Using Analytical Performance/Power Model and Fine-Grained DVFS to Enhance AI Accelerator Energy Efficiency

Zibo Wang
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China

Yijia Zhang*
Peng Cheng Laboratory
Shenzhen, China

Fuchun Wei
Huawei Technologies Co., Ltd
Shenzhen, China

Bingqiang Wang
Peng Cheng Laboratory
Shenzhen, China

Yanlin Liu
Huawei Technologies Co., Ltd
Shenzhen, China

Zhiheng Hu
Huawei Technologies Co., Ltd
Shenzhen, China

Jingyi Zhang
Huawei Technologies Co., Ltd
Shenzhen, China

Xiaoxin Xu
Huawei Technologies Co., Ltd
Shenzhen, China

Jian He
Huawei Technologies Co., Ltd
Shenzhen, China

Xiaoliang Wang
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China

Wanchun Dou
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China

Guihai Chen
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China

Chen Tian*
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China

Abstract

Recent advancements in deep learning have significantly increased AI processors' energy consumption, which is becoming a critical factor limiting AI development. Dynamic Voltage and Frequency Scaling (DVFS) stands as a key method in power optimization. However, due to the latency of DVFS control in AI processors, previous works typically apply DVFS control at the granularity of a program's entire duration or sub-phases, rather than at the level of AI operators.

The advent of millisecond-level DVFS capabilities on the latest Ascend NPU platforms enables us to set frequency individually for single or multiple operators, opening up the opportunity for further enhancing energy efficiency through fine-grained DVFS control. To ensure performance is unaffected in DVFS, our work builds performance and power models for each operator. Through in-depth timeline analysis, we demonstrate that the cycle count of an operator can be modeled as a convex piecewise linear function of frequency, resulting in a performance model with an average error of 1.96%. Moreover, we build power models that incorporate temperature-dependent terms, which enhances the model's precision and results in an average error of 4.62%.

Based on our performance and power models as well as the fine-grained DVFS functionality of Ascend NPU, we propose a DVFS strategy that integrates operator classification, preprocessing, and a genetic algorithm-based search. Experiments on applications including GPT-3 training achieve a reduction in AICore (the computing component within the Ascend NPU) power by 13.44% and NPU chip power by 4.95%, while limiting performance degradation to 1.76%.

*corresponding authors

ASPLOS '25, March 30-April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-0698-1/25/03.
https://doi.org/10.1145/3669940.3707231

CCS Concepts: • **Hardware → Chip-level power issues**; • **Computing methodologies → Modeling methodologies**; • **Computer systems organization → Processors and memory architectures**.

Keywords: AI Accelerator; Performance Model; Power Model; Fine-grained DVFS; Genetic Algorithm

ACM Reference Format:
Zibo Wang, Yijia Zhang, Fuchun Wei, Bingqiang Wang, Yanlin Liu, Zhiheng Hu, Jingyi Zhang, Xiaoxin Xu, Jian He, Xiaoliang Wang, Wanchun Dou, Guihai Chen, and Chen Tian. 2025. Using Analytical Performance/Power Model and Fine-Grained DVFS to Enhance AI Accelerator Energy Efficiency. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3669940.3707231

1 Introduction

In recent years, with the rapid advancement in deep learning, the demand for computational power has increased significantly. Accelerators such as GPUs (Graphics Processing Units) and NPUs (Neural Processing Units) have become primary computing devices in many computational systems. In the Top500 list of high-performance computing systems, 63% of the top 100 systems are equipped with accelerators [37].

High computational power offered by these accelerators comes at the cost of increased power consumption. NVIDIA GPUs' TDP (Thermal Design Power) has increased from 300W for the V100 to 400W for the A100, and up to 700W for the H100 [30], representing a significant portion of GPU server power consumption. It is estimated that 1% of global electricity resources are used in data centers [25], and this proportion is expected to further increase in the future. Consequently, enhancing the energy efficiency of these accelerators is becoming more critical than ever.

Dynamic Voltage and Frequency Scaling (DVFS) is a key method in chip power management. It adjusts the frequency and voltage of the chip to reduce power consumption while reducing chip performance, or to increase power consumption while improving chip performance. Approaches to optimize the energy efficiency of AI accelerators (especially GPUs) through DVFS have been explored in many studies [2, 3, 12, 15, 26, 32, 38, 39, 46, 47].

However, previous research typically sets their DVFS control cycle to be the entire runtime duration of an application [2, 3, 12, 15]. While some more advanced DVFS strategies are capable to adjust for sub-phases of an application [32, 38, 39, 46, 47], their DVFS control cycles are still longer than several seconds, which are not suitable for optimizing AI operators that execute in milliseconds. Although some studies have highlighted the importance of shorter-delay DVFS control [4], their work is only limited to simulations. One reason for the lack of experimental studies on fine-grained DVFS optimization is that NVIDIA data center GPUs currently lack the capability for millisecond-level DVFS control. Our measurements on NVIDIA GPU V100 found that its typical delay for frequency control is 15 milliseconds, and other studies have reported similar delay values [40].

The latest Ascend NPU offers fast-acting frequency control operators with a latency of 1 millisecond, which enables us to set frequency individually for single or multiple operators, and it opens up the opportunity for further enhancing energy efficiency. Leveraging this capability requires accurate models to predict performance and power across frequencies and voltages. Then, the trade-offs between performance and power can be considered to select suitable frequencies to enhance energy efficiency while maintaining performance.

Existing DVFS-aware performance modeling methods typically either summarize patterns from empirical results [1, 6, 12] or build black-box models using machine learning [3, 8, 41, 43]. These approaches do not provide a detailed understanding of the nature of the frequency-performance relationship. In this work, we present a white-box analysis of operator execution in multiple scenarios by comparing the time consumption of core computation and two type of memory access (load and store). Based on this analysis, we conclude that the cycle counts for executing an operator can be modeled as a piecewise linear function of the chip frequency, and with increasing frequency, the slope of each segment gradually increases. Based on these findings, we fit performance models for AI operators under DVFS control, with experimental results for over 5,000 operators on 6 frequency points showing an average error of 1.96%.

To improve energy efficiency, building a power model for AI accelerator is important, and approaches to building power models have been explored in many studies [7, 11, 13, 19, 26, 34]. However, existing work seldom considers the temperature factor in chip power, whose variations could affect a chip's subthreshold leakage current [36]. In this work, we incorporate the temperature factor into power modeling to enhance accuracy. Validation on seven workloads demonstrates that our modeling has an average error of 4.62%.

Based on our performance and power models, we apply DVFS control on AI operators to improve energy efficiency. However, the fine-grained DVFS capability introduces a large search space. To address this challenge, we integrate operator classification, preprocessing, and a genetic algorithm-based search to generate DVFS strategies. Our experiments demonstrate that our approach could reduce AICore power by 13.44%, NPU chip power by 4.95% on average, while the average performance loss is only 1.76%.

This paper summarizes our experience in end-to-end power optimization in Ascend NPU. We hope that sharing our experience can inspire similar efforts and contribute to sustainable computation. In summary, this paper makes the following contributions:

- We present timeline analysis for operators on AI accelerators, demonstrating that the cycle count of an operator can be modeled as a convex piecewise linear function of frequency. Our performance modeling based on this analysis shows an average error of 1.96%.

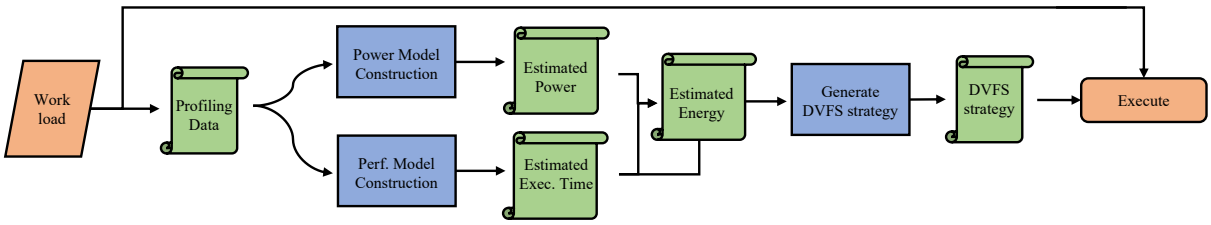


Figure 1. End-to-end energy optimization process.

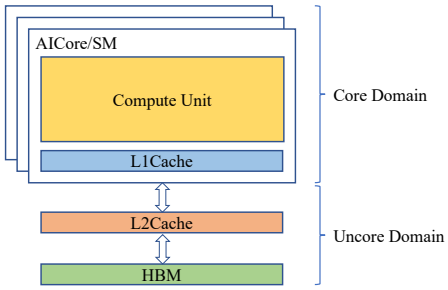


Figure 2. Memory hierarchy of GPU and NPU.

- We develop power consumption models for AI accelerator operators, incorporating a temperature-dependent term to enhance accuracy, with our experiments demonstrating an average error of 4.62%.
- We propose a DVFS strategy for accelerators supporting millisecond-level DVFS control. Our strategy integrates operator classification, preprocessing, and a genetic algorithm-based search. Experiments on applications including GPT-3 training achieve a reduction in AICore power by 13.44% and NPU chip power by 4.95%, while limiting performance degradation to 1.76%.

