# **ADAPter**: Power Efficient Model Inference on NVIDIA GPUs

*Ante Tonkovic-Capin*, *Drishan Poovaya*, *Ashu Kumar*, *Pratik Sanghavi*
*Group 11*

## 1 Introduction

Our project was initially focused on adapting a specific Qualcomm device through the Qualcomm Innovators Development Kit (QIDK) and integrating it with the python package Zeus for performing inference on Mobile [18]. While a slew of challenges prevented us from that specific objective, we did our best to adhere to the broader goal of energy efficiency during AI inference on mobile devices. Accordingly, our primary goal was to explore the feasibility of measuring and tuning GPU usage and energy consumption during inference to enable, and be able to target, the most efficient energy usage on mobile devices. As AI powered breakthroughs and innovations continue to permeate every aspect of modern life, the demand for access to newer and more demanding models is only growing. Currently, there's an estimated 17 billion mobile devices in operation worldwide[8], an average of around 2 devices for every person on the planet.

Enabling efficient energy consumption across all these devices is critical for many reasons. First, allowing efficient usage and power consumption is crucial to the performance and accuracy of the models themselves[4]. Models often require input data to be pre-processed on-device with tasks like feature extraction, down-sampling, pooling, multi-modal transformations and many more tasks to ensure the models they feed provide reliable, consistent, timely and accurate responses[19]. Second, and perhaps more critically, enabling efficient power consumption during AI inference is essential to the long-term sustainability of the technological breakthroughs being powered by advances in all fields of Artificial Intelligence[4]. Ensuring that these breakthroughs are accompanied by efficient energy-usage and a commensurately low carbon footprint should be seen as a responsibility shared by all researchers and advocates of the technology.

Given how crucial the issue is, there have been numerous studies into the the energy usage and efficiency of models during training, yet attention on inference is lacking[11]. This may sound counter-intuitive, as most of the work and focus has been on efficient GPU usage and power-consumption during model training rather than inference[10]. However, several sources, including the likes of NVIDIA and Amazon, estimate that inference can exceed the cost of training in popular LLMs, with inference accounting for up to 90% of the costs for deployed models[5, 2]. As we celebrate the exponential growth and power of AI models, it is crucial that their energy usage, their carbon footprint, does not grow exponentially with them.

With this objective, we find an opportunity in energy consumption specifically during the inference phase on mobile devices. An energy efficient approach that is both adaptive in real-time and sensitive to pre-configured performance targets, will unlock avenues for more powerful model deployments on the edge in a much more sustainable way.

## 2 Background

Graphics Processing Units (GPUs) have become an integral part of mobile devices due to their ability to accelerate graphics and compute-intensive applications. However, the high-performance nature of GPUs often leads to significant energy consumption, which is a critical concern for battery-powered mobile devices and as we discussed before, long-term sustainability of the field more broadly. To address this, various techniques have been developed to tune and adjust GPU devices for optimal energy usage during inference [6].

One common approach is Dynamic Voltage and Frequency Scaling (DVFS) [9], which adjusts the operating frequency and resulting voltage of the GPU based on the workload. By lowering or increasing the frequency according to the demand, energy consumption can be adjusted to match either a specific target latency or a priori energy usage target. However, this must be carefully managed to avoid performance degradation and ensure actual decreases in power consumption without unintended side effects. Given the challenges we faced, this was the only approach available to us. An alternative avenue which we were unable to explore would have been

workload scheduling, which involves strategically distributing tasks across the GPU's cores to maximize utilization and minimize energy consumption. This may be particularly effective in mobile devices, where the GPU may have multiple cores with different performance and power characteristics such as with Apple's newer generation iPhones[1].

Focusing on the approach used for our project, DVFS, sounds simple enough in principle, yet a number of obstacles need to be overcome. First, access to modify the frequency of the device itself. As mentioned earlier for Qualcomm and their QIDK, many device manufacturers expose specific SDKs or APIs in order to send and receive instructions to and from their GPU device. One may be tempted to modify the processor frequency or power usage via manufacturer-independent, device agnostic channels, but this is a non-starter as different mobile device manufacturers may use GPUs from different vendors (e.g., Qualcomm, ARM, Apple), each offering varying core counts and architectural designs. For example, Qualcomm's Adreno GPUs commonly found in Android devices can range from 1 to 8 or more cores depending on the device's target selling market.

