Exploiting the Tradeoff Between Energy Consumption and Execution Time in Mixed CPU-GPU Edge-Cloud Environment

By

Ravi Rathee

Under the supervision of

Dr. Arani Bhattacharya

Indraprastha Institute of Information Technology Delhi

June, 2022

Exploiting the Tradeoff Between Energy Consumption and Execution Time in Mixed CPU-GPU Edge-Cloud Environment

By

Ravi Rathee

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology

To

Indraprastha Institute of Information Technology Delhi

June, 2022

Certificate

This is to certify that the thesis titled *Exploiting the Tradeoff Between Energy Consumption and Execution Time in Mixed CPU-GPU Edge-Cloud Environment* being submitted by *Ravi Rathee* to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

June, 2022

Dr. Arani Bhattacharya

Indraprastha Institute of Information Technology Delhi

New Delhi 110020

Everything can be taken from a man but the last of the human freedoms—to choose one's attitude in any given set of circumstances, to choose one's own way.

Viktor E. Frankl

The world will ask you who you are, and if you don't know, the world will tell you.

Carl Jung

Meaning is only found in the journey uphill, and never in the fleeting sense of satisfaction awaiting at the next peak.

Unknown

Acknowledgements

I would like to sincerely thank my advisor, Dr. Arani Bhattacharya, whose constant guidance, availability, and support have enabled me to navigate every obstacle encountered in the course of this project. I have learned and grown immensely through their constructive criticism and feedback on my work.

I am also grateful for my time at IIIT Delhi, which has proved to be a turning point in my life and has provided me with the opportunity to learn from exceptionally knowledgeable professors to better myself at my craft. Every lecture has been a privilege.

Abstract

Mobile devices like smartphones can augment their low-power processors by offloading GPU-heavy applications to cloud servers. However, cloud data centers consume a lot of energy and have high latency. To mitigate the problem of high latency by data centers, recently offloading to edge devices that are in proximity to the users has become popular. Such edge devices usually have much lower compute capacity, as they need to be more widely distributed than data centers. However, the recent rise of machine learning-based workloads make it a challenge to model such execution. To resolve this challenge, we benchmark a few widely used machine learning programs on a representative edge device and a server-grade GPU. Our benchmarks show that in addition to saving network latency, edge devices also consume less energy than server-grade GPU's, albeit at the cost of higher execution time.

Based on the above trade-off between execution time, network latency and computation time, we look at the problem of scheduling a job with a sequence of machine learning workloads. We formulate this scheduling as an integer linear programming problem, where the objective is to minimize the energy consumption while ensuring that the maximum number of jobs finish within a specific deadline. We use ILP-solvers to identify the optimal solution by using Google OR-tools and demonstrate on a few examples that our technique works well in practice.

Contents

Certificate

Acknowledgements **i**

Abstract **ii**

List of Figures **v**

List of Tables **vi**

1 Introduction **1**

2 Related Work **7**

2.1 Literature Review 7

2.2 Background 11

2.2.1 Jetson Nano-4GB 11

2.2.2 NVIDIA GeForce RTX 2080 Ti 11

2.2.3 CUDA 12

3 GPU Benchmarking **13**

3.1 Need of GPU Benchmarking 13

3.2 Technical Specifications of Hardware used in my experiments 13

3.2.1 Cloud-Server 13

3.2.2 Personal Computer 14

3.2.3 Jetson-nano 14

3.3 Description of each Bench-marking service/algorithm 14

3.3.1 YOLOV5 14
3.3.2 Fast Fourier transform (FFT) 15
3.3.3 Real-time voice clone 15
3.3.4 Real-time video transcoding 15
3.3.5 Image Classification (only on jetson nano) 16
3.3.6 RL-based industrial controller (using cpu)(only on jetson-nano) 16
3.4 Preparation for bench-marking : For each bench-marking listed below, our platforms are prepared for the benchmarking 17
3.5 Assumptions : For each bench-marking listed below, we have made some assumptions 17
3.6 Methodology of GPU Benchmarking 18
3.7 Results of Benchmarking 20
3.7.1 Results of Stress Testing on server 23

4 Task Scheduling Problem **24**
4.0.1 Preliminaries 24
4.0.2 Mathematical Model 25

5 Methodology **27**
5.1 Definitions and Tools 27
5.1.1 Google OR-Tools 27
5.1.2 Linear Optimization 28
5.2 Model Preparation 29
5.3 Code Preparation from our Formulated Model 30
5.4 Summary 36
5.5 Additional Optimization 37
5.6 Observations from The Three Cases 38

6 Conclusion and Future Work **39**
6.1 Future Work 40

Bibliography **42**

List of Figures

1.1 Energy consumption of servers and data centers worldwide -forecasts to 2030 A comparison of the various forecasts shows that the possible future development of the energy consumption of data centers has a wide range. In the "best case" the energy consumption of data centers can remain largely constant. If Moore's Law ends and the performance of the data centers increases significantly, annual energy consumption may increase to almost 3,000 billion kWh/a 2

1.3 Fraction of U.S. data center electricity use in 2014, by end use. Source : Shehabi 2016 3

1.9 A hypothetical example of yolov5 running on CPU of mobile device vs GPU of mobile device, shows how gpu consumes more energy than cpu and performs gpu specific tasks faster 5

1.17 A hypothetical example of yolov5, offloading its gpu specifics functions to server and jetson nano, shows how jetson nano uses less energy compared to server but more time, we will quantify these with benchmarkings in the next chapter . . . 6

3.1 Time comparison between different platforms for each service tested 21

3.2 Power comparison between different platforms for each service tested 21

3.3 Energy comparison between different platforms for each service tested 22

3.4 Power Consumption vs Time(10 second internal), After every 10 seconds . . . 23

5.1 Two Examples for Case 1 32

5.4 Example 1 and Example 2 for Case 2 33

5.7 Example 3 and Example 4 for Case 2 33

5.10 Example 1 and Example 2 for Case 3 35

5.13 Increase in Power w.r.t. Services Run on model 36

5.16 Services Able to Run vs Power Consumption by model 37

List of Tables

1.1 An example of few services when run on Server and Edge-device(in this case jetson-nano), since most of the services are gpu focussed we are using jetson-nano, but other edge-devices which only contain cpu can also be connected . . 4

2.1 Jetson Nano Developer Kit technical specifications. 11

2.2 NVIDIA GeForce RTX 2080 Ti 12

3.1 An example of few services when run on Server and Edge-device(in this case jetson-nano), since most of the services are gpu focussed we are using jetson-nano, but other edge-devices which only contain cpu and gpu can also be connected 20

5.1 An example of few services when run on Server and Edge-device(in this case jetson-nano), since most of the services are gpu focussed we are using jetson-nano, but other edge-devices which only contain cpu can also be connected . . 30

Chapter 1

Introduction

The demand for compute and data services has given rise to cloud computing, where user devices with low compute capability can request additional services. These services are provided by servers typically located in data centers. Thus, even small IoT devices have access to large compute facilities and intelligence possible from availability of large dataset.

While cloud computing has enabled the rise of IoT, it suffers from two major limitations. The first limitation, which has been widely discussed in prior studies [], is the fact that access to the data centers come with high latency.

An interesting question in computation offloading is to decide which machines to use as server while conserving energy. On average, data center capacity of globe is exponentially increasing, along with the increase of usage of servers, power consumption is increasing exponentially as well.[1] Along with that, on average, servers and cooling systems account for the greatest shares of direct electricity use in data centers, as shown in figure 2, followed by storage drives and network devices. Energy consumption of data centres is very high, that is why a lot of effort goes into inventing cooling systems for servers. This energy consumption increases even more when we use GPU driven computations.

Most offloading framework prototypes developed so far use an in-house desktop or server machine. Currently, offloading frameworks have two distinct choices of machines that can be

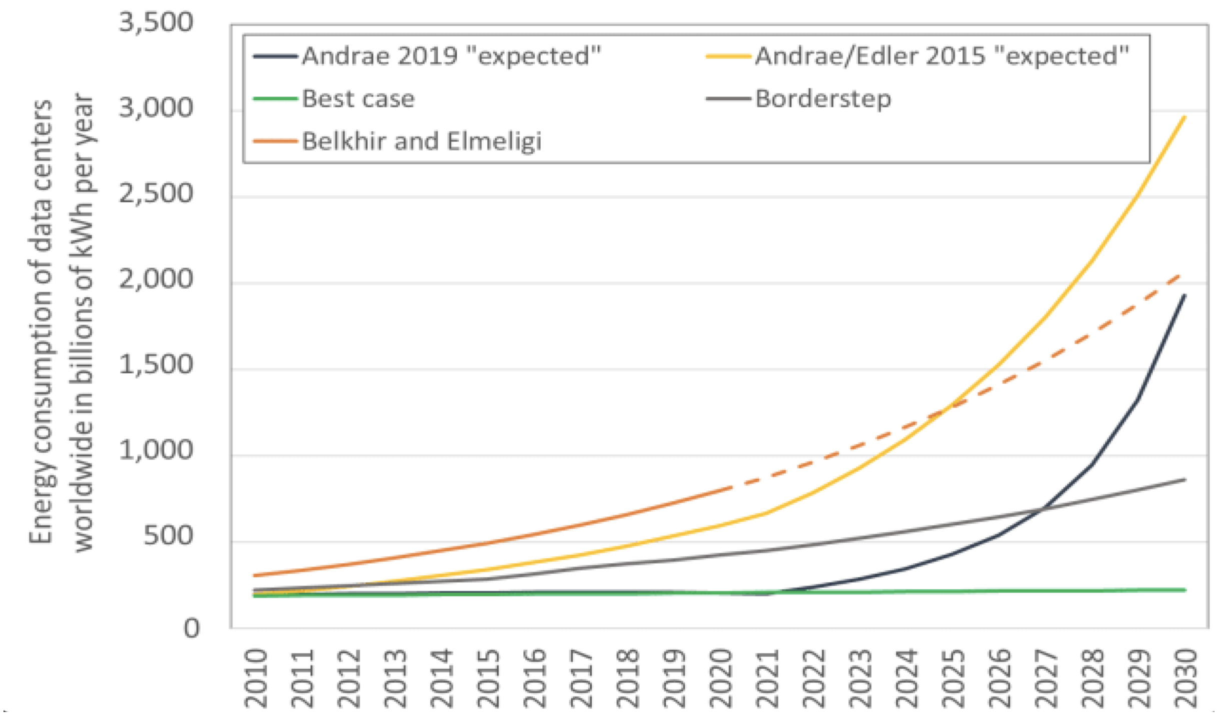


Figure 1.1: Energy consumption of servers and data centers worldwide -forecasts to 2030 A comparison of the various forecasts shows that the possible future development of the energy consumption of data centers has a wide range. In the "best case" the energy consumption of data centers can remain largely constant. If Moore's Law ends and the performance of the data centers increases significantly, annual energy consumption may increase to almost 3,000 billion kWh/a

used as server. The first choice involves offloading to commercially available cloud servers and is known as mobile cloud computing. Prototype implementations of offloading utilize this technique. The second choice, known as mobile edge computing, involves offloading to other user edge devices such as tablets, laptops, network routers and small gpu-computing devices like jetson-nano.