2 Background

2.1 DVFS

DVFS is a prevalent energy management technique for CPUs, GPUs, and NPUs. Its core aim is to dynamically adjust the voltage and frequency of a chip during operation to meet varying performance requirements, whether enhancing performance or improving energy efficiency in different scenarios. Since the dynamic power consumption of a chip is proportional to $V^2 f$ [10], DVFS can directly alter the chip's energy consumption. Except for the under voltage scenario [27], changing the chip's voltage and frequency in a proper range only affects the speed of computation without compromising correctness. Therefore, DVFS has been widely applied in various scenarios that demand high energy efficiency, such as in smartphone chips and embedded devices.

As DNNs advance and accelerator energy consumption rises, there is a growing trend to implement DVFS on accelerators. Unlike CPUs with refined DVFS capabilities, accelerators often lack optimized strategies, leading to suboptimal energy consumption [46]. Moreover, as these accelerators are often used in high-performance scenarios, energy optimization should not compromise application performance.

2.2 Accelerator Memory Hierarchy

The rapid growth of deep learning networks has led to the development of various accelerators, with Huawei's Ascend NPU being a notable example. The hardware architecture of Ascend NPU is introduced in detail in Ref. [23, 24], and interested readers can refer to them for more information. However, our work does not rely on such detailed hardware architecture but rather on an abstraction of its memory hierarchy, featuring an L1 cache in each AICore, a shared L2 cache, and High Bandwidth Memory (HBM), as illustrated in Fig. 2. This abstraction forms the basis of the generalization of our work, which will be discussed further in Sect. 8.3.

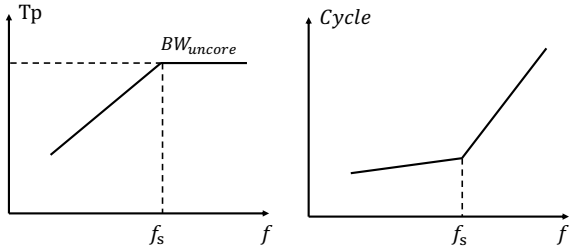
On the Ascend NPU, the core and uncore operate in separate frequency domains, allowing independent frequency adjustment. Since the L1 cache operates within the core domain and the L2 cache, along with HBM, belongs to the uncore domain, the rate of data transfer between the core domain and the uncore domain is determined by both frequencies. Our work focuses on analyzing these transfer rates, applicable regardless of computing unit design specifics.

3 Overview

For clarity, our paper will follow the organization of the end-to-end energy optimization process depicted in Fig. 1. In Sect. 4, we will introduce our timeline analysis method, derive the conclusion that the cycle count of an operator can be modeled as a convex function of frequency, and discuss our performance modeling based on this conclusion. In Sect. 5, we will explain how we incorporate temperature factors into our power modeling process. In Sect. 6, we will present our approach to address the challenges posed by the fine-grained DVFS capability using genetic algorithms for DVFS strategy generation. In addition, in Sect. 7.1, we will briefly outline how we leverage the fine-grained frequency adjustment operators provided by Ascend CANN (Compute Architecture for Neural Networks) to achieve operator-level frequency adjustment when executing DVFS strategies. Note that this work focuses on the frequency scaling of the core domain. The frequency adjustment in the uncore domain is not discussed since Ascend NPU does not support it.

4 Performance Model

In this section, we extensively analyze the mechanism of frequency adjustment on the performance of Ascend NPUs. By comparing the execution time of core computation and load



(a) Throughput-Freq. relation. (b) Cycle-Freq. relation.

Figure 3. The relationship between Ld/St throughput and frequency, and the relationship between cycle count and frequency with fixed memory access volume.

(Ld), store (St) operations in white-box timeline analysis, we identify computational and memory bottlenecks at different frequencies. Based on this analysis, we propose a method for modeling NPU performance under DVFS control.

4.1 Ld/St Bandwidth Analysis

As Ld/St involves different memory hierarchies located in the core domain and uncore domain, their throughput (Tp) is influenced by both the core frequency (f) and the uncore bandwidth limitations, following the relationship below:

$$Tp(f) = min(C \times f \times core\_num, BW_{uncore}). \quad (1)$$

In Eq. (1), C is a hardware-specific constant determined by factors such as the port width of the bus. BW_{uncore} denotes the peak uncore memory bandwidth, influenced by the L2 Cache bandwidth, HBM bandwidth, and L2 Cache hit rate.

According to Eq. (1), Ld/St throughput increases with core frequency until the uncore bandwidth is saturated, as in Fig. 3(a). The saturation frequency point f_s marks this transition and is derived from Eq. (2):

$$f_s = \frac{BW_{uncore}}{C \times core\_num}. \quad (2)$$

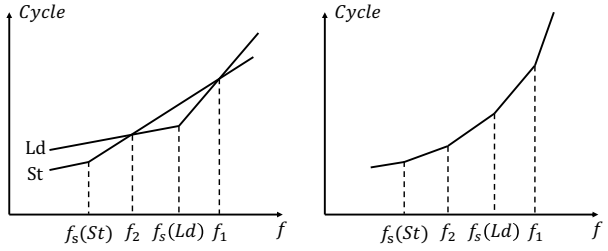
Assuming M is the Ld/St data transfer volume and T is its execution time, we have the following relationship:

$$T = \frac{M}{Tp(f)} + T_0, \quad (3)$$

$$\begin{aligned} Cycle(f) = T \times f &= \frac{M}{Tp(f)} f + T_0 f \\ &= max(\frac{M}{BW_{uncore}} f, \frac{M}{C \times core\_num}) + T_0 f. \end{aligned} \quad (4)$$

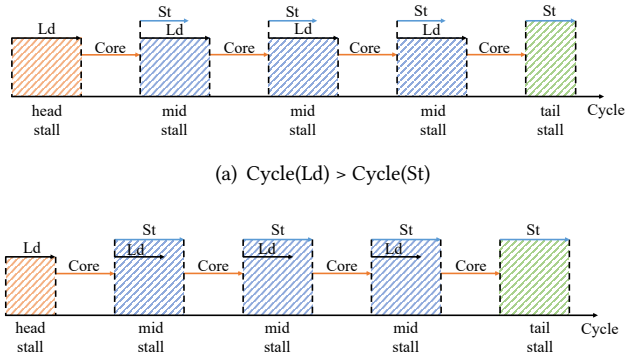
The constant T_0 here represents the fixed time overhead of memory access, including time for initiation of the operation, signal propagation, etc. With a fixed memory access volume M, Eq. (4) illustrates the cycle-frequency relationship for Ld/St, as shown in Fig. 3(b). Note that the term 'cycle' mentioned in this paper refers to the cycles in the core domain.

Our model considers the dependency of the Ld/St time T on the core frequency, an aspect that some previous works have overlooked. For instance, in Ref. [28], they assumed that the stall time of Ld/St is independent of the core frequency.



(a) Cycles of Ld/St. (b) Cycles of the operator.

Figure 4. An example of PingPong-Free, Independent Ld/St Scenarios, where $Cycle_{St}(f)$ and $Cycle_{Ld}(f)$ follow (a), and operator cycles and frequency follow (b).



(a) Cycle(Ld) > Cycle(St)

(b) Cycle(Ld) < Cycle(St)

Figure 5. The execution timeline of operators in PingPong-Free and Independent Ld/St Scenarios.

4.2 Operator Timeline Analysis

In this section, we classify operators on the Ascend NPU into four categories based on PingPong involvement and the independence of Ld and St operations and sequentially dissect each category. Our analysis considers computations and internal data transfers on the AICore as core operations. This encompasses cube and scalar computations by the cube core, scalar computations by the vector core, and MTE1 operations (which facilitate memory transfers within the AICore).

4.2.1 PingPong-Free and Independent Ld/St Scenarios. PingPong, or double-buffering, optimizes computing resource utilization by segmenting buffers to allow simultaneous data pre-transfer and current computation. This ensures data readiness for subsequent computations, reducing wait times for data transfers.

When the workload does not engage in PingPong operations and there are no dependencies between Ld and St, an AI operator that consists of n core computations requires $Cycle(Ld)$ cycles to move-in data at the beginning, and $Cycle(St)$ cycles to move-out data at the end, as well as $(n-1) * max(Cycle(Ld), Cycle(St))$ cycles to move-in and move-out data between the n core computations, as illustrated in Fig. 5. We refer to the time when AICore is not performing computations as 'stall'. The use of $max()$ corresponds to the fact that those move-in and move-out cycles

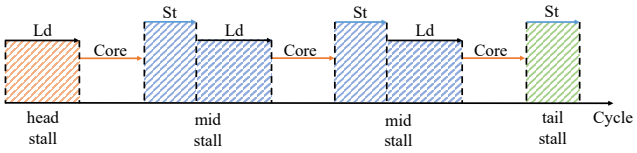


Figure 6. The execution timeline of operators in PingPong-Free and dependent Ld/St Scenarios.

can be processed simultaneously due to the Ld/St independence, and the core computation does not overlap with Ld/St due to the PingPong-free property. As a result, combined with the cycles of Ld/St expressed by Eq. (4), the function describing the number of cycles consumed by the operator in relation to frequency can be represented as:

$$\begin{aligned} Cycle(f) &= Cycle(Ld) + Cycle(St) + n \times Cycle(core) \\ &\quad + (n-1) \times max(Cycle(Ld), Cycle(St)) \\ &= max(a_1 f, c_1) + max(a_2 f, c_2) + n \times Cycle(core) \\ &\quad + (n-1) \times max(a_1 f, c_1, a_2 f, c_2) + (n+1) \times T_0 f. \end{aligned} \tag{5}$$

In this equation, there are $T_0 > 0$, and $a_1, a_2, c_1, c_2 > 0$. It is easy to find out that the given equation represents a piecewise linear function with an increasing derivative through a case-by-case discussion. An example is given as Fig. 4. When $Cycle_{St}(f)$ and $Cycle_{Ld}(f)$ satisfy the relationship in Fig. 4(a), the relationship between the cycles consumed by the operator and the frequency satisfies Fig. 4(b).

