm Cortex-A78AE CPU. This robust configuration enables the Orin NX to deliver up to 100 TOPS (Tera Operations Per Second) of AI performance, a substantial increase from the 21 TOPS offered by the Xavier NX and a massive jump from the 0.5 TOPS of the Jetson Nano. This heightened processing power allows the Orin NX to handle even more complex and demanding AI workloads, making it ideal for high-end robotics, autonomous vehicles, and advanced AI computing at the edge.

In terms of memory, the Orin NX comes with up to 16 GB of LPDDR5 RAM, which is a significant upgrade from the 8 GB LPDDR4x RAM of the Xavier NX and the 4 GB RAM of the Jetson Nano. This increased memory capacity, combined with faster memory speed, provides the bandwidth needed to support the high-throughput AI and deep learning tasks that the Orin NX is designed for. Additionally, the Orin NX offers enhanced I/O capabilities, including PCIe Gen 4, USB 4.0, and multi-Gigabit Ethernet, enabling faster and more diverse connectivity options crucial for complex systems integration.

Software support for the Orin NX is provided by the latest version of NVIDIA's JetPack SDK, which includes support for advanced AI and deep learning libraries like CUDA, cuDNN, ONNX and TensorRT. This software stack is optimized to leverage the full capabilities of the Orin NX's hardware, ensuring maximum performance and efficiency in AI processing. The Orin NX also continues to support a wide range of AI frameworks and models, further enhancing its flexibility and appeal to developers.

When compared to its predecessors, the Jetson Orin NX stands out for its significant performance enhancements. While the Jetson Nano is an entry-level AI module suitable for

basic AI tasks and learning purposes, and the Xavier series (both NX and AGX Xavier) are designed for mid to high-level AI applications, the Orin NX is targeted at the most demanding edge AI applications that require the highest levels of performance. Its superior processing power, memory, and I/O capabilities make it a formidable platform for next-generation AI applications. The Orin NX's increased capabilities do come with higher power requirements and likely a higher price point, positioning it as a premium option in the Jetson series.

2.5 Impact of CPU and GPU frequency on performance and energy

The NVIDIA Jetson series, encompassing the Jetson Nano, Jetson Xavier NX, and Jetson Orin, each offers multiple operating frequencies for both the CPU and GPU, allowing for a dynamic adjustment of performance based on the requirements of specific applications. This flexibility in operating frequencies is crucial for optimizing the balance between computational power, energy efficiency, and heat management, particularly in the diverse environments where these edge devices operate.

Starting with the Jetson Nano both the CPU and GPU can operate at various different frequencies, providing a range of performance levels. At lower frequencies, the Nano conserves energy and reduces heat output, making it ideal for continuous, less computationally intensive tasks, like basic image processing or sensor data analysis. Conversely, higher frequencies can be leveraged for short bursts of intensive work, such as complex neural network inference, albeit at the cost of increased power consumption and heat generation.

The Jetson Xavier NX and Jetson Orin further extend these capabilities. Both offer a broader range of operating frequencies. This range allows for more granular control over performance and energy consumption, catering to more demanding edge AI applications. High-frequency operations in these devices unlock the potential for running sophisticated AI models, such as those required in autonomous vehicles or advanced robotics, where rapid real-time processing is essential. Lower frequencies, on the other hand, are suitable for scenarios where power efficiency is paramount, such as in remote IoT devices or when operating on battery power. The ability to tailor the operating frequency to the specific needs of the task at hand not only maximizes the efficiency of these devices but also extends their applicability across a wider spectrum of edge computing scenarios.

In essence, the multi-frequency operational capabilities of the Jetson Nano, Xavier NX, and Orin are fundamental to their versatility and effectiveness as edge devices. They enable each device to adapt its performance characteristics to the requirements of the application,

Table 2.1 Specifications for Jetson Nano, Xavier NX, and Orin NX [55–57]

Specification	Jetson Nano	Jetson Xavier NX	Jetson Orin NX
CPU	4-core ARM A57	8-core Nvidia Carmel	8-core Nvidia Cortex
CPU Freq. range	102 MHz – 1.48 GHz	115 MHz – 1.9 GHz	115 MHz – 2.0 GHz
CPU Freq. step	100 MHz (15 steps)	77 MHz (25 steps)	77 MHz (26 steps)
GPU	Nvidia Maxwell	NVIDIA Volta	NVIDIA Ampere
CUDA Cores	128	384	1024
Tensor Cores	-	48	32
Memory	4 GB LPDDR4	8 GB LPDDR4	16GB LPDDR5
GPU Freq. range	76 MHz – 921 MHz	114 MHz – 1.1 GHz	308 MHz – 918 MHz
GPU Freq. steps	77 MHz (count 12)	90 MHz (count 15)	102 MHz (7 steps)
Throughput	472 GFLOPs	21 TOPs	100 TOPs
Power Modes	5W, 10W	10W, 15W	15W, 25W
Libraries	CUDA 10.2 + cuDNN 8.2.1	CUDA 10.2 + cuDNN 8.0.0	CUDA 11.8 + cuDNN 8.9.6

ensuring optimal balance between power, speed, and thermal management. This adaptability is a key factor in the widespread adoption and success of NVIDIA's Jetson series.

For illustration, Table 2.1 provides specifications of the NVIDIA Jetson developer boards that we use in this work: (a) Jetson Nano; (b) Jetson Xavier NX; and (c) Jetson Orin NX.

Chapter 3

Related Work

Since the advent of DNNs and its increased usefulness in recent years, several research works have been undertaken to effectively understand and optimize deep learning techniques on single-board systems and embedded platforms such as Jetson Nano, Jetson Xavier NXs, Asus Tinker, Google Coral TPU accelerators, and Raspberry Pi. This chapter presents a detailed examination of the existing scholarly work related to Deep Neural Networks (DNN), focusing on their energy consumption and performance aspects. It is methodically divided into four sections: Studies on the Impact of Energy on DNN Inference, Research on the Impact of Power and Performance on DNN Inference, Investigations into the Impact of Energy on DNN Training, and Analyses of the other aspects impacting the execution of DNN workloads on edge devices.

3.1 Studies on Energy Consumption of DNN Inference on Edge

The closest prior work to ours is the study by **Holly *et al.*** [33], which examines *the energy usage of MobileNet-V2 for different number of CPU cores, CPU and GPU frequency, and layers, on Jetson Nano*. This paper presents a detailed analysis of the energy consumption of deep neural networks (DNNs) on the NVIDIA Jetson Nano, a popular embedded system used for edge computing and AI applications. The study focuses on understanding how various factors such as hardware configurations, network structures, and individual DNN layers impact the power and energy efficiency of these systems.

Key findings of the research include the significant influence of hardware settings on power consumption and system performance. The study reveals that adjusting the number of active CPU cores and their operating frequencies, along with GPU frequency, can markedly

affect the energy efficiency of DNN operations. For instance, higher CPU and GPU frequencies generally increase power consumption but can decrease total energy usage by reducing computation time. This insight is crucial for optimizing the balance between performance and power efficiency, especially in applications where either speed or energy consumption is a priority.

Additionally, the research delves into the effects of individual layer types on power usage. It becomes evident that some network structures are more energy-efficient than others, highlighting the importance of considering power consumption as a key factor when designing or choosing DNN models for edge computing applications. The study provides guidance on selecting or optimizing DNN architectures to minimize energy consumption without significantly compromising performance.

Our work builds on this study and extends beyond MobileNet-V2 to incorporate two additional image classification models, real-time object detection, and two Transformer-based language models, on three edge devices. In addition to hardware knobs, our work also considers the energy impact of workload parameters, such as batch size, and model initialization.

Daghero *et al.* [37] *provides a comprehensive examination of techniques and strategies to enhance the efficiency of deep learning models, particularly for inference tasks on edge devices*. This focus is essential due to the inherent limitations of these devices in terms of processing power and energy resources. The paper is structured into several key sections, each addressing different aspects of the topic.

The paper offers a fundamental understanding of deep learning models and their computational aspects, emphasizing the significance of optimizing these models for energy-efficient operation on edge devices. The paper also highlights the necessity of hardware acceleration and model optimization due to the demanding computational requirements of deep learning models. It delves into various types of deep learning models such as feed-forward models, including fully connected and convolutional neural networks, and sequential models like RNNs, LSTM, and GRU. The computational challenges and characteristics of these models are thoroughly discussed.

The core of the paper is dedicated to exploring various optimization strategies for deep learning inference on edge devices. These include both static and dynamic optimizations. Static optimizations, which are applied at the design time of the model, include techniques like quantization, pruning, knowledge distillation, and collaborative inference. Dynamic optimizations are input-dependent and adapt the optimization to the processed input, including methods like conditional inference and fast exiting, hierarchical inference, and dynamic

tuning of inference algorithm parameters. The paper elaborately discusses each of these techniques, their implementation, and the impact on the efficiency of deep learning models.

While the paper offers valuable insights, it has certain limitations, due to it primarily being a position and survey paper without any experimental results. It potentially lacks extensive empirical data or real-world case studies, which are crucial for validating the theoretical approaches and optimization techniques suggested.

Boroumand *et al.* [10] *benchmark the performance and shortcomings of executing DNN models on a Google TPU during inference by analyzing the power consumption of individual layers*. The paper offers a comprehensive analysis of the performance and energy efficiency of machine learning inference on edge devices, specifically focusing on the Google Edge Tensor Processing Unit (TPU). The research identifies significant limitations in the Edge TPU's architecture and proposes a new framework, Mensa, to address these challenges.

The study begins by characterizing the performance of the Edge TPU using various Google edge neural network (NN) models. It uncovers three primary shortcomings of the Edge TPU: suboptimal computational throughput, poor energy efficiency, and a memory system that acts as a significant bottleneck. These issues are attributed to the Edge TPU's monolithic design, which fails to accommodate the diverse requirements of different NN models and layers. For instance, the Edge TPU's one-size-fits-all approach results in low utilization of its processing elements and inefficient energy use, particularly for models like LSTMs and Transducers.

To overcome these limitations, the paper introduces Mensa, a novel acceleration framework. Mensa incorporates multiple heterogeneous edge ML accelerators, each tailored to the characteristics of specific subsets of NN models and layers. This design allows for more efficient execution by matching the accelerator to the specific requirements of each NN layer, considering factors like computational intensity and memory usage. Mensa's effectiveness is demonstrated through significant improvements in energy efficiency and computational throughput compared to the Edge TPU, showcasing its potential for more efficient machine learning inference on edge devices.

While the study provides insights into the model parameters that influence energy for a DNN inference task, their study does not consider the energy impact of hardware knobs such as CPU and GPU (and in the case of this paper TPU) frequency. The work also limits the analysis to Google TPUs without considering other edge devices. The Mensa framework proposed by the paper is not DNN architecture agnostic and has three different accelerators for three different DNN layer types. Workload parameters such as batch size, are not considered during the acceleration phase, as well.

Xu *et al.* [77] *focuses on the efficient offloading of DNN inference tasks in 5G-enabled Mobile Edge Computing (MEC) networks*. The study addresses the challenge of dynamically offloading inference requests to optimize energy consumption while meeting stringent latency requirements. It proposes exact and approximate algorithms for the admission of a single inference request, along with learning-based dynamic inference offloading methods for online adaptation. The algorithms are designed to minimize energy consumption across mobile devices, cloudlets, and 5G base stations, or to maximize the number of admitted inference requests. The performance of these algorithms is evaluated through extensive simulations and real-world test-bed implementations, demonstrating their effectiveness over theoretical models and other similar heuristics.

However, the paper's focus on the energy-aware offloading of DNN inference tasks primarily addresses the offloading mechanisms in 5G-enabled MEC environments without delving into the specific architectural differences among various edge devices, which might affect their suitability for different types of DNN applications. The paper also does not explore the implications of different DNN architectures on the offloading strategy, which could be significant given the varying computational and energy requirements of different models.

Hanafy *et al.* [28] *offers a comprehensive analysis of the energy efficiency of deep neural network (DNN) inference on edge devices, particularly focusing on NVIDIA's Jetson Nano*. The study evaluates over 40 popular pre-trained DNN models to understand their performance in terms of accuracy, latency, and energy consumption. It demonstrates that while larger and more complex models generally provide higher accuracy, they also consume significantly more energy, highlighting a trade-off between model accuracy and energy efficiency. The authors propose a new metric, "*accuracy per joule*", to quantify this trade-off and introduce a model recommendation algorithm. This algorithm assists application designers in selecting the most suitable DNN model that satisfies application-specific constraints on accuracy, energy, and latency.

The experimental methodology includes testing various model families (like ResNet, EfficientNet, VGG) for classification and detection tasks, while considering model characteristics like the number of parameters and floating-point operations (FLOPs). The findings reveal that smaller models are generally more energy-efficient but offer lower accuracy. This relationship between model size, accuracy, and energy consumption is non-linear, indicating that the most accurate models are not always the most energy-efficient. The recommendation algorithm developed in the study takes these factors into account, aiming to optimize model selection based on specific application requirements.

The paper effectively addresses the design of energy-efficient deep neural network (DNN) models with a focus on maintaining accuracy, but it falls short in exploring the impact of hardware parameters of the Jetson Nano on DNN deployment. This omission is significant as hardware configurations, such as CPU and GPU utilization, memory bandwidth, and thermal constraints, play a critical role in the performance and energy efficiency of models in real-world scenarios. Additionally, the study limits the design strategy to Jetson Nano, raising generalizability concerns about migrating the work to other edge devices.

Molom-Ochir *et al.* [49] *focuses on evaluating various GPUs for their energy and cost efficiency in running deep neural network (DNN) inferences. The study compares several popular embedded and desktop GPUs, highlighting that while larger devices typically provide higher throughput, they are not always the most energy-efficient.* The authors present a novel recommendation algorithm to assist system designers in selecting the ideal hardware accelerator that balances cost, power, and performance constraints. By analyzing different types of DNN models, the paper sheds light on how energy consumption and latency vary across a range of devices, and it demonstrates that the efficiency of DNN inference is influenced not only by the hardware but also by the specific characteristics of the DNN model being run.

The methodology involves testing a variety of DNN models on multiple GPUs, including both embedded GPUs (like the Jetson family) and desktop GPUs (such as the GTX 1660 and GTX 1080). The research reveals interesting correlations between model architecture, accelerator capabilities, and energy efficiency. For example, models that are better optimized for higher parallelization in larger GPUs show more energy-efficient performance due to faster inference capabilities. The study also examines throughput per dollar, providing insights into the cost-effectiveness of different hardware setups. This leads to the development of the aforementioned recommendation algorithm, which factors in financial and throughput constraints to suggest the most suitable hardware for a given application.

A notable drawback of the paper is its reliance on Dynamic Voltage and Frequency Scaling (DVFS) for all experiments without individually examining the impact of specific device and workload parameters such as batch size, CPU, and GPU frequency. This approach may overlook how these parameters uniquely influence the energy efficiency and performance of AI inference workloads on GPUs. Batch size, for instance, can significantly affect throughput and latency, while CPU and GPU frequencies directly impact power consumption and processing speed. By not isolating and analyzing these factors in detail, the paper potentially misses out on identifying more nuanced strategies for optimizing energy and cost efficiency.

Ghasemi *et al.* [23] *presents an innovative approach for the efficient execution of a network of trained deep neural network (DNN) models on edge computing devices. The research focuses on a scenario involving two interconnected devices—a user-end device (U) with limited energy and performance capabilities, and a cloudlet (C) with higher performance and no energy constraints.* The primary objective is to distribute the computation of DNN models between these two devices to minimize the energy consumption of the user-end device while accounting for variability in wireless channel delay and performance overhead from parallel model execution. The framework is implemented using an NVIDIA Jetson Nano as the user-end device and a Dell workstation with a Titan Xp GPU as the cloudlet. Experiments demonstrate significant improvements in terms of energy consumption and processing delay, showcasing the effectiveness of the proposed approach.

