Universiteit Leiden

Master Computer Science

Frequency- and Temperature-Aware Calibration of NVML Power Measurements

Name:	Setki Fejsko
Student ID:	3676935
Date:	08/12/2025
Specialisation:	Advanced Computing and Systems
1st supervisor:	Ben van Werkhoven
2nd supervisor:	Stijn Heldens

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

Contents

Chapter 1

Introduction

Computers play a major role in our modern society. Business, entertainment, and research are just a few sectors that have been revolutionized and further developed through the integration of computer technology. In particular, supercomputers stand out as a special type of computer system offering significantly higher computing power compared to commodity ones. They are primarily used in computational tasks such as weather forecasting, meteorology, scientific simulations, or training large language models. With the introduction of Graphics Processing Units (GPUs) into supercomputers, it became possible to achieve billions or even trillions of operations per device [1]. GPUs are a type of computer hardware used in commodity computers for graphical computation or general-purpose computing (GPGPU), enabling highly parallel environments with low setup costs. In addition, GPUs are used as accelerators in computer systems to achieve significant speedups.

Although GPUs are very powerful, they also consume substantial energy, which is a primary concern for environmental impact, electricity bills, and overall system performance. In addition, powering all supercomputers in the Top500 [2] is estimated to result in around 2 million metric tons of carbon emissions annually. Aurora [3], the second-fastest supercomputer, is estimated to use 60 MW of power at its peak, which is equivalent to approximately 210 thousand metric tons of carbon dioxide annually. Nowadays, GPUs provide the vast majority of compute power for supercomputers. According to the latest Top500 list, 186 systems use GPUs as accelerators [2]. Hence, optimizing GPUs to be more energy-efficient can significantly reduce their carbon footprint and electricity consumption.

GPU programming enables the exploitation of a GPU's parallel processing capabilities. This process includes writing code in a special programming model that accelerates the computation of computationally intensive tasks. Compared to CPU programming, this process is significantly more complex because it requires a thorough understanding of the device's architecture and programming models. In addition, to develop efficient GPU applications, it is necessary to optimize memory access patterns and maximize the utilization of available cores. However, the variety of different code optimization techniques, along with the complexity of GPU architectures, makes it challenging to manually tune GPU applications for optimal performance and energy efficiency.

Auto-tuning is the technique of automatically determining the optimal GPU code parameters that yield the best performance. Varying combinations of parameter values, such as loop unrolling, block sizes, or vector data types, significantly impact application performance. Therefore, considering every possible combination of parameter values creates a large search space that cannot be explored effectively with manual configuration. Hence, auto-tuners such as Kernel Tuner [4], KTT (Kernel Tuning Toolkit) [5], and CLTune [6] play a crucial role in developing high-performance applications. Furthermore, Kernel Tuner offers tuning for both performance and energy, enabling the development of high-performance or energy-efficient applications.

An important step in optimizing the GPU for energy efficiency is to properly understand the energy consumption of a computational task. Usually, the total energy for a computational task is measured as the integral of power usage over time between two states, where the first state represents the beginning of a computation and the second state represents the end of the computation [7]. Additionally, to accurately measure energy consumption, it is necessary to ensure the power readings are as accurate as possible. Failing to do so results in inaccurate GPU optimization, leading to up to 84% higher energy consumption than with accurate measurements [8].

Two of the most popular techniques for measuring GPU power consumption are external power meters and built-in sensors [9]. High precision and high accuracy make external power meters stand as the ground truth. However, the complex process of building and installing them on each machine poses a significant challenge, especially for large systems. On the other hand, the latest GPUs come with built-in sensors that enable direct power measurement without the need for additional devices. However, a low sampling frequency introduces measurement lags while the GPU is active, thereby affecting the accuracy of energy monitoring. Thus, the average error of power profiles using internal sensors can be as high as 73% with a maximum of 300% [8].

Energy correction techniques emerged to address the issue of inaccurate power readings [10, 11]. These studies proposed calibration models that can be applied to correct power readings, ensuring that data reliability is sufficiently high. However, these methods are not always applicable to all GPU models. The presence of built-in power measurement sensors on most NVIDIA GPUs, combined with NVIDIA's leading market position, necessitates the development of a general calibration approach applicable to all NVIDIA GPUs. Hence, the main contribution of this thesis is to develop a general calibration methodology applicable to NVIDIA GPUs equipped with built-in sensors.

We introduce a calibration method for GPU power using built-in sensors. Our method corrects power based on clock frequency and temperature, treating temperature as a linear function of time and clock frequency as a linear function of power. We estimate energy by combining the corrected power with the kernel execution time.

This thesis tries to resolve the following research questions:

- RQ1 (measurement window): Can we define a method to provide reliable and accurate power measurement through NVML without extensive kernel repetition?
- RQ2 (no external sensors): How to develop a correction model that is not dependent on external sensors and can rely purely on internal sensors?
- RQ3 (auto-tuning impact): Can we reduce auto-tuning time by predicting kernel executions using correction techniques?

We evaluate our NVML-based calibration methodology on GEMM and FP32 kernels using an

NVIDIA RTX 4000 Ada. Initially, we executed GEMM and FP32 with a single set of parameters and corrected based on clock frequency and temperature. Although the initial results were promising, and our methodology kept the error within 5% of the ground truth data, extending the experiments to a more realistic auto-tuning setup revealed a higher error rate. Our results showed that each GPU configuration creates a distinct power and time profile; therefore, using a single power and time profile across the search space is not efficient. Our initial idea was to define a single power and time profile from a single configuration and use our correction method to predict power and energy across the entire search space. However, in this empirical setup, a single trained model did not generalize well to the entire search space. In fact, it resulted in a high error rate, especially during energy modeling, where we need to predict both power and time.

The remainder of the thesis is organized as follows: In Chapter 2, we provide technical background. Then, in Chapter 3, we review related work on GPU power measurements and calibration. We then present the proposed methodology in Chapter 4, which includes corrections based on clock frequency and temperature. Next, we evaluate our methodology using different experimental setups in Chapter 5. We conduct experiments using samples from both the training set and the test set. Additionally, we test our methodology with generalized auto-tuning examples. After each experiment, we discuss the results and drawbacks of our methodology. Finally, in Chapter 6, we conclude with implications and future directions.

Chapter 2

Background

This chapter provides the background information necessary to understand the contributions of this thesis. We begin with an overview of GPU architecture in Section 2.1, followed by its programming model, CUDA, in Section 2.2. Next, we discuss auto-tuners in Section 2.3. Finally, Section 2.4 examines different techniques for measuring GPU power.

2.1 GPU architecture

Although the original purpose of GPUs was graphics rendering, they quickly evolved to become the primary component for accelerating scientific simulations, large-scale data processing, and complex calculations in an extremely efficient manner. The design of GPU architecture enables an extremely parallel environment consisting of a memory hierarchy and processing units. In contrast, the traditional CPU architecture is more suitable for sequential applications, which offer low-latency execution. The GPU manages to hide latency by executing millions of threads concurrently on different data elements. Figure 2.1 shows a simplified comparison between CPU and GPU. The CPU architecture typically contains more control and memory units, whereas the GPU consists of more arithmetic and logical units (ALU).

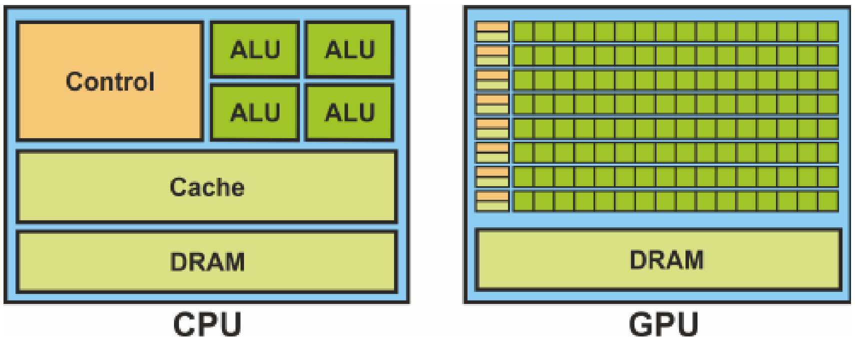


Figure 2.1: CPU vs GPU architecture [12].

The GPU architecture groups processing cores into streaming multiprocessors (SMs) to effectively manage the hardware while delivering maximum performance. Each SM consists of several processing cores (SPs). Each SM has its own control unit and memory space. At the lowest memory level are register files allocated to each thread; shared memory is a bigger memory that allows for sharing data between threads inside one SM. Additionally, global memory is used to share data across thread blocks or SM. Furthermore, modern GPUs feature various levels of caches alongside specialized processing cores, such as ray tracing cores for graphics rendering and tensor cores for training neural networks. Figure 2.2 presents an abstract view of GPU architecture.

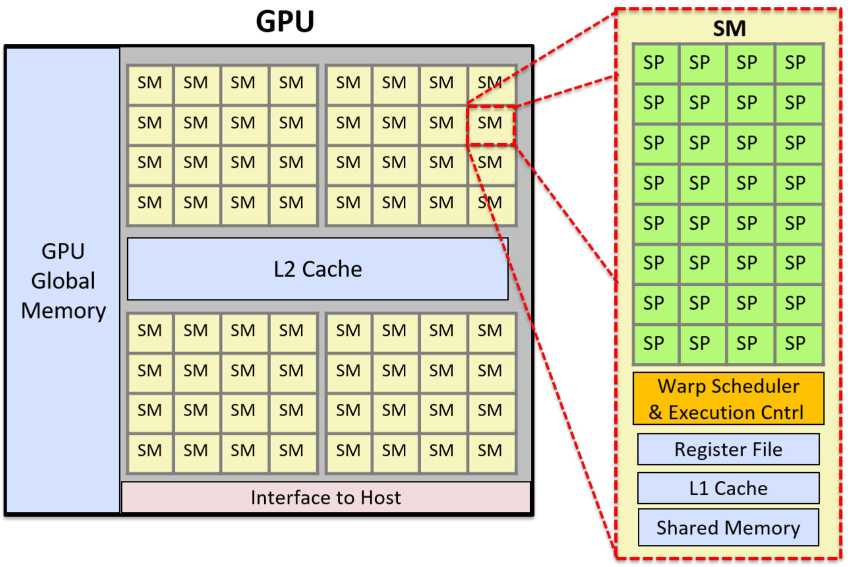


Figure 2.2: Abstract view of GPU architecture with 2048 cores. For this architecture, the GPU is divided into 64 Streaming Multiprocessors (SM) and each SM consists of 32 processing cores [13].

2.2 CUDA programming language

CUDA (Compute Unified Device Architecture) stands as the main interface for computing on NVIDIA GPUs. It is a parallel computing platform that enables software to utilize GPU hardware features to accelerate computational tasks. Currently, CUDA is available in two levels: the runtime API and the driver API. The CUDA runtime API offers easier device management with simpler code, but trades off some of the fine-grained control over the hardware. In contrast, the driver API provides more control over the hardware, but the code is significantly more complicated [14].

The GPU functions as a co-processor controlled by the CPU, as the workload is divided into host and device code. The CPU executes the host code, which transfers data to and from the GPU and between the GPU and the host. In contrast, the device code refers to highly parallel functions called kernels that are mapped to and executed by GPU hardware components. These kernels are often written in C, C++, or Fortran, and when a kernel is launched, CUDA organizes its execution into a grid of thread blocks. Each thread block contains a user-specified number of threads. Threads within a block can communicate via shared memory and synchronization, whereas blocks are independent and may be scheduled in any order of the GPU's SM. Moreover, grids can be 1-, 2-, or 3-dimensional, and the launch configuration of grid and block sizes determines the total number of threads. The number of threads can vary from thousands to millions, depending on the GPU architecture and hardware capabilities. The primary goal is to select grid and block dimensions that keep the hardware busy, which means efficiently assigning meaningful work to all threads to maximize parallelism. An abstract overview of the CUDA programming model is given in Figure 2.3.

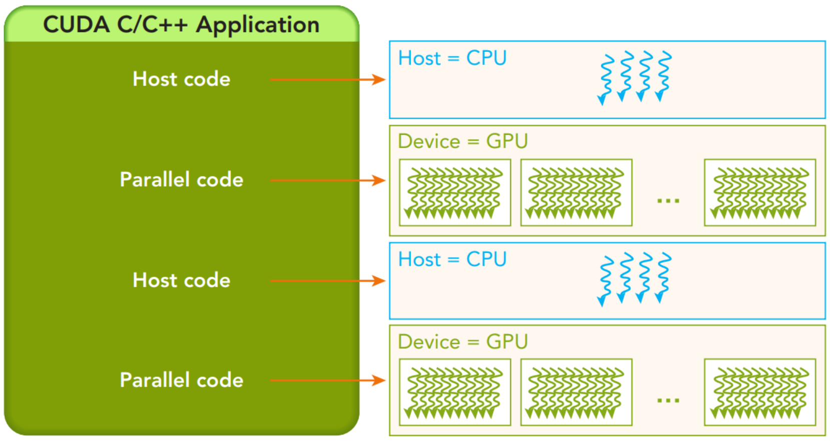


Figure 2.3: Host code with blue rectangle represents the CPU and parallel code with green rectangle represents the GPU [15].

2.3 Auto-Tuning GPU kernels

The performance of a GPU application can vary significantly depending on kernel parameter configurations, such as thread block size, grid size, memory pattern, unrolling factor, and work per thread. Additionally, for each application and GPU architecture type, different parameter configurations maximize device performance and occupancy compared to other configurations. It is feasible to try all combinations manually when the number of parameters is low. However, this approach becomes impractical for optimized applications that contain hundreds or even millions of combinations. Thus, it is essential to use auto-tuning to efficiently search for the best parameters in this vast optimization space.

Auto-Tuners like Kernel Tuner [4], CLTune [6], or KTT (Kernel Tuning Toolkit) [5] require users to define the kernel, parameters, and restrictions, and for each combination, a new GPU application is compiled, executed, and benchmarked. All benchmarked results are stored and compared during the tuning process, and the best-performing combination is identified. This methodology allows the investigation of different aspects of an application and is more flexible with different architectures.

In addition, searching the entire optimization space is not necessarily the most optimal solution, as optimization strategies such as Bayesian Optimization, the Firefly Algorithm, or the Genetic Algorithm can be used to speed up the tuning process.

2.3.1 Kernel Tuner

Kernel Tuner is a generic auto-tuner that provides a very simplified interface for identifying optimal kernel configurations. This Python-based tool allows a smooth tuning process by just providing kernels written in any language, tunable parameters, code restrictions, and metrics. The automation procedure then handles the compilation, benchmarking, and result analysis. In addition, Kernel Tuner does not require extensive changes in the kernel code before or after tuning. It offers GPU code testing in isolation within Python, provides the requested analysis, and allows the efficient kernel with the best-performing configuration to be used in production without complex code changes.