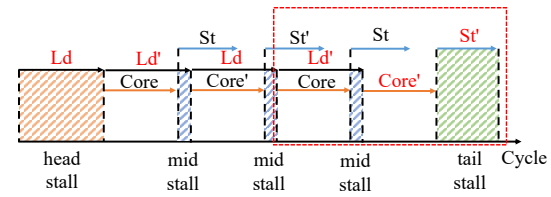
4.2.2 PingPong-Free and Dependent Ld/St Scenarios.

When the workload does not engage in PingPong operations and there are dependencies between Ld and St, data move-in and move-out cannot be processed simultaneously. Therefore, an AI operator that consist of n core computations require $nCycle(Ld)$ cycles to move-in data, $nCycle(core)$ cycles to complete computation and $nCycle(St)$ to move-out data, as illustrated in Fig. 6. Therefore, combined with the cycles of Ld/St expressed by Eq. (4), the function describing the number of cycles consumed by the operator in relation to frequency can be represented as Eq. (6).

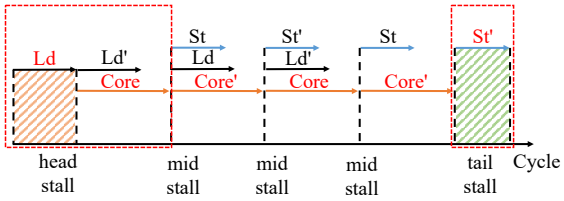
$$\begin{aligned} Cycle(f) &= n * (Cycle(Ld) + Cycle(St) + Cycle(core)) \\ &= n * (max(a_1 f, c_1) + max(a_2 f, c_2) + Cycle(core) + 2T_0 f). \end{aligned} \tag{6}$$

4.2.3 PingPong and Independent Ld/St Scenarios.

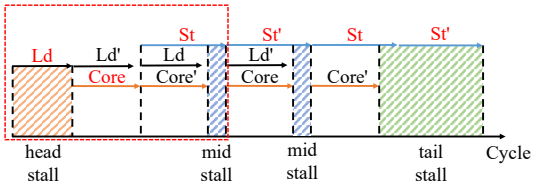
In the PingPong and independent Ld/St scenarios, we discuss the composition of $Cycle(f)$ based on the three possible cases of $max(Cycle(Ld), Cycle(St), Cycle(core))$, as presented in Fig. 7. In the figure, the critical path for each case is highlighted in red font. Through the analysis of the three graphs, we observe that, except for the Ld, St, and core computations that cannot be overlapped at the beginning or end, the remaining segments of the critical path consist of $(n-1) \times max(Cycle(Ld), Cycle(St), Cycle(core))$. Therefore,



(a) $max(Cycle(Ld), Cycle(St), Cycle(core)) = Cycle(Ld)$



(b) $max(Cycle(Ld), Cycle(St), Cycle(core)) = Cycle(core)$



(c) $max(Cycle(Ld), Cycle(St), Cycle(core)) = Cycle(St)$

Figure 7. The execution timeline of operators in PingPong and Independent Ld/St Scenarios.

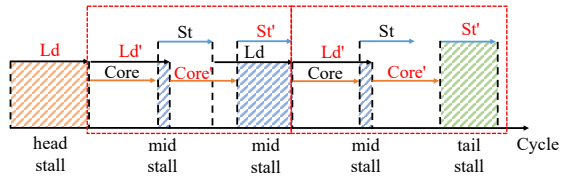
the function representing the number of cycles of the operator with respect to frequency in this scenario can be expressed as Eq. (7).

$$\begin{aligned} Cycle(f) =& Cycle(Ld) + Cycle(core) + Cycle(St) \\ &+ (n-1) * max(Cycle(Ld), Cycle(core), Cycle(St)) \\ =& max(a_1 f, c_1) + Cycle(core) + max(a_2 f, c_2) + (n-1) * max(\\ & max(a_1 f, c_1) + T_0 f, max(a_2 f, c_2) + T_0 f, Cycle(core)) + 2T_0 f \end{aligned} \tag{7}$$

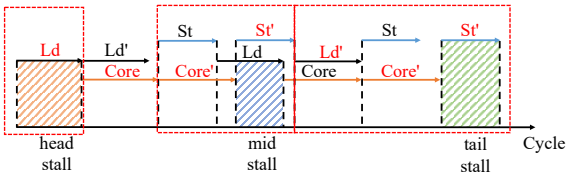
4.2.4 PingPong and Dependent Ld/St Scenarios.

In the PingPong and dependent Ld/St scenarios, we continue to analyze the composition of $Cycle(f)$ based on the three possible cases of $max(Cycle(Ld), Cycle(St), Cycle(core))$. As presented in Fig. 8, the critical path for each case is highlighted in red font. Through the analysis of the three graphs, we observe that, apart from the initial $max(Cycle(Ld), Cycle(St), Cycle(core))$ segment that cannot be overlapped, the remaining portion of the critical path consists of $\frac{n}{2} \times (Cycle(Ld) + Cycle(core) + Cycle(St))$. Therefore, the function representing the cycle count of the operator with respect to frequency in this scenario can be expressed as Eq. (8).

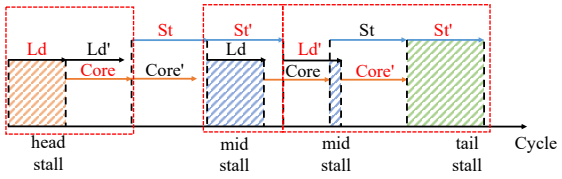
$$\begin{aligned} Cycle(f) =& \frac{n}{2} * (Cycle(Ld) + Cycle(core) + Cycle(St)) \\ &+ max(Cycle(Ld), Cycle(core), Cycle(St)) \\ =& \frac{n}{2} * (max(a_1 f, c_1) + Cycle(core) + max(a_2 f, c_2)) + max(\\ & max(a_1 f, c_1) + T_0 f, max(a_2 f, c_2) + T_0 f, Cycle(core)) + nT_0 f \end{aligned} \tag{8}$$



(a) $max(Cycle(Ld), Cycle(St), Cycle(core)) = Cycle(Ld)$



(b) $max(Cycle(Ld), Cycle(St), Cycle(core)) = Cycle(core)$



(c) $max(Cycle(Ld), Cycle(St), Cycle(core)) = Cycle(St)$

Figure 8. The execution timeline of operators in PingPong and Dependent Ld/St Scenarios.

4.2.5 Timeline Analysis Conclusion. It is easy to find out that Eqs. (5), (6), (7), (8) all represent a piecewise linear function with an increasing derivative through a case-by-case discussion. Moreover, we can find that all 4 equations are composed of a combination of $max()$ and linear functions, both of which are commonly known as convex functions in mathematics. Therefore, the composition of these functions also yields a convex function, which possesses the properties relied upon in Sect. 4.3 for performance model construction.

4.3 Performance Model Construction

Following the conclusion derived in the previous section, it is necessary to consider the relationships between the AICore's frequency range $[f_{min}, f_{max}]$ and the breakpoints of the piecewise linear function to finalize the performance model for the AICore. Taking the function shown in Fig. 4(b) as an example, different relationships between $[f_{min}, f_{max}]$ and $f_s(St)$, f_2, $f_s(Ld)$, f_1 can result in a performance model containing from one to five linear segments.

We have derived the AICore performance model, yet directly solving piecewise linear functions for each operator faces practical challenges:

- The current PMU (performance monitoring unit) cannot track the distribution of stalls within the core during operator execution. Therefore, it is impossible to identify the breakpoints, and consequently, the number of segments of the function cannot be determined.
- Modeling requires running operators at different frequencies to collect performance data, yet each run consumes time and computational resources. Thus, minimizing the number of runs at different frequencies is crucial for conserving both time and resources.

Given the limitations discussed, we opt for a fitting approach to model the cycle count consumed by operators in relation to frequency. As our analysis in Sect. 4.2, we deduce that the functions selected for fitting must be convex.