This means that any DVFS for inference on mobile is restricted to using the tools and interfaces the manufacturer exposes, or through the APIs they have already exposed, in order to modify the device frequency. Second, while there are clear leaders in the field such as NVIDIA and AMD [15], the devices used on mobile devices vary widely and even a similar device may result in disparate energy consumption when used by a different system under different conditions irrespective of manufacturer's claims of device homogeneity [16]. Accordingly, careful consideration for the device and the APIs available for it, as well as the current system its on are needed in order to understand the exact range of options available for implementing any DVFS and the power consumption changes that result from its modification.

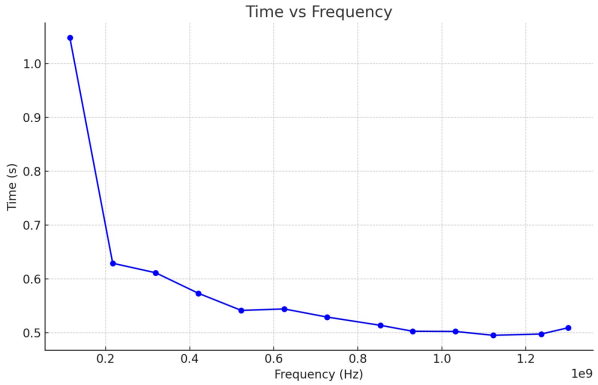


Figure 1: Frequency vs Duration Time

Another motivating factor for the DVFS approach we used is evident in the results in the time vs frequency plot, shown in Figure 1. This plot represents the time taken for inference with respect to the operating frequency of the device. It demonstrates a clear connection between total duration and frequency, showing a significant decrease in time taken to perform inference as the frequency is increased to 600 MHz. Even though the device has a maximum GPU frequency of 1.3 GHz, there's diminishing performance gains observed when increasing frequency beyond 600 MHz. Clearly, there is scope in reducing the frequency of the device, as it still performs adequately at lower frequencies.

All this to say that while our goal was broadly focused on mobile GPU devices during inference, our project was restricted by the options available for a specific version SDK on a specific GPU device running on a specific OS for the specific inference models compatible with it during the time we had access to the system. Understanding the complex environment and sensitive interfaces involved with mobile GPUs, especially when it comes to DVFS is crucial before proceeding to the design we chose, and why we chose it. Our initial results, along with our decision to target DVFS as the primary avenue to explore, lead us to the design we used in ADAPter.

# 3 Design

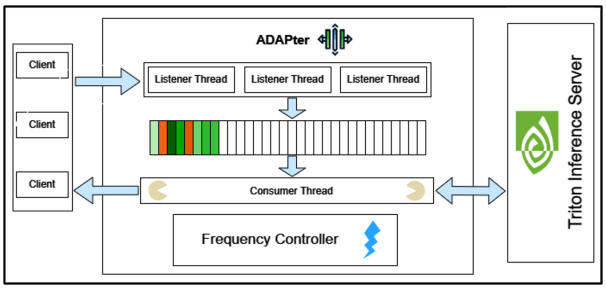


Figure 2: ADAPter design and architecture

## 3.1 Overview

Our system, ADAPter, represents an innovative approach to balance power efficiency with computational performance in machine learning inference tasks. Developed on the NVIDIA Jetson platform, this system utilizes a dynamic frequency scaling mechanism that adjusts GPU frequency based on real-time inference demands to meet predefined service level objectives (SLOs) while minimizing power consumption. The core architecture of ADAPter integrates a multi-threaded asynchronous execution model, comprising producer and consumer

threads, coupled with a frequency controller. This design ensures that the system not only responds promptly to inference requests but also operates under optimal power conditions.

