EQUILIBRIA : Co-Optimizing Energy and Latency in Online ML-based Stream Processing Systems

Sejeong Oh, Soyang Baek, Gordon Euhyun Moon and Sungyong Park*
Sogang University, Seoul, Republic of Korea
{sjoh2, goryne, ehmoon, parksy}@sogang.ac.kr

***Abstract*—An Online machine learning (ML)-based streaming processing system (SPS) combines real-time stream processing with continuous, incremental learning through simultaneous model training and inference. This system processes large, dynamic, high-velocity data streams while adapting its models to improve performance over time. However, balancing the tradeoff between latency and energy efficiency remains a critical challenge, which has not been adequately addressed in prior research. This paper introduces EQUILIBRIA, a novel framework designed to co-optimize power consumption and latency in Online ML-based SPS. EQUILIBRIA integrates dynamic voltage and frequency scaling (DVFS) with two innovative energy optimization strategies. First, a Pareto-based clock frequency adjustment mechanism dynamically tunes both core and memory clock frequencies to reduce latency while minimizing energy consumption. Second, a two-tier threshold training management technique optimizes energy use by periodically pausing and resuming model training once accuracy requirements are met, all while preserving latency. Experimental evaluations across various queries and traffic scenarios demonstrate that EQUILIBRIA achieves up to 58% energy savings without compromising latency, making a significant step forwards in energy-efficient, high-performance streaming analytics for modern, rapidly evolving data environments.**

***Index Terms*—energy efficiency, stream processing systems, machine learning, dynamic optimization, dynamic voltage and frequency scaling**

I. INTRODUCTION

Modern computing infrastructures must increasingly handle massive real-time data streams to derive instantaneous insights. Applications such as IoT sensors, social media platforms, autonomous vehicles, and large-scale online services generate diverse and enormous volumes of data at unparalleled speeds [1], [2]. To effectively optimize systems in such dynamic environments, continuous adaptation to fluctuations in input characteristics and usage patterns is essential. Traditional static or offline management techniques, however, struggle to keep up with these ever-changing dynamics.

Online machine learning (ML) has emerged as a crucial solution to meet these adaptive system requirements [3]. By continuously learning from real-time data through simultaneous model inference (predictions) and updates (training), Online ML enables rapid adaptation to dynamic performance demands [4], [5]. This capability is particularly vital for stream processing systems (SPS) [6], where real-time optimization must process continuous data streams with a responsiveness and flexibility that surpass traditional post-hoc tuning or heuristic-based methods.

In Online ML-based SPS, latency is a key performance metric, as processing bottlenecks can disrupt the entire execution cycle, undermining real-time adaptability. GPUs, with their parallel processing capabilities, are essential for accelerating both inference and training tasks. However, GPU acceleration, while effective at reducing latency, often leads to higher power consumption due to increased utilization.

Despite this, prior research [6], [7], [8], [9], [10] has primarily focused on either minimizing latency in GPU-accelerated environments [6], [8], [10] or reducing power consumption in CPU-based SPS environments [7], [9], leaving less attention on achieving both real-time processing and energy efficiency simultaneously. As a result, there is a clear need for an end-to-end optimization strategy that addresses the entire Online ML-based SPS pipeline, one that reduces the high energy consumption caused by GPU acceleration while maintaining the low latency essential for real-time analytics.

In this paper, we propose EQUILIBRIA, a novel framework for Online ML-based SPS that co-optimizes energy and latency. Through two key innovations described below, EQUILIBRIA achieves low latency and low power consumption in complex dynamic streaming environments.

Dynamic Adjustment of Clock Frequencies. To minimize energy waste, EQUILIBRIA employs a Pareto optima algorithm with dual objectives—latency and power consumption—to dynamically adjust the GPU's core clock frequency (CCF) and memory clock frequency (MCF) via dynamic voltage and frequency scaling (DVFS). DVFS reduces power consumption during runtime by dynamically adjusting device voltage and clock frequency while ensuring that performance remains within acceptable limits. Our approach selects optimal clock frequencies from the Pareto front, based on a normalized distance-based method, to meet user-defined performance requirements. This method dynamically adapts to runtime changes in user conditions by reselecting optimal frequencies from the set as needed.