We choose exponential and quadratic functions for fitting the cycle count to frequency relation, each with three parameters. To fit these, performance data across three frequencies is required per operator. However, this process involves running model training at each frequency, which, despite needing only a single training step after stabilization, adds overhead. To reduce data requirements, we refine our approach by removing the linear term from the quadratic function, keeping only the quadratic and constant terms. Please note that the objective of the function fitting discussed here is the variation of operator cycle consumption with frequency. Combined with $T(f) = Cycle(f)/f$, we finally consider using the following three functions to fit the operator's time consumption: Function 1: $T(f) = \frac{af^2+bf+c}{f}$, Function 2: $T(f) = \frac{af^2+c}{f}$, Function 3: $T(f) = \frac{ae^{bf}+c}{f}$.

Our comparison of the fitting accuracy for the three functions, detailed in Sect. 7.2, reveals that the accuracy of Func. 1 and Func. 2 remains comparable even after the linear term's removal. Moreover, when fitting Func. 2, we can directly calculate parameters a and c, whereas the other functions require the scipy curve_fit function, leading to significant computational time differences. For instance, fitting Func. 2 to 4343 operators in the ShuffleNetV2Plus model takes only 4386ms, versus 105930ms for Func. 1, highlighting a significant time saving. Consequently, we opt for Func. 2 in practical scenarios for its balance of accuracy and efficiency.

Note that the fitting presented here is a best practice derived from the characteristics of piecewise linear functions. Readers may employ other methods to achieve more accurate fitting or directly derive piecewise linear functions.

5 Power Model

Achieving precise energy consumption optimization hinges on an accurate power model. While subthreshold leakage current's temperature dependency is recognized [36], it is often overlooked in current models [7, 11, 13, 19, 26, 34]. Our approach addresses this gap by integrating temperature into our power models, refining the precision of energy consumption predictions.

5.1 Frequency-Voltage Relationship

The Ascend NPU allows frequency adjustment with automatic voltage adaptation by its firmware. It supports frequencies ranging from 1000MHz to 1800MHz in 100MHz increments. Below 1300MHz, voltage is constant regardless

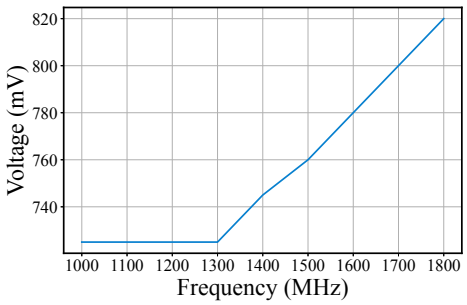


Figure 9. Voltage-Frequency on Ascend NPU.

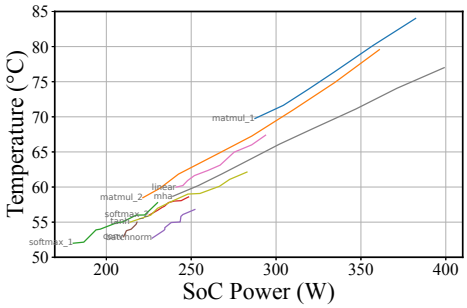


Figure 10. The relationship between temperature and SoC power. Each line is a different operator.

of frequency changes, but above this threshold, voltage increases linearly with frequency, as depicted in Fig. 9. This behavior mirrors the positive correlation between voltage and frequency observed in NVIDIA GPUs [11, 26, 34].

5.2 Power Composition Analysis

A chip's power consumption encompasses dynamic power [10] and static power [5], as formulated by:

$$P = P_{dyn} + P_{static} = AC_L V_{DD}^2 f + V_{DD} I_{leak}. \quad (9)$$

The first term represents dynamic power, where A is the Activity Factor, indicating the probability of a CMOS unit changing its state from 1 to 0 or from 0 to 1 within a clock cycle, and is influenced by circuit design and input signals. C_L is the load capacitance, V_{DD} is the voltage of the chip, and f is the frequency. It encompasses two elements: load-independent power for idle operations like memory refresh and power management ($\beta f V^2$), and load-dependent power arising from computations and communications ($\alpha f V^2$).

The static power P_{static} is primarily generated by leakage current I_{leak}, which can be decomposed into subthreshold leakage current and gate-oxide leakage current [21]. The subthreshold leakage current can be assumed to be linearly correlated with temperature [36], expressed as $I_{sub} = \gamma T + C$, where γ and C are constants. The gate-oxide leakage current is related to the chip's features and is also considered a constant. Therefore, static power can be represented as:

$$\begin{aligned} P_{static} &= (\gamma T + C + I_{gate})V \\ &= \gamma \Delta T V + (\gamma T_0 + C + I_{gate})V. \end{aligned} \quad (10)$$

In the above equation, T_0 denotes the ambient temperature, and $T = T_0 + \Delta T$. For a given accelerator model, $(\gamma T_0 + C + I_{gate})$ remains constant. To simplify, we denote this constant as θ. In summary, the overall power can be represented as:

$$P = \alpha f V^2 + \beta f V^2 + \gamma \Delta T V + \theta V. \quad (11)$$

Below, we divide the power consumption into load-dependent power and load-independent power for separate analysis.

5.3 Load-Independent Power Consumption

The load-independent power is mainly composed of $\beta f V^2$ and θV. We denote this as $P_{AICore,idle}$, such that:

$$P_{AICore,idle} = \beta f V^2 + \theta V. \quad (12)$$

The $P_{AICore,idle}$ is determined by voltage and frequency alone. With constants β and θ for a specific accelerator model, measuring power consumption in the idle state at two frequencies under normal temperature conditions establishes these values for consistent power modeling.

5.4 Load-Dependent Power Consumption

In Eq. (11), load-dependent power can be divided into temperature-independent power $P_{active} = \alpha f V^2$ and temperature-dependent power $P_{\Delta T} = \gamma \Delta T V$. We will discuss each of them separately.

5.4.1 Temperature-Independent Power Consumption.

According to Eq. (11), P_{active} can be obtained by:

$$P_{active} = \alpha_{AICore} f V^2 = P - P_{AICore,idle} - \gamma_{AICore} \Delta T V. \quad (13)$$

By following the method described in Sect. 5.4.2 to obtain γ and measuring ΔT when collecting power data in online computation part in Sect. 5.5, we can calculate α by

$$\alpha_{AICore} = \frac{P - P_{AICore,idle} - \gamma_{AICore} \Delta T V}{f V^2}. \quad (14)$$

Note that differing input shapes, even among operators of the same type, can result in varied power consumption patterns. Hence, individual α values must be calculated for each operator during modeling.

5.4.2 Temperature-Dependent Power Consumption.

The majority of electrical energy during chip operation is converted into heat, causing the chip temperature to increase under load until it reaches equilibrium, where the heat generated equals that dissipated.

To obtain $P_{\Delta T}$, we need to determine γ and ΔT. To obtain γ, we first run a test load. After the load is completed, the temperature and power consumption do not instantaneously return to their values before the load was running. Instead, they decrease gradually, the rate of decrease of power consumption with respect to ΔT is $\frac{dP}{d\Delta T} = \gamma V$. Therefore, by collecting the power consumption and corresponding temperature after the load is completed, we can obtain γ.

We obtain ΔT through experiments under different loads, with results depicted in Fig. 10. These experiments reveal a linear correlation between AICore temperature T and SoC

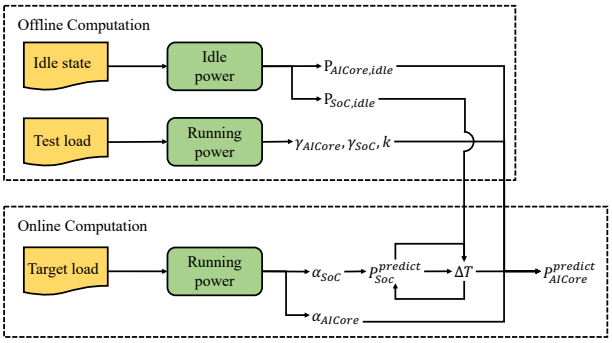


Figure 11. The Construction of a power model.

power, as follow:

$$T = T_0 + k \times P_{SoC}. \tag{15}$$

To calculate ΔT during load operation, we initially measure the SoC power, which encompasses AICore power along with other components like HBM and bus power. These can be measured separately using available tools. The SoC power modeling mirrors the approach for AICore power consumption, as follows:

$$P_{SoC} = \alpha_{SoC} f V^2 + \gamma_{SoC} \Delta T V + P_{SoC,idle}. \tag{16}$$

The interdependence of P_{SoC} on ΔT prevents direct calculation of their values. We adopt an iterative method, starting with $\Delta T = 0$ to estimate P_{SoC}, then substituting this estimate P_{SoC} into Eq. (15) to obtain ΔT, and repeating this process until the values converge, which in our experiments, takes no more than 4 iterations.

5.5 Power Model Construction

Figure 11 depicts the construction of a power model through offline and online computation. Offline involves chip analysis—idle and test load—to extract hardware-related parameters ($P_{AICore,idle}$, $P_{SoC,idle}$, γ_{AICore} and k). Online computation entails gathering power data at specific frequencies during operation to derive load-related parameters and finalize the predictive model.

6 DVFS Strategy Generation

Operators display diverse sensitivities to frequency change. Compute-bound operators like MatMul sacrifice 6.9% performance for a 7.9% power gain, while memory-bound ones like Gelu could trade a 2% performance drop for a 5% or greater power gain. Tailoring a DVFS strategy to these sensitivities is essential for optimizing a chip's energy efficiency.