## 3.2 Target Workload

Before we jump into the system design, its crucial to understand what kind of workloads our system will be optimal for. Devices such as the NVIDIA Jetson series are extensively utilized to perform edge AI inference tasks continuously over prolonged durations. These inference tasks are predominantly homogeneous, focusing consistently on a single application or use case. Moreover, the distribution of these workloads tends to be uniform across time. Such characteristics render these workloads particularly amenable to power optimization strategies. Over extended operational periods, it becomes feasible to calibrate the system to an optimal power consumption level, enhancing energy efficiency.

In contrast, workloads that exhibit bursty and heterogeneous characteristics pose significant challenges in terms of power optimization. The variability and unpredictability associated with these types of jobs complicate the ability to establish a stable and efficient power setting.

Consequently, designing systems to handle homogeneous, steady-state inference jobs can be simplified. This allows for the implementation of straightforward system architectures that minimize the overhead associated with job submission and dynamic adjustment of power levels.

## 3.3 System Components

### 3.3.1 Listener and Consumer Threads

The system architecture employs listener threads that continuously receive inference requests from clients. These threads act as producers, placing incoming requests into a shared queue. The consumer threads, on the other hand, retrieve requests from this queue and forward them to the Triton inference server running on the Jetson device. This separation of concerns allows the system to manage request handling and processing efficiently, ensuring that incoming requests are neither dropped nor delayed excessively, regardless of the current GPU frequency.

### 3.3.2 Shared Queue

The shared queue acts as a buffer that decouples the rate at which requests are received from the rate at which they are processed. This design is crucial in managing fluctuations in request volume and GPU processing capacity, particularly when adjustments in GPU frequency affect processing times.

### 3.3.3 Triton Inference Server

The Triton Inference Server provides a robust framework for deploying machine learning models and handling inference requests. It is integral to the ADAPter system, processing the requests forwarded by consumer threads. The server's performance directly influences the system's ability to meet SLOs, making its integration with the frequency controller critical.

### 3.3.4 Frequency Controller

The frequency controller is a pivotal component of the ADAPter system. It continuously monitors the deviation of actual latency from allowable latency (as a moving average), adjusting the GPU's operating frequency accordingly. The controller reduces the frequency to save power when the system operates under the acceptable latency threshold and increases it when the latency approaches or exceeds the SLO, thus ensuring that the system remains both power-efficient and effective in processing requests.

In conclusion, the ADAPter system's design leverages a sophisticated interplay between asynchronous processing and dynamic frequency scaling to address the dual challenges of power efficiency and performance in real-time inference scenarios. By integrating a producer-consumer model with intelligent frequency management, the system not only achieves significant reductions in power consumption but also upholds stringent performance criteria, making it a formidable solution for energy-conscious, high-performance computing environments. This balance of efficiency and effectiveness is expected to set a precedent for future developments in the field of machine learning infrastructure.

# 4 Evaluation

All Jetpack versions are pre-installed with a command line utility called tegrastats. They also ship with triton inference server, an open source tool by Nvidia to standardize AI model deployment. We'll be using tegrastats with appropriate arguments for measuring the impact of our system on the power draw and then compare it with the system defaults exposing the same workload patterns to the triton inference server. Check our GitHub repository for more detailed instructions on carrying out measurements: `https://github.com/Pratik-Sanghavi/Big-Data-Systems-Project`

### 4.1 DenseNet 121

We first tried our system with a 121 layer densenet model trained on ImageNet data. We intend to flood our triton inference server with a homogenous stream of images and obtain inference results. To observe the reactance of our system to a loose SLO, we kept the maximum allowable latency of every job fixed at 2s. This stream continues for about 60s after which we manually terminate sending requests. The aim is to observe that with loose bounds, ADAPter should settle at a lower frequency that just manages to meet the SLO and optimize power consumption. This is clearly visible from Figure 3 and Figure 4. While the latency of serving requests has certainly increased in ADAPter, it is still below the SLO (except for a few exceptions). The power draw clearly shows that we've become largely power efficient over the baseline.