The paper primarily emphasizes the concept of server-edge offloading, concentrating on the distribution of computational tasks between a user-end device and a more powerful cloudlet. This approach is distinct from end-to-end inference purely on edge devices. In this server-edge offloading paradigm, the computation is partitioned and shared between the local device, which has limited computational resources and energy constraints, and the cloudlet, which offers higher computational power. However, this focus on server-edge offloading means that the paper does not delve into scenarios where the entire DNN inference process occurs end-to-end on a single-edge device.

Jeong *et al.* [35] introduces a TensorRT-based framework, named JEDI, optimized for NVIDIA Jetson embedded platforms. This framework supports various optimization parameters to accelerate deep learning applications, including multi-threading, pipelining, buffer assignment, and network duplication. A key aspect of this work is a parameter optimization methodology that combines heuristic balancing of pipeline stages among heterogeneous processors with fine-tuning of optimization parameters. The framework and methodology were evaluated with nine real-life benchmarks, achieving significant performance improvements and energy reductions compared to GPU-only baselines.

While the paper does focus on reducing total execution energy for DNN inference workloads on edge devices, it does so, not by optimizing the hardware parameters, but by efficient model placement strategies to utilize heterogeneous resources such as CPU and GPU for inference. In our case, we try to specifically find optimal hardware and workload parameters for energy-efficient execution for a single edge-resource which is the GPU.

Desislavov *et al.* [14] and **Ren *et al.*** [64] *are comprehensive surveys focusing on energy trends and collaborative strategies in edge AI inference, including cloud-edge offloading.*

The first paper analyzes current trends and future directions in compute and energy consumption for AI applications at the edge. It delves into various aspects such as hardware accelerators, edge computing architectures, and energy-efficient AI algorithms. The study emphasizes the growing need for balancing computational demands with energy constraints in edge environments, highlighting the importance of energy-aware AI models and hardware.

The second paper explores different paradigms of collaborative deep neural network (DNN) inference in edge computing environments. It discusses various collaborative inference strategies involving cloud, edge, and end devices, outlining their architectures, performance optimization techniques, and application scenarios.

However, both papers primarily present theoretical analyses and classifications of existing research efforts without conducting any empirical evaluations or experiments to advance these concepts. They provide a synthesis of existing knowledge and propose theoretical frameworks and future research directions, but they do not offer new empirical data or experimental validation of the proposed ideas.

3.2 Studies on Power and Performance aspects of DNN Inference on Edge

In this subsection we investigate papers that delve into the impact of power and performance on edge devices, offering detailed insights into computational efficiency and throughput. However, they notably overlook a critical aspect of energy consumption in their analysis.

Baller *et al.* [6] *investigate power consumption on multiple edge devices. The paper presents an extensive analysis of the performance of deep neural networks (DNNs) on various edge computing devices. The study covers five different edge devices: Nvidia Jetson Nano, Google Coral Dev Board, Asus Tinker Edge R, Raspberry Pi 4, and Arduino Nano 33 BLE.* It evaluates these devices in terms of inference time, power consumption, and model accuracy across different deep-learning models and frameworks.

The findings reveal notable disparities in performance among the devices. For instance, the Google Coral Dev Board shows superior performance in terms of inference time and power consumption for TensorFlow-based quantized models. In specific scenarios, such as when computing time is less than 29.3% of the total for MobileNetV2, the Jetson Nano outperforms other devices. This comprehensive benchmarking offers insights into the strengths and limitations of these edge devices, providing valuable information for selecting the appropriate hardware for specific DNN applications.

A notable limitation in the aforementioned work is their lack of exploration into the optimization of hardware and model parameters specifically for enhancing energy efficiency. This omission means that while the paper provides valuable insights into the baseline energy consumption characteristics of these devices, it does not delve into potential strategies or modifications that could be employed to reduce energy usage.

Süzen *et al.* [69] study performance metrics for inference of CNN models on edge. *The paper provides a detailed comparison of the performance of these single-board computers when running convolutional neural networks (CNNs) for image classification.* The study evaluates the Nvidia Jetson Nano, Nvidia Jetson TX2, and Raspberry Pi 4 across various parameters such as accuracy, time, memory, CPU and GPU power consumption, and cost. Using a fashion product image dataset, the paper benchmarks the performance of a 2D CNN model developed to classify 13 different fashion products. The datasets used in the tests vary in size (from 5K to 45K images), allowing for a comprehensive comparison across different computational loads.

The paper provides a high-level performance and accuracy analysis of convolutional neural networks (CNNs) on three distinct edge devices. Its primary focus is on establishing the minimum hardware requirements necessary for executing CNNs without compromising on accuracy. This objective leads the study to concentrate on a broad comparison across devices, rather than delving into the finer details of device-specific and workload-specific parameters. Such an approach is suitable for determining baseline hardware capabilities but may not capture the nuances of optimizing performance under varying operational conditions or for different types of neural network architectures beyond CNNs.

Ahn *et al.* [1] *study the impact of quantization on latency and memory consumption. The paper focuses on the impact of quantization techniques on deep neural network (DNN) inference performance in edge computing environments.* The authors conduct their study using Intel x86 processors and a Raspberry Pi device with an ARM processor. They employ several DNN inference frameworks, including OpenVINO, TensorFlow Lite (TFLite), ONNX, and PyTorch, and test various models like MobileNetV2, VGG-19, and DenseNet-121. The research uses the MLPerf Edge Inference benchmark to measure latency and throughput. The results demonstrate that OpenVINO and TFLite are the most optimized frameworks for Intel CPUs and Raspberry Pi devices, respectively. Notably, INT8-based quantized models delivered significantly better performance over FP32 models in certain scenarios. The paper also observes no loss in accuracy except with static quantization techniques

While the study provides a thorough comparison of performance metrics like accuracy, inference time, and memory consumption, it overlooks the critical aspect of how

these operations affect energy consumption as well as the optimal deployment schema for the DNNs analyzed. Another limitation of the work is its narrow focus on the performance benefits of various quantization schemes, without addressing the sustainability and energy aspects of inference on edge devices. While the study provides valuable insights into how quantization affects latency, throughput, and accuracy, it overlooks the impact of these techniques on energy consumption and overall environmental footprint.

There are also orthogonal research works, as discussed below, that focus on studying the performance impact of specific DNN models on edge devices [58, 16, 24, 72].

Panopoulos *et al.* [58] present a comprehensive analysis of Transformer models specifically in the context of natural language processing (NLP) tasks on mobile devices. It addresses the performance and efficiency of various Transformer models, evaluating their compatibility with mobile processors and the impact of different quantization methods on their performance. The study includes a systematic benchmarking of a suite of Transformer models on different mobile devices, exploring the balance between model accuracy and computational demands.

Feng *et al.* [16] conducts a comparative analysis of YOLO (You Only Look Once) models on three different edge computing devices: NVIDIA Jetson Nano, NVIDIA Jetson Xavier NX, and Raspberry Pi 4B with Intel Neural Compute Stick2. Focusing on object detection, the study benchmarks two versions of YOLO across these devices to evaluate inference performance. The results provide empirical insights into the resource characteristics of each device, offering practical recommendations for deploying AI applications in edge environments.

Ullah *et al.* [72] presents a detailed analysis of the performance of NVIDIA Jetson platforms (Nano, TX1, and Xavier) for processing 3D point-cloud and hyper-spectral image data. It focuses on evaluating these platforms using two distinct deep learning architectures: PointNet for 3D point-cloud classification and stacked auto-encoders for hyper-spectral image classification. The study aims to understand the computational capabilities and limitations of the Jetson series for handling complex and computationally intensive data types.

The three aforementioned papers primarily focus on evaluating workload parameters of specific use-case models. They delve into the performance characteristics of various deep learning architectures like YOLO for object detection, CNNs for image classification, and specialized models for processing 3D point-cloud and hyper-spectral image data. However, a notable aspect of all these studies is their limited focus on the underlying edge hardware. They do not extensively explore the hardware capabilities, configurations, or potential hardware optimizations that could impact the performance of these models on edge devices. This lack of hardware-centric analysis leaves a gap in

understanding how different hardware configurations or enhancements could influence the overall efficiency and effectiveness of deploying these models in edge computing scenarios.

Hadidi *et al.* [26] *provides a comprehensive analysis of the performance of deep neural networks (DNNs) across a range of commercial edge devices. It evaluates several popular frameworks and models, particularly focusing on convolutional neural networks (CNNs), to determine their impact on execution performance. The study includes detailed measurements of power consumption and temperature behavior, offering insights into the efficiency and practicality of deploying DNNs on these devices.* The research aims to fill the gap in unified comparisons of edge devices, enabling a better understanding of the optimal deployment scenarios for DNNs in edge computing environments.

The paper conducts a high-level analysis, comparing the performance of deep neural networks across various commercial edge devices and edge-acceleration frameworks like TensorRT, ONNX, TensorFlow-lite etc. The study primarily focuses on the execution performance of different DNN models using popular frameworks and measures metrics like power consumption and temperature behavior. However, it does not delve into detailed hardware and workload parameters such as batch size, CPU and GPU frequencies, which can significantly influence model performance and energy efficiency.

Mittal *et al.* [48] provides a high-level review and compendium of existing work on DNN optimization in edge devices and provides broad overviews on which model architectures are best for a particular edge device architecture.

This is primarily a survey report on efficient edge deployments which summarizes results from multiple academic works, without any empirical or experimental contributions on their own.

Hao *et al.* [29] presents a comprehensive evaluation of the performance and resource heterogeneity of various edge devices using popular deep learning (DL) frameworks and deep neural network (DNN) models. *The study benchmarks several DNN models for image classification across a range of edge devices, from Raspberry Pi to high-performance GPU-equipped devices like Jetson Xavier NX. It highlights the impact of device hardware, batch sizes, and DL frameworks on DL inference performance and throughput.* The research also characterizes the resource usage patterns of edge devices, including CPU and memory usage, power consumption, and framework initialization overhead.

The paper focuses on two main aspects of edge device performance in the context of deep learning inference: performance and throughput. The study does not extend to analyzing energy consumption and power usage patterns, which are also vital factors, especially in edge computing scenarios.

3.3 Studies on Energy Consumption of DNN Training

In this section, we delve into works studying the energy consumption during the training phase of DL models. As the computational complexity and data requirements of these models escalate, understanding and optimizing energy use has become paramount. The differentiation between DNN training and DNN inference is crucial because the energy demands and optimization strategies can significantly differ between the training and inference stages of DNNs. While training involves processing large datasets to build the model and is often compute-intensive, inference is the phase where the trained model is used to make predictions or decisions, typically requiring less computational power but often needing to be highly efficient, especially in real-time or edge computing scenarios.

You *et al.* [81] *delves into the energy dynamics of deep neural network (DNN) training on GPUs. The core of the study is the introduction of Zeus, a novel framework designed to optimize energy consumption during DNN training without significantly affecting performance.* Zeus operates by intelligently adjusting job- and GPU-level configurations to find the most energy-efficient yet effective training setup. It uses an online exploration-exploitation strategy, which allows it to adapt to changes in data over time. This approach is particularly relevant as DNN models and training datasets evolve, necessitating a dynamic optimization process.

This work predominantly concentrates on the energy efficiency of training deep neural networks (DNNs) on GPUs, leaving a gap in its examination of energy consumption during the inference phase of these networks. The paper's focus on training, therefore, addresses only one part of the energy consumption puzzle in the lifecycle of DNNs/ Furthermore, the research is centered on optimizing energy consumption for deep neural network (DNN) training primarily on server-grade GPUs. This focus inherently limits the applicability of its findings to edge devices. Server-grade GPUs, characterized by their high performance and power capacities, differ significantly from the GPUs typically used in edge devices, which are designed for lower power consumption and smaller form factors. Consequently, the energy optimization strategies and results derived from server-grade GPUs in this study may not directly translate to edge computing environments, where hardware constraints and performance requirements are distinctly different. This gap highlights the need for separate investigations and optimization techniques tailored specifically to the unique characteristics of edge devices.

Prashanthi *et al.* [68] is a recent work analyzing energy-efficient *training* of DNN workloads on edge. *The paper offers an extensive analysis of the performance of NVIDIA Jetson devices (AGX Xavier, Xavier NX, and Nano) in training deep learning models. The authors experiment with different DNN models and datasets, varying hardware resources like*

CPU cores, storage media, and power modes. They also consider software configurations such as the number of data loader workers in PyTorch. The results provide insights into how these factors impact training time, energy usage, and the interplay between CPU and GPU performance. This research offers practical takeaways for optimizing DNN training on edge devices, balancing time and energy usage, and selecting suitable hardware for specific DNN workloads.

While exhaustive in scope covering both hardware and workload knobs for edge devices, the paper focuses on training rather than inference. The results from deep learning model training cannot be directly interpolated for inference due to fundamental differences in computational requirements and processes. Training involves iterative adjustments to the model's parameters across extensive datasets, which is computationally intensive and energy-consuming. In contrast, inference is about applying the trained model to new data, a process that is typically less resource-intensive and follows a different computational pattern, making direct extrapolation from training to inference inaccurate.

Wang *et al.* [75] discusses an innovative processor designed for efficient deep learning training on edge devices. *The processor, named "Trainer", supports sparse deep neural network (DNN) training by dynamically pruning weights during training iterations, significantly reducing the computational load and energy usage compared to traditional methods.* Trainer incorporates unique mechanisms such as an Implicit Redundancy Speculation Unit (IRSU) and a dynamic sparsity adaptive dataflow to handle challenges in sparse training, improving energy efficiency and computational throughput.

Hong *et al.* [34] focuses on enhancing the efficiency of training deep convolutional neural networks (DCNNs) on edge devices. *Efficient-Grad introduces a novel algorithm-hardware co-design approach, emphasizing gradient optimizations to improve energy efficiency and training throughput on edge devices, with minimal loss in validation accuracy.* It leverages a unique training approach that includes sign-symmetric feedback alignment and stochastic gradient pruning, which reduces computational load and improves energy efficiency without significantly compromising the model's accuracy.

Xun *et al.* [78] explores innovative approaches for optimizing Deep Neural Networks (DNNs) on heterogeneous embedded systems. The method divides DNN convolution layers into groups, which are then trained incrementally. At runtime, groups can be pruned or added back to adjust the balance between inference time/energy consumption and accuracy, without needing to retrain the model. By combining task mapping and Dynamic Voltage Frequency Scaling (DVFS) with the dynamic DNN, the approach enables finer trade-offs

between accuracy, power, energy, and time, across a wider dynamic range. This makes it highly adaptable to varying runtime conditions and resource availability.

Model-aware mathematical optimizations, such as sparsity-reducing algorithms, play a crucial role in enhancing the efficiency and performance of deep learning models, particularly in resource-constrained environments like edge computing. These mathematical techniques focus on reducing the complexity of models by identifying and eliminating redundant or less significant parameters, which leads to fewer computations and thus lower energy consumption. However, to achieve the best results in terms of both performance and energy efficiency, it is essential to combine these software optimizations with hardware-level adjustments or 'hardware knobs' such as clock speeds, power settings, or exploiting specialized hardware accelerators (without relying on DVFS for hardware optimization). This presents a significant gap in research for us to exploit, which is where our work differs from the three aforementioned works.