Listing 2.1 demonstrates a basic tuning example in Python that performs element-wise addition of two vectors. This code defines a CUDA kernel and the necessary parameters to tune its performance across different block sizes. This simple unit testing creates two input arrays, a and b, and a zeroed array, c, to store the output. Note that the number and the order of arguments should match those defined in the kernel. Finally, by calling the function 'tune_kernel', the Kernel Tuner initiates the tuning process by compiling and benchmarking the kernel with the same arguments but different block sizes.

The Listing 2.1 focuses on kernel performance; however, Kernel Tuner also offers energy tuning. Tuning for energy enables us to find the most efficient parameters that use less energy while not trading a lot of performance [7]. Kernel Tuner measures the energy and temperature of the GPU state through the observer module. The observer module extends the benchmarking process by monitoring various aspects of kernel activity and GPU state measurement. In this study, we use NVMLObserver to enable interaction with NVML and PowerSensorObserver for PowerSensor3 [16], the external sensor, to validate our data.

```python
import numpy
from kernel_tuner import tune_kernel

kernel_string = """
__global__ void vector_add(float *c, float *a, float *b, int n) {
    int i = blockIdx.x * block_size_x + threadIdx.x;
    if (i<n) {
        c[i] = a[i] + b[i];
    }
} """

size = 10000000

a = numpy.random.randn(size).astype(numpy.float32)
b = numpy.random.randn(size).astype(numpy.float32)
c = numpy.zeros_like(b)
n = numpy.int32(size)

args = [c, a, b, n]

tune_params = dict()
tune_params["block_size_x"] = [32, 64, 128, 256, 512]

tune_kernel("vector_add", kernel_string, size, args, tune_params)
```

Listing 2.1: Vector add kernel in CUDA implemented with Kernel Tuner [4].

2.4 Measuring GPU power

Understanding GPU power consumption has been a major challenge in academia, often described as a "true black box" by researchers [17], especially with the introduction of DVFS (Dynamic Voltage and Frequency Scaling), where clock frequency and voltage are adjusted in real time according to GPU workload [18]. Thus, various methodologies have been introduced and proposed to address this problem. In this section, some well-known techniques are described. Respectively, predictive models, analytical models, external power sensors, and built-in sensors.

2.4.1 Machine Learning models

With the rise of machine learning, researchers quickly developed models to predict GPU power consumption. The main idea is to gather data while the GPU is active and train machine-learning algorithms based on factors such as code analysis, instruction analysis, or performance counters. This model can then be used to predict the energy usage of a new kernel without having to run it.

Gene et al. [19] proposed a GPU performance and power model that relies on a machine learning classifier. Instead of focusing on a single workload, they gathered data from multiple OpenCL [20] kernels and varied different aspects of GPU hardware configurations, such as memory bandwidth, various core frequencies, compute unit counts, and performance counters, to explore several cases. Additionally, the data were divided into different clusters, and a neural network was trained to predict the unseen kernels. The proposed techniques have an average error rate of 15% for estimating performance and 10% for estimating power consumption compared to an external sensor.

Nagasaka et al. [21] trained a regression model using performance counters as independent variables and power consumption as the dependent variable. Additionally, the model was evaluated on 49 kernels from the CUDA SDK and the Rodinia dataset [22]. The regression model has an average error rate of 4.7%, but it was evaluated only on GeForce GTX 285. Braun et al. [23] based their study on a random forest classifier, a powerful machine learning technique, where the outcome depends on multiple decision trees. Compared to Nagasaka et al. [21], they included multiple datasets for training the model and utilized five different GPUs for evaluation. Regarding predicting power consumption, the evaluations yield between 1.84% and 2.94% Mean Average Percentage Error (MAPE). However, the quality of the dataset is not considered nor validated with external sensors.

Although fast and easy to predict power usage for new kernels, prediction models lack accuracy. The error can occur for various reasons, such as difficulties in modeling and predicting the dynamic power of the GPU, variations in the GPU architecture, or even variations in kernels. This can have a huge impact when dealing with a large HPC system that consists of hundreds of nodes.

2.4.2 Analytical model

Another methodology of prediction approach is analytical models. Even though they share the same end goal of defining a predictive model, it is a misconception to assume that machine learning and analytical models are the same. The analytical model is based on performance modeling of the combination of software and hardware to understand the energy consumption of the system, whereas machine learning predicts performance automatically using computers. Moreover, machine learning can adapt faster and better to new data trends.

Hong and Kim [24] defined an integrated power and performance (IPP) analytical model for a GPU. The model tends to predict the optimal number of active GPU cores for a specific kernel. While the kernel is reaching the peak memory bandwidth, adding more cores does not have any significant effect on performance, thus, using only the optimal number of cores can save up to 22.09% of GPU energy consumption [24]. Additionally, the model does not depend on measured execution times, architecture simulations, or performance counters but determines dynamic power events by predicting execution times. However, this approach results in an average error rate of 8.84%.

2.4.3 External power sensors

While predictive models lack accuracy, external hardware sensors are the most precise tools for measuring power consumption. High-resolution rates, high sampling rates, and direct current readings result in the lowest error. Moreover, these tools are installed between the power source and the targeted device, ensuring that such high-quality measurements are usually considered ground-truth data.

PowerSensor3 [16] offers instantaneous readings of the power consumption of PCIe card connections for GPUs, FPGAs, and Xeon Phis. The high time resolution provides data at a sub-millisecond scale, enabling in-depth analysis of kernel execution. It is a custom-built tool with a very low cost that consists of current and voltage sensors connected through a commodity microcontroller, which enables a 1% error rate with proper calibration. It comes as an upgraded version of PowerSensor2 [25].

PowerInsight [26] is another example of a custom external device that offers measurements of power and energy at the component level. This device is capable of measuring any commodity hardware in a cluster setup by simply connecting it between the main power supply and the targeted device. This device offers data collection and analysis from the same computer (in-band) or on a different machine (out-of-band), making it an excellent option for large distributed HPC systems. The validation experiments resulted in an error state of less than 0.3%.

External power sensors are, in fact, more accurate than any other measurement method. However, manual hardware installation, firmware installation, sensor connection, and soldering components are necessary for each targeted device. Hence, not everyone is capable of installing such devices, and making such changes is not permissible in every scenario. Finally, if we consider large cloud vendors' machines or supercomputer setups that contain thousands of these devices, the scale of manual installation for each of them is not feasible.

2.4.4 Internal sensors

Prediction models are easy to use but contain inaccuracy, whereas external power sensors provide high accuracy but require extra hardware and installation steps. Internal sensors stand as a balanced solution that combines ease of use with reasonable accuracy. These sensors are already included in GPU devices, providing direct measurements of hardware activity. In particular, NVIDIA started introducing built-in sensors from the Kepler architecture.

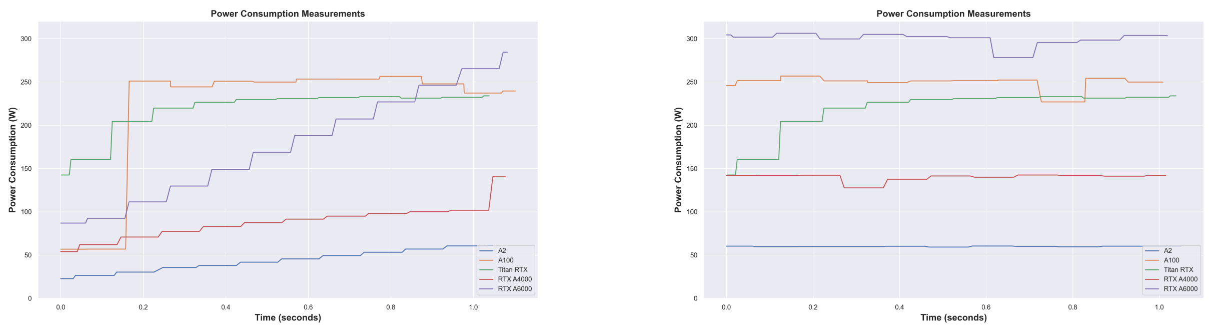
NVML (NVIDIA Management Library) [27] and nvidia-smi (NVIDIA System Management Interface) [28] are two utilities used to query, monitor, or manage device hardware properties. These tools are part of the device driver and can be used to interact with the built-in sensor. In the newest documentation of nvidia-smi [28], there are 2 options related to power readings:

1. **power.draw.average**: Provides the Average Power Consumption of the GPU for 1 second

2. **power.draw.instant**: Provides Instantaneous Power samples of the GPU every 25 milliseconds

Accuracy is a concern with this type of sensor, especially if it is not handled correctly; the error can exceed that of the prediction models. This error is primarily introduced by the low sampling rate, which makes it challenging to capture the dynamic power of device activity. Additionally, there is often uncertainty about how these sensors are manufactured, how they capture data, and the quality of their documentation is low.

Figure 2.4 shows the power readings of the Matrix Multiplication kernel executed continuously for 1 second, measured with the average option (left) and the instant option (right). Since we are dealing with short-running kernels, we can observe that the average option 2.4a is still capturing the rising power of the GPU. Hence, we observe this staircase effect rather than a plateau. This introduces a high error rate for kernels that have short execution times. On the other hand, instant readings 2.4b clearly solve the staircase problem, which, instead of averaging samples over a 1-second range, tries to provide the instantaneous power recorded through the GPU sensor. Despite their accuracy, it is not recommended to rely solely on sensor data without further analysis or processing, even with instant readings. Hence, we believe that with a proper strategy, the error percentage can be reduced to a level where data accuracy is sufficient, while also providing fast measurements for auto-tuners. Since the sensor is already included in the device, it is more applicable in HPC or cloud environments.



(a) power.average (b) power.draw.instant

Figure 2.4: Average power measurements (left), Instant power measurements (right).

Chapter 3

Related Work

This chapter provides an overview of prior research on GPU power modeling and energy measurements. We begin in Section 3.1 with studies that combine power models with GPU simulators. Section 3.2 then discusses prediction methods based on linear and non-linear regression. Following this, Section 3.3 presents methodologies that apply models to steer auto-tuners for power or performance. Finally, Section 3.4 examines different approaches for correcting power measurements collected from internal GPU sensors.

3.1 GPU simulator

Chen et al. [29] designed a tree-based random forest model. This model captures the correlation between power consumption and performance variables. To further extend the relationship between these variables, Chen et al. [29] applied a GPU simulator to collect additional insights. Leng et al. [30] defined a power model with microarchitecture components and integrated this model into a power simulation framework called GPUWatch. Although their simulation method is useful for modeling different architectures, the average error ranges from 9.9% to 13.4%. The inaccuracy of GPUWatch increased further with new GPU architectures like Volta and Turing; therefore, Kandiah et al. [31] introduced the AccelWattch power modeling framework. The proposed power model accurately captures static, dynamic, and constant power consumption by combining SM occupancy, thread divergence, power gating, and intra-warp functional units for a given kernel. In addition, collecting measurements with nvidia-smi or NVML and integrating the framework with GPGPU-Sim and Accel-Sim makes AccelWattch a reliable choice for predicting the performance and power consumption of new GPU architectures. Nevertheless, regarding validation, the average error is within 7.5–9.2 $\pm$ 2.1–3.1% tested on only the NVIDIA Volta Quandro GV100 GPU.

3.2 Regression model

Shalvai et al. [32] used external hardware to collect accurate measurements and trained a regression model. The goal of this model was to properly calibrate the internal GPU sensor. They targeted only the NVIDIA Jetson, and the calibration managed to keep an error state of 3%. Ghosh et al. [33] developed a non-linear regression model to estimate kernel power and energy consumption. Their study aims to investigate the relationships among power, energy, hardware counters, and GPU kernel stats such as operations per second. This approach was tested in a multi-GPU platform, which resulted in a 5% average error. Ma et al. [34] collected data using FLUKE 2680 A, a specialized system for capturing power and GPU run-time workload signal, and NVIDIA PerfKit, a tool for analyzing kernel performance. In addition to the acquired dataset, they trained a regression model and validated it across eight workloads, achieving error rates ranging from 1.7% to 39%.

3.3 Model steered auto tuning

Cheema and Khan [35] developed the Multi-Objective Kernel Auto-Tuner (MOKAT) framework, which employs optimization techniques to improve the power consumption and performance of GPU kernels. Multi-objective optimization involves optimizing multiple objective functions, such as power, performance time, or energy. The defined performance and energy models use measurements from the internal GPU sensor to steer the optimization process. Like Kernel Tuner 2.3.1, this framework creates the search space by combining all values of the given variables but targeting only OpenCL applications. The framework is evaluated and tested on a Tesla GPU using a highly tunable 2D convolution kernel [36], which includes various tunable variables, such as work per thread, caching, and the loop unrolling factor. The optimal parameter values for the 2D Convolution kernel managed to reduce the power consumption by 30 % and a 4% decrease in execution time compared to non-optimized configurations.

Similar to our findings, Schoonhoven et al. [7] observed the power lag problem of NVML. They executed a matrix-multiplication kernel on the Titan RTX, NVIDIA A100, and NVIDIA A6000. Titan RTX and A100 power measurements stabilized after 0.3 seconds, while the A6000 measurements gradually increased until they reached their maximum device power. They showed that the reported time-averaged from NVML has a relatively low frequency; thus, the power lag is present. To solve this issue, they suggest running the kernel multiple times and reporting the aggregated new values as the final power and energy consumption. Additionally, several aspects of energy tuning are discussed. Techniques such as adjusting the device clock frequency or power capping were compared while also developing a GPU power model. This model defines an efficient range of device clock frequencies, thereby speeding up the tuning of the optimization space. From the experimental results, they found that trading 27.5% of performance can lead to 50.9% better energy efficiency for the matrix multiplication kernel. The proposed model significantly reduced the search space by 77.8% to 82.4%. Finally, all these new optimization and monitoring features are included in the Kernel Tuner.

3.4 Correction method

Burtscher et al. [10] were the first to dive into the GPU internal sensor. They observed various anomalies during the sampling of power consumption. False data, delayed measurements, and low sampling rates are among the reasons why the kernel activity did not match the expected square wave shape as observed with the external power sensor [37]. All measurements were collected through NVML (NVIDIA Management Library), and a calibration model was proposed that computes the true power and energy consumption. This model relies on the capacitance constant, which describes the capacitors' charging and discharging effects, thereby minimizing the error between the modeled data and the true power consumption. However, this capacitor constant is not universal for all architectures; therefore, a new value must be determined for each different GPU architecture. The authors did not mention a proper strategy for defining this constant, nor did they validate the experimental results with ground truth data. Finally, they targeted only the Tesla GPU family and used only the n-body algorithm throughout their study.