In a production setting, we initiate DVFS strategy generation by executing the target workload, such as training a model. Simultaneously, we utilize the CANN profiler and lpmi_tool to collect comprehensive performance and power data, which is then used for performance and power modeling of individual operators. Subsequently, operators are classified and preprocessed. Finally, we apply a genetic algorithm, integrating the performance and power models to identify effective DVFS strategies. Current long-lived AI workloads essentially involves the continuous repetition of the same task (iteration). Each iteration invokes a sequence of operators that remains consistent. Therefore, once we optimize a single iteration, the generated policy can be applied to all subsequent iterations.

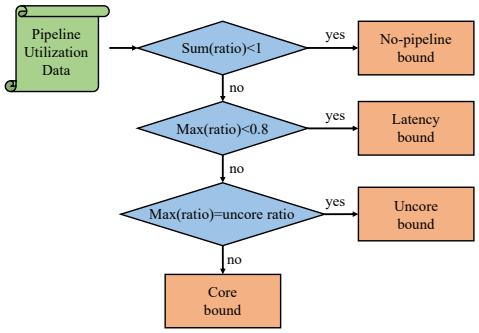


Figure 12. The flowchart of bottleneck classification.

6.1 Classification

The AICore within Ascend NPU comprises components like cube, vector, scalar, and memory transfer units, which may operate concurrently during an operator's execution, each with a distinct utilization rate, termed its 'ratio'. Based on the pipeline utilization data obtained from the CANN profiler, we categorize operators into different bottleneck types:

- An operator is classified as no-pipeline bound if the sum of its ratio in all pipelines is less than 1, indicating the presence of free time during execution. These operators typically exhibit short execution times, with a notable portion of time spent on pre- and post-processing tasks.
- Latency-bound operators, defined by a maximum ratio of less than 0.8, suffer from suboptimal pipeline arrangement. This deficiency may arise due to factors like the lack of PingPong strategies or inherent design flaws.
- Operators are classified as uncore-bound when their maximum ratio pertains to pipelines in the uncore domain. For instance, Ld/St reflect data transfers between the uncore and core domains. The performance of such operators is influenced by frequencies of both domain.
- When the maximum ratio is attributed to the pipeline in the core domain, it is classified as core-bound. Examples include cube-bound, scalar-bound, vector-bound, and MTE1-bound operations. In such cases, the performance of the operator depends on the frequency within the core.

Table 1. Classification of operators based on their sensitivity to AICore frequency.

AICore Frequency-Sensitive Operators	cube-bound, scalar-bound, vector-bound, MTE1-bound, latency-bound operators
AICore Frequency-Insensitive Operators	Ld-bound, St-bound, AICPU, idle and communication operators

Besides compute operators, on the Ascend NPU, there are also AICPU operators, communication operators, and idle modes generated by scheduling, which are minimally affected by the AICore frequency. We thus categorize operators into two types: those sensitive to AICore frequency and those insensitive, as detailed in Table 1.

6.2 Preprocessing

Given the large number of operators in deep learning models, a brute-force search for each in frequency space is impractical due to high computational costs. Therefore, preprocessing operators with knowledge of their bottlenecks is essential. The process involves four key steps, as depicted in Fig. 13. These steps entail:

1. We initially gather the execution sequence and profiling data of operators using the CANN profiler, considering significant gaps between executions as idle time.
2. Based on the profiling data of the operators, we analyze the bottleneck types according to the classification method proposed in Sect. 6.1.
3. We divide the execution process into Low Frequency Candidate (LFC) and High Frequency Candidate (HFC) stages based on the operators' sensitivity to frequency changes, with HFC for those more affected and LFC for the less sensitive. The start of each stage serves as the initial frequency candidate point.
4. Frequency candidates are refined based on the frequency adjustment interval (e.g., 5ms). Candidates with intervals shorter than the threshold are merged with adjacent candidates to obtain new frequency candidates.

6.3 Search

We employ genetic algorithms (GA) [9, 14, 22, 33] to optimize our solution space effectively, leveraging their natural-inspired mechanisms—crossover, mutation, and selection—to evolve towards global optima. This section outlines our GA's initialization and objective function configurations.

Based on the preprocessing described in Sect. 6.2, we obtain n frequency candidate points, denoted as $\{s_1, s_2, ..., s_n\}$, with durations $\{d_1, d_2, ..., d_n\}$. We denote the hardware-supported frequency points as $\{f_1, f_2, ..., f_m\}$.

6.3.1 Initialization. ***Performance Baseline***: We define the performance and power at the highest frequency as the baseline, denoted as $Per_{baseline}$ and $Power_{baseline}$, respectively. When the performance loss target is less than 2%, the lower bound of performance is $Per_{lb} = Per_{baseline} \times 0.98$.
Search Space: As described earlier, the search space can be represented as $\{f_{i,1}, f_{i,2}, \ldots, f_{i,n}\}$, where i denotes different individuals in GA, and each element $f_{i,j}$ belongs to $\{f_1, f_2, \ldots, f_m\}$.
First Generation Population: Our initial generation population comprises baseline and prior individuals, with baseline frequencies maximized at 1800MHz and prior individuals' LFC set to 1600MHz, HFC set to 1800MHz. The rest of the population receives random frequencies ranging from 1000 to 1800 MHz in 100 MHz steps.

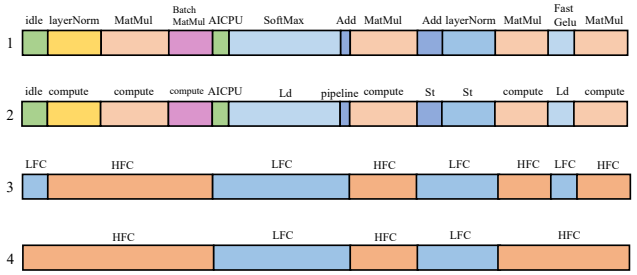


Figure 13. Illustrations of each preprocessing step.

6.3.2 Individual Scoring. For each individual $\{f_{i,1}, f_{i,2}, \ldots, f_{i,n}\}$, we use the obtained performance model and power model to predict the performance $\overline{Per_{i,j}}$ and power consumption $\overline{Power_{i,j}}$ during the interval $[s_j, s_j + d_j]$ for each stage. Then, we sum up the predicted data for all n stages to obtain the performance $\overline{Per_i}$ and power consumption $\overline{Power_i}$.

To ensure compliance with the performance loss threshold, we assign different scores to individuals based on their performance. As per Eq. (17), those not meeting the target (i.e., $\overline{Per_i} < Per_{lb}$) receive lower scores as a penalty.

$$Score_i = \begin{cases} 2 \times \frac{\overline{Per_i}^2}{\overline{Power_i}}, & \overline{Per_i} \geq Per_{lb} \\ \frac{\overline{Per_i}^2}{\overline{Power_i}}, & \overline{Per_i} < Per_{lb} \end{cases} \tag{17}$$

6.3.3 Generating New Individuals. In each iteration, we need to generate the next generation population through crossover and mutation. For crossover, with a certain probability, we exchange the last k frequency settings of two individuals i_1 and i_2, where k is a randomly chosen value. This involves swapping $\{f_{i_1,n-k+1}, f_{i_1,n-k+2}, \ldots, f_{i_1,n}\}$ with $\{f_{i_2,n-k+1}, f_{i_2,n-k+2}, \ldots, f_{i_2,n}\}$. For mutation, with a certain probability, we change $f_{i,j}$ to f_t, where i, j, and f_t are randomly chosen values, and f_t belongs to $\{f_1, f_2, \ldots, f_m\}$.

In each iteration, individuals receive scores that reflect their environmental adaptation and requirement fulfillment, with higher scores denoting superior adaptation. The likelihood of selection for crossover and mutation is score-based, ensuring the propagation of advantageous traits in subsequent generations.

7 Evaluation

In this section, we initially provide an overview of our approach to utilizing fine-grained frequency adjustment operators. Subsequently, we detail the experimental outcomes obtained on the latest Ascend NPU.

7.1 Frequency Setting Mechanism

Ascend CANN features a SetFreq operator for rapid frequency adjustment within 1ms, significantly faster than Nvidia GPU V100's 15ms delay[40]. This capability enables us to execute fine-grained DVFS at the operator level. In

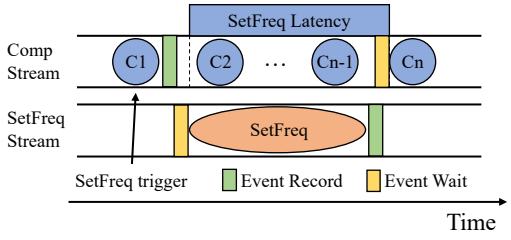


Figure 14. Timeline diagram of executing SetFreqs.