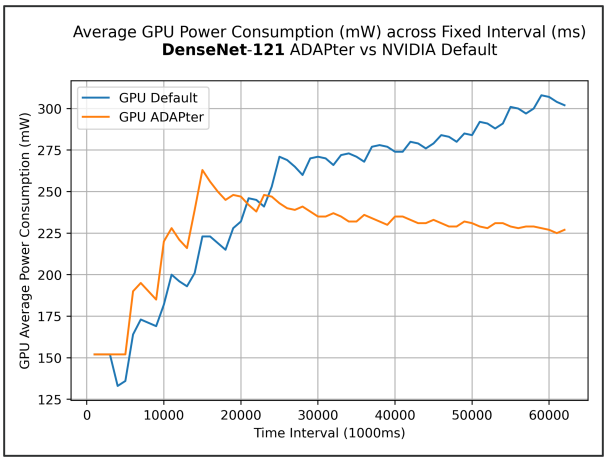


Figure 3: GPU consumption for ADAPter vs NVIDIA default on DenseNet-121

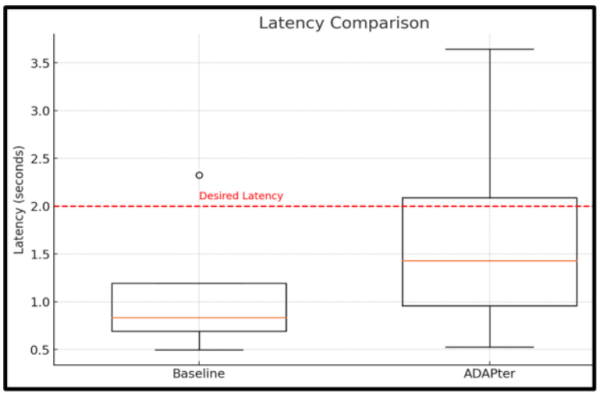


Figure 4: Latency Comparison, ADAPter vs NVIDIA default with target latency

### 4.2 RoBerta-110

In order to test the model agnostic nature of the system, we also tried a large language model called RoBERTa-110, which is almost 4x larger than DenseNet-121 in terms of the number of parameters. Again, we intentionally chose a very achievable SLO to measure the power efficiency of our approach with respect to the baseline. As can be seen from Figure 5, we're 2x more power efficient here as well. Clearly it pays to be adaptive and SLO aware!

While the experiments conducted are very loose with latency constraints to demonstrate the power efficiency of our approach, we can also easily show that with tighter bounds on latency, our system becomes no more power hungry than the baseline, justifying the "adapt" in our ADAPter design and process.

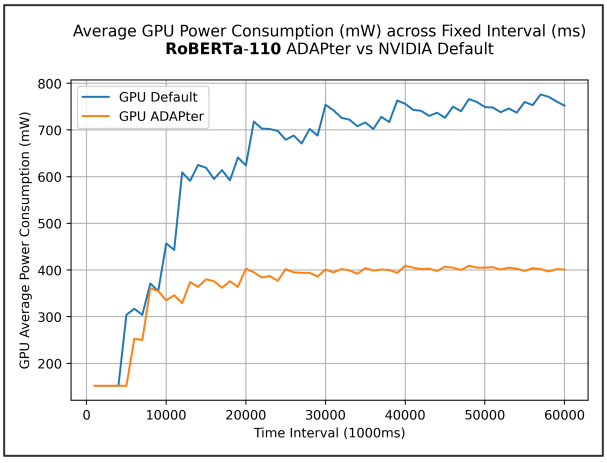


Figure 5: GPU consumption for ADAPter vs NVIDIA default on RoBerta-110

## 5 Related Work

The literature review will categorize existing research into three main buckets: model-level optimizations, energy-efficient AI model training and model selection strategies for inference on general-purpose computing devices as well as popular accelerators. This will allow us to explore the range of methodologies employed and the resulting outcomes achieved. By examining the state of energy efficiency in AI applications, we can lay the groundwork for finding opportunities for mobile efficient inference jobs.