Aslan et al. [11] noticed the same power lag problem with NVIDIA Jetson, which are low-power consumption devices and are mostly used as edge devices. This study demonstrated that the capacitor model by Burtscher et al. [10] is not the general solution for all GPU architectures. The authors had to define a new value for the capacitor constant and noticed a higher error rate in the trailing and ending edges from the measurements. The resolution rate of the built-in sensor caused this wobbling effect on the edges. To address this issue, the data were filtered using a 9-point Moving Average Filter (MAF) to remove some noise, followed by the application of the capacitor model to define the true power. Despite lowering the error rate, they experimented only with Jetson devices and used only the Matrix Multiplication algorithm (MUL).

In a recent study, Yang et al. [38] conducted a detailed investigation of NVIDIA's internal sensor. Their study included experiments to properly define the number of kernel repetitions, the rise time, and the steady-state of average and instantaneous power readings, and to emulate boxcar averaging, a technique that averages results over multiple samples. Their study included over 70 types of GPUs, spanning different architectures. Moreover, they utilized an external sensor, the Power Measurement Device (PMD) [39], to validate the results. To keep the error state within 5%, the authors suggest following these steps when measuring power with nvidia-smi:

1. Run the application for 32 executive runs or at least 5 seconds.
2. Conduct four separate trials.
3. Post-process data and shift the data with the proposed offset from their result.

This work shows that NVIDIA is finally acknowledging the issue with average readings, where the error was significantly higher. However, despite this admission, their documentation still claims the error is within 5 watts rather than 5%. This claim can be dangerous because, in a large data center, it can result in millions of dollars in additional electricity bills, as reported by Yang et al. [38].

3.5 Summary

To sum up everything that has been stated so far, the work of Chen et al. [29], Leng et al. [30], and Kandiah et al. [31] are highly dependent on the GPU simulator. On the other hand, Shalvai et al. [32], Ghosh et al. [33], and Ma et al. [34] used statistical approaches to predict GPU activity. Cheema and Khan [35] and Schoonhoven [7] focused on steering auto-tuners to define the optimal power or performance approach. Finally, Burtscher et al. [10], Aslan et al. [11], and Yang et al. [38] focused on providing new insights or refining techniques for nvidia-smi or NVML.

The main difference and one of the biggest challenges of our study is working with low-resolution measurements. Most previous work overcomes this obstacle by increasing the kernel runtime to seconds or hundreds of iterations while constantly sampling. However, this is not feasible in our case because tuning a large search space can last months of benchmarking. Additionally, our methodology will enable a more general solution that is not dependent on external devices or machine learning models, but rather focuses on calibrating the built-in device sensor's instantaneous power readings.

Chapter 4

Methodology

This chapter presents the main contribution of our work. It defines the primary methodologies used to enhance the accuracy of GPU power measurements utilizing internal sensors. The main goal is to adjust power data obtained through NVML by modeling and correcting for two critical influential factors: temperature and clock frequency. In section 4.1, we model and correct the effect of clock frequency on power. Next, in Section 4.2, we model and correct temperature-induced bias. Finally, in Section 4.3 we elaborate on various smoothing techniques to remove noise captured during the tuning process.

4.1 Clock Frequency correction

GPU power consumption is heavily influenced by clock frequency. To account for this effect, a modeling-based approach is applied using parameterized fits derived from power and frequency measurements recorded with Kernel Tuner.

4.1.1 Dependence of Power on Clock Frequency

The relation between clock frequency and power was modeled in the form used in Schoonhoven et al. [7]. The following Equation was fitted to model power consumption as a function of clock frequency.

$$P_{\text{estimated}} = P_{\text{static}} + a \cdot f \cdot V^2 \tag{4.1}$$

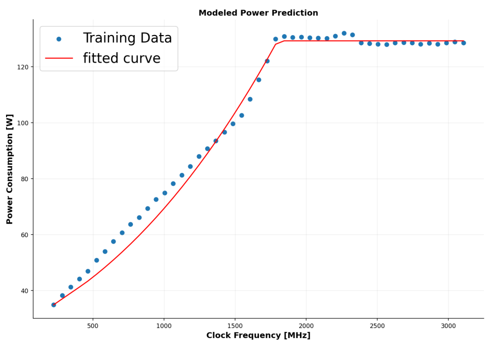
Where:

1. $P_{estimated}$ is the estimated power
2. P_{static} is the idle power consumption
3. a is the constant
4. f is the clock frequency
5. V is voltage

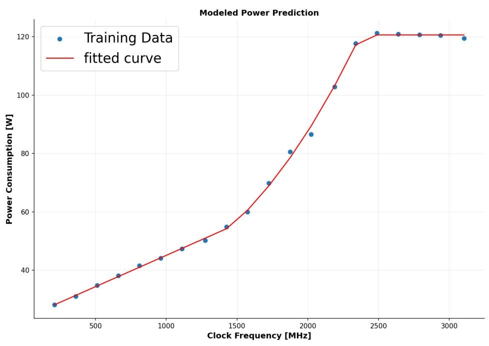
4.1.2 Frequency Correction Model: Fitting and Application

We evaluate the correlation between clock frequency and power with two different kernels: GEMM and FP32, which we shall introduce in more detail in Section 5.2. For GEMM, we use the parameters in Table 5.2, whereas for FP32, we use the parameters in Table 5.3. The only difference in parameters is that, instead of varying `REPEAT` from 1 to 2000, we fix the clock frequency at several increasing values to isolate the frequency–power correlation.

The Figure 4.1 shows the fit of Equation 4.1 to the measured frequency–power data. The fitted curve captures both the rising slope of power consumption with frequency and the eventual saturation region, reflecting the GPU's hardware limits, where additional clock frequency requests do not translate into increased power. A similar observation was reported by Schoonhoven et al. [7].



(a) GEMM.



(b) FP32.

Figure 4.1: Clock Frequency vs. Power correlation. The red line represents the fitted model described in Equation 4.1.

The fitted model provided an estimated power curve, $P\hat{}(f)$, describing the expected GPU power at any frequency f. For any chosen reference frequency f_{ref}, the corresponding modeled power is $P_{\text{ref}} = P\hat{}(f_{\text{ref}})$. As we describe in Section 5, for our experiments, we evaluate both an in-sample reference and an out-of-sample reference from data described in Figure 4.1.

For each kernel run at measured frequency f_k, the power measurement reported by NVML $P_{\text{meas}}(f_k)$ was corrected to the reference frequency by subtracting the modeled power and adding back the modeled value of the reference frequency f_{ref}.

$$P_{\text{corr}}(f_k) = P_{\text{meas}}(f_k) - P\hat{}(f_k) + P_{\text{ref}} \tag{4.2}$$

Where:

1. $P_{corr}(f_k)$ is the corrected power
2. $P_{meas}(f_k)$ is the measured power at clock frequency f_k
3. $P\hat{}(f_k)$ is the modeled power from equation 4.1
4. P_{ref} is the modeled power at the reference frequency f_{ref}

Additionally, the corrected energy was obtained by multiplying the corrected power by the measured execution time Δt:

$$E_{\text{corr}}(f_k) = P_{\text{corr}}(f_k) \cdot \Delta t \tag{4.3}$$

This correction removes frequency-driven biases while preserving workload- and temperature-related variations. Hence, it enables fair comparison across kernel executions performed at different DVFS (Dynamic Voltage and Frequency Scaling) states.

Furthermore, Figure 4.2 illustrates the relationship between GPU clock frequency and kernel execution time. Modeling this relationship is necessary because, for unseen clock frequencies, execution time must be estimated to predict total energy consumption. The fitted curve uses the Equation 4.5, and it captures how execution time generally decreases as frequency increases, until it reaches a saturation point where requesting a higher frequency no longer reduces runtime. This execution-time model is then combined with the corrected power values to compute energy, as implemented in the correction step:

$$E_{\text{model}} = P_{\text{corrected}} \cdot T_{\text{model}}(f) \tag{4.4}$$

In this work, we propose to model the execution time T_{model}(f) using Equation 4.5. This functional form was chosen because it captures both the rapid initial decrease in execution time as frequency increases and the eventual plateau at higher frequencies.

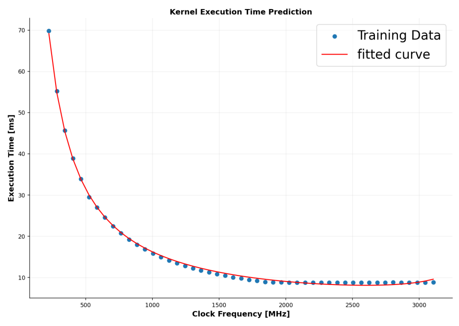
$$f(x) = \frac{a}{1 + b \cdot f} + \frac{c}{1 + d \cdot f} \tag{4.5}$$

Parameter Interpretation:

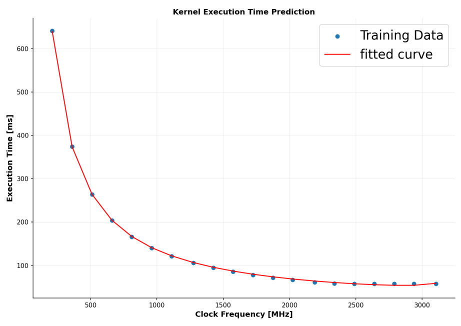
1. a and c represent the contributions to execution time from different bottlenecks (e.g., compute-bound and memory-bound components)
2. b and d determine how quickly each component decreases as frequency increases.
3. f represents the measured clock frequency

4.2 Temperature correction

Accurate GPU power modeling must account for thermal effects. Even with high-resolution external sensors, temperature variations across runs can bias energy measurements. In the following subsection, we outline the methodology for isolating and modeling temperature correction. Temperature plays a crucial yet often underestimated role in GPU power behavior. To understand its influence and validate the need for temperature correction, we show a series of experiments under varying thermal and frequency conditions. We observed that the correlation between power and temperature depends strongly on how the GPU clock frequency is managed. Specifically, whether it's left in default DVFS mode or fixed manually.



(a) GEMM.

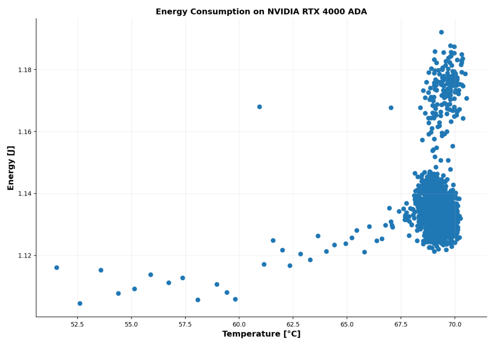


(b) FP32.

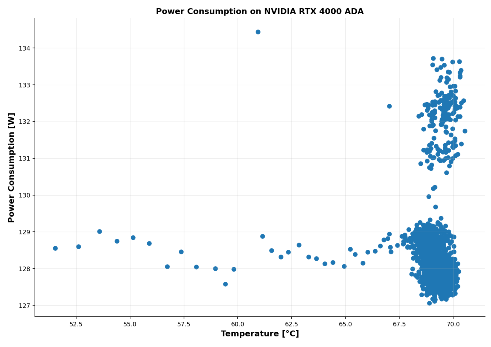
Figure 4.2: Relationship between Clock Frequency and Execution Time for GEMM kernel (left) and FP32 (right). The red line represents the fitted rational model defined at Equation 4.5.

4.2.1 Weak Temperature Correlation under Default Clock Frequency

In the default configuration (DVFS enabled), the GPU dynamically adjusts its clocks to match the workload. Figure 4.3 shows the temperature reported by NVML versus energy (left) and power (right). The data seem to be scattered and weakly correlated, with no clear trend. With DVFS active, frequent changes in core frequency and voltage mask the correlation between temperature and power.



(a) Temperature vs Energy.



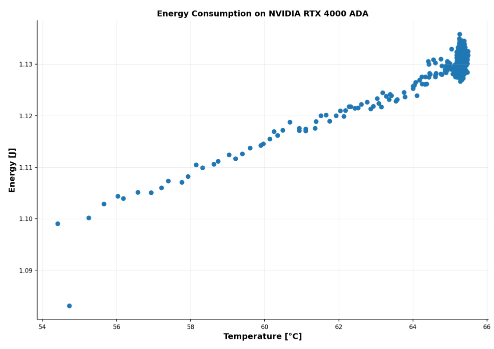
(b) Temperature vs Power.

Figure 4.3: Relationship between Temperature vs Energy (left) and Temperature vs Power (right) for GEMM kernel on NVIDIA RTX 4000 ADA GPU.

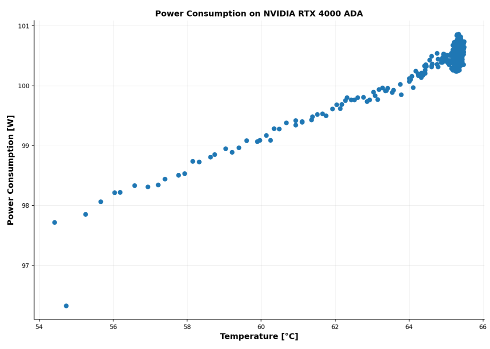
The scatter plots indicate a lack of a consistent relationship between temperature and power or energy due to the dynamic frequency behavior of the GPU. As the temperature increases, the power and energy values fluctuate significantly, and these fluctuations are not consistent across the temperature range. This inconsistency arises because the GPU dynamically adjusts its clock frequency, which introduces variability in power and energy consumption.

4.2.2 Strong Linear Correlation at Fixed Clock Frequency

To properly isolate the impact of temperature on power, we manually fixed the GPU clock to a stable clock frequency of 1425 MHz. This enabled us to remove the effects of both clock frequency and voltage scaling, ensuring that our measurements reflect the impact of temperature alone. Under these assumptions, Figure 4.4 shows a strong linear relationship between temperature and both power and energy, indicating that thermal effects become observable only when other variables are held constant.



(a) Temperature vs Energy.



(b) Temperature vs Power.

Figure 4.4: Relationship between Temperature vs Energy (left) and Temperature vs Power (right) for GEMM kernel on NVIDIA RTX 4000 ADA GPU with fixed clock frequency 1425 MHz.

The power (or energy) increases at a steady rate with rising temperature, and a linear model provides a good fit to the data. The slope of this trend line, denoted α, represents the additional power drawn per increase in GPU temperature of °C. This slope coefficient is used in the temperature-correction model introduced in Section 5.