Ascend CANN, this operator can be dispatched and executed in the same way as computational operators. We integrate it into the PyTorch Adaptor[1] and implement a DVFS Executor. This executor reads the strategy generated in the DVFS Strategy Generate phase and automatically inserts SetFreq operators without modifying user code. As shown in Fig. 14, thanks to the low latency and stable activation time of the SetFreq operator, we subtract the SetFreq latency (1ms) from the frequency adjustment time point and identify the last operator before the resulting time point as the SetFreq trigger. At this trigger time, we dispatch and execute a SetFreq operator on a dedicated SetFreq stream. To ensure that the SetFreq operator begins execution at the intended location and finishes execution before the operator at the frequency adjustment time point, we utilize the Event Record and the Event Wait mechanisms in PyTorch to synchronize between the compute stream and the SetFreq stream, achieving precise asynchronous DVFS at the operator level.

7.2 Performance Model

To assess the performance model's accuracy, we used the CANN profiler to gather execution times for operators across seven models: Resnet50, Vit_base, Bert, Deit_small, AlexNet, ShufflenetV2plus, and VGG19. Running each model once at specific frequencies suffices to collect comprehensive performance data for all operators, making the data collection process simplified.

A subset of operators with execution times below 20 microseconds, accounting for 58.3% of the total, show high variability and contribute minimally, at 0.9%, to the total execution time. This suggests that prediction errors on these operators are unlikely to significantly impact the overall prediction errors of the models. Therefore, we opted to exclude them from our analysis. This decision still left us with substantial data, covering over 5,000 operators across 6 frequency points, totaling over 30,000 data points.

For each operator, we apply the three functions from Sect. 4.3 to fit performance data across two to three frequency points. We then use the fitted functions to predict the performance at other frequency points and compare these predictions with the actual performance data obtained from testing. The cumulative distribution function (CDF) of the error rates for the functions is shown in Fig. 15.

[1]https://gitee.com/ascend/pytorch

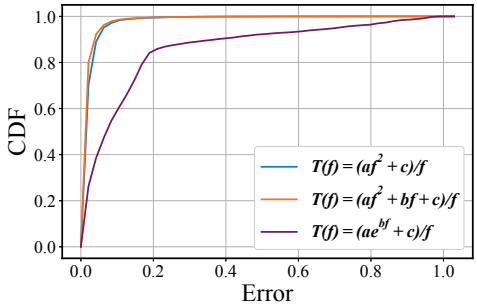


Figure 15. CDF graphs of performance modeling error rates using three functions for fitting.

We select five representative operators—Add, RealDiv, ReduceMean, Conv2D, and BNTrainingUpdate—spanning execution times from 20us to 300us. We present their predicted performance and prediction error rates in Fig. 16. From the examples shown in the figure, we can observe that in most cases, the fitting error rates of Func. 2 are relatively low, accurately capturing the variations in operator running times.

Please note that due to the propensity for overflow when fitting using Func. 3 in Python, we have to limit the range of parameter b to $[0, 10]$, thereby compromising fitting accuracy. Therefore, we conclude that Func. 3 is not appropriate for this scenario. Conversely, Func. 2, which is employed for the final model, yields over 90% accurate predictions within a 5% error margin, exceeds 98% within 10%, and demonstrates an average error of 1.96%, affirming the model's reliability.

Table 2. Error of predicted results of our power model. The first row shows the error ranges, and the second row shows the percentages of predictions.

(0, 1%]	(1%, 5%]	(5%, 10%]	(10%, +∞)	Avg
22.2%	42.6%	42.2%	19.4%	4.62%

7.3 Power Model

To validate the accuracy of the power model, we use the training of GPT3, Bert, VGG19, ResNet50, ViT models, and Softmax and Tanh operators as our test subjects. Power data was collected at varying frequencies with Ascend's lpmi_tool, and we use the 1000MHz and 1800MHz data to build our model. This model was then applied to predict consumption at additional frequencies, with results detailed in Table 2.

Table 2 shows that our power model achieves an error of less than 1% for 22.2% of predictions, less than 5% for 64.8% of predictions, and less than 10% for over 80% of predictions. The average error is 4.62%.

Additionally, by setting the temperature coefficient γ to zero, we assessed the model's prediction accuracy without considering thermal effects, resulting in an average error of 4.97%. In fact, if our model does not consider thermal effects, the temperature-dependent power $P_{\Delta T} = \gamma \Delta T V$ will be classified as active power $P_{active} = \alpha f V^2$. If we approximately assume $V = kf$, then there are $P_{\Delta T} \propto f$, $P_{active} \propto f^3$, resulting in an increased rate of change, thus introducing

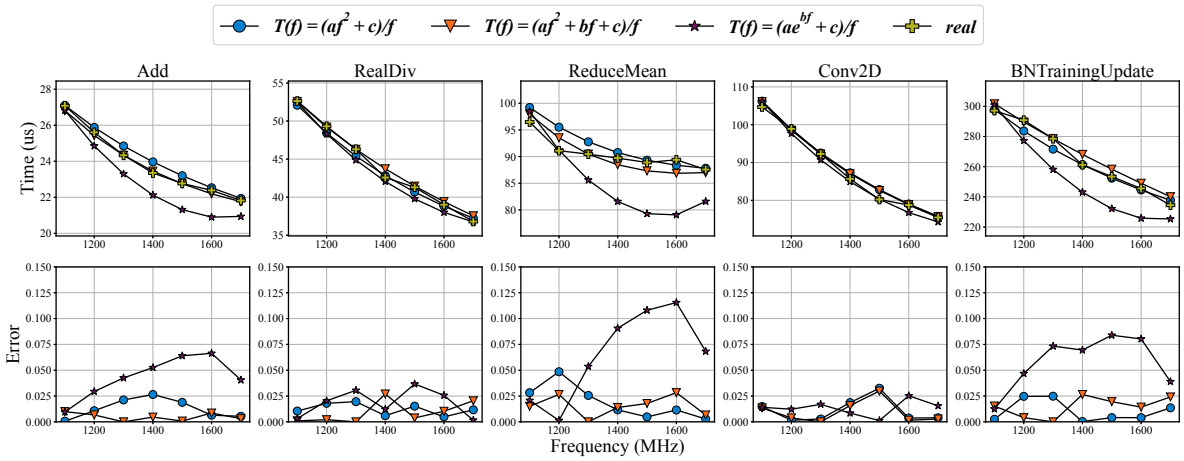


Figure 16. The performance prediction results and error rates of five example operators.

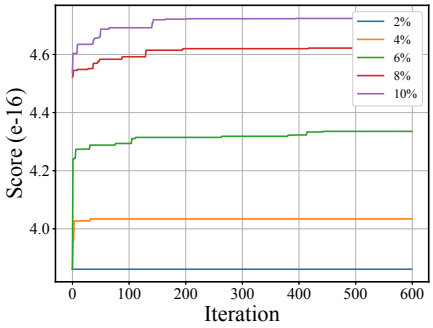


Figure 17. Score of the fittest individual during the search process under different performance lower bounds.

modeling errors. Our measured data indicate that the $P_{\Delta T}$ of AICore is approximately 3 to 8 W, accounting for about 10% to 20% of the AICore's total power. Given this relatively low proportion, the accuracy improvement brought about by introducing the temperature factor is diminished. Nevertheless, this does not imply that our efforts are unimportant, as our theoretical derivation can be translated into a small amount of engineering code implementation, which can further enhance the accuracy of energy consumption modeling.

7.4 End-to-End Energy Optimization

We perform end-to-end energy optimization on several common deep learning models, following the process in Fig. 1. We start by training models at 1000MHz and 1800MHz, collecting performance and power data once stable training is achieved. These data are then used to construct performance and power models as described in Sect. 4 and 5.

We then integrate the developed performance and power models into the DVFS Strategy Generation process for individual scoring. Utilizing the generated strategy, we conduct a subsequent training iteration. Upon stabilization, we collect and compare performance and power data against the baseline 1800MHz frequency results.

When generating DVFS strategy, we set the frequency adjustment interval to 5ms, the population size per round to 200, the mutation rate of the population to 0.15, and run 600 iterations. Using GPT3 model training as a test workload, we evaluated the search algorithm's performance and the effectiveness of strategies under various performance loss targets. As depicted in Fig. 17, stricter performance loss targets accelerate search convergence. When the performance loss target is set to 2%, the prior individuals we introduce are already optimal. Across all configurations, the search converges within 500 rounds, each within 2.5 seconds.

In our experiments, GPT3 has around 18,000 operators per iteration. The generated policy triggers 821 SetFreq, averaging one SetFreq for every 20 operators. With the performance loss target set at 10%, our policy sets the LFC to low values, around 1200 MHz, while the frequency for the HFC remains high. To validate the policy, we reviewed the visualized trace collected by the CANN profiler during GPT-3 training. For example, we find an instance where, right before executing a compute-bound Matmul operator, the AICore frequency is increased from 1100MHz to 1800MHz. After the operator finished, the frequency reverted to 1100 MHz,, which aligns with the required frequency scaling logic.