Cloud servers and user edge devices have two major differences:

- Processors of cloud servers have faster processors. For example, running the benchmark CoreMark [3] shows that a Google Cloud Platform [4] processor is around 6 times faster than a mobile device and on average 1.2 times faster as per our benchmarks, as shown in figure 3.
- Power Consumption of cloud servers is higher than edge devices. By running a few test

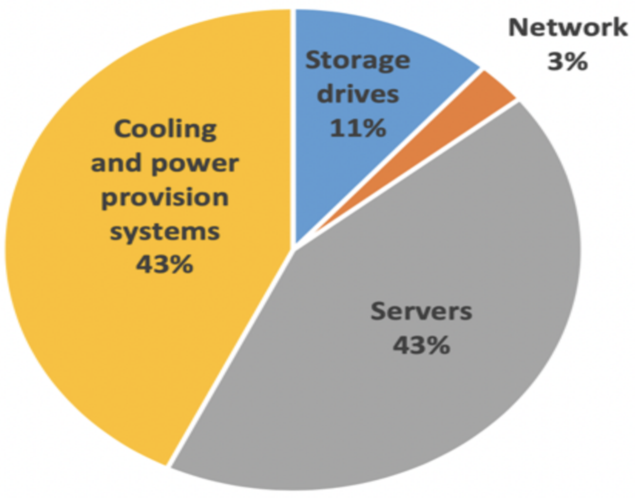


Figure 1.3: Fraction of U.S. data center electricity use in 2014, by end use. Source : Shehabi 2016

examples we have found that power consumption of could server is 20-30% more compared to mobile device like jetson-nano.

One of the most important factors of profitability of data centres is less power consumption. So, computer scientists are always searching for efficient ways to run programs on cloud in such a way that we don't compromise on time yet yield efficient power consumption results.

In 2019, Andrae and Edler published a study, [1] which showed rise in energy consumption by servers. It also predicts from given data points, what energy consumption of data servers in future may look like, as shown in figure 1.1, there is exponential rise in electricity use after 2020, and this trend only continues further, worsening the total power usage of this industry.

We performed a set of bench-markings on a server and jetson-nano, running various GPU specific tests. For each service/test we ran, we measured its time(in seconds) and power consumption(in Watts), on both server and jetson-nano, as shown in figure 3.

From our bench-marking results, we have found that on average server performs 22% faster than edge-devices(here jetson-nano), while consuming 19% more energy than edge-devices(here

jetson-nano).
Thus, a scheduling algorithm has to balance the trade-off between more execution of tasks on the edge-devices to decrease latency and ensuring less energy consumed by servers.

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$7.93s/49.75W$	$28.2s/2.9W$
Real Time Video Transcoding	$6s/82W$	$10.22s/51W$	$30.22s/3.21W$
$OnAverage$	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Table 1.1: An example of few services when run on Server and Edge-device(in this case jetson-nano), since most of the services are gpu focussed we are using jetson-nano, but other edge-devices which only contain cpu can also be connected

Table 1.1, illustrates an example of time and power usage of a yolov5 program, run on different platforms (server and jetson-nano).
When yolov5 runs entirely on Mobile device, it takes 1000s and consumes 50W of power. When it runs entirely on GPU of mobile device it takes 400s (comparatively less time) and 100W (comparatively more power). This is because a mobile-device is a fairly low-powered device, mostly runs on battery power, so it uses less energy hence the longer computing times. But we can offload some of the GPU-specific load to a server.
Figure 6 a), illustrates offloading of a GPU specific program on server. When the program is scheduled on server, it takes 100s (very less compared to mobile device) but 400W (significantly more power than mobile device). This shows how computational offloading reduces execution time of our program drastically.
Figure 6 b), illustrates offloading of a GPU specific program on jetson-nano. When the program is scheduled on jetson-nano, it takes 150s (more time than server) but only 100W (less energy than server).

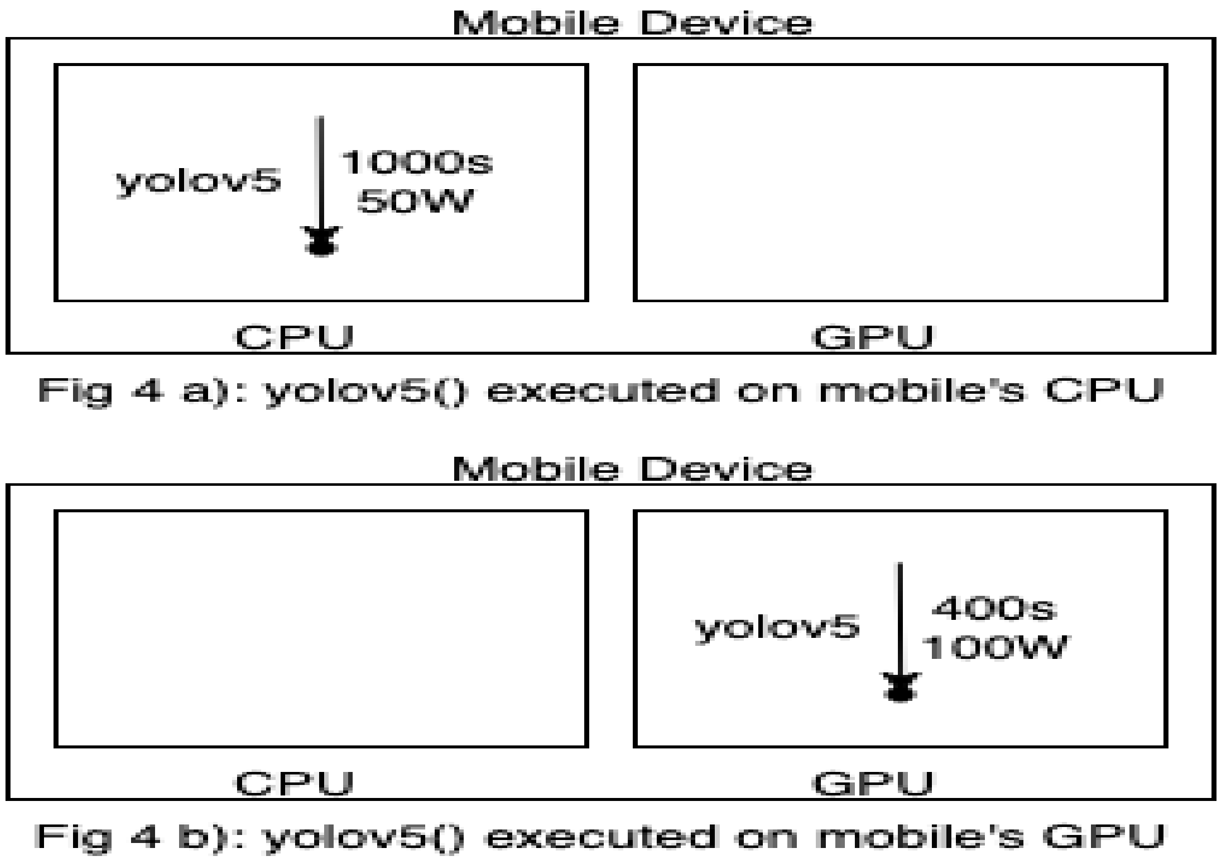

Mobile Device
yolov5
1000s
50W
CPU
GPU
Fig 4 a): yolov5() executed on mobile's CPU
Mobile Device
yolov5
400s
100W
CPU
GPU
Fig 4 b): yolov5() executed on mobile's GPU

Figure 1.9: A hypothetical example of yolov5 running on CPU of mobile device vs GPU of mobile device, shows how gpu consumes more energy than cpu and performs gpu specific tasks faster

From above figures, it is quite clear how computational offloading is useful in reducing execution times of our program. But what happens when more than one program request is generated by a mobile device? To illustrate this problem, let us consider one more program, namely FFT. FFT does not uses a lot of gpu power but takes more time than yolov5(if a huge file for yolov5 is used and a huge array for FFT is used) to compute the results.

To overcome this problem, we propose a scheduling algorithm be used to schedule remote mobile-device's requests in such a way that our requests complete within a given time frame, while using least amount of energy.

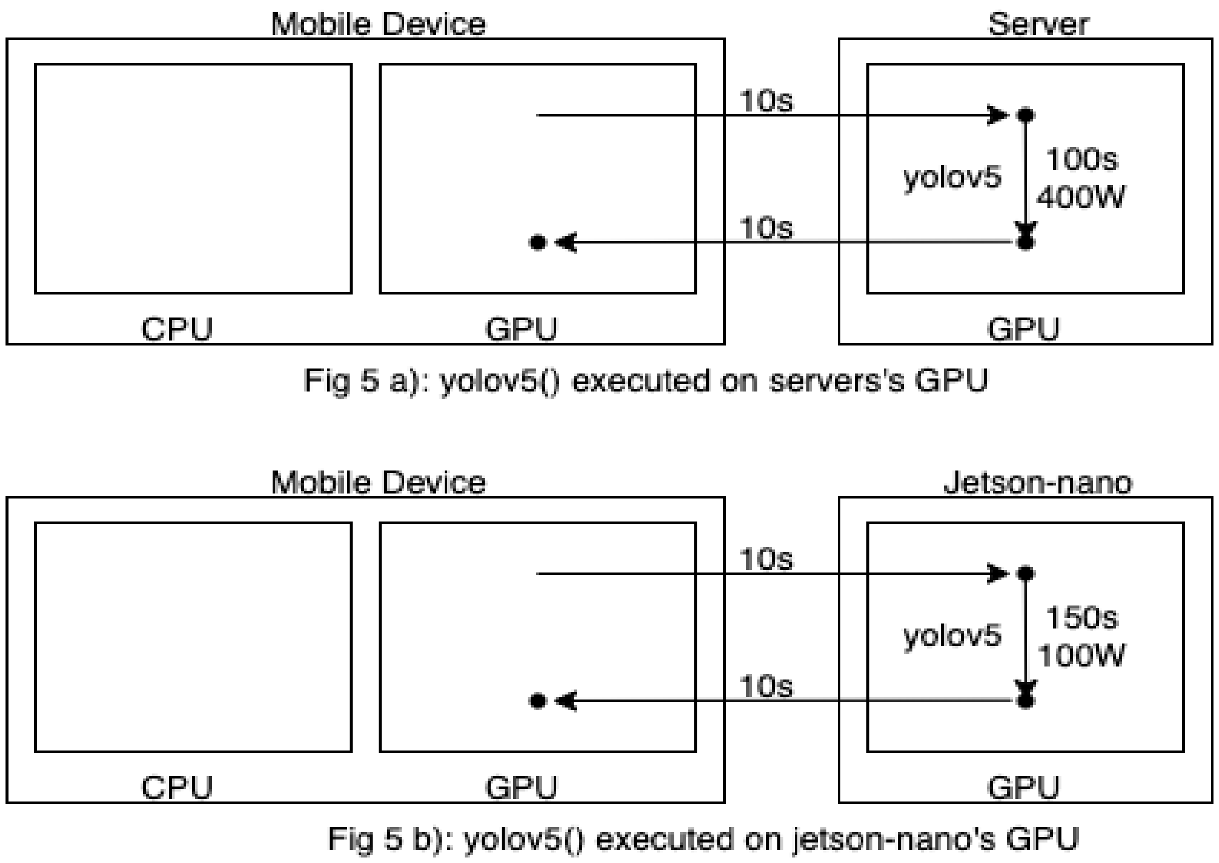


Figure 1.17: A hypothetical example of yolov5, offloading its gpu specifics functions to server and jetson nano, shows how jetson nano uses less energy compared to server but more time, we will quantify these with benchmarkings in the next chapter