4.2.3 Temperature Correction Model: Fitting and Application

At a fixed GPU frequency, a clear linear correlation was observed between die temperature and NVML-reported power, consistent with temperature-sensitive leakage currents and related effects in the GPU power circuitry [24]. To quantify this relationship, we fit a linear model to estimate the slope of the line.

$$\alpha = \frac{\Delta P}{\Delta T} \text{ [W/°C]} \tag{4.6}$$

Equation 4.6 represents the change in power per degree Celsius. Using this fitted model, all power readings were normalized to a chosen reference temperature T$_{\text{ref}}$. This reference is typically set to the lowest temperature recorded during auto-tuning to model measurements in the GPU's cold state. Hence, the final temperature correction formula takes the form:

$$P_{\text{corrected}} = P_{\text{measured}} - \alpha \cdot (T - T_{\text{ref}}) \tag{4.7}$$

Where:

1. $P_{corrected}$ is the corrected power
2. $P_{measured}$ is the measured power
3. α is slope from regression
4. T is the run temperature
5. T_{ref} is the reference temperature

This normalization procedure ensures that measured power values are directly comparable between runs executed at different thermal states, isolating workload-related variations from temperature-induced bias.

4.3 Smoothing techniques

In addition to clock frequency and temperature correction, we investigated signal smoothing techniques. To improve the clarity and reliability of the power readings from NVML, several filtering techniques were applied to smooth out the inherent noise and fluctuations in the instantaneous power readings. Specifically, we selected three lightweight, commonly used filters: Moving Average Filtering (MAF), GH filter, and a one-dimensional Kalman filter, because they are simple to implement and run in constant time per sample, making them suitable for real-time data processing.

4.3.1 Moving Average Filtering

A standard moving average filter was applied with a sliding window size of 9, which corresponds to approximately 80 ms of data. The formula for a window size n is:

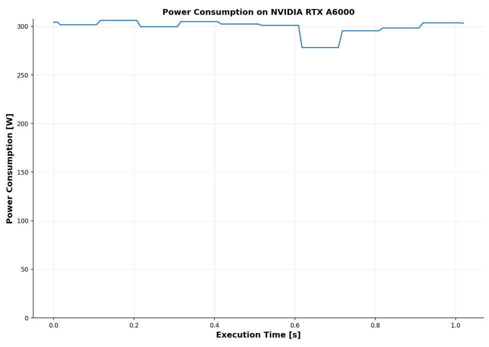
$$y_i = \frac{1}{k}\sum_{r=0}^{k-1} x_{i+r}, \quad i = 1, \ldots, N-k+1 \tag{4.8}$$

In the equation 4.8, x_i is the input power and y_i is the filtered output. We plot the filtered signal in Figure 4.5, where we can see the difference between the raw power measurements on the left and the filtered signal on the right. The moving-average filtering works by averaging the power readings over a sliding window of fixed size $n = 9$, which reduces high-frequency noise.

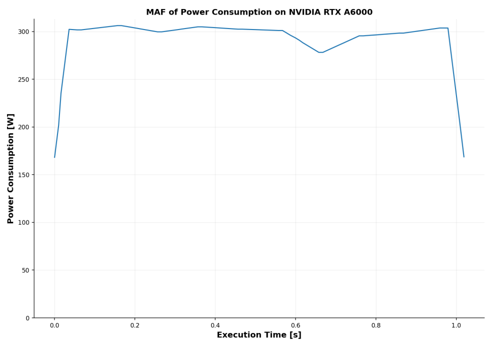
4.3.2 GH Filter

An adaptive GH (or alpha-beta) filter was evaluated, inspired by the FilterPy framework [40]. We set the following parameters, and for each measurement, we apply the GH filtering.

- Initial state x (e.g., initial power): The starting estimate of the signal level. In our case, we set it to the first NVML power sample.
- Initial rate $\dot{x}$ (dx): The starting estimate of the time derivative of the signal (rate of change). We set $\dot{x}_0 = 0$ to assume no initial trend, the filter quickly adapts this from the data.



(a) Original measurements.

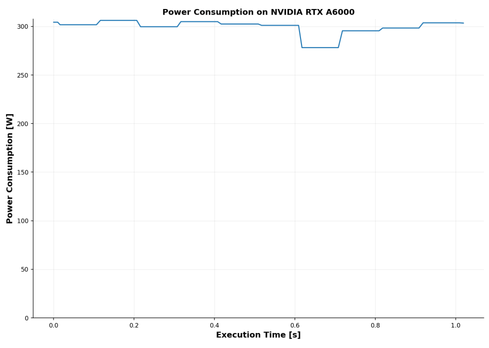


(b) Filtered signal.

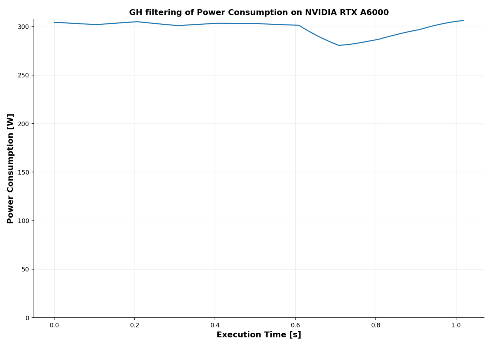
Figure 4.5: Power measurements of GEMM kernel on NVIDIA RTX A6000 repeated for 1 second. The left is raw measurements, whereas the right is filtered.

- Gain g: a scalar that controls how strongly the level estimate x follows the measurement each step. Larger g = more responsive, less smooth.
- Gain h: a scalar that controls how strongly the rate $\dot{x}$ is adjusted by each new measurement. Larger h = faster adaptation of the rate.
- Time step Δt (dt): The sampling interval used in the prediction step. We set the average of timestamps recorded from all measurements.
- Measurement z (per step): The observed value of the signal at the current time. In our case, z is the NVML instantaneous power reading at time t_k.

GH filtering adapts more effectively to trends in the data by utilizing both predictive and corrective mechanisms. As shown in Figure 4.6, this filter demonstrated promising results for smoothing, especially in reducing short-term power spikes.



(a) Original measurements.



(b) Filtered signal.

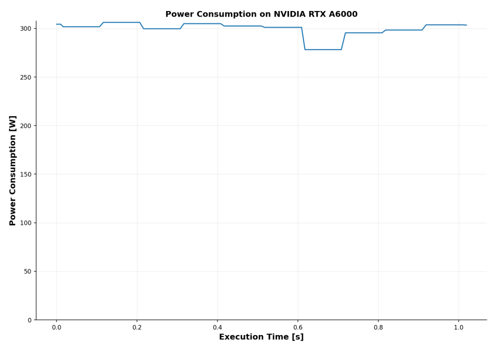
Figure 4.6: Power measurements of GEMM kernel on NVIDIA RTX A6000 repeated for 1 second. The left is raw measurements, whereas the right is filtered.

4.3.3 One-Dimensional Kalman Filtering

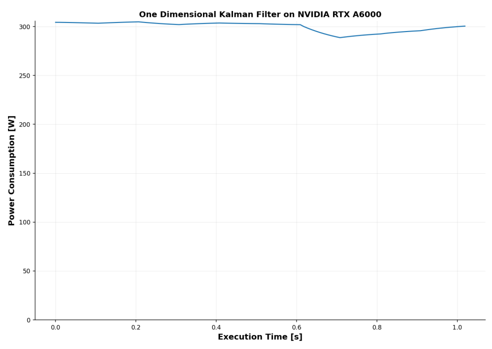
Similarly, one-dimensional Kalman filtering was applied using the FilterPy framework [40]. The parameters that we set are:

- Initial state x: Starting estimate of the signal level. In our case, we set the first NVML power sample.
- Initial variance P: Uncertainty of the initial estimate x. Larger P means "trust the measurements more than the prior".
- Process noise variance Q: Variance of how much the true signal can wander between steps (model noise). A Larger Q makes the filter more responsive (less smooth).
- Measurement noise variance R: Variance of measurement noise. Larger R means you trust the measurements less (more smoothing).
- Measurement z (per step): The observed value at the current time.

The Figure 4.7a shows the raw measurements and the filtered signal using one-dimensional Kalman.



(a) Original measurements.



(b) Filtered signal.

Figure 4.7: Power measurements of GEMM kernel on NVIDIA RTX A6000 repeated for 1 second. The left is raw measurements, whereas the right is filtered.

4.3.4 Summary

While these techniques were useful in improving the visual quality of the data and making it easier to interpret trends in power consumption, they were ultimately not included in the final correction model. The reason for this decision was the scalability and tuning challenges associated with each filtering method. Both the moving average and GH filters required careful selection of window sizes or gain parameters, which varied across kernel configurations. In addition, the Kalman filter, though effective, introduced significant computational overhead and required continuous parameter tuning to optimize its performance for different types of kernel runs. Given the variability in kernel parameters and the potential performance impact of applying these filters during real-time auto-tuning, they were excluded from the final correction process. Instead, the focus was placed on applying temperature and frequency-based corrections using empirical models that could be applied consistently across kernel configurations.

Chapter 5

Experiments & Results

This chapter presents the comprehensive verification and validation of the correction methodology proposed in Chapter 4. The experiments are structured in two main phases to test the model's accuracy and practical applicability.

First, we establish a baseline by evaluating the methodology on two distinct single-configuration workloads: GEMM and FP32. For each workload, we test the model against both "seen" (in-sample) and "unseen" (out-of-sample) clock frequencies defined in Figure 4.1.

Building on this validation, the second phase assesses the methodology's generalization in a realistic auto-tuning context. We apply our model to a search space of 216 GEMM kernel configurations. The primary goal is to determine if the model can accurately predict energy consumption across this diverse parameter space and at clock frequencies that were not explicitly benchmarked.

5.1 Experiments Setup

To ensure consistent and reproducible evaluation of GPU power behavior, we used the following experimental setup, which combines kernel-level tuning, internal sensor measurements, and validation of our data with external sensor measurements.[1]

All the following experiments were performed on an NVIDIA RTX 4000 Ada generation, a modern GPU architecture. This GPU was selected because it offers a balanced compute-memory ratio, supports detailed measurement via NVIDIA NVML, and is widely used in both scientific and production contexts. A detailed list of experiment setup is provided in the following Table 5.1.

Component	**Version**
CUDA	12.8
Driver	570.133.07
Python	3.11.8
Linux Distribution	Rocky Linux 8.10

Component	**Version**
Linux Kernel	6.5.0
Kernel Tuner	1.0
Cluster	DAS6 ASTRON [41]

Table 5.1: System configuration.

[1]The source code for all experiments and data analysis is publicly available at: `https://github.com/SetkiF/nvml-power-correction`

Although at first glance it appears that we are comparing the same quantities, it is worth noting that the setups for external measurements and internal measurements via Kernel Tuner differ. For the external sensor, we utilize the PowerSensor3's ability to capture instant power readings while only repeating the kernel a fixed number of times. For the internal sensor, we employ continuous benchmarking, running the kernel repeatedly for 1 second per data point to obtain the mean power. As we show later in Section 5.3.4, this difference can cause reported NVML measurements to be influenced more by temperature than those from PowerSensor3 (ground-truth data).

5.1.1 Experimental Procedure

To properly evaluate the correction methodology, we designed a comprehensive experimental setup. We began by running each kernel with fixed configuration parameters to establish a baseline for comparison. Subsequently, we performed generalized tuning by varying the kernel configuration to simulate a real-world use case.

During all experiments, we simultaneously recorded energy and power from both external and internal sources. To ensure measurement stability, each configuration was repeated 32 times.

The experiments were structured in three main phases:

1. Validation of a frequency–power model correction
2. Validation of a temperature–power model correction
3. Model generalization across different parameters

This experimental strategy provided us with data that compares high-frequency ground-truth sensor data against NVML measurements, enabling us to identify both the limitations of built-in power sensors and the conditions under which power prediction models can or cannot generalize while auto-tuning.

5.2 Kernels

To ensure the robustness of our evaluation, we selected two representative GPU workloads: GEMM (General Matrix Multiplication) and an FP32 synthetic kernel. These kernels were chosen because they represent common computational patterns in scientific computing while exhibiting distinct power and performance characteristics.

1. **GEMM Kernel**: General matrix multiplication is a fundamental building block in numerical linear algebra and deep learning frameworks. GEMM kernels are widely studied for performance tuning due to their sensitivity to memory bandwidth, data reuse, and thread-block configuration. GEMM generates sustained memory traffic; we use it as a memory-intensive workload to provide a baseline for understanding how resource allocation affects GPU power.
2. **FP32 Kernel**: The FP32 [42] synthetic kernel represents a compute-intensive workload that heavily stresses the floating-point units of the GPU. As highlighted by Schoonhoven et al. [7], the FP32 kernel provides valuable insight into the trade-offs between clock frequency, power, and thermal characteristics.

These kernels provide a contrasting pair of workloads: GEMM heavily utilizes memory bandwidth, whereas FP32 performs the fewest memory operations. This allowed us to analyze how GPU power consumption varies with and without reliance on memory bandwidth.

5.3 GEMM with seen clock frequency 1485 MHz

The following experiment was designed to establish a baseline for modeling GPU power and energy when executing the GEMM kernel. Although the GPU does not operate natively at 1485 MHz by default, this frequency was selected as a reference and validation point. Default NVML power measurements, recorded with DVFS (Dynamic Voltage and Frequency Scaling) enabled, were corrected as if they had been recorded at 1485 MHz. These corrected values were then compared against PowerSensor3 [16] measurements taken directly at 1485 MHz, which served as the external ground-truth baseline.

The measurements were recorded using Kernel Tuner [4], and the applied parameter set is summarized in Table 5.2. For this experiment, we used a single set of parameters and repeated the same setup 2000 times.

Parameter	Value	Parameter	Value
GEMMK	0	KWI	2
MWG	128	VWM	4
NWG	128	VWN	4
KWG	16	STRM	1
MDIMC	8	STRN	1
NDIMC	16	SA	1
MDIMA	16	SB	1
NDIMB	16	KREG	1
PRECISION	32	REPEAT	1–2000

Table 5.2: Parameter set values for GEMM kernel executed with Kernel Tuner. A detailed description of the parameters can be found at [43].

5.3.1 Frequency Power modeling

We next model power as a function of clock frequency. The fitting model described in Chapter 4.1 is illustrated in Figure 4.1a, which demonstrates the rise in power with frequency and the resulting fitted curve. The data demonstrate a monotonic increase in power with frequency up to approximately 1800-2000 MHz, beyond which the curve reaches a plateau. This saturation reflects the GPU's hardware limits, where even if higher clock frequencies are requested, the GPU does not actually run faster, preventing a proportional increase in dynamic power. The fitting model captures the real data very closely, following both the rising region and the plateau, which indicates that the model can interpolate power values across a broad frequency range.

5.3.2 Execution time fitting

Execution time data were also analyzed as a function of GPU clock frequency. To capture the correlation between execution time and clock frequency, the time model from Equation 4.5

was used, which accurately models the sharp decrease in runtime at low frequencies, followed by a gradual asymptotic convergence at higher frequencies. This functional form reflects the physical behavior of the system: at lower frequencies, runtime is dominated by limited compute throughput, while at higher frequencies, execution time reaches a lower bound dictated by memory bandwidth and parallelism constraints.