3.4 Studies on other aspects of Deep Learning on Edge

In this section, we analyze works that investigate a range of other critical optimization techniques for deploying Deep Neural Networks (DNN) on edge devices. We explore various strategies that enhance the overall performance and functionality of DNNs in edge computing environments, encompassing aspects such as mathematical optimizations impacting model size, model pruning to reduce the computational complexity of the workload, transfer learning for faster training, and dynamic model architectures.

Lee *et al.* [41] presents a method for real-time Deep Neural Network (DNN) inference and training. It introduces SubFlow, a strategy that dynamically constructs and executes a sub-network of a DNN to meet varying time constraints while maintaining comparable performance to the full network. This approach involves two algorithms for the dynamic construction of a sub-network and its time-bound execution and is demonstrated to be effective with popular DNN models and in a practical autonomous robot application.

Han *et al.* [27] proposes a novel framework for efficiently scaling Deep Neural Networks (DNNs) in mobile vision systems. It achieves this by block-level model scaling and retraining, allowing for a large exploration space of model sizes with minimal training time. The framework dynamically combines these scaled blocks at runtime to optimize for accuracy under specific resource and latency constraints. Extensive evaluations demonstrate that LegoDNN significantly outperforms existing model scaling and FLOP scaling techniques in terms of inference accuracy, latency, and energy consumption in various mobile vision applications.

Park *et al.* [59] resents a method for optimizing Deep Neural Networks (DNNs) for efficient real-time inferencing in automotive applications, particularly for object detection in Advanced Driver Assist Systems (ADAS). The authors utilized transfer learning with a pre-trained YOLOv3 model, followed by layer fusion and optimization techniques to enhance performance. The optimized model significantly increased the frame rate and accuracy in vehicle detection tasks on an NVIDIA Jetson embedded device, demonstrating its suitability for real-time applications in embedded systems.

Lou *et al.* [45] proposes a novel method, Dynamic-OFA, for optimizing Deep Neural Network (DNN) performance on heterogeneous embedded platforms. Dynamic-OFA utilizes a pre-trained Once-For-All (OFA) network, from which it samples a variety of sub-networks tailored for either CPU or GPU performance. This approach allows for real-time switching between these sub-networks to meet varying performance requirements and hardware constraints, without needing additional training. The method demonstrates significant performance improvements in terms of speed and accuracy for DNN tasks on platforms like the Nvidia Jetson Xavier NX.

Yi *et al.* [79] presents NNV12, an innovative system engine designed to optimize cold inference performance for deep neural networks on edge devices. NNV12 employs three key optimization strategies: kernel selection, post-transformed weights caching, and pipelined execution. These optimizations collectively address the challenges of cold inference, where a DNN model needs to be loaded and initialized before execution, often leading to significant delays. The system demonstrated substantial improvements in reducing cold inference latency and energy consumption compared to existing DNN engines.

The collection of papers discussed in this section, while significantly contributing to the field of efficient deployment of DNN workloads on edge devices, primarily focus on algorithmic and architectural optimizations rather than direct energy optimization or the exploration of optimal hardware settings like CPU and GPU frequencies for deployment. For instance, methods like the dynamic sub-network strategy in SubFlow, block-grained scaling in LegoDNN, and the runtime architecture switching in Dynamic-OFA, all aim to enhance performance and inference efficiency on constrained devices. Similarly, the TensorRT-based optimization framework JEDI and NNV12's approach to boosting cold inference latency mainly target speed and operational efficiency. While these innovations invariably lead to improved energy efficiency as a secondary benefit, they do not explicitly focus on tuning device-level parameters such as CPU/GPU frequencies or batch sizes, which have a direct impact on the deployment of DNN models. This highlights a distinct gap in the current research landscape, one that our work aims to address by specifically targeting the optimization of these crucial device and

workload parameters to achieve a more balanced and efficient deployment of DNNs on edge devices. This focus on fine-tuning hardware and software parameters stands orthogonal to the existing body of work, offering a complementary approach to the prevalent strategies in the field.

Chapter 4

Design and Methodology

In this chapter, we outline our experimental design and approach, encompassing the detailed setup of our experiments, the criteria for model selection, and the specific techniques employed in data collection and analysis. This comprehensive blueprint provides a clear understanding of the processes and principles guiding our research.

4.1 Experimental Setup

We used Jetson Nano, Jetson Xavier NX, and Jetson Orin NX for our experiments. The device specifications for all three devices are shown in Table 2.1. All three devices provide multiple onboard sensors measuring power for different components. Each device has a separate sensor for full module power, which we used for reporting our power readings. The I2C (Inter-Integrated Circuit) interface and Tegrastats utility are integral components of NVIDIA Jetson boards, including the Jetson Nano, Xavier NX, and Orin NX, offering detailed insights into power measurements and system monitoring. While we use both tools for polling the power readings, we use the readings provided by the I2C interface as it provides better control over the granularity of polling frequency.

The *I2C* interface on the NVIDIA Jetson boards provides a way to interact with I2C devices using the Linux file system, particularly for tasks like monitoring power consumption. The *sysfs* (System Filesystem) in Linux is a virtual filesystem that presents information about various kernel subsystems, hardware devices, and associated device drivers from the kernel's device model to user space and is used to poll the I2C measurements. The Jetson Nano's I2C bus can connect to various INA3221 power monitors which is a feature of Jetson boards. These devices can measure voltage and current, and thus calculate power consumption. Through the Sysfs interface, these I2C-connected power monitors appear as devices in the file system. Each device gets a directory in /sys/bus/i2c/devices/, typically

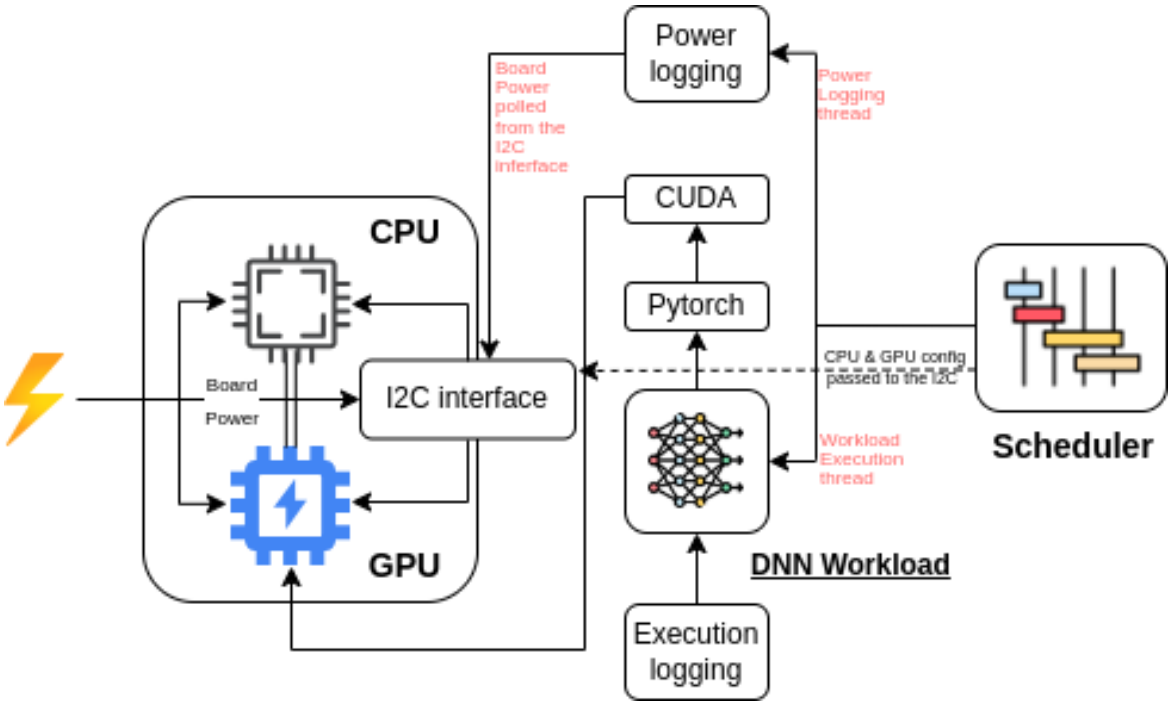


Fig. 4.1 Energy Measurement Workflow

named i2c-<bus number>/<I2C address>. Inside each device's directory, there are files that represent the various registers or data points that the device offers. Reading from these files can give us real-time data on voltage, current, and power consumption. Since these readings are accessible as file reads, they can be easily integrated into power measurement scripts or monitoring tools.

We manually polled and logged these readings via *I2C interface* every 100ms as this provided the best fine-grained granularity over the measurements without increasing the polling energy overhead. The energy overhead of 100ms polling was less than 0.5%. We found that more frequent polling (*e.g.*, 10ms or 1ms) resulted in high energy overhead of more than 2%.

4.2 DNN Workloads

We chose diverse DNN workloads that have practical inference applications in edge devices [58, 42, 82]. While benchmarking suites like MLPerf [51] do exist, we assembled our own benchmarking framework as we needed finer control over workload parameters and inputs for evaluation. We also try to cover all possible use-cases such as image classification,

Table 4.1 DNN workload specifications

Model	Layers	Params	Ops (GFLOPs)	Batch Size	Input
AlexNet	8	61M	0.727	8, 16, 32, 64	Tensor (3,224,224)
ResNet-18	18	11M	2	8, 16, 32, 64	Tensor (3,224,224)
MobileNet-V2	53	3.4M	0.57	8, 16, 32, 64	Tensor (3,224,224)
YOLOv4-Tiny	29	6.1M	6.9	8, 16, 32, 64	Tensor (3,416,416)
BERT-Tiny	4	4.4M	0.0353	8, 16, 32, 64	String (512 words, 1.1kb)
DistilBERT	6	43.2M	4.3	8, 16, 32, 64	String (512 words, 1.1kb)

object detection, and natural language inference and textual entailment, which might have downstream inference applications in edge devices.

For image classification task, we used the following three models:

1. **AlexNet [38]:** AlexNet is a convolutional neural network model comprising five convolutional layers, some of which are followed by max-pooling layers, and three fully connected layers at the end. It employs filters of varying sizes in its convolutional layers and uses overlapping max pooling, a technique that helped reduce the network's error rate.

2. **ResNet-18 [32]:** ResNet-18 is a convolutional neural network model that consists of 18 layers, featuring residual connections or "skip connections" around each of two-layer blocks. This design enables the network to avoid the vanishing gradient problem, allowing for deeper architectures without degradation in training performance.

3. **MobileNet-v2 [65]:** MobileNet-v2 is an efficient convolutional neural network architecture optimized for mobile and edge devices. It features an inverted residual structure with linear bottlenecks and shortcut connections, designed to provide high accuracy while being lightweight and low on computational cost, making it well-suited for performance-constrained environments like edge computing.

For natural language classification, we use the following Deep Learning models as our workload:

1. **BERT-Tiny [71]:** BERT-Tiny is a compact version of the BERT (Bidirectional Encoder Representations from Transformers) model, featuring a smaller architecture with fewer transformer blocks and attention heads. It is specifically designed for edge devices and resource-constrained environments, offering a balance between model size, latency, and performance suitable for edge-based natural language processing tasks.

2. **DistilBERT [66]:** DistilBERT is a streamlined version of the BERT architecture, designed to maintain most of BERT's performance while being smaller and faster. It achieves this by using a technique called knowledge distillation, where the model is trained to replicate the behavior of a larger BERT model, resulting in a lighter architecture with fewer transformer layers.

For Object detection, we used the following Deep Learning model as our workload:

- **YOLOv4-Tiny [36]:** YOLOv4-Tiny is a scaled-down version of the YOLOv4 (You Only Look Once) object detection architecture, designed for speed and efficiency with fewer convolutional layers and channels. It retains the core YOLOv4's principle of single-shot detection, making it suitable for real-time object detection in resource-constrained environments.

Table 4.1 lists the specifications for our models. All models were implemented in Python. We use OpenCV for YOLOv4 inference script and PyTorch for the other workloads. The inputs were carefully selected so that input size stays about the same across different models for better comparison and parity. In our evaluation, we primarily varied the batch size of the DNN workload (shown in Table 4.1). We select batch size values in increments of powers of 2, starting from 8 and ending at 64.

4.3 Hardware Knobs

All three devices have three main hardware knobs that can be changed during runtime—CPU clock frequency, GPU clock frequency, and the number of cores. The number of cores did not impact the power consumption in our experiments as the DNN models are implemented in Python, which is effectively single-threaded due to the Global Interpreter Lock (GIL) mechanism employed. The *Global Interpreter Lock* (GIL) is a mechanism used in the implementation of Python that limits the execution of multiple threads in a Python process. Essentially, the GIL is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once. This means in a multi-threaded program,

even if there are multiple threads available, only one thread can execute Python code at a time [11].

The impact of the GIL is particularly noticeable in CPU-bound and multi-threaded Python programs. In such cases, despite having multiple cores in the CPU, the GIL allows only one thread to execute at a time, leading to the underutilization of multi-core processors [31]. Consequently, Python's ability to perform parallel execution, especially in the context of computationally intensive tasks like DNN (Deep Neural Network) model training or inference, is limited. This limitation is a significant reason why adjusting the number of cores in a multi-core system doesn't markedly impact the power consumption or performance efficiency of Python-based DNN models [46].

Therefore, in experiments involving DNN models implemented in Python, running on devices like the Jetson Nano, Xavier NX, and Orin NX, altering the number of active CPU cores does not lead to significant changes in performance. Each Python process can only utilize one core effectively at a time due to the GIL, making the presence of additional cores redundant for improving the speed of Python DNN executions. As a result, the main hardware knobs that can influence the performance and power consumption in such scenarios are the CPU and GPU clock frequencies, rather than the number of CPU cores.

As such, we primarily focus our efforts on finding optimal CPU and GPU frequency for DNN inference on the 6 workloads.

4.3.1 Impact of CPU frequency on DNN workloads

In Deep Learning workloads, especially those running on Jetson edge devices like the Jetson Nano, Xavier NX, and Orin NX, certain processes are not accelerated by the GPU and thus are directly impacted by the CPU frequency. These non-GPU processes include [25, 43]:

- **Data Preprocessing:** Before feeding data into a neural network, it often needs to be preprocessed. This can include tasks like resizing images, normalizing values, or encoding text. These preprocessing steps are typically performed by the CPU.
- **Model Loading and Initialization:** Loading the Deep Learning model from storage into memory and initializing it for inference are tasks that rely on the CPU. Higher CPU frequencies can speed up these initial setup processes.
- **Data Transfer Operations:** The movement of data between different components of the system, such as from memory to the GPU or between different layers of a neural network, often involves the CPU. Faster CPU speeds can reduce bottlenecks in these data transfer operations.

- **Post-Processing:** After the GPU completes the inference, there might be additional processing needed on the output, such as converting raw output into a more interpretable form, applying thresholds, or performing non-maximum suppression in object detection tasks. These post-processing steps are handled by the CPU.
- **Control Flow Operations:** Tasks involving decision-making logic, looping, and other control flow operations within the inference process are generally executed by the CPU.

All three Jetson class devices offer a large range of CPU frequencies. Jetson Nano, Xavier NX, and Orin NX have 15, 25, and 26 unique CPU frequencies respectively [55–57].