Chapter 2

Related Work

2.1 Literature Review

Mobile Edge Computing(MEC) Mobile edge computing, as claimed by ETSI is “MEC provides cloud computing capabilities and Internet Technology environment within the radio access network near the proximity of mobile subscribers [2]. Cloud computing ability is provided by MEC within the confines of the RAN. MEC enables the user to bridge straight to the imminent cloud service-enabled edge network, rather than permitting direct mobile traffic between the end-user and the core network [3].

Main components of Mobile Edge Computing(MEC):-

A. Computational Offloading Computation offloading is the process of offloading the computational tasks from the user device to external resource rich devices or systems to enhance the capacity of mobile devices to carry out computation power requiring tasks.

In 2016, Chen et al. [4] proposed an offloading model scheme in which Game theory helps in making the correct decision for the connected users when connecting to a wireless channel. While performing the offload computation activities, if all the user devices using the same wireless channel, it will cause interference in the signal with one another and therefore led to the reduction in the wireless quality. The NP-hard problem of computation offloading sustained by multiuser computation offloading and provides a solution by attaining the Nash equilibrium of multiuser computation.

B. Energy Efficiency MEC reduces the energy consumption of the user devices by migrating the compute intensive tasks to different resource rich devices.
In 2014, Gao [5] proposed an opportunistic peer-to-peer MCC framework that consists of peer mobile devices which are connected with their short-range radios. Depending on their available capacity, the mobile devices are sharing both computational resources and energy.

More information related to basic definitions used in my paper, along with more information related to MEC can be found in this paper. [6] [6] [7] These two papers also highlights the survey on mobile edge computing's efficient energy management system.
MEC holds the potential to give a platform to the numerous content marketers and services towards the handheld and mobile industry. The improvement in networking and newer and efficient technological advancements make MEC a highly potent possibility.

A lot of research is going on in this field, related to energy aware placement algorithms on edge-servers while saving time, we will look at some of the prominent researches :-
[8] **Edge server placement problem** is a hot topic in mobile edge computing. In this paper, we study the problem of energy-aware edge server placement and try to find a more effective placement scheme with low energy consumption. Then, we formulate the problem as a multi-objective optimization problem and devise a particle swarm optimization based energy-aware edge server placement algorithm to find the optimal solution. We evaluate the algorithm based on the real dataset from Shanghai Telecom and the results show our algorithm can reduce more than 10% energy consumption with over 15% improvement in computing resource utilization, compared to other algorithms.

There are also some papers which focused mostly on the time, energy and cost performance of wimpy heterogeneous systems for Edge Computing. This paper [9] focuses on 3 main points :
1) Heterogeneous systems with GPU for both the edge and the cloud since they provide time-energy savings of up to 70% for compute-intensive applications.
2) We establish an equivalence ratio of 12 wimpy edge nodes that achieve the same or better

performance compared to a single brawny cloud instance. Using this time-performance equivalence ratio, we show that edge computing using wimpy systems results in cost savings compared to brawny cloud computing servers except for the case where the manpower cost does not get amortized because of small cluster sizes.
3) Counter-to-intuition, we observe that recent Jetson TX1 system exhibits lower time-cost performance compared to the older Jetson TK1 system. This is due to lower operating core clock frequency and the unexpectedly low Instructions-per-Cycle (IPC) of the TX1 GPU on some compute-intensive applications.
These claims are promising for starting my research in the field of edge-cloud heterogeneous use. Since, it can not only imporove user experience but also reduce energy usage upto 70%.

There are various other papers written which emphasize the importance of energy saving in servers and mobile computing being the strong hold for these types of researches. Many researches were conducted but with slight modifications on their basic workings, these researches had different applications with same ideology of saving energy from servers, some examples of such researches are :- 1) [10] shows energy-efficient task scheduling in **vehicular edge computing networks**. A little bit different from previous paper but clearly shows another way to make energy-efficient task scheduling algorithms.
2) [11] shows real-time heterogeneous Edge Computing system for **Social Sensing Applications**
3) [12] shows minimizing Energy for **Caching Resource Allocation** in Information-Centric Networking with Mobile Edge Computing
4) [13] shows energy efficient cooperative Edge Computing with Multi-Source Multi-Relay Devices, this paper also aims at finding a solution to energy consumption minimization problem. It was a good starting point for my research, as it defined its models into two parts :- a) network model b) channel model, and finds best algorithm to reduce energy needs to a significant level.

Then some papers also used machine learning to solve benchmarkings on gpus of edge systems. Below are some interesting papers related to research in this field
Reddi et al. [14] present their benchmarking method for evaluating ML inference systems. MLPerf prescribes a set of rules and best practices to ensure comparability across systems

with wildly differing architectures. They presented MLPerf Inference, a standard ML inference benchmark suite with proper metrics and a benchmarking method to measure the inference performance of ML hardware, software, and services. They explain why designing the right ML benchmarking metrics, creating realistic ML inference scenarios, and standardizing the evaluation methods enables realistic performance optimization for inference quality.

Richins et al. [15] deploy and characterize Face Recognition, an AI-centric edge video analytics application built using open source and widely adopted infrastructure and ML tools. They evaluated its end-to-end behavior in a production-size edge data center and revealed the "AI tax" for all the involved processing. They also discuss how their conclusions generalize beyond Face Recognition as many AI-centric applications at the edge rely upon the same underlying software and hardware infrastructure.

This article [16] first reviews articles published over the past three decades to trace the history of performance benchmarking from tightly coupled to loosely coupled systems. It then systematically classifies previous research to identify the system under test, techniques analyzed, and benchmark runtime in edge performance benchmarking. This article focuses on the following aspects like tracing the history of the development of performance benchmarking over the past three decades for high-performance computing (HPC) and cloud systems, examining different edge performance benchmarks, and reviewing the system under test, and benchmark runtime that facilitate edge performance benchmarking. Additionally, they suggested that real-time edge performance benchmarking can be integrated with automated edge software development and adaptive edge orchestration platforms to select the most appropriate edge resource for deployment based on current performance and network conditions. In this context, edge performance benchmarks will be of significant interest to any edge application developer.

An edge user allocation policy [17] determines how to allocate service requests from mobile users to MEC servers. Current state-of-the-art techniques assume that the total resource utilization on an edge server equals the sum of the individual resource utilization of services provisioned from the edge server. One of the main reasons for shifting computing power from the cloud to its data source is to reduce latency. At many times, the edge device needs to perform elementary calculations to run its operation near to real-time where these low-powered single-board computers like Raspberry pi and Jetson Nano are used.

2.2 Background

2.2.1 Jetson Nano-4GB

We have selected Jetson Nano (4GB variant) 2.1 as our edge computing device. It's made for embedded designers, researchers, and DIY makers, delivering the power of modern AI in a compact, easy-to-use platform with full software programmability. Jetson Nano provides 472 GFLOPS of computing performance with a quad-core 64-bit ARM CPU and a 128-core integrated NVIDIA GPU. It also includes 4GB LPDDR4 memory in an efficient, low-power package with 5W/10W power modes and 5V DC input. It features an L1 data cache of 32K, an L1 instruction cache of 48K, and an L2 cache of 2048K.

Jetson Nano	
CPU	64-bit Quad-core ARM A57 @ 1.43GHz
GPU	128-core NVIDIA Maxwell @ 921MHz
Memory	4GB 64-bit LPDDR4 @ 1600MHz | 25.6 GB/s
Video Encoder*	4Kp30 | (4x) 1080p30 | (2x) 1080p60
Video Decoder*	4Kp60 | (2x) 4Kp30 | (8x) 1080p30 | (4x) 1080p60

Table 2.1: Jetson Nano Developer Kit technical specifications.

Here, * indicates the maximum number of concurrent streams up to the aggregate throughput. Supported video codecs: H.265, H.264, VP8, VP9 (VP9 decode only)

2.2.2 NVIDIA GeForce RTX 2080 Ti

The NVIDIA GeForce RTX 2080 Ti 2.2 is the fastest consumer desktop graphics card and the fastest Turing RTX card at launch. It offers 4,352 shaders, 11 GB GDDR6 with a 352-bit interface (14 GHz, 616 GB/s for the Founders Edition). Furthermore, the 2080Ti includes 544 Tensor cores and 68 raytracing cores.

NVIDIA manufacturers the TU102 chip on a 12 nm FinFET process and includes features like Deep Learning Super Sampling (DLSS) and Real-Time Ray Tracing (RTRT), which should combine to create more realistic lighting effects than older GPUs.

NVIDIA GeForce RTX 2080 Ti	
NVIDIA CUDA Cores	4352
Boost Clock (MHz)	1545
Base Clock (MHz)	1350
Memory Speed	14 Gbps
Standard Memory Config	11 GB GDDR6
Memory Interface Width	352-bit
Memory Bandwidth (GB/sec)	616 GB/s

Table 2.2: NVIDIA GeForce RTX 2080 Ti

2.2.3 CUDA

The CUDA architecture is a revolutionary parallel computing architecture that delivers the performance of NVIDIA's world-renowned graphics processor technology to general-purpose GPU Computing. Applications that run on the CUDA architecture can utilize an installed base of over one hundred million CUDA-enabled GPUs in desktop and notebook computers, professional workstations, and supercomputer clusters.