The fitted curve in Figure 4.2a closely matches the measured execution times, indicating that the model is well-suited to describe the correlation between time and clock frequency. Importantly, this model provides a robust mathematical basis for subsequent energy modeling, as energy consumption is determined jointly by power and execution time.

5.3.3 Baseline recording with NVML

Initial measurements were collected using the NVML observer. As shown in Figure 5.1, the reported power values are clustered near 128 W. Interestingly, we noticed some outlier points above this line. This could be due to the kernel itself, temperature effects, or measurement noise.

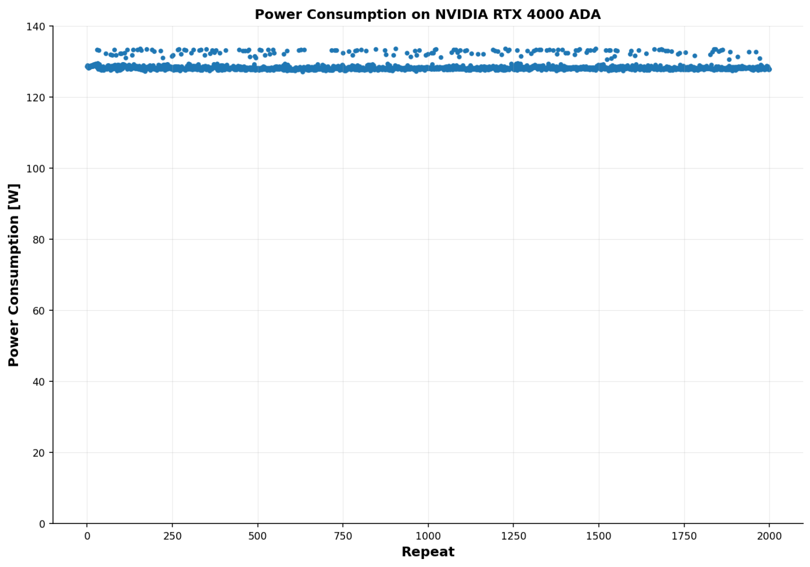


Figure 5.1: Default measurements of GEMM kernel.

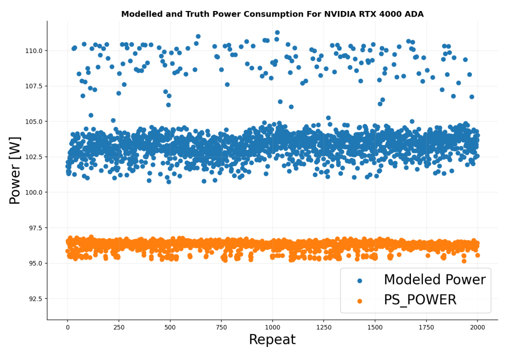
5.3.4 Clock Frequency Correction

Using the reference clock frequency of 1485 MHz, we extended the linearity of the frequency-power relationship into a practical modeling framework for predicting both power and energy consumption.

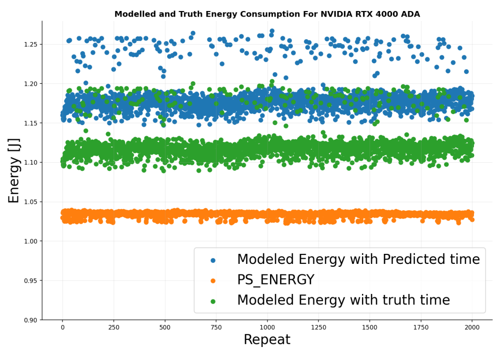
We started by modeling power solely as a function of frequency scaling, ignoring thermal effects. As shown in Figure 5.2a, the modeled power (blue) consistently overestimated the external PowerSensor3 ground truth (orange). While the frequency scaling effectively captured the overall magnitude and trends, it failed to accurately reproduce the absolute power levels. The offset indicated that frequency alone was insufficient, suggesting that other factors, particularly temperature, play a critical role.

Additionally, we present an energy analysis in Figure 5.2b, where two modeled energy estimates are compared against PowerSensor3 ground truth (orange). The green trace corresponds to the energy correction model using the true kernel execution time of 10.81 ms, while the orange trace represents the model using the predicted execution time of 11.39 ms, obtained from the rational function fit. Both versions systematically overestimate the ground-truth PS_ENERGY, with the truth-time model performing slightly better and showing a narrower spread than the predicted-time variant. The gap between modeled and measured energy arises because temperature correction has not yet been applied, meaning thermal effects are still unaccounted for.

The temperature difference comes from the measurement methods in PowerSensor3 and NVML in Kernel Tuner. For instance, each NVML power sample is obtained by executing the kernel 32 times to extend the sampling window and to allow the GPU to warm up. On the other hand, PowerSensor3 has a very high sampling rate and can produce accurate measurements from a few iterations. As shown in Figure 5.3b, the temperature levels during the GEMM kernel execution between NVML and PowerSensor3 are significantly different (around 15°C). Hence, incorporating temperature effects is essential to achieve closer alignment with sensor-based measurements.



(a) Correcting Power based on clock frequency using Equation 4.2.

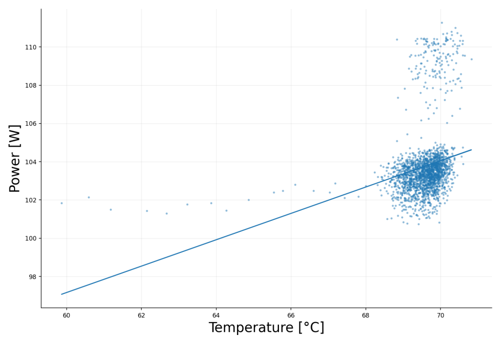


(b) Correcting Energy based on clock frequency using Equation 4.3.

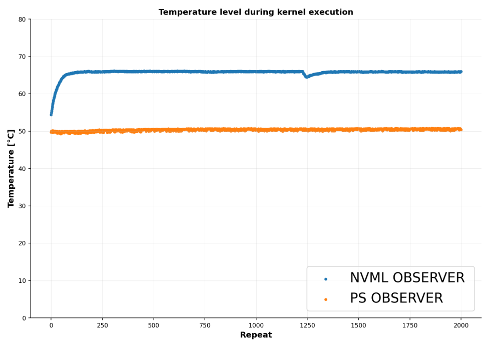
Figure 5.2: Power correction (left) and Energy Correction (right).

5.3.5 Temperature Correction

To address discrepancies between the modeled power and PowerSensor3 measurements, we added a temperature correction step. The analysis revealed a positive slope. Figure 5.3a indicates that power consumption increases with temperature at approximately 0.69 W per degree Celsius. Although the overall correlation strength was moderate, the regression clearly demonstrated that even modest temperature variations can systematically shift power readings. Importantly, we anchored the correction to the lowest recorded temperature during kernel execution, serving as a "cold GPU" baseline. This ensured that the adjusted model accounted only for the incremental increase in power associated with thermal rise, without artificially shifting the entire curve.



(a) Slope of Temperature vs Power linear model.



(b) Temperature Level during GEMM kernel execution.

Figure 5.3: Temperature VS Power (left) and Repeat VS Temperature (right).

5.3.6 Final Corrected Model

The complete correction methodology, combining both frequency and temperature corrections, is illustrated in Figure 5.4a. The figure compares four sets of measurements:

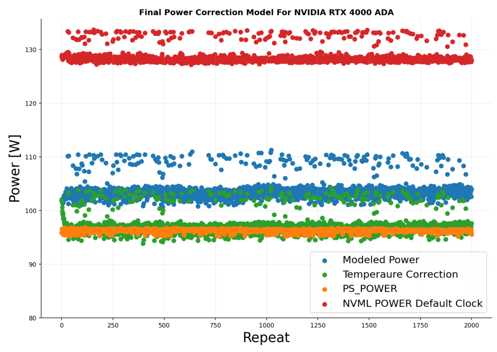
1. Red: NVML default power measurements
2. Blue: Frequency-corrected power, which captures the general trend but retains a systematic offset
3. Green: Frequency- and temperature-corrected power, which aligns with the ground-truth mean but is prone to transferring the outlier trends from the default measurements
4. Orange: External PowerSensor3 measurements, serving as ground-truth reference data.

The results demonstrate that while frequency scaling alone transforms the raw NVML outputs (red) into frequency-corrected power (blue) with a 7.79% MAPE (mean absolute percentage error) relative to PowerSensor3 truth data. This means that it is still insufficient to fully reproduce sensor-level accuracy; hence, we incorporated the temperature correction, which eliminated most of the residual offset, reducing the MAPE to 1.19% and aligning the green trace closely with the orange ground-truth readings. The corrected model overlapped with PowerSensor3 values across nearly all repetitions, diverging only for a small number of outliers, likely due to transient thermal fluctuations or measurement noise.

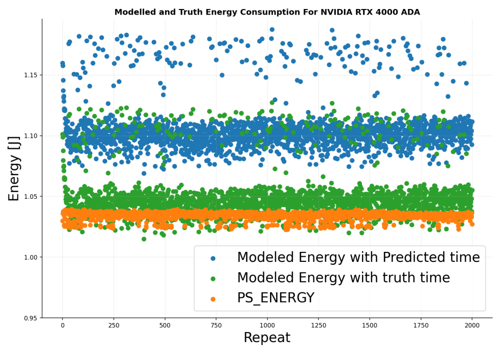
This outcome confirms that temperature-aware frequency scaling substantially enhances the fidelity of NVML-based power modeling. By accounting for both dynamic scaling effects and thermal contributions, the final corrected model provides a reliable proxy, with a 1.19% MAPE on high-frequency external measurements, bridging the gap between NVML measurements and hardware-level ground truth.

Moreover, we extended the correction framework to predict cumulative energy consumption. Energy was computed from both the true kernel execution time and the predicted execution time from the rational function fit and compared with the external PowerSensor3 ground-truth measurements.

As shown in Figure 5.4b, the green trace represents modeled energy using the true execution



(a) Final Power correction.



(b) Final Energy Correction.

Figure 5.4: Power correction (left) and Energy Correction (right).

time (10.81 ms). This version aligns closely with the MAPE of the PowerSensor3 energy values (orange), at 1.61%, though the scatter shows that some outliers in the power traces are directly transferred into the energy estimates. In contrast, the blue trace, which relies on the predicted execution time (11.39 ms), is systematically shifted upward, resulting in a biased MAPE of 6.92%. This shift reflects the limitations of the rational function in accurately capturing kernel runtime, leading to compounding errors when estimating energy.

With the full methodology in place, the results demonstrate that accurate execution-time measurement is critical for reliable energy modeling. While the truth-time model reproduces the mean energy consumption with reasonable fidelity, additional corrections are still needed to fully reconcile the modeled energy with the ground truth.

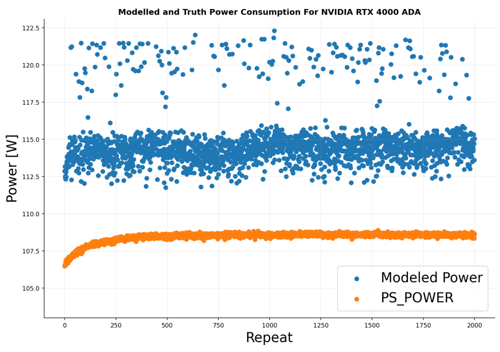
5.4 GEMM with unseen clock frequency 1620 MHz

The following experiment extends the baseline analysis for the seen clock frequency described in Section 5.3. We extend the experiment to an unseen clock frequency to evaluate the model's ability to generalize to unseen data. The model was trained at the reference frequency of 1485 MHz (seen/in-sample) and then tested at 1620 MHz (held-out sample), which was not part of the training set. The goal of this experiment is to validate whether the frequency and temperature corrections can reliably interpolate to new operating points.

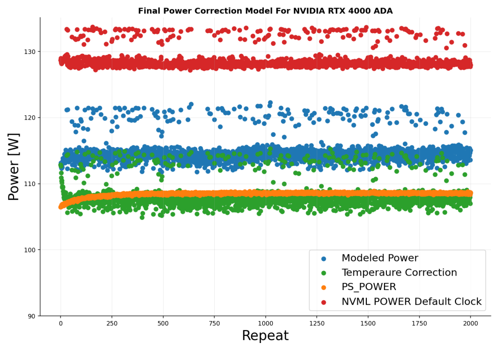
5.4.1 Power Correction

When the frequency-scaled model was applied directly at 1620 MHz, the predicted power values (blue) showed a systematic offset relative to the PowerSensor3 ground truth (orange), as illustrated in Figure 5.5a. Although the model correctly captured the trend and variance across repetitions, the predictions consistently overestimated the truth data by an error rate of 5.82% (MAPE). This confirmed that frequency scaling alone is insufficient for accurate extrapolation to new frequencies.

To refine the prediction, the same temperature correction procedure from Section 5.3.5 was applied. As shown in Figure 5.5b, the addition of this correction aligned the modeled power, with temperature correction (green), much more closely with the PowerSensor3 measurements. The MAPE offset was significantly reduced to 1.18% from 5.82%, and the final corrected model reproduced the external sensor readings across most repetitions, with residual deviations limited to transient fluctuations.



(a) Correcting Power based on clock frequency using Equation 4.2.



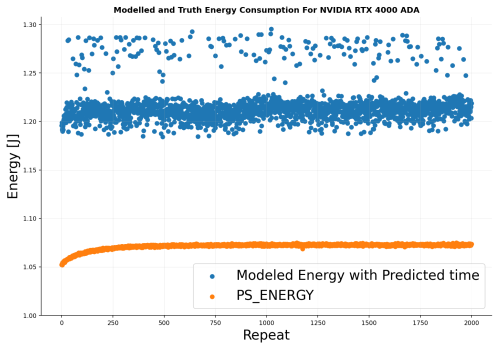
(b) Final Power correction.

Figure 5.5: Power correction with only clock frequency correction (left) and Final Power correction including temperature correction (right).

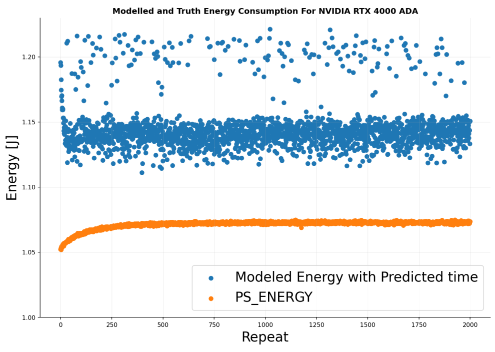
5.4.2 Energy Correction

For an unseen clock frequency, modeled energy was computed using the predicted kernel execution time from the rational function fit in Equation 4.5, since the true execution time was not available. The rational model predicted an execution time of 10.59 ms.

The modeled energy (blue) in Figure 5.6a consistently lies above the PowerSensor3 ground-truth values (orange) with an error rate of 13.48% (MAPE). Note that the modeled power in Figure 5.6a is still not corrected for temperature; hence, this offset reflects the inaccuracy of both power and predicted runtime.