4.3.2 Impact of CPU frequency on DNN workloads

In the context of NVIDIA Jetson edge devices such as Jetson Nano, Xavier NX, and Orin NX, the GPU frequency has a considerable impact on Deep Learning workload inference. The GPU, primarily responsible for accelerating the computationally intensive tasks in Deep Learning, operates more efficiently at higher frequencies. Key tasks affected by the GPU frequency include [3, 47]:

1. **Neural Network Computation:** The core operations of Deep Learning models, particularly the forward pass during inference (such as convolution, pooling, and activation functions), are heavily parallelized and executed on the GPU. Higher GPU frequencies allow these operations to be processed faster, reducing the inference time.
2. **Matrix Multiplications and Linear Algebra:** Deep Learning heavily relies on matrix operations, which are optimized and accelerated by the GPU. Increasing the GPU frequency can significantly speed up these calculations, directly impacting the overall performance of model inference.
3. **Graphical Data Processing and Parallel Processing of Large Datasets:** In addition to numerical computations, GPUs on Jetson devices also handle graphical data processing, essential in applications like computer vision and video analytics. Tasks like image and video analysis where processing large datasets is common, a higher GPU frequency can lead to quicker throughput and faster handling of batched data.

All three Jetson class devices offer a large range of GPU frequencies, which can be tuned as well. Jetson Nano, Xavier NX, and Orin NX have 12, 15, and 7 unique GPU frequencies respectively [55–57].

Given the large range of CPU and GPU frequency that can be optimized, the parameter space of each of the three edge devices is as follows:

- **Jetson Nano:** 180 possible GPU and CPU frequency combinations (15 CPU freq $\times$ 12 GPU freq)
- **Jetson Xavier NX:** 375 possible configurations (25 CPU freq $\times$ 15 GPU freq)
- **Jetson Orin NX:** 182 possible GPU and CPU frequency combinations (26 CPU freq $\times$ 7 GPU freq)

We set static CPU and GPU frequencies for our DNN inference workload execution via the `I2C interface` in the `sysfs config filesystem`. The devices also provide a handful of Power Modes, representing a much smaller subset of the frequency knobs; we chose to instead explore the full frequency range.

4.4 DVFS Baselines

The devices also offer a default dynamic frequency module (DVFS), which we compare against for energy savings. DVFS allows the system to dynamically adjust the voltage and frequency levels of the CPU and GPU, during runtime, based on workload demands. DVFS governors are mechanisms within the operating system that manage the voltage and frequency scaling of a processor. DVFS governors are designed to optimize the performance and energy efficiency of a system. By controlling the processor's voltage and frequency, they help reduce energy consumption when full processing power is not needed and boost performance during computationally intensive tasks. The governors monitor the system's workload and decide the appropriate voltage and clock frequency for the processor. This decision is based on predefined policies, heuristics, or algorithms that aim to maintain a balance between power saving and performance [40, 76, 70].

Common DVFS governor policies for optimizing CPU performance are:

1. **Performance Governor:** Always runs the processor at its maximum frequency to deliver the highest performance, disregarding energy savings.
2. **Powersave Governor:** Operates the processor at the minimum frequency to prioritize energy efficiency over performance [12].
3. **Ondemand Governor:** Scales the CPU frequency dynamically according to current usage. It boosts the frequency to the maximum when the system is busy and reduces it when the load is light [9].

4. **Conservative Governor:** Similar to demand but more gradual in changing frequencies. It slowly ramps up the CPU frequency to avoid rapid spikes in power usage [8].

5. **Schedutil Governor:** Dynamically adjusts CPU frequency based on real-time CPU load information from the scheduler, balancing performance and power efficiency. Can make both reactive and proactive adjustments [67].

Common DVFS governor policies for optimizing GPU performance (for NVIDIA GPUs) are:

1. **nvhost_podgov:** DVFS governor native to NVIDIA's Tegra processors dynamically adjusts the GPU's voltage and frequency based on workload and thermal conditions, optimizing for power efficiency and performance while preventing overheating [2, 15].

2. **wmark_active** DVFS governor is designed to adjust the voltage and frequency of a processor based on active workload thresholds, specifically watermarks (high and low marks). It increases the frequency when the workload exceeds a high watermark, indicating a high demand for processing power, and reduces the frequency when the workload falls below a low watermark, signifying lower processing demands [54].

We encounter 'Ondemand' and 'Conservative' governor policies for GPUs as well.

4.5 Workload Knobs

In the realm of Deep Learning, workload parameters like batch size and model initialization methods (static vs. dynamic) play pivotal roles in determining the efficiency and performance of model training and inference. Batch size influences how much data is processed at once, affecting memory usage and computational speed, while the choice between static and dynamic model initialization impacts how the model adapts to varying data inputs, with implications for flexibility and resource utilization.

4.5.1 Batch Size Optimization

The selection of batch size in Deep Neural Network (DNN) inference on edge devices is crucial, as it directly influences both the computational efficiency and the memory usage of the device. A larger batch size allows for more data to be processed simultaneously, which can lead to better utilization of the device's GPU, resulting in faster overall inference times due to parallel processing capabilities. However, this increase in batch size also demands more memory, which can be a limiting factor on edge devices with constrained resources.

Conversely, a smaller batch size reduces memory requirements and can be more manageable for devices with limited computational power, but it may lead to underutilization of the GPU, resulting in slower inference [50]. Therefore, finding the optimal batch size is a balancing act that requires consideration of the specific computational and memory constraints of the edge device, as well as the requirements of the DNN application. This optimization is key to achieving efficient and effective DNN inference in resource-constrained edge computing environments [15].

4.5.2 Static vs Dynamic Model Initialization

Frameworks like PyTorch and TensorFlow allow static and dynamic loading when a DNN model is initialized. This refers to different approaches to defining and initializing neural network models, which can significantly impact Deep Neural Network (DNN) inference.

Static Loading (TensorFlow default, can be enabled in PyTorch through JIT): TensorFlow traditionally uses a static graph approach, where the computational graph (defining all the operations and data flows) is created and compiled before the actual runtime. This means the structure of the network and the operations are defined and fixed before the model runs. Static loading allows for optimizations during compilation, which can lead to efficient execution during inference. However, it lacks flexibility as changes to the model architecture require recompilation of the graph [19].

Dynamic Loading (PyTorch default, Can be enabled in TensorFlow through 'Graph Execution mode'): PyTorch, on the other hand, typically employs dynamic or "eager" execution. Here, the computational graph is built on-the-fly during runtime. This approach offers high flexibility, allowing for changes and adjustments to the model on the fly, which is particularly beneficial during the development and debugging stages. However, since the graph is created at runtime, it might not be as optimized for efficiency as a statically compiled graph [63].

Static loading is often favored in DNN (Deep Neural Network) inference due to its efficiency and predictability. When using static loading, the computational graph of the neural network is defined and compiled in advance, before the actual inference process begins. This pre-compilation allows for optimizations such as layer fusion, memory allocation, and operation scheduling to be performed ahead of time, leading to more efficient execution [44].

4.6 Experimental Methodology

We executed each of the 6 DNN inference workloads and logged the execution checkpoints, CUDA events (*e.g.*, model loading and execution), and the requisite timestamps. In a separate thread in the background, we logged power readings using a Python process that reads the sysfs power entries from the I2C interface. Merging these logs and correlating with inference events (*e.g.*, model initialization, inference start and end) allowed us to compute the power consumption, execution time, and energy for specific intervals of interest, such as model loading and model inference. During testing, we disconnected all peripherals from the device and killed unnecessary background processes, including the GUI.

Each experiment consisted of running a DNN inference workload with a specified batch size under a specific CPU and GPU frequency setting. The workload in every experiment was kept constant at 3,200 inferences. Each experiment was repeated 10 times and average values were reported to ensure that the variation between different runs was low (much less than 5%).

Chapter 5

Evaluation

This chapter presents our experimental results on the impact of workload and device parameters/knobs on DNN inference energy consumption for all 6 aforementioned workloads on all three Jetson class edge devices. We also present some observations on other non-trivial factors such as programming implementation, which might impact the inference energy on devices.

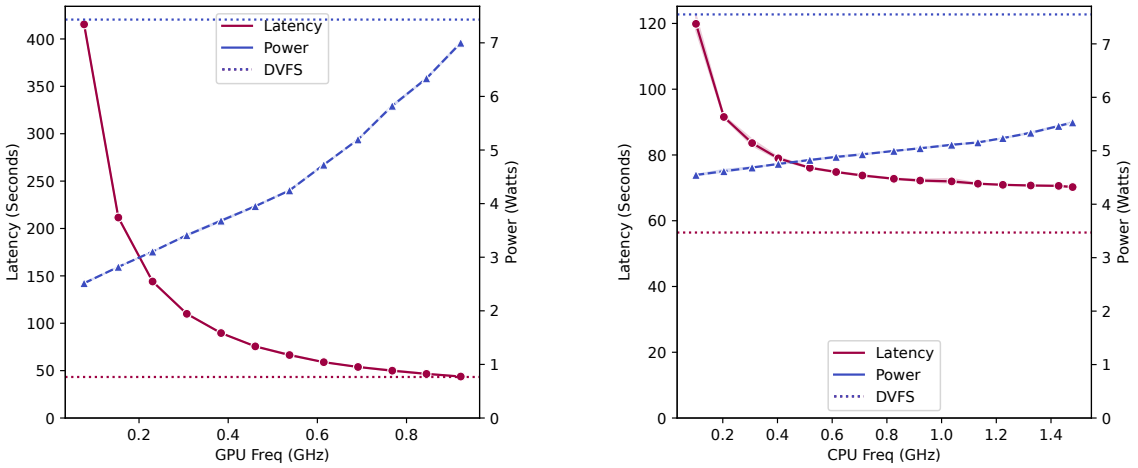
5.1 Frequency Scaling Sweep and DVFS for Nano

We start with a simple experiment to understand the impact of frequency scaling on power and inference time; in the next section, we analyze the impact on energy consumption, which is the aggregated power consumption over the workload execution.

Figure 5.1(a) shows the time taken for inference (left y-axis) and power consumed (right y-axis) as a function of GPU frequency for Jetson Nano using AlexNet inference with batch size of 8; here, we fix the CPU frequency at 1132.8 MHz. Note that the inference time, or latency, is the time to complete the full workload of 3,200 inferences.

We see that power consumption increases almost linearly with GPU frequency. However, the *decrease in inference latency starts to plateau out at higher GPU frequencies* as GPU is no longer the bottleneck and the performance is limited by other components such as memory and I/O bandwidth; similar effects have been noted in prior work on servers [74].

Both power and latency vary greatly as GPU frequency changes from lowest to highest; the span of power consumed is 4.7W, whereas the span of latency is 532s. Figure 5.1(b) shows a similar result, but for changing CPU frequencies; here, we fix the GPU frequency at 0.6912 GHz. While the trend is somewhat similar, we see that the span of power (1W) and span of latency (60s) is much narrower, indicating that ***CPU frequency has a smaller impact compared to GPU frequency***. This is because computationally intensive operations, such as



(a) Changing GPU frequency

(b) Changing CPU frequency

Fig. 5.1 Comparison of inference latency and power consumed under different CPU/GPU frequencies when running AlexNet on Jetson Nano.

tensor operations, are done by the GPU, whereas the CPU is responsible for less intensive tasks such as data preprocessing, initialization, and control flow.

Next, we evaluate the impact of DVFS on power and latency. The dashed lines in Figure 5.1 show the behavior when using DVFS instead of manually setting the frequencies; note that the DVFS values are same in both subfigures. We see that DVFS affords low latency but incurs very high power consumption, suggesting that *DVFS opts for higher frequencies*. This was confirmed by profiling, which revealed that DVFS operated at the highest CPU frequency 89% of the time and at the highest GPU frequency 83% of the time. Note that DVFS power usage can be slightly higher than that observed by manually setting the highest frequency because DVFS aggressively scales up the operating voltage when increasing frequency. We also experimented with the other DVFS governors available (*e.g.*, 'powersave', 'performance', 'ondemand'). We found that 'powersave' consumes <0.01% more energy than the default governors, primarily because although the average power is reduced by 30%, the execution time of the workload is increased by 43%. Likewise, we found that all other DVFS governors perform similarly to the default governor, with the difference in energy consumption being within 1%. With CPU DVFS, the best among other governors had 2.9% lower power but with 2.6% higher inference latency.

Takeaway 1. ***While GPU frequency scaling significantly impacts both power consumption and inference latency for deep learning tasks on the Jetson Nano, with diminishing returns at higher frequencies, CPU frequency scaling has a comparatively lesser impact on these***

metrics. DVFS tends to favor higher frequencies, leading to increased power consumption, but does not necessarily result in proportional improvements in energy.

5.2 Energy Topology under Jetson Nano

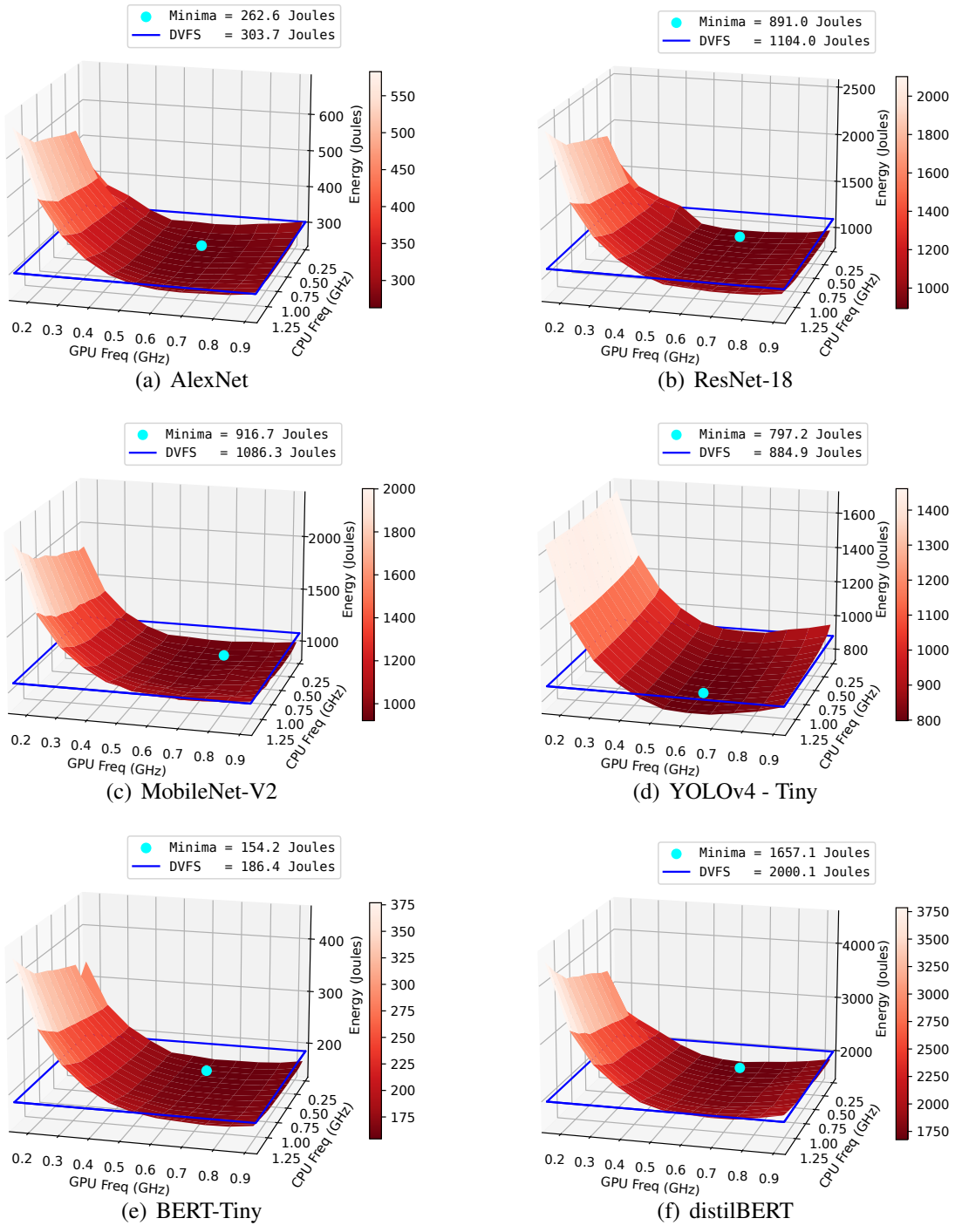
To study the impact of energy, we plot the energy consumed for inference as a function of CPU and GPU frequency for the 6 DNN workloads for Jetson Nano in Figure 5.2. The batch size is fixed at 16 for all workloads except MobileNet-v2, for which we use a batch size of 8 due to memory constraints. We see that energy does not change much with CPU frequency for a given GPU frequency. However, for a given CPU frequency, the ***energy consumption does change substantially with GPU frequency***. While not always visible, the *impact of GPU frequency on energy is not monotonic*; there exist some moderately high GPU frequencies at which the energy is minimized, as shown by the cyan dot in the plots.