The CUDA architecture consists of several components, in the green boxes below:

1. Parallel compute engines inside NVIDIA GPUs
2. OS kernel-level support for hardware initialization, configuration, etc
3. User-mode driver, which provides a device-level API for developers
4. PTX instruction set architecture (ISA) for parallel computing kernels and functions

Chapter 3

GPU Benchmarking

In this chapter, we first explain the need of GPU benchmarking, then methodology of benchmarking and finally the results. We then utilize these results to reach our conclusion.

3.1 Need of GPU Benchmarking

We discussed the idea that servers in general consume more energy than edge-devices like jetson-nano, in the chapter Introduction, and how 43% effort goes into cooling servers, servers are a bit faster than jetson-nanos and latency of servers is higher than jetson-nano. To quantify these notions, we may attempt to run a few algorithms/services to benchmark a particular cloud-server, edge-device like jetson nano and personal computers and see how much faster and energy hungry can servers be compared to edge-devices.

3.2 Technical Specifications of Hardware used in my experiments

3.2.1 Cloud-Server

CPU : 48 cores x Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz, 12 cores, 24 threads

GPU : NVIDIA GeForce RTX 2080 Ti, It offers 4,352 shaders, 11 GB GDDR6 with a 352 bit interface (14 GHz, 616 GB/s for the Founders Edition). Furthermore, the 2080Ti includes 544 Tensor cores and 68 raytracing cores.

3.2.2 Personal Computer

CPU : 4 cores x Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz

GPU : GeForce GTX 1050Ti

3.2.3 Jetson-nano

Jetson Nano delivers 472 GFLOPS of compute performance with a quad-core 64-bit ARM CPU and a 128-core integrated NVIDIA GPU. It also includes 4GB LPDDR4 memory in an efficient, low-power package with 5W/10W power modes and 5V DC input. It features L1 data cache of 32K, L1 instruction cache of 48K and L2 cache of 2048K.

3.3 Description of each Bench-marking service/algorithm

3.3.1 YOLOV5

YOLO an acronym for ’You only look once’, is an object detection algorithm that divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. YOLO is one of the most famous object detection algorithms due to its speed and accuracy. We ran pre-trained models based on YOLOV5 on Jetson Nano and GPU server. It uses Pytorch for training and inferencing.

3.3.2 Fast Fourier transform (FFT)

FFT: The FFT is a divide-and-conquer algorithm for efficiently computing discrete Fourier transforms of complex or real-valued data sets. It is one of the most important and widely used numerical algorithms in computational physics and general signal processing. The cuFFT library provides a simple interface for computing FFTs on an NVIDIA GPU, which allows users to quickly leverage the floating-point power and parallelism of the GPU in a highly optimized and tested FFT library. cuFFT: The cuFFT Library provides GPU-accelerated FFT implementations that perform up to 10X faster than CPU-only alternatives. cuFFT is used for building commercial and research applications across disciplines such as deep learning, computer vision, computational physics, molecular dynamics, quantum chemistry, and seismic and medical imaging. Using cuFFT, applications automatically benefit from regular performance improvements and new GPU architectures.

3.3.3 Real-time voice clone

Recent advances in deep learning have shown impressive results in the domain of text-to-speech. Text-to-speech takes text as an input and mimic exactly same voice on which it had been trained for cloning. A recent research introduced a three-stage pipeline that allows to clone a voice unseen during training from only a few seconds of reference speech, and without retraining the model. The experiments for Real-time voice cloning were performed on GPU server as well as Jetson Nano. It's an implementation of Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis (SV2TTS) with a vocoder that works in real-time. For this, pytorch, FFmpeg and necessary requirements were installed. It accepts input as an audio file and a sentence which we want to be convert into similar voice to that audio file.

3.3.4 Real-time video transcoding

Video transcoding is one of the most important steps in the streaming video process, but it can be difficult to facilitate. Video providers must be able to accommodate an increasing number of display devices with screen resolutions and frame rates that creep higher over time. And as devices, resolutions, and frame rates increase, so does the transcoding time required for complex video that is distributed on-demand. FFMPEG is one of the most popular open-source multimedia manipulation tools with a library of plugins that can be applied to various parts of the audio and video processing pipelines and have achieved wide adoption across the world. Video encoding, decoding and transcoding are some of the most popular applications of FFmpeg. I have done benchmarking of Real-time video transcoding, on GPU server using FFmpeg. Fisrt, I have installed FFmpeg, then necessary packages and libraries. Then a video file was provided as an input for transcoding. Finally transcoding is performed using GPU, for example video format conversion like H.264 to H.265, resolution change like 1080p to 720p.

3.3.5 Image Classification (only on jetson nano)

Image classification is a technique that is used to classify or predict the class of a specific object in an image. The main goal of this technique is to accurately identify the features in an image. We are using pretrained imagenet model to inference on jetson nano. Imagenet Model: ImageNet is a large dataset of annotated photographs intended for computer vision research so basically, it is a project which aims to provide a large image database for research purposes. Image classification experiment is done using jetson-inference code on Jetson Nano. For this, necessary packages were installed, configured and then compiled. Then a pre-trained model was run on few images of imagenet dataset for our experiments.

3.3.6 RL-based industrial controller (using cpu)(only on jetson-nano)

Reinforcement learning: Reinforcement Learning is a subset of machine learning. It enables an agent to learn through the consequences of actions in a specific environment. Industrial control system (ICS) is a collective term used to describe different types of control systems and associated instrumentation, which include the devices, systems, networks, and controls used to operate and/or automate industrial processes. We've recently come across the some applications of industrial control systems that are a good fit for RL. For this benchmarking, we used containerized example of reinforcement learning using docker. That will remove itself after exiting. For that necessary packages were installed and then experiments were run to get the output and analysis.

3.4 Preparation for bench-marking : For each bench-marking listed below, our platforms are prepared for the benchmarking

1) It was made sure that no other service was running on the platform and platform was given sufficient time to cool down, so temperature of cpu and gpu should be at minimum possible level, so performance losses due to temperature are minimized.
2) For each service, it was made sure that service doesn't produce any errors.
3) For each service, it was made sure that it uses gpu of the platform.
4) For each platform, same service with same parameters was used, so comparison is optimal.

3.5 Assumptions : For each bench-marking listed below, we have made some assumptions

1) **Latency is neglected** :
Even though we are using non-remote servers, jetson-nanos and personal computers, they are bound to have some latency, but that latency is considered and kept negligible for our experiments.

2) **Cloud server used in our experiment and commercial cloud server have similar cooling capabilities** :
Cloud server we have used lies in the premises of IIITD college. We have assumed it will have similar cooling capabilites as used in customer grade commercially used servers and will have similar cooling effects on the server.

3) **No service error decreasing throughput and generating psuedo heat**

In real world, services requested by remote users can have errors or might halt in between tasks, this might reduce the throughput of servers. It might look like servers are not performing much tasks but in reality those tasks are not getting completed in a given time frame.

It might also appear as if servers are consuming more power per unit work done, and are generating more heat than normal, called psuedo-heat.

So, it is essential to use services which do not produce errors or to monitor services which produce errors and rerun service all over again.

4) **No service is halted while running and generating psuedo heat**

In real world, services requested by remote users might be haulted before completion, this might reduce the throughput of servers. It might look like servers are not performing much tasks but in reality those tasks are not getting completed in a given time frame.

It might also appear as if servers are consuming more power per unit work done, and are generating more heat than normal, called psuedo-heat.

So, it is essential to use services which do not produce errors or to monitor services which produce errors and rerun service all over again.

3.6 Methodology of GPU Benchmarking

1) Hardware is prepared for bench-marking, as stated in section, Preparation for benchmarking.

2) Each service is downloaded from their respective github pages and made sure they are running on the platform and they do not contain any errors, as stated in section No service error.

3) **Measuring Time of each Service on each Platform :** Every service's code is edited to measure time it takes for it to run on the platform, all measurements for time are done automatically via code.

4) a) **Measuring Energy of each Service on Server and Personal Computer :**

Since, both server and personal computer can run linux, we can measure energy of both of them using similar code.

We use the command : **nvidia-smi stats -i <device> -d pwrDraw** It measures power Draw of GPU, every second. We save output in a textfile, clean it and produce average power draw using a python file, uploaded to github given here.

4) b) **Measuring Energy of each Service on Jetson-nano :**

We use the command : **sudo tegrastats** It measures power Draw of GPU, every second. We save output in a textfile, clean it and produce average power draw using a python file, uploaded to github given here.

5) **Stress Testing of GPU on Server**

Since, both server and personal computer can run linux, we can stress test both of them using similar code.

For this purpose, I have used yolov5 code given here.

We try to run as many instances possible of yolov5 on server, this will help us find out increase in power usage with each instance of yolov5.

Algorithm/Service	Server (Time/Power)	Personal Computer(Time/Power)	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$7.93s/49.75W$	$28.2s/2.9W$
Real Time Video Transcoding	$6s/82W$	$10.22s/51W$	$30.22s/3.21W$
$OnAverage$	$36.12s/81.64W$	$70.4s/50.9W$	$806.1s/3.03W$

Table 3.1: An example of few services when run on Server and Edge-device(in this case jetson-nano), since most of the services are gpu focussed we are using jetson-nano, but other edge-devices which only contain cpu and gpu can also be connected

3.7 Results of Benchmarking

From our bench-marking results, we have found that on average server performs 2131.73% faster than edge-devices(here jetson-nano), while consuming 2594.39% more power than edge-devices(here jetson-nano).
Thus, a scheduling algorithm has to balance the trade-off between more execution of tasks on the edge-devices to decrease latency and ensuring less energy consumed by servers.

Table 3.1, illustrates an example of time and power usage of a yolov5 program, run on different platforms (server and jetson-nano).
When yolov5 runs entirely on Edge device, it takes 1052s and consumes 3W of power. When it runs entirely on GPU of server it takes 50s (significantly less time) and 85W (significantly more power). This is because a server is a fairly high-powered device, and a lot of power is used in cooling the server, so it uses very-high energy and results in very less compute times. But we can offload some of the GPU-specific load to a edge-device and reducing power usage and decreasing latency as well in the process.