(a) Correcting Energy based on clock frequency using Equation 4.3.



(b) Final Energy correction.

Figure 5.6: Energy correction with only clock frequency correction (left) and Final Energy correction including temperature correction (right).

We then incorporate temperature correction into the modeled power to further correct energy modeling. Figure 5.6b compares the temperature-corrected modeled energy (blue) against

PowerSensor3 (orange) ground-truth data. While the correction improved the MAPE alignment from 13.48% to 6.88%, the offset remained evident. This demonstrates that even when power is accurately corrected, imperfect runtime prediction continues to bias the energy model. The discrepancy further underscores the need to integrate more robust runtime modeling to complement temperature-aware power corrections.

5.5 FP32 with seen clock frequency 1425 MHz

The following experiment focused on the FP32 kernel, a compute-intensive workload that fully loads the GPU. Unlike the GEMM kernel, FP32 performs as few memory operations as possible, placing heavier stress on the compute units and often amplifying noise and variance in power measurements. Similar to Experiment 5.3, we chose 1425 MHz as a reference point because these data were used during the training process. The same approach of correcting default measurements with DVFS (Dynamic Voltage and Frequency Scaling) enabled was used, and we then compared them with PowerSensor3.

The FP32 kernel was executed with the configuration parameters summarized below. These values defined the problem size, block dimensions, and number of repetitions, ensuring consistent execution and stable measurements for both NVML and PowerSensor3.

Parameter	**Value**
block_size_x	1024
nr_outer	64
nr_inner	1024
REPEAT	1–2000

Table 5.3: Parameter set values for the FP32 kernel executed with Kernel Tuner.

The choice of a large block size of 1024, combined with high repetition factors nr_outer = 64 and nr_inner = 1024, was intended to fully saturate the GPU's compute resources while minimizing random fluctuations in power reporting. By systematically controlling both inner and outer repetition loops, the kernel provided a sufficiently long execution window to capture realistic power, frequency, and temperature behavior under sustained load. This configuration ensured that the FP32 kernel allowed robust analysis of the interactions among clock frequency scaling, thermal effects, and power consumption.

5.5.1 Frequency Power Modeling

The FP32 kernel was first analyzed by fitting the linear model described in Equation 4.1. The measured data and fitted curve are shown in Figure 4.1b, where power increases monotonically with frequency before saturating within the operating range.

Furthermore, we fit a rational function defined in Equation 4.5 to predict kernel execution time. The fitted curve in Figure 4.2b shows excellent agreement with the training data, with runtime decreasing at low frequencies and approaching an asymptotic lower bound at high frequencies. Together, the power–frequency model and the rational execution time fit establish a solid foundation for subsequent energy modeling of the FP32 kernel by coupling frequency-dependent power with the predicted kernel execution time.

5.5.2 Baseline recording with NVML

Figure 5.7 shows the raw NVML observer power readings at the baseline clock frequency. Compared to GEMM, the FP32 traces exhibited lower noise and variance. Lower noise and variance can be caused by factors such as longer execution time (nvml_observer has more sample points), less noise during measurements, or because the FP32 kernel is not accessing memory.

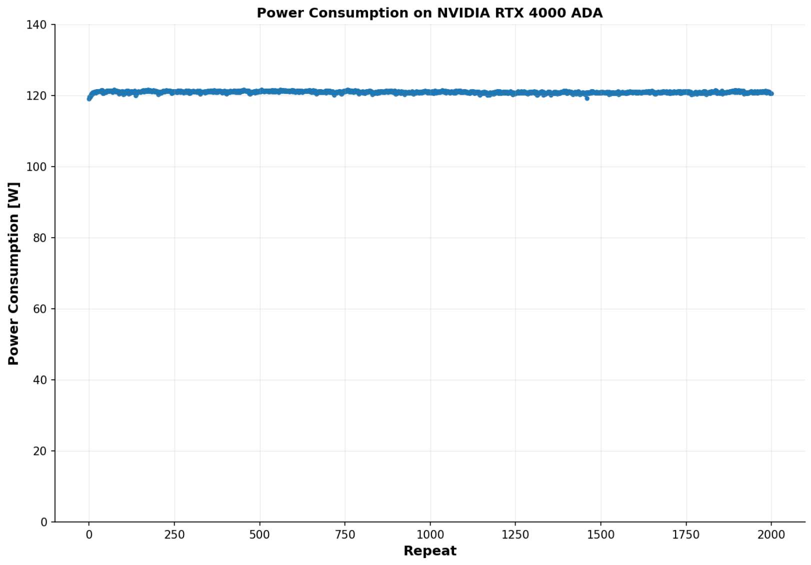


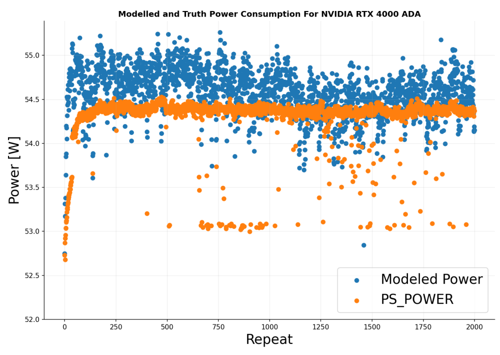
Figure 5.7: Default measurements of FP32 kernel.

5.5.3 Clock Frequency Correction

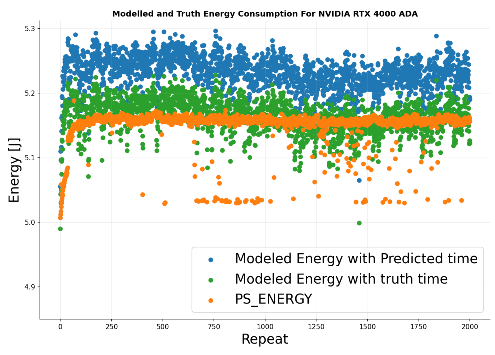
To refine the raw NVML measurements, a frequency-scaling correction was applied to the FP32 kernel power data. The intent of this step was to transform and normalize power values to the reference GPU clock of 1425 MHz. As shown in Figure 5.8a, the corrected modeled power (blue) closely followed as the MAPE is only 0.63% when compared to the PowerSensor3 values (orange). This confirmed that the frequency-based correction reproduced the FP32 kernel's general scaling behavior for the given clock reference.

However, the correction also propagated the variance from NVML readings. Instead of producing a smooth trajectory, the modeled power showed substantial scatter across repetitions. This contrasted with the PowerSensor3 data, which remained comparatively stable and smooth. This divergence highlights an important limitation of frequency-only correction. While it is sufficient to approximate the average behavior of the kernel, the correction cannot eliminate short-term fluctuations.

In addition, Figure 5.8b represents the modeled energy consumption. The blue trace represents modeled energy using the predicted execution time of 95.85 ms, whereas the green trace represents modeled energy using the true execution time of 94.60 ms. Compared to the ground-truth data (orange), the modeled mean, with the true execution time, aligns more closely with



(a) Correcting Power based on clock frequency using Equation 4.2.



(b) Correcting Energy based on clock frequency using Equation 4.3.

Figure 5.8: Power correction (left) and Energy Correction (right).

the PowerSensor3 mean energy consumption; however, outliers present in the power traces also propagate directly into the energy estimate.

Similar to the previous energy modeling, the predicted-time model shows a slightly upward shift in its mean, reflecting the error introduced by runtime overestimation in the rational function fit. This analysis demonstrates that while truth-time-based modeling provides a reasonable baseline for energy estimation, reliance on predicted execution time introduces a small bias. Accurate runtime modeling is, therefore, essential for reliable energy predictions, particularly for workloads such as FP32 that exhibit noisier behavior than GEMM.

5.5.4 Temperature Correction

To further improve the accuracy of the FP32 power model, a temperature correction step was introduced. The relationship between temperature and default power measurements is illustrated in Figure 5.9, where a linear regression captures the upward trend in power as temperature increases. The slope of the fitted line indicates a positive correlation, with power rising approximately 0.55 W per degree Celsius.

Compared to GEMM, the linear correlation with FP32 is more noticeable because of the kernel runtime. Longer execution times make the power measurements more prone to temperature effects. For instance, the FP32 average execution time is 57.98 ms, whereas GEMM is 8.85 ms. We can observe the strong clustering of data points at higher temperatures (around 66°C), which suggests that the system reaches thermal equilibrium during extended FP32 operations, resulting in more consistent and predictable power-temperature relationships.

5.5.5 Final Corrected Model

The complete FP32 correction methodology, which combines frequency and temperature corrections, is presented in Figure 5.10. The figure compares four sets of measurements.

1. Red: NVML default power measurements
2. Blue: Frequency-corrected power

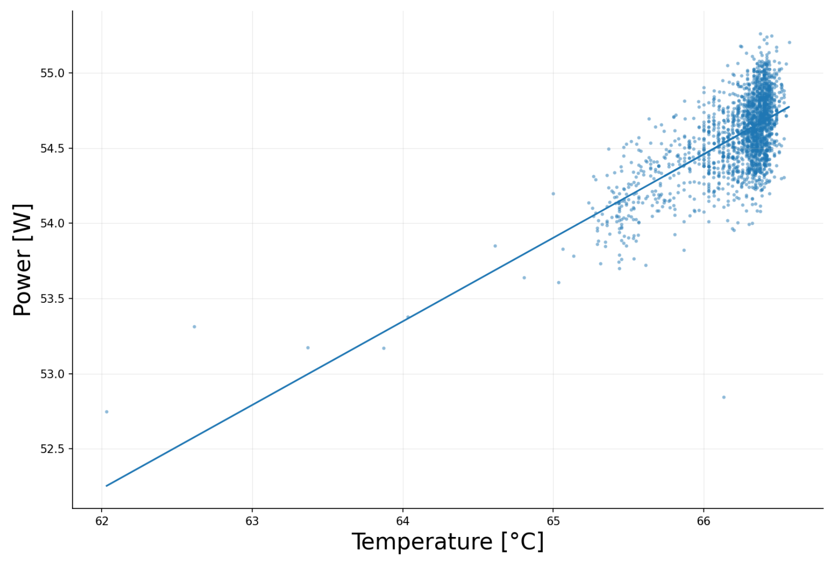
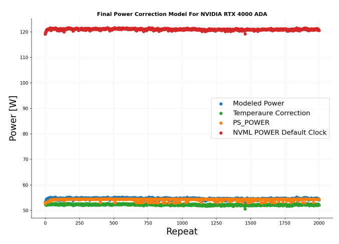


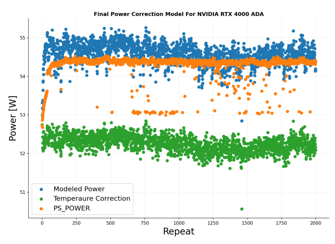
Figure 5.9: Slope of Temperature vs Power linear model.

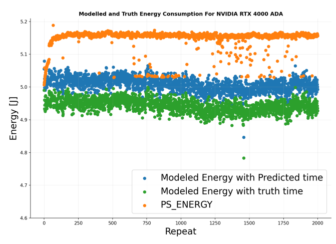
3. Green: Frequency and temperature-corrected power
4. Orange: External PowerSensor3 measurements

Interestingly, for FP32 at the reference point of 1425 MHz, only the clock-frequency correction aligns closely with the PowerSensor3 mean. At first glance, the temperature correction in Figure 5.10b appears to underestimate the ground-truth power levels; however, it aligns well with the initial iterations of the PowerSensor3 measurements. This indicates that PowerSensor3 is also affected by temperature when running the FP32 kernel, stemming from the FP32 kernel's longer execution time. Based on Figure 5.10b, we can assume that PowerSensor3 data should probably also be corrected for temperature.

Moreover, temperature correction reduces some of the systematic bias present in the frequency-only corrected data by compensating for thermal effects that accumulate during kernel execution.







(a) Final Power Correction. (b) Final Power Correction. (c) Final Energy Correction.

Figure 5.10: Final Correction model based on clock frequency and temperature correction.

In Figure 5.10c, we show the corresponding modeled energy after applying temperature correction. Two variants were computed:

1. Blue: Predicted-time modeled energy based on the rational function estimate of 95.85 ms.
2. Green: Truth-time modeled energy using the actual kernel execution time of 94.60 ms.
3. Orange: PowerSensor3 energy.

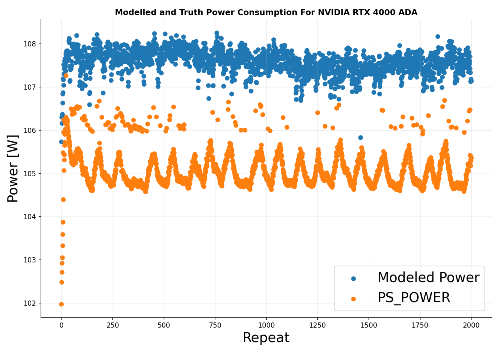
From Figure 5.10c, we observe that PowerSensor3 also shows temperature-induced drift. As the number of repetitions increases, energy consumption trends upward, confirming that thermal load increases the average power draw.

5.6 FP32 with unseen clock frequency 2235 MHz

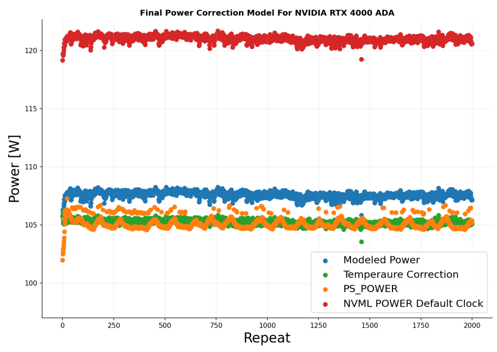
To further evaluate the generalization ability of the FP32 kernel model, we tested it at an unseen clock frequency of 2235 MHz, which was not included in the baseline training. This experiment served as a higher-frequency extension of Experiment 3, enabling the assessment of how well the correction framework scales beyond the mid-range operating points.

5.6.1 Power Correction

Figure 5.11a shows the modeled power (blue) compared to PowerSensor3 ground truth (orange). The frequency-only correction reproduced the overall trend but overestimates the measured values. Moreover, the modeled data exhibited considerable variance, while the external measurements followed a smoother but oscillatory profile, likely reflecting subtle thermal cycling effects under sustained load. The oscillating pattern from PowerSensor3 is more likely a high clock frequency reference point. The GPU is unable to maintain this frequency consistently, even when set manually. Hence, it is dropping to a more stable one. This mismatch emphasizes that frequency scaling alone cannot reliably capture the true power characteristics at elevated frequencies. Hence, we present the temperature correction of modeled power in Figure 5.11b.



(a) Correcting Power based on clock frequency using Equation 4.2.