The earliest methods[3] that sought to achieve this relied on static and dynamic optimization techniques. Static techniques such as quantization, pruning, knowledge distillation and collaborative inference reduce time,

memory and energy requirements but they are input-independent causing such deployments to miss the less salient inputs due to their simplified representations. Dynamic optimizations solve this but place the burden of determining the correct technique on the engineer in addition to placing additional demands on memory. Since edge devices are resource constrained, striking a balance between computational efficiency and model accuracy is crucial for successful deployment in edge computing scenarios.

Modern body of work takes a different approach. A recurring theme of papers such as [7, 12, 18] is that although a tradeoff between energy consumption and performance exists, such a tradeoff is not linear. This non-linearity can be leveraged to obtain energy efficient, model training and serving.

Other popular techniques focus on selecting the right model for the right inference target at run time, as in Mobile Deep Inference(MODI)[13] or by performing the inference itself on the cloud, as in CNNSelect[14] . By varying model size and the batching strategy, we're changing the power draw by the core act of inference. However, this does place a requirement for a remotely located repository of models.

While the first group addresses model training which is run only periodically, the second group of papers addresses inference but needs to sync with a remotely located server to get the appropriate model and/or inference results which isn't suitable for the more latency/privacy sensitive workloads.

There is a third group which includes PolyThrottle[17] which addresses both these pain points. It relies on a static optimization approach as opposed to a more adaptive strategy that responds to real-time conditions. This raises the question: why not explore a hybrid strategy that integrates the merits of both approaches?

# 6 Challenges

The project initially aimed to leverage the Zeus package in conjunction with the Qualcomm Innovators Development Kit (QIDK). However, a significant early challenge was acquiring the hardware. Due to nonavailability of the kit, as well as insufficient documentation around power tuning on the Qualcomm device, we pivoted to using an NVIDIA Jetson device.

Next, we encountered challenges in accessing the Jetson device and installing packages on it. These issues were due to the proxy not set up correctly. Once this was resolved, we had access to the device, but faced issues installing the correct version of various packages such as PyTorch and TensorFlow. After going through the Jetson release notes, we were able to locate the model backends and Triton inference server which we set up and used for the project going forward. We continued to face challenges in setting up larger models as they kept crashing. The device had limited storage capacity as well, which made it challenging to load large models and data onto it.

A major aspect of our work centered around changing the device frequency. Changing frequencies is not a deterministic process as the device only provides access to the minimum and maximum allowed frequencies, making it harder to validate accurately in a controlled manner. Another issue encountered was, at higher frequencies, the device very quickly hit a bottleneck due to thermal throttling. At this point, we lost the ability to control frequencies as the device lowered it automatically. We believe providing better cooling and airflow to the device will allow it to perform at peak frequencies for longer, but could not verify this due to not having physical access to the device.

Overall, the project was marked by a series of unforeseen challenges that tested the adaptability and resilience of our research team. From the initial hardware procurement failures to the complexities of modifying and validating device performance under new constraints, each obstacle required innovative solutions and adjustments. This challenges not only tested our technical skills but also demanded a high degree of flexibility and creative problem-solving.

# 7 Future Work

In this research, we have established a foundational approach for optimizing the deployment of machine learning models in real-time applications. Moving forward, several enhancements are envisioned to refine and expand our current methodology. Firstly, an adaptive tuning of operational constants, such as request timeouts and batch sizes, will be crucial. This entails a systematic analysis and dynamic adjustment of these constants based on real-time performance metrics, allowing for optimal responsiveness under varying conditions.

Additionally, a more sophisticated batching strategy is proposed that aims to amortize the costs associated with enqueuing and dequeuing processes within the ADAPter system. By dynamically adjusting batch sizes and processing times based on fluctuations in incoming request volumes, and potentially incorporating predictive analytics, this strategy could significantly enhance system throughput and reduce latency during traffic surges.