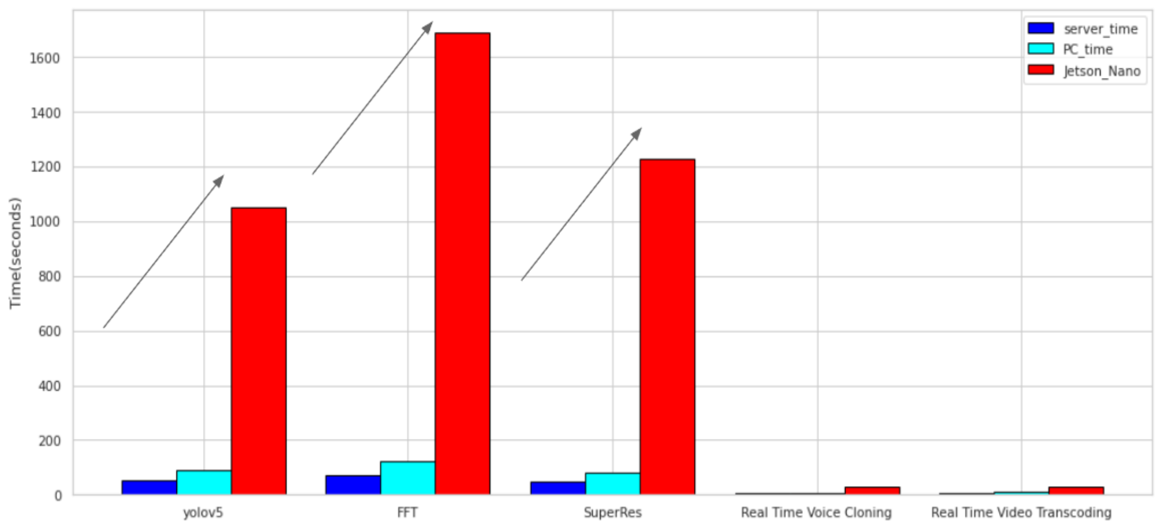


Figure 3.1: Time comparison between different platforms for each service tested

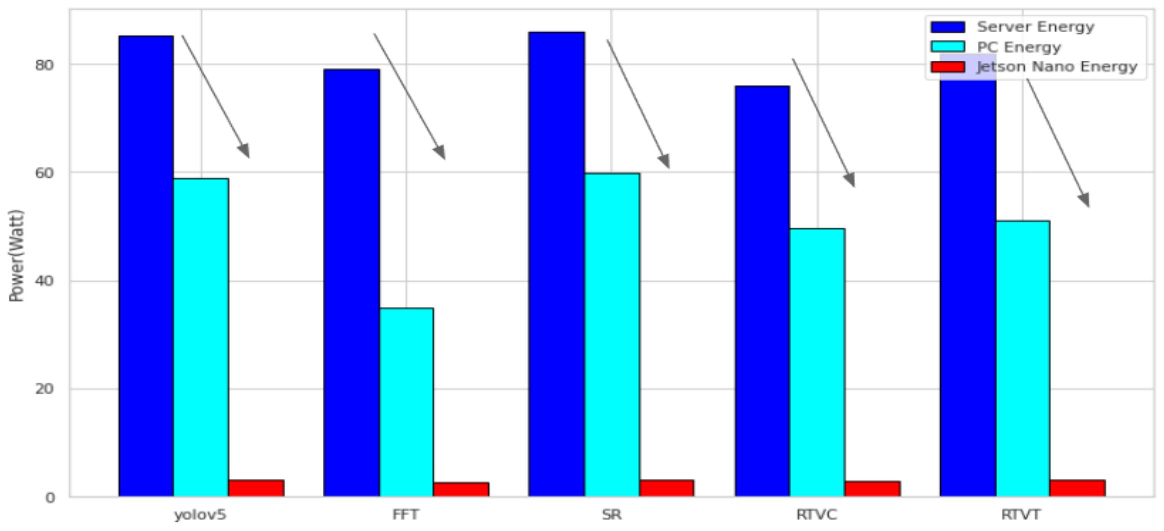


Figure 3.2: Power comparison between different platforms for each service tested

Figure 3.1 and 3.2, shows that time taken by edge-devices like jetson nano are fairly low powered devices when compared to servers hence take more time to run services.

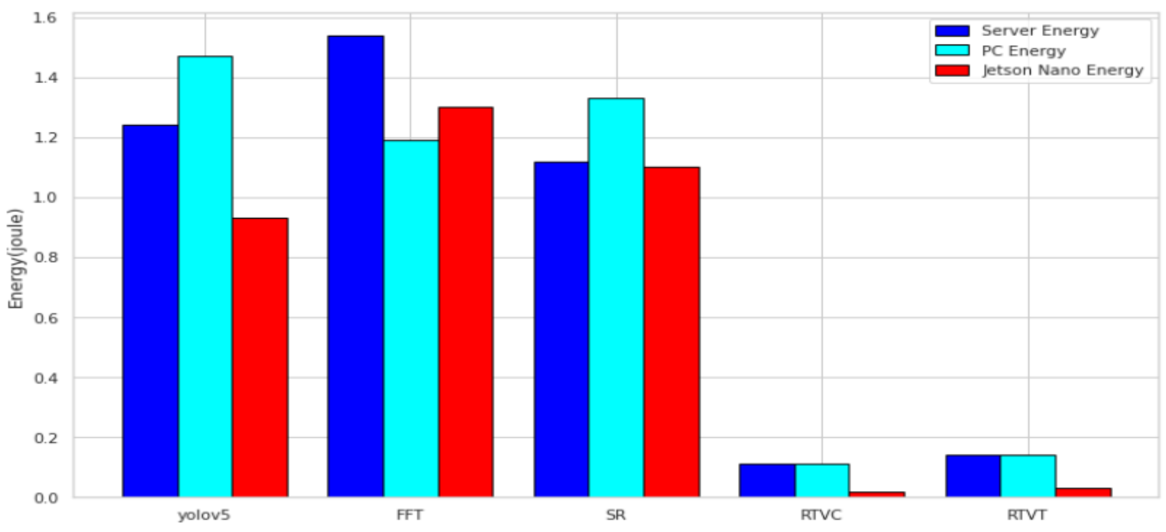


Figure 3.3: Energy comparison between different platforms for each service tested

Figure 3.3, shows that energy consumption of different services on different platforms. As we can see there is a comparable difference between their energies. Servers taking comparably more energy than edge-devices, indicating a trade-off between energy-consumption and time in Edge-Cloud Environment.

3.7.1 Results of Stress Testing on server

Since, both server and personal computer can run linux, we can stress test both of them using similar code.

For this purpose, I have used yolov5 code given here.

We try to run as many instances possible of yolov5 on server, this will help us find out increase in power usage with each instance of yolov5.

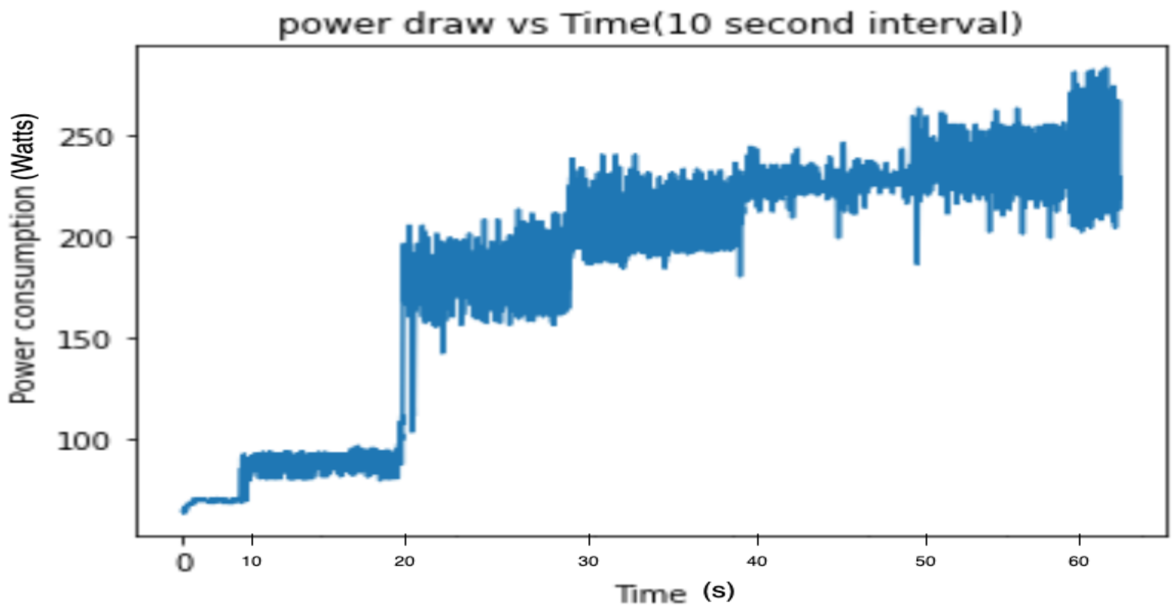


Figure 3.4: Power Consumption vs Time(10 second internal), After every 10 seconds

Chapter 4

Task Scheduling Problem

In this section, we try to use the concept of computation offloading to utilize this technique to develop its formal mathematical model.

4.0.1 Preliminaries

We represent the set of platforms p_i as $P = \{p_1, p_2, ..., p_m\}$, where $(i \leq m)$
We represent the set of services s_i as $S = \{s_1, s_2, s_3, s_4, ..., s_n\}$, where $(j \leq n)$

A GPU computation offloading system consists of several platforms p_i, each platform has its own GPU and CPU. We represent the set of platforms as $P = \{p_1, p_2, ..., p_m\}$. P is a set of "m" available platforms, where $(i \leq m)$.
S_j is a service requested by a user remotely, which can be executed on either a CPU or GPU of a Platform P_i. We represent the set of services as $S = \{s_1, s_2, s_3, s_4, ..., s_n\}$. S is a set of "n" available requests, where $(j \leq n)$.

Each service s_i when run on platform p_i , takes certain Time and Energy.
On a cpu/gpu of platform p_i a service s_j takes t_{ij} time to execute. On a cpu/gpu of platform p_i a service s_j takes e_{ij} energy to execute.The value of t_{ij} and e_{ij} depends on the service s_j and the power of the cpu/gpu of the platform p_i. An application begins and ends on the mobile device.

We assume that execution time t_{ij} and energy consumed e_{ij} are known a priori by running these applications already on server and jetson-nano and obtaining the table shown in figure 3. We consider that obtaining this table is a common in scheduling problems.