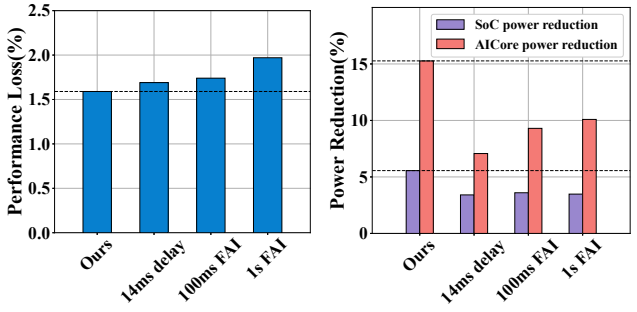
Table 3 displays the impact of varying performance loss targets on power consumption using the GPT3 model. A 2% performance loss target emerges as optimal, achieving substantial power savings with minimal performance impact. Beyond this target, the power reduction rate slows, indicating diminishing returns. When applying the 2% target in production, we observe an average AICore power reduction of 13.44% and a 4.95% decrease in SoC power across four models, with an average performance loss of 1.76%. Among these, the BERT model achieves the highest AICore power reduction at 17.08%.

To demonstrate the significance of the millisecond-level DVFS control and fine-grained DVFS, we conduct some comparative experiments on the training of GPT-3. In the first experiment, we delay the deployment of the SetFreq operator by 14ms to simulate the delay of the frequency adjustment operator in NVIDIA V100. As in Fig. 18, this delay resulted in a 1.69% performance drop but significantly less power savings: 7.07% for AICore and 3.41% for SoC. This is an expected

Table 3. End-to-end experimental results.

Model	Performance loss target	Original iteration time	Iteration time under DVFS	Performance loss	Original SoC power	SoC power under DVFS	SoC power reduction	Original AICore power	AICore power under DVFS	AICore power reduction
GPT3	2%	11.29s	11.47s	1.59%	250.04W	236.14W	5.56%	45.92W	38.91W	15.27%
GPT3	4%	11.29s	11.66s	3.28%	250.04W	232.58W	6.98%	45.92W	36.62W	20.25%
GPT3	6%	11.29s	11.85s	4.96%	250.04W	226.65W	9.35%	45.92W	34.13W	25.68%
GPT3	8%	11.29s	12.10s	7.17%	250.04W	223.40W	10.65%	45.92W	32.25W	29.77%
GPT3	10%	11.29s	12.26s	8.59%	250.04W	220.11W	11.97%	45.92W	31.22W	32.01%
BERT	2%	0.309s	0.315s	1.78%	261.8W	244.5W	6.61%	56.2W	46.6W	17.08%
ResNet50	2%	0.317s	0.322s	1.8%	232.7W	224.7W	3.44%	43.8W	38.96W	11.05%
ResNet152	2%	0.637s	0.649s	1.88%	235.6W	225.7W	4.20%	46.94W	42.07W	10.37%

result. Due to the delay of the frequency adjustment operator, some operators in the LFC run at higher frequencies before the lower frequency takes effect, while some operators in the HFC run at lower frequencies before the higher frequency takes effect. Consequently, this delay not only leads to a greater performance decline but also reduces the energy savings. This result underscore the enhanced power optimization capabilities of millisecond-level DVFS control.



(a) Performance Loss (b) Power Reduction

Figure 18. Comparative experiments on GPT-3 training.

In the second set of experiments, we generate policy with different frequency adjustment interval (FAI in Fig. 18), i.e. 100 ms and 1 s, and merge frequency candidates shorter than this granularity. When the frequency adjustment interval is set to 100 ms, the generated policy triggers 38 SetFreq commands per training iteration, resulting in a 1.74% performance drop, along with a 3.60% reduction in SoC power and a 9.30% reduction in AICore power. When the interval is set to 1 s, only 4 SetFreq commands are triggered, leading to a 1.97% performance drop, a 3.48% SoC power reduction, and a 10.09% AICore power reduction. These results indicate that with a larger frequency adjustment interval, fewer SetFreq commands are triggered, causing many memory-bound and compute-bound operators to run at the same frequency. This not only misses many opportunities to reduce energy consumption but also leads to greater performance degradation.

8 Discussion

In this section, we will discuss the advantages over model-free approaches, limitation of NPU DVFS domain, generalization to other hardware, and model inference scenarios.

8.1 Advantages over model-free approaches

Our work builds performance and power models to guide DVFS. Another potential approach is a model-free DVFS method, which uses feedback from real systems rather than model-predicted results for individual scoring in the genetic algorithm. Our work adopts a modeling approach instead of a model-free approach because employing performance/power modeling enables us to rapidly assess policies. For instance, with GPT-3, by incorporating performance and power model within the genetic algorithm, we can evaluate a policy in just milliseconds. Furthermore, we can utilize multi-processing techniques to concurrently execute this process, allowing us to assess 20,000 strategies within 5 minutes. If a purely experimental approach were used, each training round, which takes 11 seconds, could only evaluate one policy. Within the same timeframe, only 30 individuals could be assessed, significantly slowing down the search process.

8.2 Limitation of NPU DVFS domain

We have currently implemented DVFS only on the AICore, achieving around a 15% reduction in power consumption. However, other uncore components on the SoC, such as HBM and AICPU, lack frequency-tuning capabilities. Depending on the different AICore utilization rates of various operators, the power consumption of these uncore components can range from 10% to 90% of the SoC's total power consumption, averaging around 80%, which limits the overall power savings, which limits the overall power savings. This limitation is also seen in other AI accelerators, like the Nvidia A100, where DVFS is restricted to computing cores. In the future, when hardware supports frequency tuning for these uncore components, we will utilize these capabilities to further enhance the benefits of frequency tuning.

8.3 Generalization to other hardware

This work consists of three parts: the performance model, the power model, and the policy generation. We discuss the generalization of each component separately.

Regarding the performance model, as described in Sect. 2.2, our model is built upon an abstraction of the memory hierarchy shown in Fig. 2, rather than the detailed hardware architecture. Therefore, we believe the proposed performance model can in principle be applied to hardware platforms that share the same memory hierarchy, such as Nvidia GPUs [29] and Google TPUs [17]. For example, similar abstraction has been used in building the CRISP performance model for GPUs [28]. We also analyze the performance data presented in several GPU performance modeling works [2, 16, 46] and

find that the relationship between cycles and frequency is a convex function, which aligns with our analysis conclusions. Thus, our analysis conclusions can also be applied in those works to derive the performance models. On the other hand, our performance model cannot currently generalize to out-of-order accelerator, and Fig. 5, 6, 7 and 8 may not cover more complex execution scenarios. Therefore, our model may not be applicable to processors with more intricate execution mechanisms, such as those featuring multiple in-flight memory accesses and overlapping out-of-order speculative execution characteristics, like CPUs.

As for power model, since it is not based on any hardware-specific details but instead relies on a combination of physical principles, we believe it represents a general technique that can be easily applied to other hardware platforms.

As for policy generation, the classification in Sect. 6.1 and preprocessing in Sect. 6.2 are tailored to our experimental experience on the Ascend NPU platform and the CANN Profiler output data, making this aspect specific to our hardware. Generalizing it to other platforms requires adjusting algorithm parameters based on profiling data collected from operators running on the target hardware. Nonetheless, the core concept—classifying memory-bound operators as LFC and compute-bound operators as HFC—is broadly applicable. Additionally, the use of a genetic algorithm for strategy generation can serve as a reference for similar optimization tasks on other hardware platforms.

8.4 Model inference scenarios

Optimizing energy efficiency for large model inference is a critical issue [18, 35]. While our work focuses on model training, it can theoretically extend to inference tasks as well. Our preliminary experiment on the NPU based on llama2 inference codes from ModelLink[2] shows that by lowering the frequency of all operators to 1300 MHz, we can achieve a 2.48% performance degradation in exchange for an 11.26% reduction in SoC power consumption and a 25.06% reduction in AICore power consumption. This significant reduction in power consumption is a result of the host-bound nature of model inference tasks on our device. In this scenario, the CPU dispatches operators at a slower rate than the NPU can execute them, causing the NPU to experience idle periods. As a result, when we lower the NPU frequency, although the operator execution time increases, this primarily fills the existing NPU idle time. This allows us to reduce the NPU's frequency to 1300 MHz or even lower without significant performance degradation. However, since efficient model inference systems typically require complex scheduling designs [31, 44], further research is needed to fully explore and optimize DVFS for these systems.

[2]https://gitee.com/ascend/ModelLink

9 Related Work

Many studies have investigated performance modeling methods for AI accelerators, which can be broadly categorized into analytical models [28, 40, 42] and statistical models [3, 8, 41, 43]. Statistical models leverage machine learning algorithms to predict performance based on hardware counter metrics collected from accelerators. Analytical models, on the other hand, emphasize analyzing the operational mechanisms of accelerators, estimating performance from perspectives like instruction stalls, pipeline arrangements, and resource competition. Our work concentrates on analyzing the impact of DVFS on performance. By properly discussing multiple important scenarios, our analytical model demonstrates that the execution cycle count of operators is a convex piecewise linear function of frequency, providing strong support for our selection of fitting functions in performance modeling.

Power modeling methods for AI accelerators have also been extensively studied [7, 11, 13, 19, 26, 34], including both analytical models, and statistical models. Refs. [19, 26] provide a good summary of the impact of DVFS on chip power. However, these studies typically do not consider the temperature dependence of power. Our work enhances existing DVFS-aware models by incorporating temperature effects.