(b) Final Power correction.

Figure 5.11: Power correction with only clock frequency correction (left) and Final Power correction including temperature correction (right).

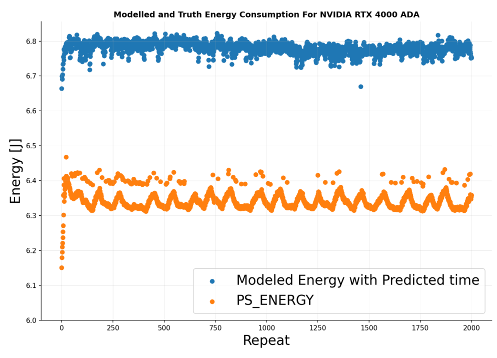
The final corrected data (green) shifted downward and aligned more closely with the PowerSensor3 readings. The mean offset was substantially reduced, particularly in regions where the raw model had overestimated power consumption. Nonetheless, some residual discrepancies remained, with the corrected model unable to fully track the oscillatory fluctuations present in the external sensor data.

5.6.2 Energy Correction

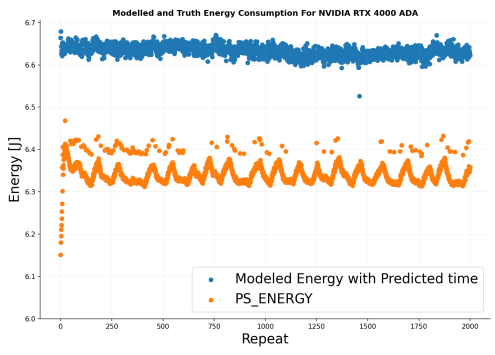
For the unseen-frequency validation of FP32 at 2235 MHz, we extended the analysis to include cumulative energy modeling. Unlike the case of seen frequency at 1425 MHz, we did not have access to the true kernel execution time. Instead, the energy estimates relied solely on the predicted execution time from the rational function, which, for this kernel, was 92.10 ms.

As shown in Figure 5.12a, the mean of the modeled energy is higher than the ground-truth data. This systematic offset directly reflects the inaccuracy of the predicted runtime: while the power correction captures the mean scaling reasonably, even small deviations in execution time translate into significant energy bias. Furthermore, the oscillatory pattern observed in the PowerSensor3 energy is not reproduced by the modeled results, underscoring the limitations of runtime prediction at high clock frequencies.

Furthermore, we added a temperature correction to the modeled power to bring the modeled mean energy closer to the ground-truth data. However, we did not observe any significant improvement, as shown in Figure 5.12b. This outcome mirrors the behavior observed in the GEMM unseen-frequency experiments 5.4.2: when the true execution time is not available, reliance on the rational model introduces error that propagates into energy estimates. Improving runtime modeling is therefore critical to achieving robust energy prediction at unseen frequencies.



(a) Correcting Energy based on clock frequency using Equation 4.3.



(b) Final Energy correction.

Figure 5.12: Energy correction with only clock frequency correction (left) and Final Energy correction including temperature correction (right).

5.7 Generalization

Previous experiments were conducted by repeating the same kernel configuration multiple times. However, this is not realistic, especially during auto-tuning, when we want to test different configurations to find the kernel that has the best performance or is the most energy-efficient within the search space. Hence, the following experiments include tuning the GEMM kernel with 216 different configurations. Each configuration was defined by a distinct set of tuning parameters, including matrix workgroup sizes (MWG, NWG), kernel workgroup dimensions

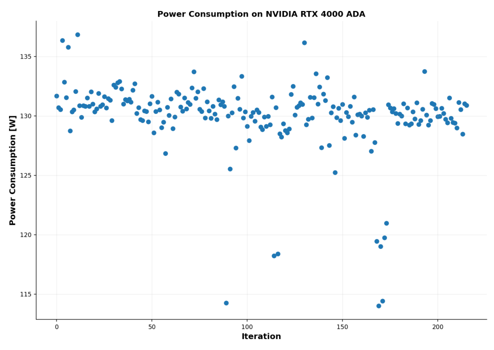
(KWG), tiling parameters (MDIMC, NDIMC, MDIMA, NDIMA, MDIMB, NDIMB), vector widths (VWM, VWN), and shared memory flags (SA, SB). The parameters defined in Table 5.4 collectively shape the execution workload and directly influence the GPU's power behavior.

Parameter	Value(s)	Parameter	Value(s)
GEMMK	0	KWI	2
MWG	16, 32, 64, 128	VWM	4, 8
NWG	16, 32, 64, 128	VWN	4, 8
KWG	16	STRM	0
MDIMC	8, 16, 32	STRN	0
NDIMC	8, 16, 32	SA	1
MDIMA	8, 16, 32	SB	1
NDIMB	8, 16, 32		
PRECISION	32		

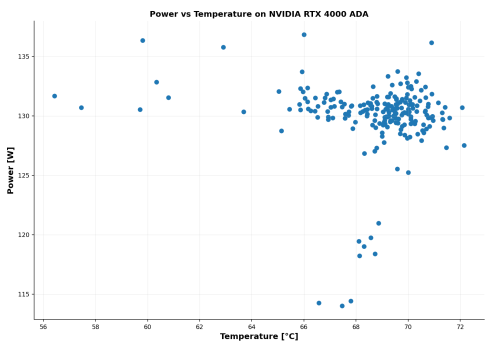
Table 5.4: Parameter set values for the GEMM kernel tuned with the Kernel Tuner framework.

Compared to the controlled single-configuration runs, the power traces in Figure 5.13a showed greater variance, both in absolute magnitude (ranging from 114 W to above 135 W) and in stability across repetitions. Instead of forming a straight line, the data points vary widely. This immediately highlights the challenges of generalization: when kernel parameters are varied, the GPU operates under substantially different efficiency regimes, and a single correction model is unlikely to capture clock-frequency or temperature effects across different configurations.

Interestingly, as shown in Figure 5.13b, temperature levels were nearly constant, despite the measurements being recorded with the NVML observer. Perhaps the 216 configurations are not enough to warm the GPU to a point where the thermal effects are noticeable. On the other hand, with 2000 repetitions in the single-configuration case Figure 5.3a, thermal effects were noticeable, and a correlation between temperature and power was observed. Therefore, in the following experiments, we exclude temperature correction and rely only on clock frequency correction.



(a) Default measurements of GEMM.



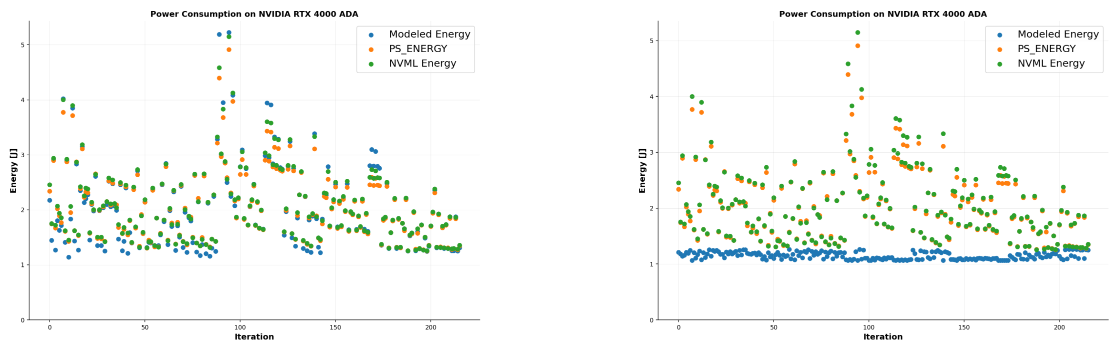
(b) Temperature VS Power.

Figure 5.13: Default measurements for GEMM (left) Temperature VS Power correlation.

5.7.1 Modeling energy with default clock frequency

In Figure 5.14, modeled energy predictions were evaluated against PowerSensor3 and NVML data to assess accuracy across the 216 kernel configurations. We compared two different types of measurements. On the left 5.14a, we modeled energy with the true kernel execution time recorded with the runtime observer, and on the right, we modeled energy with the predicted time based on the rational function defined in 4.2a.

As shown in Figure 5.14a, the modeled energy (blue) closely follows the PowerSensor3 (orange) and NVML (green) measurements, with an acceptable MAPE of 4.7%. On the other hand, when energy is modeled using the predicted time 5.14b, the modeled energy (blue) significantly underestimates the values from PowerSensor3 and NVML observer, and the error significantly increases to 38.28%. This underestimation occurs because the predicted execution time is always shorter than the actual one (the mean predicted time is 15.95 ms, whereas the mean true time is 22.83 ms), and since the modeled power is identical in both cases, we can assume that the rational function in 4.2 does not accurately predict runtime across configurations.



(a) Energy Modeling with truth kernel execution time.

(b) Energy modeling with predicted kernel execution time using model defined at 4.5.

Figure 5.14: Energy Modeling VS NVML vs PowerSensor3.

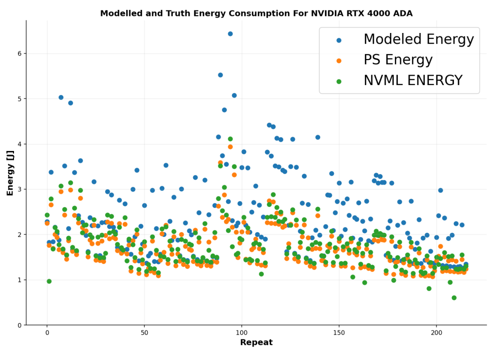
5.7.2 Modeling energy with fixed clock frequency

We extend our general experiments to fixed clock frequencies, specifically at 1485 MHz and 1620 MHz. To model power, we use the fitted model described in Figure 4.1, and to model kernel execution time, we use the model described in Figure 4.2. The idea is to test whether we can model energy without even running the kernel at that particular clock frequency. This way, we can reduce overall benchmarking time; instead of running all configurations for each tuned clock frequency, we can run the tuned space once and use our methodology to explore the other clock frequencies.

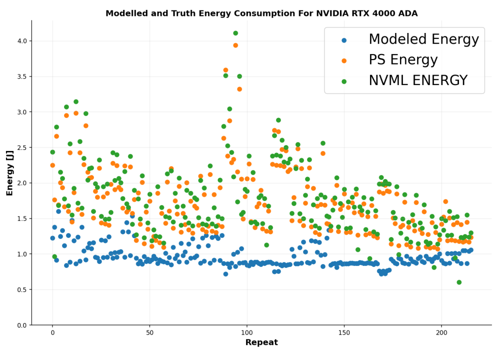
The left side of the plots in Figure 5.15 shows energy modeling with ground truth times. That is, the modeled energy (blue) is defined by multiplying the modeled power (obtained by correcting the default measurements from Figure 5.13a using 1485 and 1620 as clock reference points) by the execution time recorded via the runtime observer in Kernel Tuner. On the right side, the modeled energy is defined by multiplying the same modeled power by the rational function defined in Figure 4.2. This ensures that we evaluate both the power model 4.1 and the time model 4.2 against ground-truth data from PowerSensor3 (orange) and the NVML observer (green).

At 1485 5.15a and 1620 5.15c MHz, the modeled energy (blue) overestimates the energy relative to PowerSensor3 and NVML with a MAPE of 35.82% for 1485 MHz and 37.90% for 1620 MHz. Compared to the energy modeling with default clock frequencies in Figure 5.14a, the error margin in this experiment is notably higher. We can assume the source of the error is the power modeling, since we use the true kernel execution time. In addition, when using predicted times, the modeled energy underestimates the PowerSensor3 and NVML observer data for both 1485 MHz 5.15b and 1620 MHz 5.15d, with MAPE of 39.51% and 34.45%, respectively.

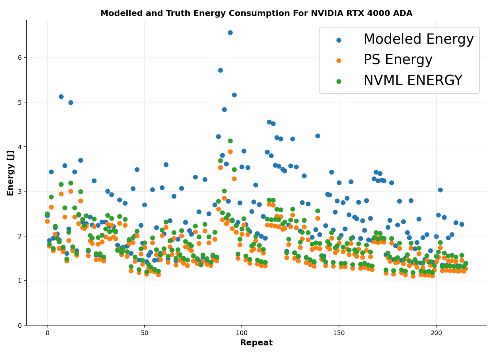
This indicates that for the previous experiment 5.7.1, the error was solely in execution time. However, in these experiments, we observe that the error originates from both linear power modeling and rational function time prediction. With these results, we conclude that, unfortunately, we cannot accurately predict energy or power measurements without actually running the kernels.



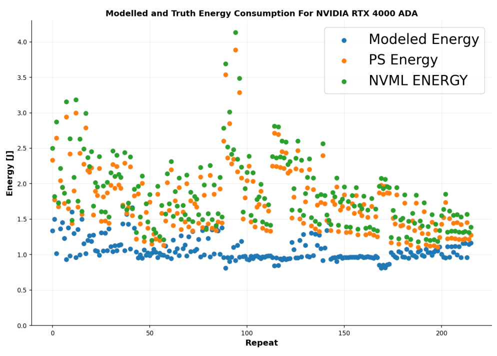
(a) 1485 MHz truth time.



(b) 1485 MHz predicted time.



(c) 1620 MHz truth time.

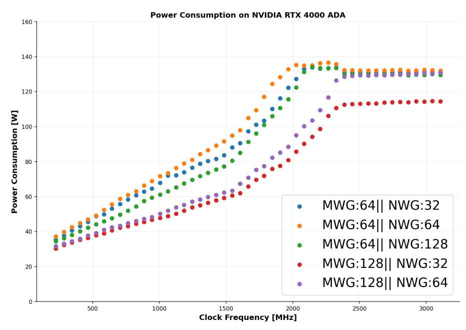


(d) 1620 MHz predicted time.

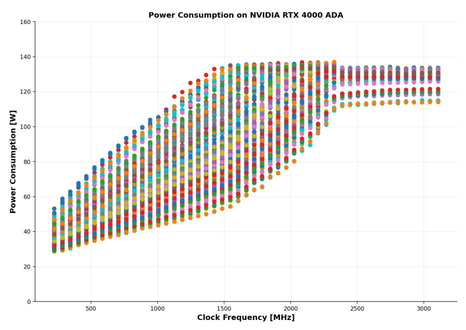
Figure 5.15: Energy Modeling for 1485 (Upper part) and 1620 (Lower).