4.0.2 Mathematical Model

The scheduling algorithm needs to decide the platform p_i on which each service s_i executes. To denote this, let x_{ij} be a binary decision variable such that:

$$x_{ij} = \begin{cases} 1, & \text{if s}_i \text{ is executed on } p_j \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

The scheduling algorithm needs to decide the processor/component of the platform p_i on which each service s_i executes.
To denote this, let y_{ij} be a binary decision variable such that:

$$y_{ij} = \begin{cases} 1, & \text{if s}_i \text{ is executed on GPU of } p_j \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

We want to reduce the total energy consumption of our servers, Thus, our objective is to minimize the "total energy" E consumption of our platforms.

$$\text{Min} \;\; E \tag{4.3}$$

Total Energy (E) consumed can be defined as :

$$E = \sum_{j=1}^{n} \left[\sum_{i=1}^{m} \left[\left(e_{ij} \cdot y_{ij} + e_{ij} \cdot (1 - y_{ij}) \right) \cdot x_{ij} \right] \right] \tag{4.4}$$

Total Time (T) consumed can be defined as :

$$T = \max_{j=1}^{n} \left[\sum_{i=1}^{m} \left[\left(t_{ij} \cdot y_{ij} + t_{ij} \cdot (1 - y_{ij}) \right) \cdot x_{ij} \right] \right] \tag{4.5}$$

Total Time cant go beyond a given time constraint D

$$\text{T} \quad \leq \text{D} \tag{4.6}$$

Chapter 5

Methodology

5.1 Definitions and Tools

5.1.1 Google OR-Tools

Google OR-Tools is a free and open-source software suite developed by Google for solving linear programming (LP), mixed integer programming (MIP), constraint programming (CP), vehicle routing (VRP), and related optimization problems.

OR-Tools is an open source software suite for optimization, tuned for tackling the world's toughest problems in vehicle routing, flows, integer and linear programming, and constraint programming.

After modeling your problem in the programming language of your choice, you can use any of a half dozen solvers to solve it: commercial solvers such as Gurobi or CPLEX, or open-source solvers such as SCIP, GLPK, or Google's GLOP and award-winning CP-SAT.

5.1.2 Linear Optimization

Linear programming (LP, also called linear optimization) is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships. Linear programming is a special case of mathematical programming (also known as mathematical optimization).

More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. Its feasible region is a convex polytope, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine (linear) function defined on this polyhedron. A linear programming algorithm finds a point in the polytope where this function has the smallest (or largest) value if such a point exists.

Linear programs are problems that can be expressed in canonical form as :-
Find a vector x
that maximizes/minimizes $c^T x$
subject to $Ax \leq b$ and $x \geq 0$

Here the components of x are the variables to be determined, c and b are given vectors (with c^T indicating that the coefficients of c are used as a single-row matrix for the purpose of forming the matrix product), and **A** is a given matrix.
The function whose value is to be maximized or minimized ($c^T x$ in this case) is called the objective function.

Linear programming can be applied to various fields of study. It is widely used in mathematics, and to a lesser extent in business, economics, and for some engineering problems. Industries that use linear programming models include transportation, energy, telecommunications, and manufacturing. It has proven useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design.

For our purpose, we have also made on ILP, and we would try to solve it via OR Tools.

5.2 Model Preparation

The scheduling algorithm needs to decide the platform p_i on which each service s_i executes. To denote this, let x_{ij} be a binary decision variable such that:

$$x_{ij} = \begin{cases} 1, & \text{if s}_i \text{ is executed on } p_j \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

We have also defined a variable y_i which decides the processor/component(CPU or GPU) of the platform p_i on which each service s_i executes, but for our experiment we will assume that we run our program only on GPU, as we have chosen GPU specific tasks which perform better on GPU, like Yolov5 algorithm and FFT algorithm, as described in Benchmarking section .

.

$$y_{ij} = \begin{cases} 1, & \text{if s}_i \text{ is executed on GPU of } p_j \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

Keeping $y_i = 1$, our equation for Energy consumption and Total Time taken becomes :-
Total Energy (E) consumed becomes :

$$E = \sum_{j=1}^{n} \left[\sum_{i=1}^{m} \left[e_{ij} \cdot x_{ij} \right] \right] \tag{5.3}$$

Total Time (T) consumed becomes :

$$T = \max_{j=1}^{n} \left[\sum_{i=1}^{m} \left[t_{ij} \cdot x_{ij} \right] \right] \tag{5.4}$$

While the objective function and constrains remain the same.

$$\text{Objective Function}: \ Min \ \ E \tag{5.5}$$

Total Time cant go beyond a given time constraint D

$$\text{Constraint}: \ \text{Total Time (T)} \ \leq \text{D} \tag{5.6}$$

5.3 Code Preparation from our Formulated Model

From our Benchmarkings, we know how much time and energy each service takes on all the three platforms.

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$7.93s/49.75W$	$28.2s/2.9W$
Real Time Video Transcoding	$6s/82W$	$10.22s/51W$	$30.22s/3.21W$
$OnAverage$	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Table 5.1: An example of few services when run on Server and Edge-device(in this case jetson-nano), since most of the services are gpu focussed we are using jetson-nano, but other edge-devices which only contain cpu can also be connected

We have in total, 5 services to run on 3 platforms.

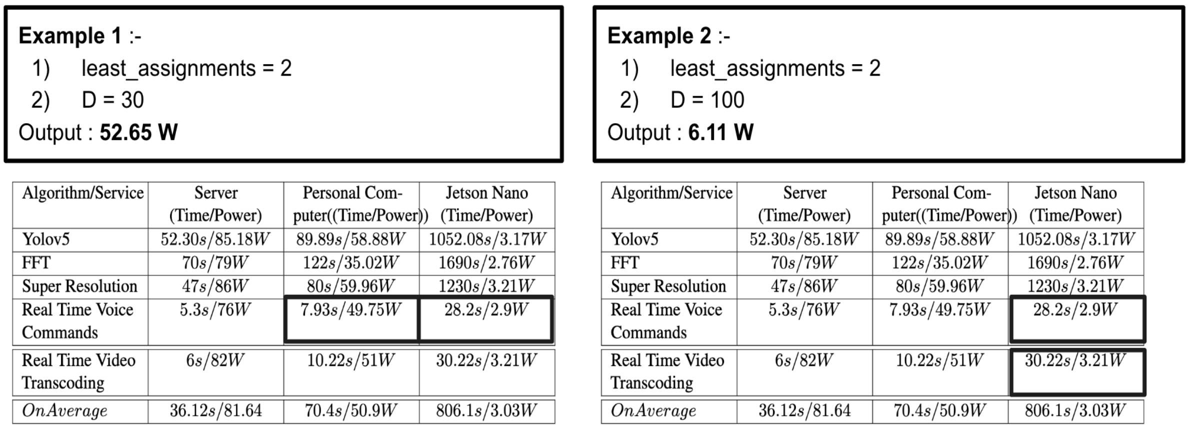
To setup our code, its clear from our mathematical model that :-

1) "m" represents total number of services, hence for our case, m=5.

2) "n" represents total number of platforms, hence for our case, n=3.

Now there can be 3 cases, for 3 different situations. There can be more cases, but we have chose these 3 cases because they might be most common use cases. We will understand those 3 cases and prepare our code for all the three cases :-

Case 1 : Fixed No. of Assignments for a given time(D)
Case 2 : Fixed no. of assignments for a given time interval(LB $\leq$ time $\leq$ UB)
Case 3 : Maximum No. of Assignments for a given time(D)



Example 1 :-
1) least_assignments = 2
2) D = 30

Output : **52.65 W**

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$7.93s/49.75W$	$28.2s/2.9W$
Real Time Video Transcoding	$6s/82W$	$10.22s/51W$	$30.22s/3.21W$
OnAverage	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Example 2 :-
1) least_assignments = 2
2) D = 100

Output : **6.11 W**

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$7.93s/49.75W$	$28.2s/2.9W$
Real Time Video Transcoding	$6s/82W$	$10.22s/51W$	$30.22s/3.21W$
OnAverage	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Figure 5.1: Two Examples for Case 1

Lets discuss each case in detail, and see how we can prepare our code for each case.

Case 1 : Fixed No. of Assignments for a given time(D)

We define two variables :-

1) **least_assignments**

It forces our ILP to allocate ‘least_assignments’ no. of services on available platforms.

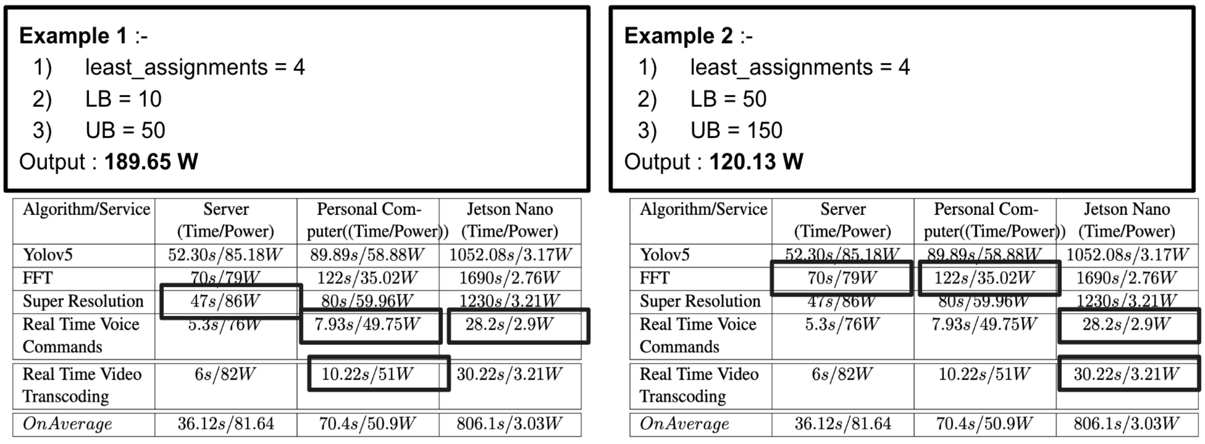
2) given time,**D**

It forces our ILP to allocate ‘least_assignments’ no. of services on available platforms but Total Time taken by all platforms to complete their tasks must not exceed D.

We have to schedule atleast a given number of services on our platforms represented by the variable **"least_assignments"**.

for example : if **least_assignments** = 2, D = 30 then our ILP will be forced to assign atleast two assignments, with total time $\leq$ D and minimum energy possible.

But since, our ILP is trying to minimize energy it will only assign 2 services on our platforms.