The minimum energy obtained by sweeping over all the CPU and GPU frequencies does lower energy significantly when compared to DVFS, as noted in the legend for each subfigure. The percentage reductions in energy afforded by the minima over DVFS for Figures 5.2(a)–5.2(f) are 13.5%, 19.3%, 15.6%, 9.9%, 17.3%, and 17.2%, respectively. However, we found that the latency under the minima configuration is typically 28%–35% higher than that achieved under DVFS. The minima usually occurred at a GPU frequency of 614.4 MHz, with the optimal CPU frequency varying across the workloads.

Takeaway 2. ***Optimal energy configuration for all 6 DNN workloads on the Jetson Nano is achieved at moderate GPU frequencies, leading to energy savings to the tune of 19% compared to DVFS, albeit at the cost of increased latency.***

5.3 Impact of Batch size on Jetson Nano

Figure 5.3(a) shows the impact of batch size on energy consumption under Jetson Nano for AlexNet for a fixed CPU frequency (of 1.48GHz) and different GPU frequencies. We observe that energy drops almost linearly with batch size, with the energy consumed under the highest batch size (64) being 40–45% lower than that under the smallest batch size (8). Since the workload is constant (3,200 inferences) across batch sizes, a higher batch size reduces the total number of mini-batches, resulting in the GPU performing less I/O and preprocessing. The drop is less pronounced at higher batch sizes because the GPU computation throughput starts to saturate as the batch size increases. Note the higher energy profile for the lowest

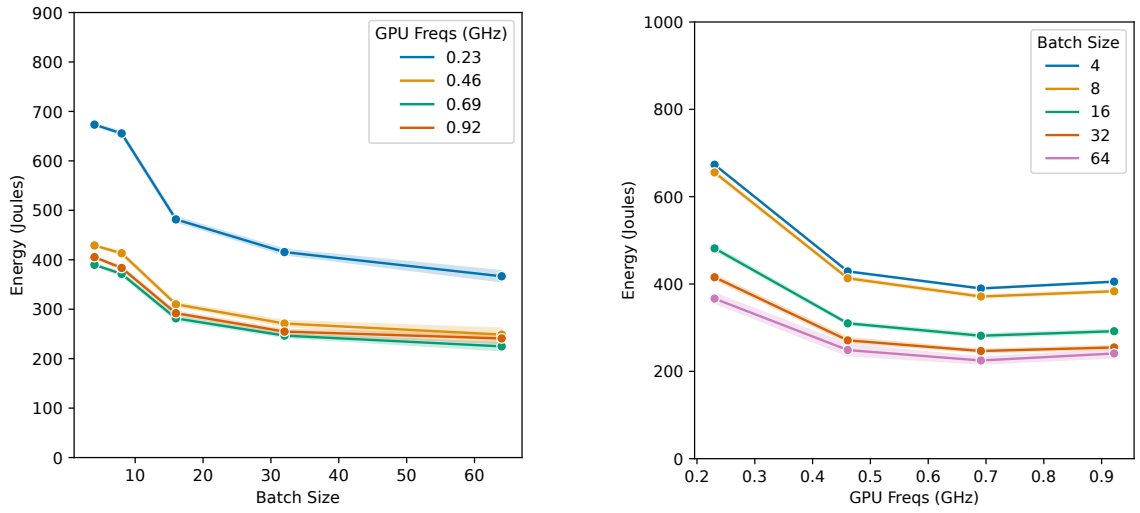


(a) AlexNet (b) ResNet-18

(c) MobileNet-V2 (d) YOLOv4 - Tiny

(e) BERT-Tiny (f) distilBERT

Fig. 5.2 Inference energy consumption as a function of CPU and GPU frequency under Jetson Nano; the color shade denotes the energy consumption, also shown on z-axis. Also highlighted is the point at which energy is minimized. The blue line represents the energy consumed under DVFS.



(a) Impact of batch size

(b) Impact of GPU frequency

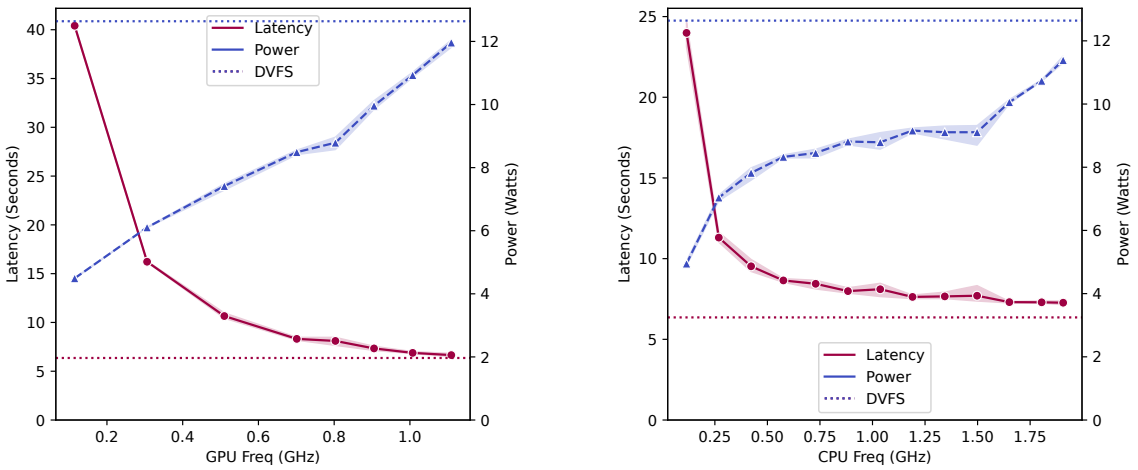
Fig. 5.3 Comparison of energy consumed for different batch sizes and GPU/CPU frequencies when running AlexNet inference on Jetson Nano.

GPU frequency; this is consistent with the higher z-axis values for lower GPU frequencies in Figure 5.2(a). Results for the impact of batch size on inference energy were qualitatively similar when we fixed a GPU frequency and varied the CPU frequencies; however, the energy difference between different CPU frequency curves was much smaller, consistent with the observation from Figure 5.2(a) that CPU frequency has a small impact on inference energy. Figure 5.3(b) presents the same results but with GPU frequency on the x-axis and distinct curves for different batch sizes. We see that a larger batch size reduces inference energy; visually, the energy curve for a larger batch size is "shifted" lower, without impacting the trend.

Takeaway 3. ***Increasing the batch size in AlexNet inference on the Jetson Nano significantly reduces energy consumption, as larger batch sizes decrease the number of mini-batches, thereby reducing GPU I/O and preprocessing activities.***

5.4 Frequency Scaling Sweep and DVFS for Xavier NX

Similar to our experiments for Nano, Figure 5.4(a) shows the time taken for inference (left y-axis) and power consumed (right y-axis) as a function of GPU frequency for Xavier NX using AlexNet inference with batch size of 16; here, we fix the CPU frequency at 1.0368 GHz. Inference time, or latency, is the time to complete the full workload of 3,200 inferences.



(a) Changing GPU frequency

(b) Changing CPU frequency

Fig. 5.4 Comparison of inference latency and power consumed under different CPU/GPU frequencies when running AlexNet on Xavier.

Again, we see that power consumption increases almost linearly with GPU frequency. The plateau effect for inference latency at higher GPU frequency values is observed for Xaviers as well [74]. The span of power consumed for Xavier NX is 9.8W which is higher than Nano, whereas the span of latency is 20s. *Inference tasks on the NVIDIA Jetson Xavier NX are significantly faster compared to the Jetson Nano, primarily due to the Xavier NX's support for higher power input and its more advanced architecture (higher CUDA and Tensor cores). The Xavier NX, designed with enhanced processing capabilities and a more robust power framework, can handle more complex computations at a quicker rate, which is consistent with our observations.*

Both power and latency vary greatly for Xavier as well, as GPU frequency changes from lowest to highest; the span of power consumed is 7.5W, whereas the span of latency is 20s.

Figure 5.4(b) shows the impact of changing CPU frequencies; here, we fix the GPU frequency at 0.80325 GHz. Similar to Nanos, the inference latency values tend to plateau faster at higher CPU frequencies, indicating hat ***CPU frequency has a smaller impact compared to GPU frequency for Xaviers as well.***

Takeaway 4. ***The NVIDIA Jetson Xavier NX demonstrates faster inference and higher power consumption than the Jetson Nano, with a significant difference in power span due to its more advanced architecture and higher power capacity, though both devices show similar trends in GPU frequency impacting power and latency.***

Similar to observation for Nanos, we see that DVFS affords low latency but incurs very high power consumption, suggesting that *DVFS opts for higher frequencies*. This was

confirmed by profiling, which revealed that DVFS operated at the highest CPU frequency 87% of the time and at the highest GPU frequency 79% of the time.

We also experimented with the other DVFS governors available (*e.g.*, 'powersave', 'performance', 'ondemand'). We found that 'powersave' consumes $<$1.3% less energy than the default governors. primarily because although the average power is reduced by 36%, the execution time of the workload is increased by 54%. Similar to Nanos, we found that all other DVFS governors perform similarly to the default governor, with the difference in energy consumption being within 2%.

5.5 Energy Topology under Xavier NX

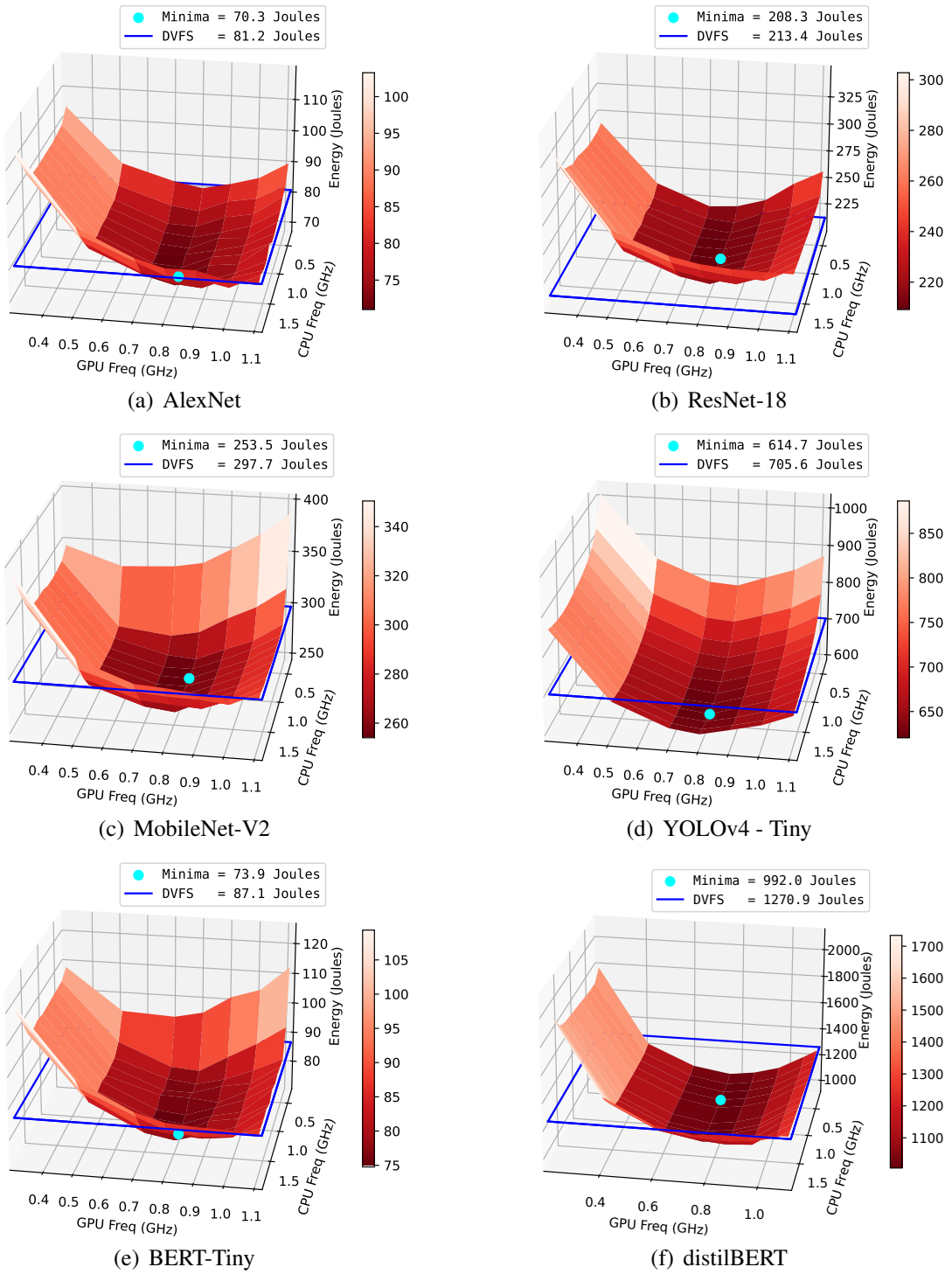
Figure 5.5 shows the familiar energy vs.CPU/GPU frequencies plot for all six workloads with batch size 16, but under Xavier NX. We see similar trends here as with the Nano, with the minima affording an energy reduction of 13.42%, 2.5%, 14.84%, 12.9%, 15.1%, and 21.95% for AlexNet, ResNet-18, MobileNet-V2, YOLOv4-Tiny, BERT-Tiny, and distilBERT respectively. The energy minima usually occurred at GPU frequency of 803.25MHz.

Comparing the energy topologies of Nano (Figure 5.2) and Xavier NX (Figure 5.5), we clearly see that the ***energy consumption of Xavier NX is significantly lower***, often by at least 2×, and sometimes as much 4×. Given the newer generation of Xavier NX, this is not wholly unexpected. We also observe a greater increase in energy (compared to the minima) at higher frequencies for Xavier NX, resulting in a steeper trough-like surface; by contrast, the energy values appear to plateau and only slightly increase at higher frequencies under Nano.