Many studies have explored DVFS strategies on AI accelerators. These works mainly focus on NVIDIA GPU platforms [2, 8, 12, 20, 39, 45, 47], or on AMD GPU platforms [2, 32]. Our work is the first to study the DVFS strategy on Ascend NPU platforms. Moreover, while existing GPU DVFS studies control DVFS at the granularity of program runtime or sub-stages, our work investigates the frequency selection at the operator level on AI accelerators.

10 Conclusion

In this work, we present our experience with end-to-end energy optimization during model training. This includes white-box timeline analysis of operator performance with respect to frequency, consideration of temperature in power modeling, and generating effective DVFS strategies based on genetic algorithms. Experiments conducted on Ascend NPU demonstrate that our approach achieves 13.44% power reduction while maintaining performance degradation below 2%. Our methodology is not tied to specific hardware designs, hence we believe that this work can be readily extended to other types of accelerators, including GPUs, TPUs, and others. We hope that sharing our experience can inspire similar efforts and contribute to sustainable computation.

Acknowledgments

The authors would like to thank the anonymous reviewers and our shepherd David Meisner for their insightful comments. This work was supported by the National Natural Science Foundation of China under Grant Numbers 62325205, 62072228, and 62172204.

References

[1] Ghazanfar Ali, Sridutt Bhalachandra, Nicholas J. Wright, Mert Side, and Yong Chen. Optimal gpu frequency selection using multi-objective approaches for hpc systems. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2022.

[2] Ghazanfar Ali, Mert Side, Sridutt Bhalachandra, Nicholas J. Wright, and Yong Chen. An automated and portable method for selecting an optimal gpu frequency. *Future Generation Computer Systems*, 149:71–88, 2023.

[3] Ghazanfar Ali, Mert Side, Sridutt Bhalachandra, Nicholas J. Wright, and Yong Chen. Performance-aware energy-efficient gpu frequency selection using dnn-based models. In *Proceedings of the 52nd International Conference on Parallel Processing*, ICPP '23, page 433–442, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Srikant Bharadwaj, Shomit Das, Kaushik Mazumdar, Bradford M. Beckmann, and Stephen Kosonocky. Predict; don't react for enabling efficient fine-grain dvfs in gpus. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ASPLOS '23, page 253–267, New York, NY, USA, 2024. Association for Computing Machinery.

[5] J. Adam Butts and Gurindar S. Sohi. A static power model for architects. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 33, page 191–201, New York, NY, USA, 2000. Association for Computing Machinery.

[6] Enrico Calore, Alessandro Gabbana, Sebastiano Fabio Schifano, and Raffaele Tripiccione. Evaluation of dvfs techniques on modern hpc processors and accelerators for energy-aware applications. *Concurrency and Computation: Practice and Experience*, 29(12):e4143, 2017. e4143 cpe.4143.

[7] Bishwajit Dutta, Vignesh Adhinarayanan, and Wu-chun Feng. Gpu power prediction via ensemble machine learning for dvfs space exploration. In *Proceedings of the 15th ACM International Conference on Computing Frontiers*, CF '18, page 240–243, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Kaijie Fan, Biagio Cosenza, and Ben Juurlink. Predictable gpus frequency scaling for energy and performance. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP '19, New York, NY, USA, 2019. Association for Computing Machinery.

[9] David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[10] R. Gonzalez, B.M. Gordon, and M.A. Horowitz. Supply and threshold voltage scaling for low power cmos. *IEEE Journal of Solid-State Circuits*, 32(8):1210–1216, 1997.

[11] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. Gpgpu power modeling for multi-domain voltage-frequency scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 789–800, 2018.

[12] João Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomás. Dvfs-aware application classification to improve gpgpus energy efficiency. *Parallel Computing*, 83:93–117, 2019.

[13] João Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomás. Modeling and decoupling the gpu power consumption for cross-domain dvfs. *IEEE Transactions on Parallel and Distributed Systems*, 30(11):2494–2506, 2019.

[14] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[15] Shashikant Ilager, Rajeev Muralidhar, Kotagiri Rammohanrao, and Rajkumar Buyya. A data-driven frequency scaling approach for deadline-aware energy efficient scheduling on graphics processing units (gpus). In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 579–588, 2020.

[16] Joseph Issa and Silvia Figueira. A performance estimation model for gpu-based systems. In *2012 2nd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)*, pages 279–283, 2012.

[17] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, page 1–12, New York, NY, USA, 2017. Association for Computing Machinery.

[18] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. Slo-aware gpu dvfs for energy-efficient llm inference serving. *IEEE Computer Architecture Letters*, 23(2):150–153, 2024.

[19] Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G. Rogers, Tor M. Aamodt, and Nikos Hardavellas. Accelwattch: A power modeling framework for modern gpus. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 738–753, New York, NY, USA, 2021. Association for Computing Machinery.

[20] Dong-Ki Kang, Yun-Gi Ha, Limei Peng, and Chan-Hyun Youn. Cooperative distributed gpu power capping for deep learning clusters. *IEEE Transactions on Industrial Electronics*, 69(7):7244–7254, 2022.

[21] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, 2003.

[22] Davis Lawrence. Handbook of genetic algorithms. *Van Nostrand Reinhold*, 1991.

[23] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 789–801, 2021.

[24] Heng Liao, Jiajin Tu, Jing Xia, and Xiping Zhou. Davinci: A scalable architecture for neural network computing. In *Hot Chips Symposium*, pages 1–44, 2019.

[25] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[26] Xinxin Mei, Qiang Wang, and Xiaowen Chu. A survey and measurement study of gpu dvfs on energy conservation. *Digital Communications and Networks*, 3(2):89–100, 2017.

[27] Francisco Mendes, Pedro Tomás, and Nuno Roma. Decoupling gpgpu voltage-frequency scaling for deep-learning applications. *Journal of Parallel and Distributed Computing*, 165:32–51, 2022.

[28] Rajib Nath and Dean Tullsen. The crisp performance model for dynamic voltage and frequency scaling in a gpgpu. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 281–293, 2015.

[29] NVIDIA. Accessed: 2024-05. cuda c++ programming guide. *https://docs.nvidia.com/cuda/cuda-c-programming-guide/*.

[30] NVIDIA. Accessed: 2024-05. nvidia h100 tensor core gpu. *https://www.nvidia.com/en-us/data-center/h100/*.

[31] Hyungjun Oh, Kihong Kim, Jaemin Kim, Sungkyun Kim, Junyeol Lee, Du-seong Chang, and Jiwon Seo. Exegpt: Constraint-aware resource scheduling for llm inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 369–384, New York, NY, USA, 2024. Association for Computing Machinery.

[32] Indrani Paul, Wei Huang, Manish Arora, and Sudhakar Yalamanchili. Harmonia: Balancing compute and memory power in high-performance gpus. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, page 54–65, New York, NY, USA, 2015. Association for Computing Machinery.

[33] Kumara Sastry, David Goldberg, and Graham Kendall. *Genetic Algorithms*, pages 97–125. Springer US, Boston, MA, 2005.

[34] Richard Schoonhoven, Bram Veenboer, Ben Van Werkhoven, and K. Joost Batenburg. 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)*, pages 48–59, 2022.

[35] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. Dynamollm: Designing llm inference clusters for performance and energy efficiency. *arXiv:2408.00741*, 2024.

[36] Hameedah Sultan, Shashank Varshney, and Smruti R Sarangi. Is leakage power a linear function of temperature? *arXiv:1809.03147*, 2018.

[37] TOP500. Accessed: 2023-11. top500 list (november 2023). *https://www.top500.org/lists/top500/2023/11/*.

[38] Farui Wang, Meng Hao, Weizhe Zhang, and Zheng Wang. Model-free gpu online energy optimization. *IEEE Transactions on Sustainable Computing*, 9(2):141–154, 2024.

[39] Farui Wang, Weizhe Zhang, Shichao Lai, Meng Hao, and Zheng Wang. Dynamic gpu energy optimization for machine learning training workloads. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2943–2954, 2022.

[40] Qiang Wang and Xiaowen Chu. Gpgpu performance estimation with core and memory frequency scaling. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2865–2881, 2020.

[41] Qiang Wang, Chengjian Liu, and Xiaowen Chu. Gpgpu performance estimation for frequency scaling using cross-benchmarking. GPGPU '20, page 31–40, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Zhuowei Wang, Xiaoyu Song, Lianglun Cheng, Hai Wan, Wuqing Zhao, and Tao Wang. Warp-aware adaptive energy efficiency calibration for multi-gpu systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(5):1676–1690, 2023.

[43] Gene Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. Gpgpu performance and power estimation using machine learning. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 564–576, 2015.

[44] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA, July 2022. USENIX Association.

[45] Junyeol Yu, Jongseok Kim, and Euiseong Seo. Know your enemy to save cloud energy: Energy-performance characterization of machine learning serving. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 842–854, 2023.

[46] Yijia Zhang, Qiang Wang, Zhe Lin, Pengxiang Xu, and Bingqiang Wang. Improving gpu energy efficiency through an application-transparent frequency scaling policy with performance assurance. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 769–785, New York, NY, USA, 2024. Association for Computing Machinery.

[47] Pengfei Zou, Ang Li, Kevin Barker, and Rong Ge. Indicator-directed dynamic power management for iterative workloads on gpu-accelerated systems. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 559–568, 2020.