Example 1 :-
1) least_assignments = 4
2) LB = 10
3) UB = 50

Output : **189.65 W**

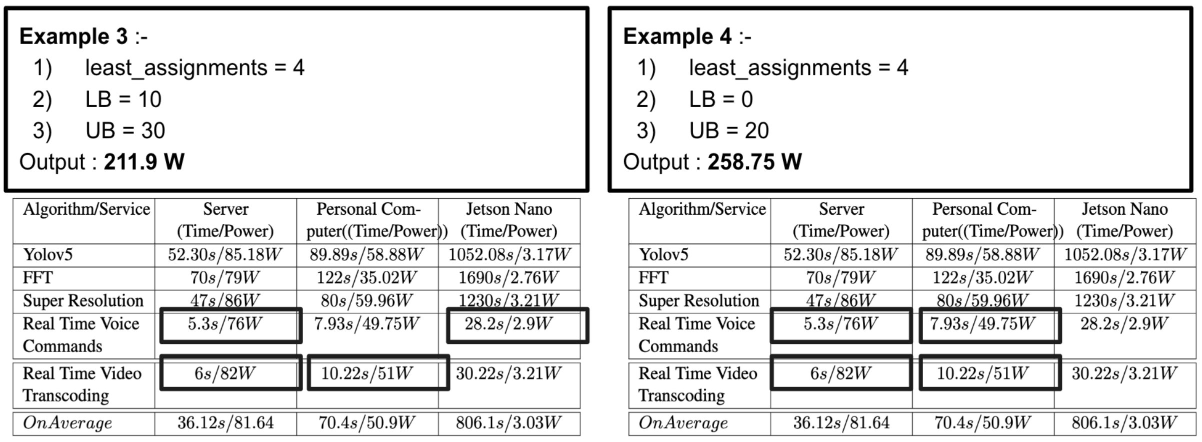
Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$\boxed{47s/86W}$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$\boxed{7.93s/49.75W}$	$\boxed{28.2s/2.9W}$
Real Time Video Transcoding	$6s/82W$	$\boxed{10.22s/51W}$	$30.22s/3.21W$
OnAverage	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Example 2 :-
1) least_assignments = 4
2) LB = 50
3) UB = 150

Output : **120.13 W**

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$\boxed{70s/79W}$	$\boxed{122s/35.02W}$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$7.93s/49.75W$	$\boxed{28.2s/2.9W}$
Real Time Video Transcoding	$6s/82W$	$10.22s/51W$	$\boxed{30.22s/3.21W}$
OnAverage	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Figure 5.4: Example 1 and Example 2 for Case 2



Example 3 :-
1) least_assignments = 4
2) LB = 10
3) UB = 30

Output : **211.9 W**

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$\boxed{5.3s/76W}$	$7.93s/49.75W$	$\boxed{28.2s/2.9W}$
Real Time Video Transcoding	$\boxed{6s/82W}$	$\boxed{10.22s/51W}$	$30.22s/3.21W$
OnAverage	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Example 4 :-
1) least_assignments = 4
2) LB = 0
3) UB = 20

Output : **258.75 W**

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$\boxed{5.3s/76W}$	$\boxed{7.93s/49.75W}$	$28.2s/2.9W$
Real Time Video Transcoding	$\boxed{6s/82W}$	$\boxed{10.22s/51W}$	$30.22s/3.21W$
OnAverage	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Figure 5.7: Example 3 and Example 4 for Case 2

Case 2 : Least no. of assignments for a given time interval(LB $\leq$ time $\leq$ UB)

We define three variables :-

1) least_assignments

It forces our ILP to allocate 'least_assignments' no. of services on available platforms.

2) Lower Bound, LB and 3) Upper Bound, UB

It forces our ILP to allocate 'least_assignments' no. of services on available platforms but Total Time taken by all platforms to complete their tasks must not exceed UB and must not go below LB.

for example : if **LB** = 10, UB = 30 then our ILP will be forced to find least number of assignments which can fulfill this range.

Case 3 : Maximum No. of Assignments for a given time(D)

We define two variables:-

1) least_assignments

It forces our ILP to allocate 'least_assignments' no. of services on available platforms.

2) given time, D

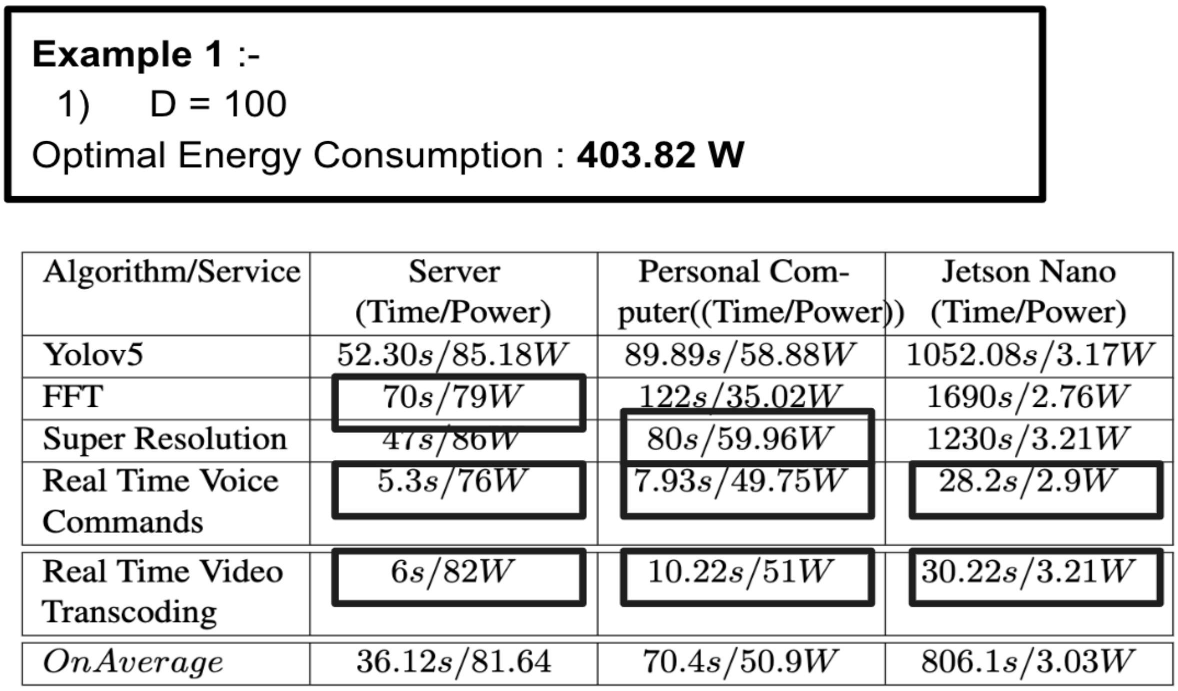
It forces our ILP to allocate 'least_assignments' no. of services on available platforms but Total Time taken by all platforms to complete their tasks must not exceed D.

In this case, we initialize variable "least_assignments", with maximum possible value i.e. m*n
Then in a loop, we decrease its value till least possible value of assignments i.e. 1

Now, while decreasing the variable we try to find if that many number of assignments are possible on our platforms or not.

Finally, we will converge at a point, which indicated maximum assignments possible on a given network of services and systems.

for example : **least_assignments** = m*n = 15(initially), D = 100. **least_assignments** will go from 15 to 1
For each value of least_assignments we will check if that assignment is possible, for each value, our ILP will be forced to assign "least_assignments" no. of assignments, with total time $\leq$ D while minimizing energy.


Example 1 :-

1) D = 100

Optimal Energy Consumption : **403.82 W**

Algorithm/Service	Server (Time/Power)	Personal Computer((Time/Power))	Jetson Nano (Time/Power)
Yolov5	$52.30s/85.18W$	$89.89s/58.88W$	$1052.08s/3.17W$
FFT	$70s/79W$	$122s/35.02W$	$1690s/2.76W$
Super Resolution	$47s/86W$	$80s/59.96W$	$1230s/3.21W$
Real Time Voice Commands	$5.3s/76W$	$7.93s/49.75W$	$28.2s/2.9W$
Real Time Video Transcoding	$6s/82W$	$10.22s/51W$	$30.22s/3.21W$
OnAverage	$36.12s/81.64$	$70.4s/50.9W$	$806.1s/3.03W$

Figure 5.10: Example 1 and Example 2 for Case 3

Algorithm for finding the optimal value for Case 3:

```
START
Let **least_assignments** = m*n (maximum possible service allocations, each service gets a )
Let i = least_assignments (declare loop variable i)
while ( i > 0) :
    Run ILP Solver Script to get schedules and optimal energy consumption
    if (valid Schedule): (atleast one service is allocated on atleast one platform)
        return Schedules and Energy consumption
        break
    else
        print("The problem does not have an optimal solution")
    i - - ( decrement loop variable i )
STOP
```

5.4 Summary

After we have prepared our code, we can run our code for execution. Out of above three cases :-

Case 1 : Fixed No. of Assignments for a given time(D)

Case 2 : Least no. of assignments for a given time interval(LB <= time <= UB)

Case 3 : Maximum No. of Assignments for a given time(D)

Case 3, seems to be of most use, as it takes into account time constraint while delivering maximum assignments possible for any network of services and systems.

Below figure illustrates, how power consumption increases with increase in Scheduled Services on our entire system, it increases to a point where we are no longer able to deploy any more services or else it would break our Time constraint.

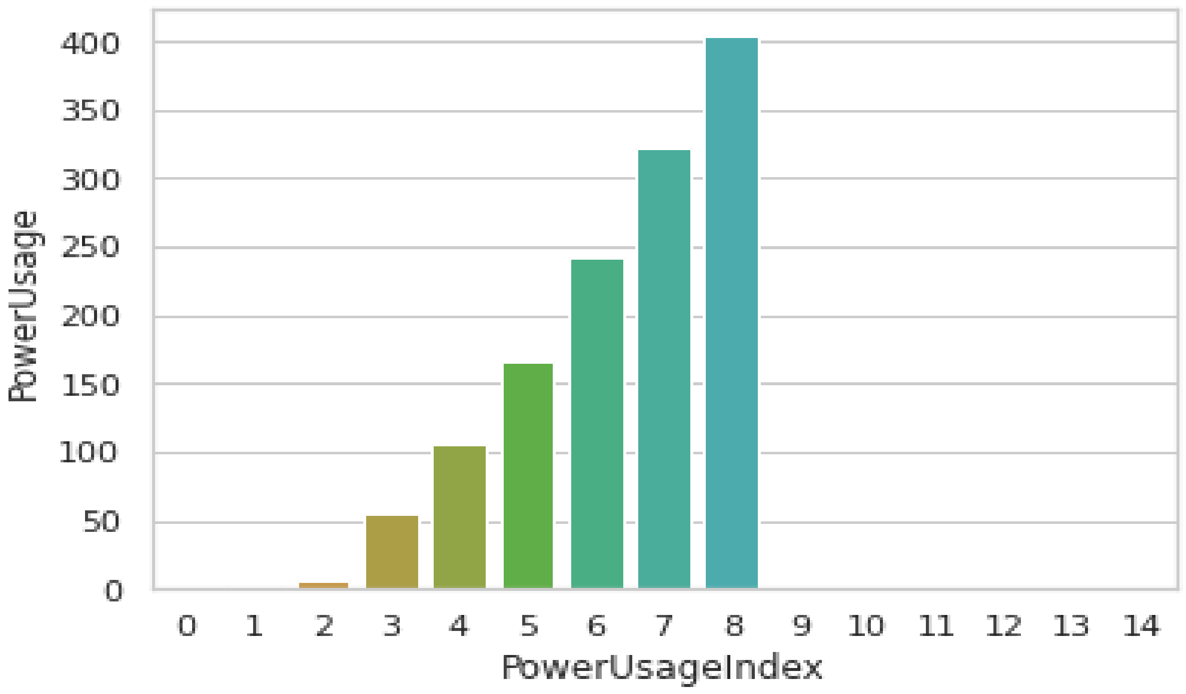


Figure 5.13: Increase in Power w.r.t. Services Run on model

Below figure illustrates, how power consumption increases with increase in Scheduled Services on our entire system, it increases to a point where we are no longer able to deploy any more services or else it would break our Time constraint.