Takeaway 5. ***The non-monotonicity of the inference energy topology is even more apparent in Xaviers as compared to Nanos, thus demonstrating that the problem of finding an optimal CPU-GPU energy configuration is not trivial.***

5.6 Impact of Batch size on Xavier NX

Figure 5.6(a) shows the impact of batch size on energy consumption under Xavier NX for AlexNet for a fixed CPU frequency (of 1.0368GHz) and different GPU frequencies. We observe that energy drops almost linearly with batch size, with the energy consumed under the highest batch size (64) being 35–42% lower than that under the smallest batch size (8). Similar to Nanos, we see that a larger batch size reduces inference energy independent of

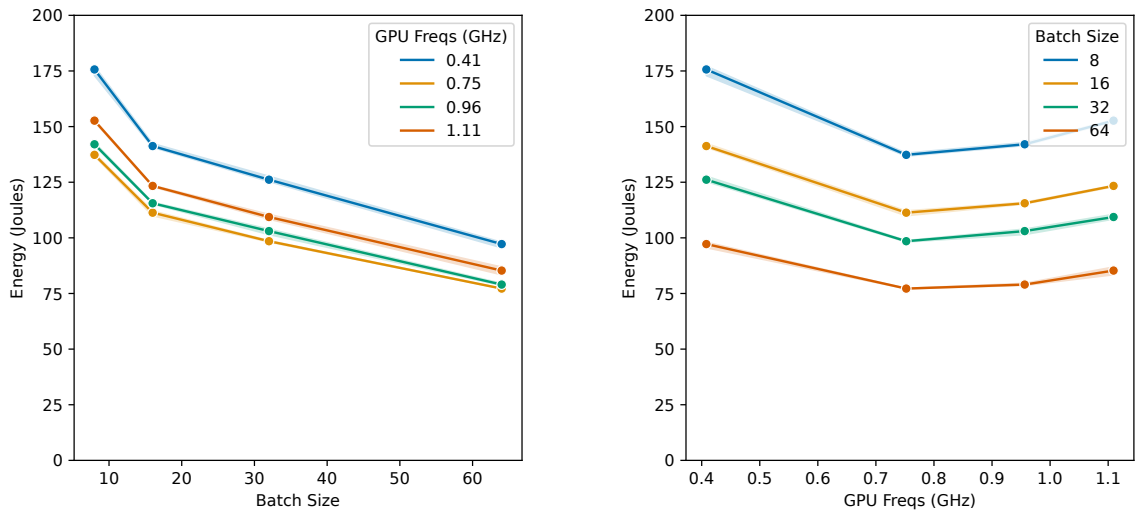


(a) AlexNet (b) ResNet-18

(c) MobileNet-V2 (d) YOLOv4 - Tiny

(e) BERT-Tiny (f) distilBERT

Fig. 5.5 Inference energy consumption as a function of CPU and GPU frequency under Xavier NX; the color shade denotes the energy consumption, also shown on z-axis. Also highlighted is the point at which energy is minimized. The blue line represents the energy consumed under DVFS.



(a) Impact of batch size

(b) Impact of GPU frequency

Fig. 5.6 Comparison of energy consumed for different batch sizes and GPU/CPU frequencies when running AlexNet inference on Xavier NX.

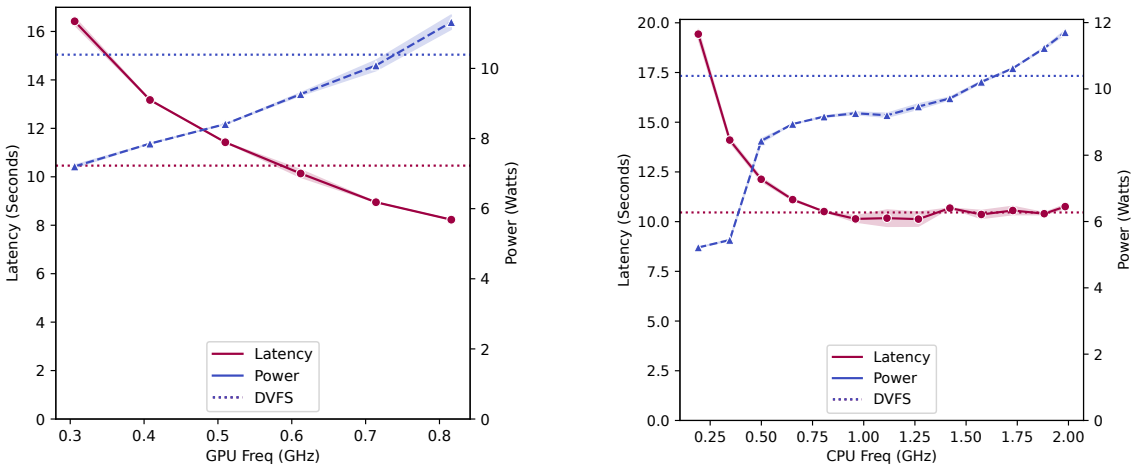
CPU and GPU frequency and thus joint optimization of execution frequencies and batch size is necessary to obtain greater energy savings.

Takeaway 6. ***Joint optimization of execution frequencies (CPU and GPU frequencies) and batch size may yield energy savings to the tune of 45% for both Jetson Nanos and Xaviers.***

5.7 Frequency Scaling Sweep and DVFS for Orins

Similar to our experiments for Nano and Xavier, Figure 5.7(a) shows the time taken for inference (left y-axis) and power consumed (right y-axis) as a function of GPU frequency for Orin NX using AlexNet inference with batch size of 16; here, we fix the CPU frequency at 0.96 GHz. Inference time, or latency, is the time to complete the full workload of 3,200 inferences.

Analogous to our previous results for Xaviers and Nanos, we see that power consumption increases almost linearly with GPU frequency. *However, the plateau effect for inference latency at higher GPU frequencies values, that we observed for Xaviers and Nanos, is not as apparent for Orins. This can be attributed to the fact that Orins (built on Ampere architecture) have substantially larger memory and more advanced I/O capabilities (4th-generation PCI-Express [52]) between system and GPU, as compared to Xaviers and Nanos and as such memory and I/O bottlenecks do not occur, even at very high GPU frequencies.*



(a) Changing GPU frequency

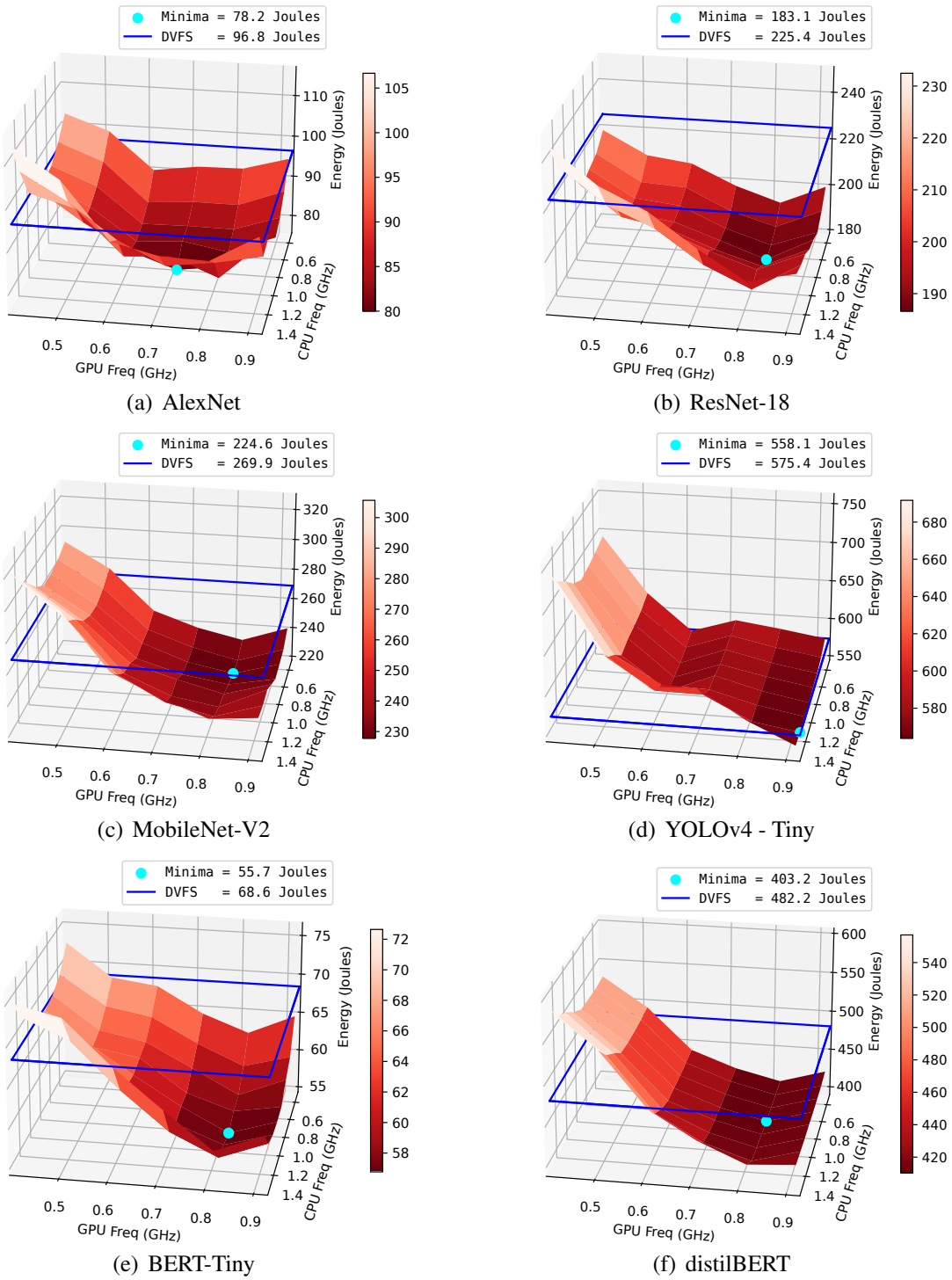
(b) Changing CPU frequency

Fig. 5.7 Comparison of inference latency and power consumed under different CPU/GPU frequencies when running AlexNet on Orin.

The span of power consumed for Orin NX is 5.1W which is lower than both Nano and Xavier, whereas the span of latency is 6s. *It is interesting to note that while both Xavier and Orins have similar operating power ranges, Orin is able to deliver much higher GPU throughput at lower operating GPU frequencies. This is primarily due to a plethora of architectural optimizations implemented in the Ampere GPUs which includes acceleration sparse matrix operations, better mixed-precision and double-precision FP support, increased size of shared memory, and ability to bypass L1 cache via asynchronous copy instructions [17].*

We do however see the plateau effect occur for the CPU frequency sweep. Figure 5.7(b) shows the impact of changing CPU frequencies; here, we fix the GPU frequency at 0.612 GHz. Similar to Nanos and Xaviers, the inference latency values tend to plateau faster at higher CPU frequencies, indicating hat ***CPU frequency has a smaller impact compared to GPU frequency for Xaviers as well.*** We are constrained by a fixed GPU frequency. This limits the GPU throughput, and given our workloads are large GPU-dependent, we are bottlenecked by the GPU even at very high CPU frequencies.

For our Orin experiments, we see that DVFS relatively affords low latency and incurs relatively high power consumption. This was confirmed by profiling, which revealed that DVFS operated at the highest CPU frequency only 56% of the time and at the highest GPU frequency 59% of the time. This slightly conservative behavior of DVFS in Orins can be attributed to Power-Capping and Thermal Throttling ([53, 4]), which are implemented in all Ampere architecture GPUs. Power capping is a feature that modifies existing DVFS



(a) AlexNet (b) ResNet-18

(c) MobileNet-V2 (d) YOLOv4 - Tiny

(e) BERT-Tiny (f) distilBERT

Fig. 5.8 Inference energy consumption as a function of CPU and GPU frequency under Orin NX; the color shade denotes the energy consumption, also shown on z-axis. Also highlighted is the point at which energy is minimized. The blue line represents the energy consumed under DVFS.

heuristics to set an upper limit on the power consumption of the GPU. It prevents the GPU from exceeding a predefined power budget. While explicit power budgets can be assigned to Ampere GPUs, the DVFS also has implicit power governance. Thermal Throttling is a mechanism that modifies default DVFS heuristics to automatically reduce the GPU's clock speed and voltage when it reaches a certain temperature threshold. The purpose of power throttling is to prevent overheating, which can lead to hardware damage or reduced performance over time. The Ampere GPUs monitor their thermal state and dynamically adjust their operating parameters to maintain safe temperatures.

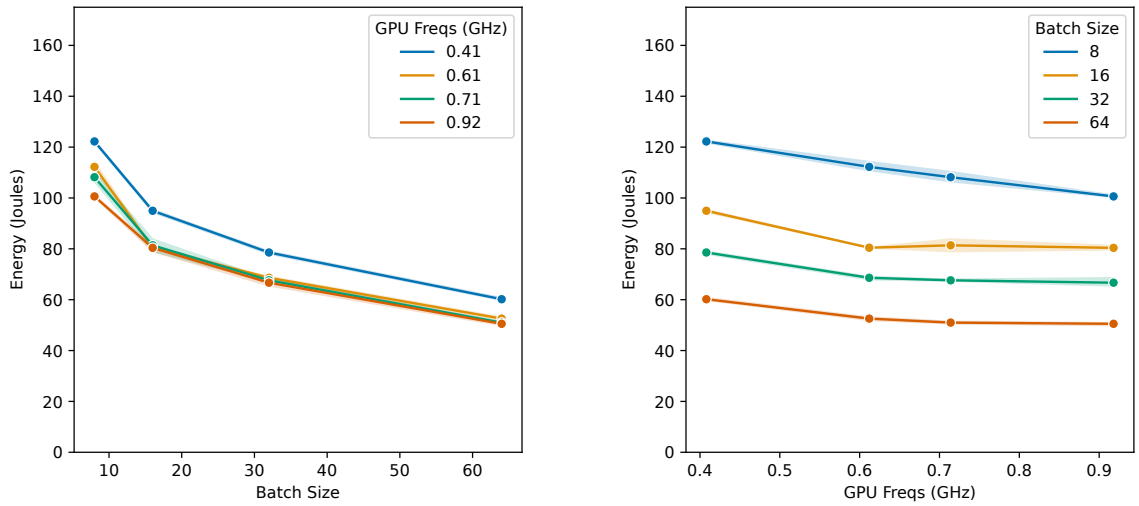
We also experimented with the other DVFS governors available (*e.g.*, 'powersave', 'performance', 'ondemand'). We found that 'powersave' consumes $<$1.5% more energy than the default governors, primarily because although the average power is reduced by 28%, the execution time of the workload is increased by 44%. Similar to Nanos, we found that all other DVFS governors perform similarly to the default governor, with the difference in energy consumption being within 2%.

Takeaway 7. ***The NVIDIA Orin NX, leveraging the Ampere architecture, shows a linear increase in power with GPU frequency and a less pronounced latency plateau at higher frequencies compared to Nano and Xavier, thanks to its larger memory and advanced I/O capabilities. While Orin's power span is lower than both Nano and Xavier, it achieves higher GPU throughput at lower frequencies due to architectural optimizations. Additionally, DVFS in Orin is more conservative due to power capping and thermal throttling compared to the default governors.***

5.8 Energy Topology under Orin NX

Figure 5.8 also shows the familiar energy vs.CPU/GPU frequencies plot for all six workloads with batch size 16, but under Orin NX. We see similar trends here with Orins as with the Nano and Xavier (thus omitting figures for other workloads), with the minima affording an energy reduction of 19.21%, 18.76%, 16.78%, 3.00%, 18.77%, and 16.38% for AlexNet, ResNet-18, MobileNet-V2, YOLOv4-Tiny, BERT-Tiny, and distilBERT respectively. The energy minima usually occurred at GPU frequency of 0.714 GHz.