Another significant area of future work involves a deeper integration of our operational logic within the architecture of the Triton Inference Server. By embedding our enhancements directly into the server's core func-

tionalities, we can achieve more streamlined data handling and resource management, which are critical for handling diverse and intensive workloads.

Moreover, the implementation of machine learning techniques to determine and adjust the frequency floors and ceilings for each model based on historical performance data represents an innovative approach to resource allocation. This strategy ensures that each model receives adequate resources tailored to its operational demands, thereby optimizing both performance and resource utilization.

Lastly, expanding our model selection process to include the impact of different quantization techniques will further enhance our deployment framework. This involves evaluating not only the suitability of a model for a specific task but also its computational efficiency under various quantization levels. Such an approach will be particularly beneficial in resource-constrained environments, improving both the effectiveness and efficiency of deployed models.

These future directions will not only strengthen the capabilities of our current system but will also make significant contributions to the field of real-time machine learning operations. Through these enhancements, we aim to develop a robust, scalable, and efficient framework for the deployment and management of machine learning models across a diverse range of operational environments.

## Acknowledgements

Our group would like to thank and acknowledge Shivaram Venkataraman and Minghao Yan for their help and support with this project. Shivaram Venkataraman for his guidance and direction, and Minghao Yan for allowing us the use of his system and device, we hope we kept our part of the bargain and didn't change anything that would cause issues for you after we finished. Thank you!

## References

[1] D. Banerjee. A microarchitectural study on apple's a11 bionic processor. *Arkansas State University: Jonesboro, AR, USA*, 2018.

[2] J. Barr. Amazon ec2 update–inf1 instances with aws inferentia chips for high performance costeffective inferencing.

[3] F. Daghero, D. Pagliari, and M. Poncino. Energy-efficient deep learning inference on edge devices. *Advances in Computers*, pages 247–301, 2021.

[4] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo. Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857, 2023.

[5] M. J., L. B., F. N., T. D., G. V., and S. S. Great power, great responsibility: Recommendations for reducing energy for training language models. 2022.

[6] S. Jiang, L. Ran, T. Cao, Y. Xu, and Y. Liu. Profiling and optimizing deep learning inference on mobile gpus. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '20, page 75–81, New York, NY, USA, 2020. Association for Computing Machinery.

[7] A. Krzywaniak, P. Czarnul, and J. Proficz. Dynamic gpu power capping with online performance tracing for energy efficient gpu computing using depo tool. *Future Generation Computer Systems*, 145:396–414, 2023.

[8] F. Laricchia. Number of Mobile Devices Worldwide 2020-2025, March 2023.

[9] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: the laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, page 1–8, USA, 2010. USENIX Association.

[10] A. L.F.W., K. B., and S. R. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. 2020.

[11] D. Li, X. Chen, M. Becchi, and Z. Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom), BDCloud-SocialCom-SustainCom*, pages 477–484, 2016.

[12] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *2012 41st International Conference on Parallel Processing*, pages 48–57, Pittsburgh, PA, USA, 2012.

[13] S. S. Ogden and T. Guo. MODI: Mobile deep inference made efficient by edge computing. 2018.

[14] S. S. Ogden and T. Guo. Characterizing the deep neural networks inference performance of mobile applications. *CoRR*, 2019.

[15] N. Otterness and J. H. Anderson. AMD GPUs as an Alternative to NVIDIA for Supporting Real-Time Workloads. In M. Völp, editor, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[16] P. Sinha, A. Guliani, R. Jain, B. Tran, M. D. Sinclair, and S. Venkataraman. Not all gpus are created equal: Characterizing variability in large-scale, accelerator-rich systems. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 01–15, 2022.

[17] M. Yan, H. Wang, and S. Venkataraman. Polythrottle: Energy-efficient neural network inference on edge devices. *ArXiv*, abs/2310.19991, 2023.

[18] J. You, J. W. Chung, and M. Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 119–139, 2023.

[19] C. V. G. Zelaya. Towards explaining the effects of data preprocessing on machine learning. In *2019 IEEE 35th international conference on data engineering (ICDE)*, pages 2086–2090. IEEE, April 2019.