5.7.3 Limitation of the model for generalization

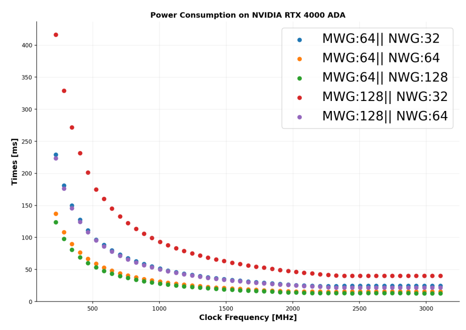
The primary limitation identified during generalization is the assumption that a single power or time profile applies across different kernel configurations. Similar to Schoonhoven et al. [7], we assumed that power consumption follows a consistent pattern regardless of how kernels are tuned. However, each kernel configuration produces a unique power or kernel execution model based on variations in resource utilization, memory access patterns, and compute density. As we can see in Figure 5.16, each color represents one kernel configuration from the search space defined in Table 5.4. As a result, the frequency-correction model defined in Figure 4.1a and kernel execution model defined in Figure 4.2a failed to generalize across different kernel configurations.



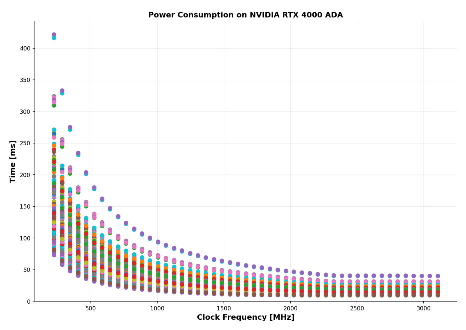
(a) Clock Frequency VS Power for 5 different configurations.



(b) Clock Frequency VS Power for search space defined at 5.4.



(c) Clock Frequency vs Time for 5 different configurations.



(d) Clock Frequency vs Time for search space defined at 5.4.

Figure 5.16: Clock Frequency vs Power (Upper part) Execution time vs Clock Frequency (Lower).

Chapter 6

Conclusion

The primary motivation for this work was to develop an efficient and generalizable method for predicting GPU power and energy consumption without the use of external sensors, which can be costly and challenging to scale across a set of machines. The proposed methodology relies on NVML measurements, a built-in sensor that provides on-device, real-time power measurements from the GPU itself. The real-time data provides consumption estimation without the need for additional hardware implementation, making the approach cost-effective and portable across various environments.

The correction methodology proposed in this work implements two techniques to improve the accuracy of power and energy prediction:

- Clock frequency correction: Taking into account that power consumption is proportional to the clock frequency, we introduced a correction model for scaling the power values with the operation frequency of the GPU.
- Temperature correction: As temperature is one of the most significant factors influencing GPU power, we added a temperature correction step to account for the thermal effects on power consumption over time.

6.1 Main Findings

This work aimed to develop a generalized and efficient model for predicting GPU power and energy consumption using NVML telemetry, incorporating clock frequency scaling and temperature correction. The findings offer valuable insights into the model's limitations and the complexity of GPU behavior. The primary conclusions indicated from the experiments are as follows:

- We observed that the power consumption increases with the clock frequency. However, each configuration had a distinctive power signature, and a fair amount of variance can be noticed between the configurations. This shows that using a single power model cannot represent all possible configurations. This result contradicts prior work, which assumed that a single model can be used to represent the kernel configurations for predicting power consumption. In contrast, our results show that it is necessary to utilize configuration-specific models, as the power scaling behavior of each configuration differs.

- Another key point in our study was the usage of temperature correction to the power model. By including a linear temperature correction, we modified the power model to account for the thermal effects that contribute to the power usage. Interestingly, for kernels with longer execution times, we observed that even external sensors must account for thermal effects. The downside of this approach is the linear assumption between temperature and power, as observed during general tuning. If correlation is constant or negative, this method can drastically increase the measurement error.
- Another important aspect of this study was the generalization of the correction methodology. As we already observed in our generalization experiments, it was very difficult to apply the same model to different kernel configurations. The model was trained on single-frequency training data and tested on a large number of configurations. This approach led to poor generalization, as the model was unable to fully accommodate the broad variability in power characteristics observed between different kernel configurations. We discovered that the assumption that all kernel configurations can be modeled by a single model was an oversimplification. This generalization failed because different kernel configurations have different power characteristics, which are not accurately represented by the model. Notably, this paper highlights that it is impossible to single out a general model for all GPU configurations and that kernel-specific models are required to capture the power behaviors for a variety of workloads.

These results demonstrate the complexities of GPU power prediction and the importance of developing more adaptive power prediction models that consider the diversity of GPU kernel configurations. While our methodology demonstrated the successful use of NVML telemetry for rapid and efficient power modeling, it also highlighted major challenges related to generalization and temperature effects.

6.2 Methodology Limitations

While the methodology brings new observations in the field, it is important to note some limitations:

- Empirical Nature: The model relies heavily on empirical data, which means that if the training data is incorrect or unrepresentative, the entire methodology will fail. This highlights the importance of collecting data carefully and accurately for effective training.
- Sensitivity to Outliers: As demonstrated in the default clock frequency measurements, outliers in the data tend to be transferred to the modeled predictions. This is a key limitation of correction methodology, as outliers can significantly skew the results and affect the model's performance, especially when testing with unseen frequencies or different configurations.

6.3 Conclusion on Research Questions

This study aimed to address key research questions regarding the development of an efficient, generalized methodology for predicting GPU power and energy consumption. The findings reveal several strengths of the methodology, as well as some challenges that highlight areas for future refinement. Below, we revisit the research questions defined in Section 1 and discuss the corresponding conclusions drawn from our experiments:

- RQ1 (measurement window): Our goal was to avoid multi-second or minute-long kernel repetitions while still producing reliable power and energy estimates. However, because our methodology failed to generalize, this was not achieved. We are still dependent on running kernels for at least 1 second during auto-tuning to obtain reliable measurements.
- RQ2 (no external sensors): One of the primary goals of this study was to develop a method that does not rely on external sensors, such as PowerSensor3. The methodology is successful to some extent in achieving this goal, but we encountered limitations in generalization and energy accuracy due to the absence of detailed, high-fidelity ground-truth data.
- RQ3 (auto-tuning impact): Our ultimate goal in this work was to enhance the auto-tuning process by modeling various executions. This way, we could significantly improve benchmarking time. However, since our kernel generalization failed, it may not be possible to accurately model GPU power behavior when factors such as clock frequency, time, and temperature are not bounded.

6.4 Future Work

While this study lays a solid foundation for GPU power and energy modeling, several areas need further development to enhance accuracy and applicability:

- Kernel-Specific Models: The current model assumes a universal power scaling curve; however, our findings indicate the need for kernel-specific models to capture distinct power profiles across different workloads. Future work should focus on adapting the model to diverse kernels for more accurate power predictions.
- Improved Runtime Estimation: The reliance on predicted execution times introduced errors, particularly for FP32. Developing a more robust runtime estimation framework, perhaps utilizing real-time feedback or machine learning, could reduce reliance on predicted values and enhance energy modeling accuracy.
- Custom observer for Kernel Tuner: There is a difference between the measurements from PowerSensor3 and the NVML observer. We considered PowerSensor3 as the truth data, but the way the data is recorded by these observers differs. The NVML observer requires significantly more iterations to increase data frequency; however, this increases the sensitivity of the measurements to thermal effects. Hence, having a custom observer where the measurement method is the same for both observers will contribute to a more accurate comparison of measurement results.

References

[1] William J Dally, Stephen W Keckler, and David B Kirk. “Evolution of the graphics processing unit (GPU)”. In: *IEEE Micro* 41.6 (2021), pp. 42–51.

[2] *TOP500*. November 2023. URL: `https://www.top500.org/lists/top500/2023/11/highs/`.

[3] Tiffany Trader. *How Argonne Is Preparing for Exascale in 2022*. September 2021. URL: `https://www.hpcwire.com/2021/09/08/how-argonne-is-preparing-for-exascale-in-2022/`.

[4] Ben van Werkhoven. “Kernel Tuner: A search-optimizing GPU code auto-tuner”. In: *Future Generation Computer Systems* 90 (2019), pp. 347–358.

[5] Jiří Filipovič, Filip Petrovič, and Siegfried Benkner. “Autotuning of OpenCL kernels with global optimizations”. In: *Proceedings of the 1st workshop on autotuning and aDaptivity approaches for energy efficient HPC systems*. 2017, pp. 1–6.

[6] Cedric Nugteren and Valeriu Codreanu. “CLTune: A generic auto-tuner for OpenCL kernels”. In: *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. IEEE. 2015, pp. 195–202.

[7] Richard Schoonhoven et al. “Going green: optimizing GPUs for energy efficiency through model-steered auto-tuning”. In: *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE. 2022, pp. 48–59.

[8] Muhammad Fahad et al. “A comparative study of methods for measurement of energy of computing”. In: *Energies* 12.11 (2019), p. 2204.

[9] Robert A Bridges, Neena Imam, and Tiffany M Mintz. “Understanding GPU power: A survey of profiling, modeling, and simulation methods”. In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–27.

[10] Martin Burtscher, Ivan Zecena, and Ziliang Zong. “Measuring GPU power with the K20 built-in sensor”. In: *Proceedings of Workshop on General Purpose Processing Using GPUs*. 2014, pp. 28–36.

[11] Büşra Aslan and Ayse Yilmazer-Metin. “A study on power and energy measurement of nvidia jetson embedded gpus using built-in sensor”. In: *2022 7th International Conference on Computer Science and Engineering (UBMK)*. IEEE. 2022, pp. 1–6.

[12] Abel Paz and Antonio Plaza. “A new morphological anomaly detection algorithm for hyperspectral images and its GPU implementation”. In: *Satellite Data Compression, Communications, and Processing VII*. Vol. 8157. SPIE. 2011, pp. 82–89.

[13] Tia Newhall Suzanne J. Matthews and Kevin C. *Dive into Systems*. Webb. 2020. URL: `https://diveintosystems.org/`.

[14] Kamran Karimi, Neil G Dickson, and Firas Hamze. "A performance comparison of CUDA and OpenCL". In: *arXiv preprint arXiv:1005.2581* (2010).

[15] John Cheng, Max Grossman, and Ty McKercher. *Professional CUDA c programming*. John Wiley & Sons, 2014.

[16] Steven van der Vlugt et al. "PowerSensor3: A Fast and Accurate Open Source Power Measurement Tool". In: *arXiv preprint arXiv:2504.17883* (2025).

[17] Qiang Wang and Xiaowen Chu. "GPGPU power estimation with core and memory frequency scaling". In: *ACM SIGMETRICS Performance Evaluation Review* 45.2 (2017), pp. 73–78.

[18] João Guerreiro et al. "Modeling and decoupling the GPU power consumption for cross-domain DVFS". In: *IEEE Transactions on Parallel and Distributed Systems* 30.11 (2019), pp. 2494–2506.

[19] Gene Wu et al. "GPGPU performance and power estimation using machine learning". In: *2015 IEEE 21st international symposium on high performance computer architecture (HPCA)*. IEEE. 2015, pp. 564–576.

[20] Khronos group. *Open Computing Languag*. 2024. URL: `https://www.khronos.org/opencl/`.

[21] Hitoshi Nagasaka et al. "Statistical power modeling of GPU kernels using performance counters". In: *International conference on green computing*. IEEE. 2010, pp. 115–122.

[22] Shuai Che et al. "Rodinia: A benchmark suite for heterogeneous computing". In: *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee. 2009, pp. 44–54.

[23] Lorenz Braun et al. "A simple model for portable and fast prediction of execution time and power consumption of GPU kernels". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 18.1 (2020), pp. 1–25.

[24] Sunpyo Hong and Hyesoon Kim. "An integrated GPU power and performance model". In: *Proceedings of the 37th annual international symposium on Computer architecture*. 2010, pp. 280–289.

[25] John W Romein and Bram Veenboer. "Powersensor 2: A fast power measurement tool". In: *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. 2018, pp. 111–113.

[26] James H Laros, Phil Pokorny, and David DeBonis. "Powerinsight-a commodity power measurement capability". In: *2013 International Green Computing Conference Proceedings*. IEEE. 2013, pp. 1–6.

[27] *NVML*. NVIDIA. Sept. 2025. URL: `https://docs.nvidia.com/deploy/pdf/NVML_API_Reference_Guide.pdf` (visited on 11/25/2025).

[28] *nvidia-smi — NVIDIA System Management Interface program*. NVIDIA. URL: `https://docs.nvidia.com/deploy/nvidia-smi/index.html` (visited on 11/25/2025).

[29] Jianmin Chen et al. “Statistical GPU power analysis using tree-based methods”. In: *2011 International Green Computing Conference and Workshops*. IEEE. 2011, pp. 1–6.

[30] Jingwen Leng et al. “GPUWattch: Enabling energy optimizations in GPGPUs”. In: *ACM SIGARCH computer architecture news* 41.3 (2013), pp. 487–498.

[31] Vijay Kandiah et al. “AccelWattch: A power modeling framework for modern GPUs”. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, pp. 738–753.

[32] Neda Shalavi et al. “Accurate Calibration of Power Measurements from Internal Power Sensors on NVIDIA Jetson Devices”. In: *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE. 2023, pp. 166–170.

[33] Sayan Ghosh, Sunita Chandrasekaran, and Barbara M Chapman. *Power and Energy Prediction of Multi-GPU Kernels using Nonlinear Regression*. 2013.

[34] Xiaohan Ma et al. “Statistical power consumption analysis and modeling for GPU-based computing”. In: *Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems (HotPower)*. Vol. 1. 2009.

[35] Sunbal Cheema and Gul Khan. “GPU Auto-tuning framework for optimal performance and power consumption”. In: *Proceedings of the 15th Workshop on General Purpose Processing Using GPU*. 2023, pp. 1–6.

[36] Ben Van Werkhoven et al. “Optimizing convolution operations on GPUs using adaptive tiling”. In: *Future Generation Computer Systems* 30 (2014), pp. 14–26.

[37] Satyabrata Sen, Neena Imam, and Chung-Hsing Hsu. “Quality assessment of gpu power profiling mechanisms”. In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2018, pp. 702–711.

[38] Zeyu Yang, Karel Adamek, and Wesley Armour. “Part-time Power Measurements: nvidia-smi’s Lack of Attention”. In: *arXiv preprint arXiv:2312.02741* (2023).

[39] *Elmorlabs pmd-usb*. 2024. URL: `https://www.elmorlabs.com/`.

[40] Roger Labbe. “Kalman and bayesian filters in python”. In: *Chap* 7.246 (2014), p. 4.

[41] Henri Bal et al. “A medium-scale distributed system for computer science research: Infrastructure for the long term”. In: *Computer* 49.5 (2016), pp. 54–63.

[42] *FP32 synthetic kernel source code*. Kernel Tuner. URL: `https://github.com/KernelTuner/kernel_tuner/blob/master/kernel_tuner/energy/energy.py#L13-L38` (visited on 11/25/2025).

[43] Kazuya Matsumoto, Naohito Nakasato, and Stanislav G Sedukhin. “Performance tuning of matrix multiplication in OpenCL on different GPUs and CPUs”. In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE. 2012, pp. 396–405.