Comparing the energy topologies of Nano and Xavier, with Orin (Figure 5.2) and Xavier NX (Figure 5.5, with Figure 5.8), we clearly see that the ***energy consumption of Orin NX is significantly lower than Nanos***, often by at least 2×, and sometimes as much 5×. However, Xavier NXs and Orin NXs have similar energy profiles for all 6 workloads. This similarity arises from a balance between power consumption and inference times for both the devices. The Xavier series, equipped with mid-tier hardware components, tends to consume



(a) Impact of batch size

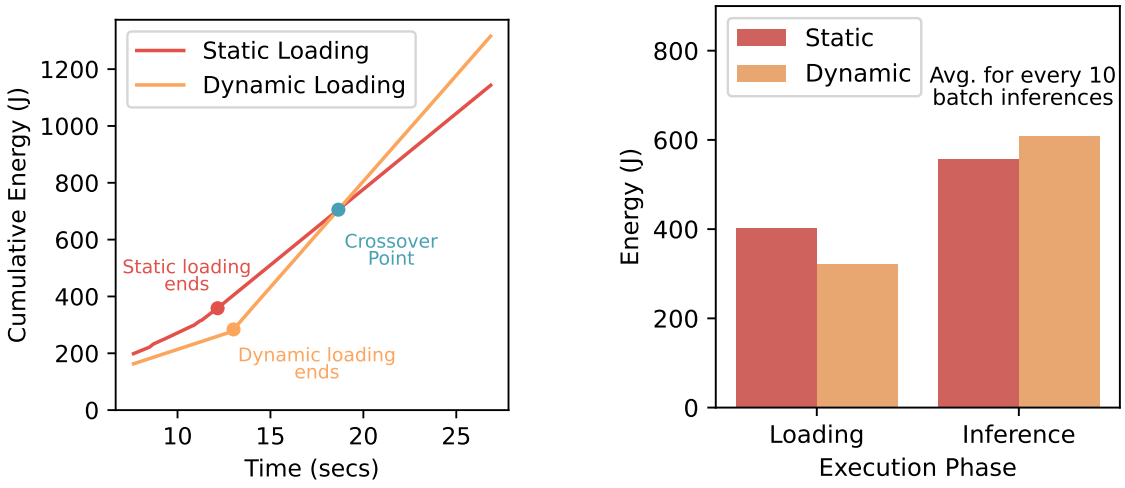
(b) Impact of GPU frequency

Fig. 5.9 Comparison of energy consumed for different batch sizes and GPU/CPU frequencies when running AlexNet inference on Orin NX.

slightly less power as compared to Orins. This lower power consumption is a result of its less demanding hardware which requires fewer resources to operate. However, this also translates to slightly higher inference times for deep learning tasks due to its relatively limited computational power compared to the more advanced Orin series.

On the other hand, the Orin series, built with larger and more powerful hardware components, consumes slightly more power. This increased power consumption is due to its advanced capabilities and resource-heavy hardware, which inherently require more energy to function. However, these robust components enable the Orin to achieve lower inference times, as it can process the same tasks slightly faster than the Xavier.

Takeaway 8. ***A key takeaway from this section is that optimizing DVFS for DNN workloads is challenging due to their computational complexity, varying demands across different layers, and the need to balance performance, energy efficiency, and thermal constraints within diverse hardware architectures. This complexity is compounded by non-linear performance gains and real-time adjustments required in response to dynamic workload and system. Despite having a thermal and power-aware DVFS system, Orins have very little energy gains as compared to Xavier NX.***



(a) Timeline of cumulative energy

(b) Energy usage of different phases

Fig. 5.10 Comparison of energy consumed with static and dynamic computational graph initialization for AlexNet inference under Jetson Nano.

5.9 Impact of Batch size on Orin NX

Figure 5.9(a) shows the impact of batch size on energy consumption under Orin NX for AlexNet for a fixed CPU frequency (of 0.96GHz) and different GPU frequencies. Similar to Nano and Xavier, We observe that energy drops almost linearly with batch size, with the energy consumed under the highest batch size (64) being 15–19% lower than that under the smallest batch size (8). Similar to Nanos and Xaviers, we see that a larger batch size reduces inference energy independent of CPU and GPU frequency and thus joint optimization of execution frequencies and batch size is necessary to obtain greater energy savings.

Takeaway 9. ***Joint optimization of execution frequencies (CPU and GPU frequencies) batch size may yield energy savings to the tune of 19% for both Orins.***

5.10 Impact of Computational Graph Initialization

PyTorch provides two ways for initializing the computational graph of a DNN model. Static Computational Graphs (SCG), also known as eager loading or preloading in TensorFlow, initialize and compile the full DNN graph, including configuring the weights and operations, *before* execution. Dynamic Computation Graphs (DCG) construct and modify the graph at runtime. Computation steps are executed immediately without explicitly defining a graph in advance.

To study the impact of static vs. dynamic graphs on energy consumption, we considered AlexNet inference with batch size 32 on Jetson Nano. Figure 5.10(a) shows the timeline of cumulative energy consumed under both graph initializations. While loading a static graph consumes ∼25% *higher* energy, it is more energy-efficient in the long run to use static graphs as the energy consumed per inference is lower (shown in Figure 5.10(b)). The timeline in Figure 5.10(a) is for 10 batches (of size 32) of inference; by the end of the plot, static graph had 8.5% *lower* energy consumption. We noticed the continued divergence in energy from Figure 5.10(a) for longer runs as well.

5.11 Impact of other factors

The implementation of the DNN model can also impact its inference energy. We perform a comparison of AlexNet implementation in PyTorch with our implementation of AlexNet using LibTorch in C++. We find that the ***PyTorch implementation consumes 16% more energy compared to LibTorch***, likely due to Python overheads. However, LibTorch is not commonly used during prototyping as it needs to be compiled every time a model parameter changes. LibTorch is more commonly used in high-performance systems [60, 73]. We also investigated other optimizations in our experiments such as turning off lazy-loading and garbage collection, and found a 1.82% and 1.65% reduction in total energy, respectively.

Chapter 6

Conclusion

This RPE report analyzes the impact of workload parameters and hardware knobs on the energy consumption of DNN inference on smart edge devices. The research provides an in-depth analysis of DNN inference on three edge devices, focusing on energy efficiency. Our literature survey shows that while there is extensive literature focused on the efficient deployment of Deep Neural Network (DNN) models on edge devices, emphasizing algorithmic and architectural optimizations, there is a notable scarcity of research specifically addressing energy optimization through the in-depth study of hardware and workload parameters. Few works delve into the intricacies of how different hardware settings, such as CPU/GPU frequencies, and workload characteristics, like batch sizes, directly impact the energy efficiency of DNN deployments on edge devices, highlighting a gap in the current research landscape.

We explore various factors affecting energy consumption during DNN inference, including batch sizes, GPU/CPU frequencies, and different computational graph initializations along with a few other system aspects which might affect the overall deployment of the workload. We perform this comparative analyses of energy consumption across three different hardware platforms (Nano, Xavier NX, Orin NX) under various hardware configurations and workloads. Our energy topology plots demonstrate that energy profiles are non-monotonic over the CPU/GPU frequency parameter space, thus showing that the problem of finding an optimal CPU-GPU energy configuration is not trivial. Our experimental results also show that significant improvements in energy consumption (as much as 22%) can be achieved over default DVFS settings by choosing the optimal frequencies by searching the parameter space.

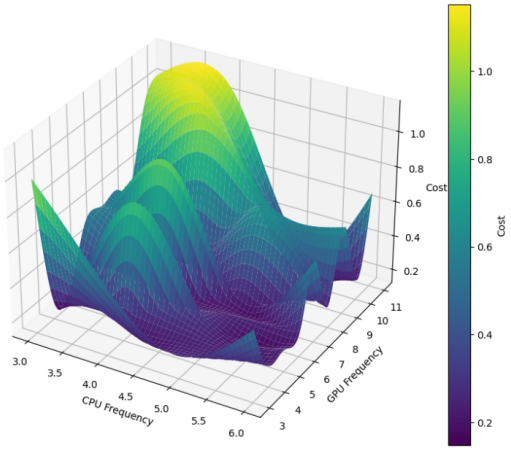
Our research illustrates that optimizing DNN workloads deployments by relying on DVFS to manage the hardware parameters is challenging due to the computational complexity of DL models, varying demands across different layers, and the need to balance performance, energy efficiency, and thermal constraints within diverse hardware architectures of the edge devices. This complexity is compounded by non-linear performance gains and real-time

adjustments required in response to dynamic workload and system. Our work on profiling energy topology of DVFS heuristics, even with advanced features such as power capping and thermal awareness (as in the case of Orins), leads to sub-optimal deployments for DNN inference. Our results reveal that all the DVFS governors perform similar to each other when it comes to effective deployment of DNN workloads, with the difference in energy consumption being within 2%. We also show that the 'powersave' governor, which is designed for efficient power management of the edge device, is unable to effectively optimize the energy consumption of DNN workloads. This is the case because, heuristically setting the scaling frequencies to lowest possible, merely increases execution time and inference latency, thus increasing overall energy consumption.

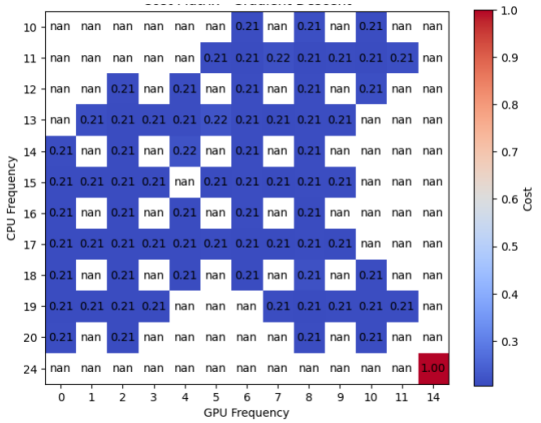
We also find that workload parameters (*e.g.*, the batch size) and model settings (*e.g.*, graph initialization strategies) can impact inference energy. We generally find that larger batches reduce overall execution energy as it decreases the number of mini-batches, thereby reducing GPU I/O and preprocessing activities. We also find joint optimization of execution frequencies (CPU and GPU frequencies) and batch size may yield energy savings to the tune of 45%. Our results also indicate that static computational graphs provide long-term energy gains when deployed for production inference frameworks. Lastly we assess the impact of other deployment factors such as programming platform, background processes, etc., and find that while background processes such as garbage collection and lazy loading do not play an important role in contributing to energy noise, programming language choice does affect overall energy of the DNN inference workload (comparing Python and C++).

6.1 Future Work

The continuous evolution in the field of deep neural network (DNN) inference on edge devices necessitates the development of more efficient methods for parameter space optimization. The advent of the latest generation of edge devices (specially Xaviers and Orins) is characterized by enhanced memory capacities and cutting-edge hardware. Thus these smart edge devices support large batch sizes and a wide range of CPU/GPU frequency. The substantially large parameter space in these smart edge devices, while beneficial in terms of flexibility and potential performance optimization, presents a challenge in searching and identifying the optimal deployment configuration for DNN inference. The vastness of this parameter space makes traditional, static optimization approaches both time-consuming and often impractical, especially in dynamic, real-world scenarios.



(a) Topology of linear cost function with adversarial weights for energy and time



(b) Sample Cost Matrix, the online descent algorithm has to solve starting from the highest CPU/GPU frequency configuration

Fig. 6.1 Description of the online descent problem for a linear cost function jointly optimizing Inference Energy and Inference Time

As part of future work, we will attempt a real-time optimization to traverse a surface as shown in Figure 6.1(a) which can be represented as follows:

$$
\begin{aligned}
\text{Minimize} \quad & f(x^{energy\_cost}_{cpu\_freq,gpu\_freq,batch\_size}, y^{time\_cost}_{cpu\_freq,gpu\_freq,batch\_size}) \\
\text{where} \quad & x_{cpu\_freq,gpu\_freq,batch\_size} = Inference\_Energy_{cpu\_freq,gpu\_freq,batch\_size} \\
& y_{cpu\_freq,gpu\_freq,batch\_size} = Inference\_Latency_{cpu\_freq,gpu\_freq,batch\_size} \\
& cpu\_freq \in Device_{all\_cpu\_frequencies} \\
& gpu\_freq \in Device_{all\_gpu\_frequencies} \\
& batch\_size \in Device_{all\_supported\_batch-sizes}
\end{aligned}
\tag{6.1}
$$

Our ongoing efforts are focused on exploring online descent algorithms that can traverse the CPU/GPU frequency and workload batch size parameter space more rapidly and accurately. This section delves into three promising algorithms: Full Parameter Space Sweep, Thomson Sampling, and Online Gradient Descent, discussing their potential applications and benefits in optimizing DNN inference.

6.1.1 Full Parameter Space Sweep

The Full Parameter Space Sweep algorithm represents a comprehensive approach. This method involves systematically exploring the entire parameter space, assessing each possible combination of CPU and GPU frequencies along with batch sizes.

Although this approach is exhaustive and potentially time-consuming, it ensures that no possible configuration is overlooked. The key advantage here is the guarantee of finding the optimal setting, albeit at the cost of higher computational overhead. To make this feasible in a real-time scenario, we plan to implement a more efficient search strategy, possibly leveraging heuristics that prioritize more promising regions of the parameter space based on preliminary evaluations.

6.1.2 Thompson Sampling

Thompson Sampling stands out as a probabilistic approach that balances exploration and exploitation. In the context of parameter space traversal, this method would involve creating a probabilistic model of the performance for different configurations and continuously updating this model as more data is gathered. This approach is particularly promising for adapting to changing conditions in real-time, as it inherently adjusts its exploration-exploitation balance based on the observed performance of different configurations [22].

6.1.3 Online Gradient Descent

Online Gradient Descent (OGD) offers a more dynamic approach, where the algorithm iteratively adjusts parameters in the direction that seems to be reducing a predefined loss function (such as energy consumption or latency). Unlike traditional gradient descent, OGD does this in an online manner, constantly updating the parameters based on new data. This approach is particularly suited for environments where the performance landscape is continuously changing. Implementing OGD will require developing a model that can effectively predict the performance impact of small changes in parameters, and robust mechanisms to ensure stability and convergence of the algorithm [80, 83, 30].

The motivation for employing online descent algorithms in managing the CPU/GPU frequency parameter space lies in their ability to dynamically and efficiently handle the complexities introduced by the advanced capabilities of the latest edge devices. This approach is vital for harnessing the full potential of these devices while ensuring energy-efficient and responsive operation in diverse applications. By implementing and comparing the efficacy of aforementioned algorithms, we aim to develop more adaptive, efficient, and intelligent

systems for managing the complex interplay of energy consumption, latency, and accuracy in real-time scenarios. This research could pave the way for more sustainable and effective deployment of AI technologies in energy-constrained environments, such as edge computing, mobile devices, and embedded systems.

References

[1] AHN, H., CHEN, T., ALNAASAN, N., A. SHAFI, M. A., SUBRAMONI, H., AND PANDA, D. Performance characterization of using quantization for dnn inference on edge devices. In *7TH IEEE INTERNATIONAL CONFERENCE ON FOG AND EDGE COMPUTING* (May 2023).

[2] AKIKI, S., YANG, Z., LIU, C., TANG, J., AND LIU, S. Energy-aware automatic tuning of many-core platform via gradient descent. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)* (2018), pp. 1199–1203.

[3] ALI, G., SIDE, M., BHALACHANDRA, S., WRIGHT, N. J., AND CHEN, Y. An automated and portable method for selecting an optimal gpu frequency. *Future Generation Computer Systems 149* (2023), 71–88.