We see a general upward trend and it falls off as soon as limit is reached.

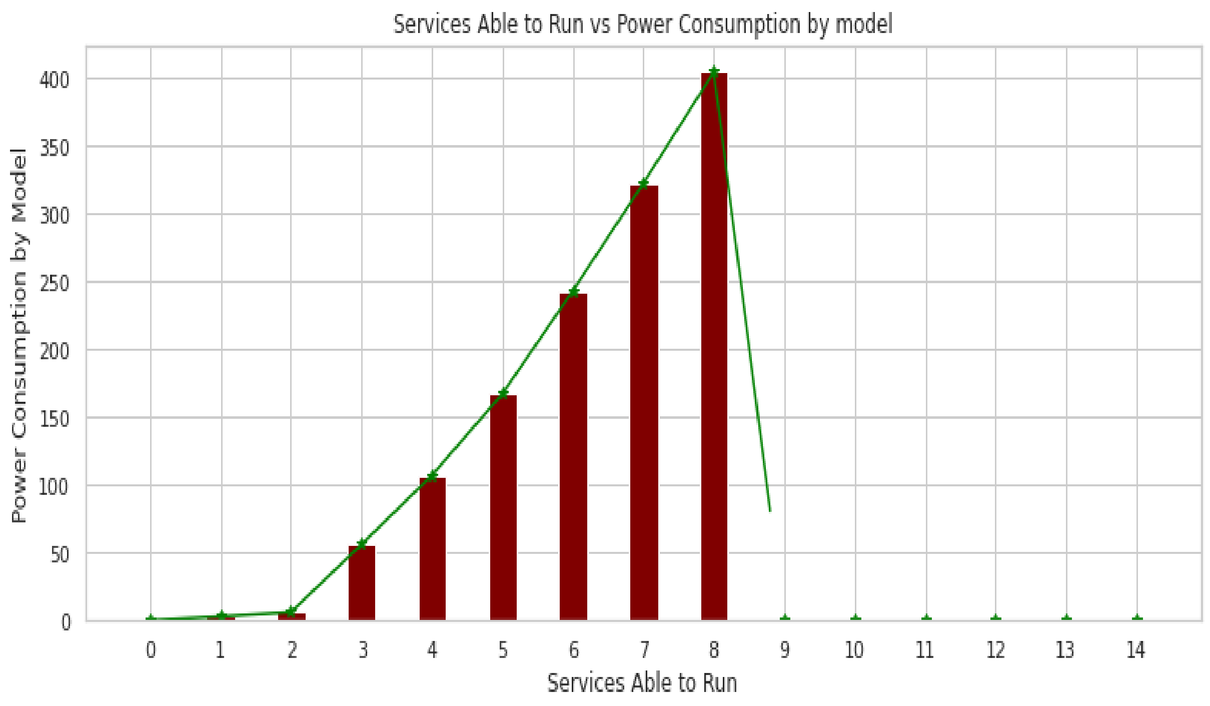


Figure 5.16: Services Able to Run vs Power Consumption by model

5.5 Additional Optimization

We can further optimize our search for a viable and effective energy consumption point using customized binary search on all possible "least_consumption" variable value.

Since, one part of our results show a non-existent part i.e. points on y-axis corresponding to their x-axis inputs, where there is no power consumption as there are no solution for those points, as they violate our constraints, to be in particular Time Constraint.

Hence, we can modify general binary search function and converge it to the optimal point, to further optimize our solution.

5.6 Observations from The Three Cases

1) As we start reducing the time constraint D, we observe that our scheduler starts allocating services to server platform, because server is faster than any other platforms (like a personal computer, edge-device (jetson-nano)).

2) With increase in assignments of services on all platforms, we see an increase in total Energy Consumption of our system.

Chapter 6

Conclusion and Future Work

In this work, we have utilized Google's OR-Tools to solve ILP formulated for the problem of "Tradeoff between energy and execution time in mixed cpu-gpu edge-cloud environment", after conducting and gathering extension research data of performance of different algorithms/programs on different type of systems(especially gpu's). The data includes energy and time consumption details of all algorithms on specified systems(like server grade gpu, desktop grade gpu and jetson-nano's gpu).

In Chapter 1, we discussed numerous studies and published articles which shows that power consumption of servers has grown exponentially and in near future it is expected to grow even further. With such high demands of power usage, servers also have a special characteristic, it takes 43% of its total power consumption for cooling purposes. When we increase total power usage of servers, indirectly we are also increasing the power consumption of cooling systems of servers. We also used several examples to show how offloading gpu-specific work to either server or jetson nano can save time for mobile devices, so overall its beneficial for users, but to save energy we have to dig deeper into statistics of computational offloading to server or jetson-nano.

In Chapter 2, based on these insights from 1, we start to look at work related in this field and try to do the literature review. We quantify the capabilities and specifications of different gpu's. We also learn about cuda and cuda-core's importance in our project.

Based on these insights, in chapter 3, we benchmarked available systems especially with different gpu's. We discuss the need of gpu benchmarking and the algorithms we ran on different systems and finally we discuss the results. When yolov5 runs entirely on Edge device, it takes 1052s and consumes 3W of power. When it runs entirely on GPU of server it takes 50s (significantly less time) and 85W (significantly more power). This is because a server is a fairly high-powered device, and a lot of power is used in cooling the server, so it uses very-high energy and results in very less compute times. But we can offload some of the GPU-specific load to a edge-device and reducing power usage and decreasing latency as well in the process. Stress testing on server reveals that power consumption of server also increases dramatically with increasing load, hence it becomes extremely useful to design a scheduling algorithm which can offload much of load to edge-devices to decrease energy use.

Based on our benchmarking results in chapter 3, in chapter 4, we discuss the Task scheduling problem, on how to schedule our tasks in such a way that we can reduce our energy consumption to an optimal level, without compromising on users delay tolerance level. We develop a mathematical model using an ILP and try to solve it using ILP solvers and modified code to run optimally.

We can finally conclude that there exists a tradeoff between energy consumption and execution time in Mixed CPU-GPU Edge-Cloud Environment, which can be exploited to save energy in many cases.

6.1 Future Work

Since, we are solving ILPs, which are considered to be NP-hard problem, our work remains open to better ILP solvers. Apart from improving time complexity of ILP solvers, in future, we can focus mainly on developing a better mathematical model. That mathematical model might take into account Dual objective function characteristic of our problem i.e. we are trying to reduce energy while maximizing usage of our system. This might result in an "Multi-objective Optimization" problem, which might require completely different sets of tools to solve them.

Our work is also open to finding more cases, which might be helpful to our users, such as applying machine learning to find out which scheduling is more suitable for a different kind of systems. ML scheduling might help to develop a more robust system, which can fine tune to more different scenarios possible. Moreover, our work is open to Online-model of our problem i.e. we don't know power and time consumption data before hand and that problem might require completely different types of tools or algorithms to reach to a satisfactory conclusion.

Bibliography

[1] A. S. G. Andrae and T. Edler, “On global electricity usage of communication technology: Trends to 2030,” *Challenges*, vol. 6, no. 1, pp. 117–157, 2015. [Online]. Available: https://www.mdpi.com/2078-1547/6/1/117

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—a key technology towards 5g,” *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[3] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa, “The future of mobile cloud computing: integrating cloudlets and mobile edge computing,” in *2016 23rd International conference on telecommunications (ICT)*. IEEE, 2016, pp. 1–5.

[4] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[5] W. Gao, “Opportunistic peer-to-peer mobile cloud computing at the tactical edge,” in *2014 IEEE Military Communications Conference*, 2014, pp. 1614–1620.

[6] C. Guleria, K. Das, and A. Sahu, “A survey on mobile edge computing: Efficient energy management system,” in *2021 Innovations in Energy Management and Renewable Resources(52042)*, 2021, pp. 1–4.

[7] H. Yu, Q. Wang, and S. Guo, “Energy-efficient task offloading and resource scheduling for mobile edge computing,” in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2018, pp. 1–4.

[8] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *2018 IEEE International Conference on Edge Computing (EDGE)*, 2018, pp. 66–73.

[9] D. Loghin, L. Ramapantulu, and Y. M. Teo, "On understanding time, energy and cost performance of wimpy heterogeneous systems for edge computing," in *2017 IEEE International Conference on Edge Computing (EDGE)*, 2017, pp. 1–8.

[10] P. Dong, Z. Ning, R. Ma, X. Wang, X. Hu, and B. Hu, "Noma-based energy-efficient task scheduling in vehicular edge computing networks: A self-imitation learning-based approach," *China Communications*, vol. 17, no. 11, pp. 1–11, 2020.

[11] D. Zhang, N. Vance, and D. Wang, "Demo abstract: Real-time heterogeneous edge computing system for social sensing applications," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 101–102.

[12] Y. Tang, "Minimizing energy for caching resource allocation in information-centric networking with mobile edge computing," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2019, pp. 301–304.

[13] M. Yao, L. Chen, T. Liu, and J. Wu, "Energy efficient cooperative edge computing with multi-source multi-relay devices," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 865–870.

[14] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[15] D. Richins, D. Doshi, M. Blackmore, A. T. Nair, N. Pathapati, A. Patel, B. Daguman, D. Dobrijalowski, R. Illikkal, K. Long *et al.*, "Missing the forest for the trees: End-to-end

ai application performance in edge data centers," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 515–528.

[16] B. Varghese, N. Wang, D. Bermbach, C.-H. Hong, E. de Lara, W. Shi, and C. Stewart, "A survey on edge performance benchmarking," 2020, accepted at ACM Computing Surveys.

[17] S. P. Panda, A. Banerjee, and A. Bhattacharya, "User allocation in mobile edge computing: A deep reinforcement learning approach," in *2021 IEEE International Conference on Web Services (ICWS)*, 2021, pp. 447–458.