[4] ALI, G., SIDE, M., BHALACHANDRA, S., WRIGHT, N. J., AND CHEN, Y. Performance-aware energy-efficient gpu frequency selection using dnn-based models. In *Proceedings of the 52nd International Conference on Parallel Processing* (New York, NY, USA, 2023), ICPP '23, Association for Computing Machinery, p. 433–442.

[5] ANANTHANARAYANAN, G., SHU, Y., COX, L., AND BAHL, V. Project rocket platform—designed for easy, customizable live video analytics—is open source. Microsoft Research Blog, January 2020.

[6] BALLER, S., JINDAL, A., CHADHA, M., AND GERNDT, M. Deepedgebench: Benchmarking deep neural networks on edge devices. In *2021 IEEE International Conference on Cloud Engineering (IC2E)* (Los Alamitos, CA, USA, oct 2021), IEEE Computer Society, pp. 20–30.

[7] BARROSO, L. A., AND HÖLZLE, U. The Case for Energy-Proportional Computing. *IEEE Computer 40*, 12 (2007), 33–37.

[8] BASMADJIAN, R., AND DE MEER, H. Modelling and analysing conservative governor of dvfs-enabled processors. In *Proceedings of the Ninth International Conference on Future Energy Systems* (2018), pp. 519–525.

[9] BASMADJIAN, R., NIEDERMEIER, F., AND DE MEER, H. Modelling performance and power consumption of utilisation-based dvfs using m/m/1 queues. In *Modelling Performance and Power Consumption of Utilisation-Based DVFS Using M/M/1 Queues* (New York, NY, USA, 2016), e-Energy '16, Association for Computing Machinery.

[10] BOROUMAND, A., GHOSE, S., AKIN, B., NARAYANASWAMI, R., OLIVEIRA, G. F., MA, X., SHIU, E., AND MUTLU, O. Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)* (2021), pp. 159–172.

[11] CASIAN, O. Improving performance of threads in python using c/c++ extensions. *Tehnica UTM* (2012).

[12] COSTA, G. D., AND PIERSON, J.-M. Dvfs governor for hpc: Higher, faster, greener. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (2015), pp. 533–540.

[13] DENG, Q., MEISNER, D., BHATTACHARJEE, A., WENISCH, T. F., AND BIANCHINI, R. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture* (Vancouver, British Columbia, Canada, 2012), MICRO '12, pp. 143–154.

[14] DESISLAVOV, R., MARTÍNEZ-PLUMED, F., AND HERNÁNDEZ-ORALLO, J. Compute and energy consumption trends in deep learning inference. *CoRR abs/2109.05472* (2021).

[15] DUTT, A., RACHURI, S. P., LOBO, A., SHAIK, N., GANDHI, A., AND LIU, Z. Evaluating the energy impact of device parameters for dnn inference on edge. *Power 3*, 4 (2023), 5.

[16] FENG, H., MU, G., ZHONG, S., ZHANG, P., AND YUAN, T. Benchmark analysis of yolo performance on edge intelligence devices. *Cryptography 6*, 2 (2022).

[17] FOSTER, B., TANEJA, S., MANZANO, J., AND BARKER, K. Evaluating energy efficiency of gpus using machine learning benchmarks. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2023), pp. 42–50.

[18] FOUNDATION, L. Sharpening the edge: Overview of the lf edge taxonomy and framework, Jul 2020.

[19] GALEONE, P. *Hands-on neural networks with TensorFlow 2.0: understand TensorFlow, from static graph to eager execution, and design neural networks*. Packt Publishing Ltd, 2019.

[20] GANDHI, A., HARCHOL-BALTER, M., DAS, R., AND LEFURGY, C. Optimal power allocation in server farms. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems* (Seattle, WA, USA, 2009), SIGMETRICS '09, pp. 157–168.

[21] GANDHI, A., HARCHOL-BALTER, M., AND KOZUCH, M. The case for sleep states in servers. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems* (Cascais, Portugal, 2011), HotPower '11.

[22] GHALME, G., JAIN, S., GUJAR, S., AND NARAHARI, Y. Thompson sampling based mechanisms for stochastic multi-armed bandit problems. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (2017), pp. 87–95.

[23] GHASEMI, M., HEIDARI, S., KIM, Y. G., LAMB, A., WU, C.-J., AND VRUDHULA, S. Energy-efficient mapping for a network of dnn models at the edge. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)* (2021), pp. 25–30.

[24] GONDI, S., AND PRATAP, V. Performance evaluation of offline speech recognition on edge devices. *Electronics 10*, 21 (2021).

[25] GURUSHANTHAPPA, P. B. *Performance Analysis of Artificial Intelligence Workloads*. PhD thesis, Texas A&M University, 2019.

[26] HADIDI, R., CAO, J., XIE, Y., ASGARI, B., KRISHNA, T., AND KIM, H. Characterizing the deployment of deep neural networks on commercial edge devices. In *2019 IEEE International Symposium on Workload Characterization (IISWC)* (2019), pp. 35–48.

[27] HAN, R., ZHANG, Q., LIU, C. H., WANG, G., TANG, J., AND CHEN, L. Y. Legodnn: Block-grained scaling of deep neural networks for mobile vision. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking* (New York, NY, USA, 2021), MobiCom '21, Association for Computing Machinery, p. 406–419.

[28] HANAFY, W. A., MOLOM-OCHIR, T., AND SHENOY, R. Design considerations for energy-efficient inference on edge devices. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems* (New York, NY, USA, 2021), e-Energy '21, Association for Computing Machinery, p. 302–308.

[29] HAO, J., SUBEDI, P., KIM, I. K., AND RAMASWAMY, L. Characterizing resource heterogeneity in edge devices for deep learning inferences. In *Proceedings of the 2021 on Systems and Network Telemetry and Analytics* (New York, NY, USA, 2021), SNTA '21, Association for Computing Machinery, p. 21–24.

[30] HAZAN, E., RAKHLIN, A., AND BARTLETT, P. Adaptive online gradient descent. *Advances in neural information processing systems 20* (2007).

[31] HE, F., HU, X., LIU, S., LI, T., ZHU, K., BAO, X., AND JIANG, C. Algorithm for improving processor utilization in multi-core processor environment by python language. In *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (2021), vol. 4, IEEE, pp. 775–779.

[32] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition, 2015.

[33] HOLLY, S., WENDT, A., AND LECHNER, M. Profiling Energy Consumption of Deep Neural Networks on NVIDIA Jetson Nano. In *Proceedings of the 11th International Green and Sustainable Computing Workshops (IGSC)* (2020), pp. 1–6.

[34] HONG, Z., AND YUE, C. P. Efficient-grad: Efficient training deep convolutional neural networks on edge devices with gradient optimizations. *ACM Trans. Embed. Comput. Syst. 21*, 2 (feb 2022).

[35] JEONG, E., KIM, J., AND HA, S. Tensorrt-based framework and optimization methodology for deep learning inference on jetson boards. *ACM Trans. Embed. Comput. Syst. 21*, 5 (oct 2022).

[36] JIANG, Z., ZHAO, L., LI, S., AND JIA, Y. Real-time object detection method based on improved yolov4-tiny, 2020.

[37] KIM, S., AND DEKA, G. C. Chapter eight - energy-efficient deep learning inference on edge devices. In *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, vol. 122 of *Advances in Computers*. Elsevier, 2021, pp. 247–301.

[38] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012), F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc.

[39] KU, Y.-J., AND DEY, S. Sustainable vehicular edge computing using local and solar-powered roadside unit resources. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)* (2019), pp. 1–7.

[40] KUMAR, H., CHAWLA, N., AND MUKHOPADHYAY, S. Biasp: A dvfs based exploit to undermine resource allocation fairness in linux platforms. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design* (New York, NY, USA, 2020), ISLPED '20, Association for Computing Machinery, p. 223–228.

[41] LEE, S., AND NIRJON, S. Subflow: A dynamic induced-subgraph strategy toward real-time dnn inference and training. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2020), pp. 15–29.

[42] LI, B., QIN, L., ZHAO, F., LIU, H., YU, J., HE, M., WANG, J., AND LIU, K. Research on edge detection model of insulators and defects based on improved yolov4-tiny. *Machines 11*, 1 (2023).

[43] LI, J., ZHANG, C., CAO, Q., QI, C., HUANG, J., AND XIE, C. An experimental study on deep learning based on different hardware configurations. In *2017 International Conference on Networking, Architecture, and Storage (NAS)* (2017), IEEE, pp. 1–6.

[44] LOOKS, M., HERRESHOFF, M., HUTCHINS, D., AND NORVIG, P. Deep learning with dynamic computation graphs, 2017.

[45] LOU, W., XUN, L., SABET, A., BI, J., HARE, J., AND MERRETT, G. V. Dynamic-ofa: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (June 2021), pp. 3110–3118.

[46] MCLEOD, C. A framework for distributed deep learning layer design in python. *arXiv preprint arXiv:1510.07303* (2015).

[47] MENDES, F., TOMÁS, P., AND ROMA, N. Decoupling gpgpu voltage-frequency scaling for deep-learning applications. *Journal of Parallel and Distributed Computing 165* (2022), 32–51.

[48] MITTAL, S. A survey on optimized implementation of deep learning models on the nvidia jetson platform. *Journal of Systems Architecture 97* (2019), 428–442.

[49] MOLOM-OCHIR, T., AND SHENOY, R. Energy and cost considerations for gpu accelerated ai inference workloads. In *2021 IEEE MIT Undergraduate Research Technology Conference (URTC)* (2021), pp. 1–5.

[50] NABAVINEJAD, S. M., REDA, S., AND EBRAHIMI, M. Batchsizer: Power-performance trade-off for dnn inference. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference* (New York, NY, USA, 2021), ASPDAC '21, Association for Computing Machinery, p. 819–824.

[51] NVIDIA. MLPerf Benchmarks. https://www.nvidia.com/en-us/data-center/resources/mlperf-benchmarks.

[52] NVIDIA. NVIDIA Ampere Architecture. https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/.

[53] NVIDIA. NVIDIA Ampere Architecture. https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf.

[54] NVIDIA. NVIDIA Jetson Clocks. https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-325/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/clocks.html.

[55] NVIDIA. NVIDIA Jetson Nano System-on-Module. https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs.

[56] NVIDIA. NVIDIA Jetson Xavier NX System-on-Module. https://en.miivii.com/uploads/file/20200511/NV_Jetson_Xavier_NX_DataSheet_v0.1.pdf.

[57] NVIDIA. NVIDIA Orin NX System-on-Module. https://files.seeedstudio.com/wiki/reComputer/module/jetson-orin-nx-datasheet.pdf.

[58] PANOPOULOS, I., NIKOLAIDIS, S., VENIERIS, S. I., AND VENIERIS, I. S. Exploring the performance and efficiency of transformer models for nlp on mobile devices, 2023.

[59] PARK, J., ARYAL, P., MANDUMULA, S. R., AND ASOLKAR, R. P. An optimized dnn model for real-time inferencing on an embedded device. *Sensors 23*, 8 (2023).

[60] PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., DESMAISON, A., KOPF, A., YANG, E., DEVITO, Z., RAISON, M., TEJANI, A., CHILAMKURTHY, S., STEINER, B., FANG, L., BAI, J., AND CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (2019), H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc.

[61] POELLABAUER, C., SINGLETON, L., AND SCHWAN, K. Feedback-based dynamic voltage and frequency scaling for memory-bound real-time applications. In *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium* (2005), pp. 234–243.

[62] RACHURI, S. P., BRONZINO, F., AND JAIN, S. Decentralized modular architecture for live video analytics at the edge. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges* (New York, NY, USA, 2021), HotEdgeVideo '21, Association for Computing Machinery, p. 13–18.

[63] RAUBER, J., BETHGE, M., AND BRENDEL, W. Eagerpy: Writing code that works natively with pytorch, tensorflow, jax, and numpy, 2020.

[64] REN, W., QU, Y., DONG, C., JING, Y., SUN, H., WU, Q., AND GUO, S. A survey on collaborative dnn inference for edge intelligence, 2022.

[65] SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A., AND CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[66] SANH, V., DEBUT, L., CHAUMOND, J., AND WOLF, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[67] SHIN, P., KIM, D., AND HONG, S. Memory-aware dvfs governing policy for improved energy-saving in the linux kernel. In *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)* (2023), IEEE, pp. 57–66.

[68] S.K, P., KESANAPALLI, S. A., AND SIMMHAN, Y. Characterizing the performance of accelerated jetson edge devices for training deep learning models. *Proc. ACM Meas. Anal. Comput. Syst. 6*, 3 (dec 2022).

[69] SÜZEN, A. A., DUMAN, B., AND ŞEN, B. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)* (2020), pp. 1–5.

[70] TANG, Z., QI, L., CHENG, Z., LI, K., KHAN, S. U., AND LI, K. An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. *J. Grid Comput. 14*, 1 (Mar. 2016), 55–74.

[71] TURC, I., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Well-read students learn better: On the importance of pre-training compact models, 2019.

[72] ULLAH, S., AND KIM, D.-H. Benchmarking jetson platform for 3d point-cloud and hyper-spectral image classification. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)* (2020), pp. 477–482.

[73] VANHOY, G., LICHTMAN, M., HOARE, R. R., AND BREVIK, C. Rapid prototyping framework for intelligent arrays with heterogeneous computing. In *2022 IEEE International Symposium on Phased Array Systems & Technology (PAST)* (2022), pp. 01–06.

[74] WANG, Q., AND CHU, X. Gpgpu performance estimation with core and memory frequency scaling. *IEEE Transactions on Parallel and Distributed Systems 31*, 12 (2020), 2865–2881.

[75] WANG, Y., QIN, Y., DENG, D., WEI, J., CHEN, T., LIN, X., LIU, L., WEI, S., AND YIN, S. Trainer: An energy-efficient edge-device training processor supporting dynamic weight pruning. *IEEE Journal of Solid-State Circuits 57*, 10 (2022), 3164–3178.

[76] WU, C.-M., CHANG, R.-S., AND CHAN, H.-Y. A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters. *Future Generation Computer Systems 37* (2014), 141–147. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.

[77] XU, Z., ZHAO, L., LIANG, W., RANA, O. F., ZHOU, P., XIA, Q., XU, W., AND WU, G. Energy-aware inference offloading for dnn-driven applications in mobile edge clouds. *IEEE Transactions on Parallel and Distributed Systems 32*, 4 (2021), 799–814.

[78] XUN, L., TRAN-THANH, L., AL-HASHIMI, B. M., AND MERRETT, G. V. Incremental training and group convolution pruning for runtime dnn performance scaling on heterogeneous embedded platforms. In *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)* (2019), pp. 1–6.

[79] YI, R., CAO, T., ZHOU, A., MA, X., WANG, S., AND XU, M. Boosting dnn cold inference on edge devices. In *Boosting DNN Cold Inference on Edge Devices* (New York, NY, USA, 2023), MobiSys '23, Association for Computing Machinery, p. 516–529.

[80] YING, Y., AND PONTIL, M. Online gradient descent learning algorithms. *Foundations of Computational Mathematics 8* (2008), 561–596.

[81] YOU, J., CHUNG, J.-W., AND CHOWDHURY, M. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (Boston, MA, Apr. 2023), USENIX Association, pp. 119–139.

[82] ZHAO, F., WANG, X., LIN, P., AND CHEN, Y. Comprehensive analysis of the heterogeneous computing performance of dnns under typical frameworks on cloud and edge computing platforms. *Expert Systems with Applications 229* (2023), 120475.

[83] ZHAO, P., WANG, J., WU, P., JIN, R., AND HOI, S. C. H. Fast bounded online gradient descent algorithms for scalable kernel-based online learning. *CoRR abs/1206.4633* (2012).