Two-Tier Threshold Training Management. EQUILIBRIA employs a two-tier threshold management technique to optimize GPU utilization for energy efficiency during model training. This method continuously monitors the model's real-time accuracy and compares it to a moving average, using predefined thresholds to decide when to pause or resume training. When training is paused, the GPU enters a low-

*Corresponding author.

2993-2114/25/$31.00 ©2025 IEEE
DOI 10.1109/CCGRID64434.2025.00061

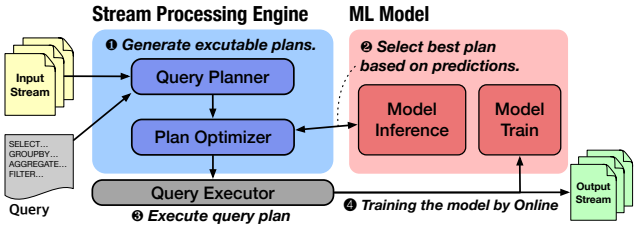


Fig. 1: An overview of Online ML-based SPS.

power idle state, remaining there until training resumes. This cycle repeats until the accuracy reaches the stop threshold once more.

By leveraging these two innovative approaches, EQUILIBRIA achieves energy efficiency while maintaining low latency. Experimental evaluations across various queries and traffic scenarios demonstrate that our EQUILIBRIA enhances energy efficiency by up to 58% without sacrificing latency.

In summary, this paper presents the following contributions:

- We propose a novel management framework that employs a Pareto optima approach to simultaneously optimize latency and energy efficiency in SPS environments.
- We introduce a two-tier threshold training management strategy to enable energy-efficient use of ML models.
- We implement a novel Pareto optimization algorithm specifically designed for the SPS environment and GPU DVFS, deploying it on a real system and evaluating its performance with real-world workloads.
- Beyond its application in Online ML-based SPS, the Pareto optima approach can be readily adapted to traditional GPU-based SPS, enhancing overall system efficiency.

II. BACKGROUND AND MOTIVATION

A. Online ML-based SPS

In recent years, Online ML has gained significant attention across diverse application domains due to its ability to continuously update and utilize models for inference with real-time data streams [3]. Unlike traditional offline learning methods, Online ML rapidly adapts to dynamic environments, such as changes in data distribution or sudden traffic spikes, ensuring consistent model performance over time. Integrating Online ML with SPS enables real-time optimization of large-scale streaming data by performing both inference and model training as new data arrives.

Fig. 1 illustrates the workflow of an Online ML-based SPS. The system processes an input stream according to a given query through several stages. First, it generates multiple executable plans capable of addressing the query ❶. Next, it selects the optimal plan by leveraging an ML model that predicts the real-world performance of each candidate ❷. After selecting the best plan, the system executes it ❸, continuously refining the ML model in real time by learning from log data generated during execution ❹. This iterative process enables the Online ML-based SPS to identify and execute plans with significantly higher accuracy and efficiency than traditional approaches, such as heuristic or cost-based methods, thus improving overall system performance.

B. GPU Energy Efficiency

With the rapid advancements in GPU performance, various strategies have been developed to optimize GPU power consumption [11]. One prominent technique, DVFS, balances power consumption and performance by dynamically adjusting voltage and frequency to align computational resources with workload demands [12]. For instance, GPU-based DVFS techniques typically focus on tuning the GPU's CCF and MCF during tasks such as deep neural network (DNN) training [13]. Increasing CCF enhances the speed of instruction execution, while higher MCF improves data transfer rates. However, both adjustments lead to higher power consumption. Conversely, when computational demand is low, reducing CCF and MCF minimizes unnecessary energy usage. In scenarios requiring high performance, scaling up these frequencies maximizes computational throughput.

This flexible adjustment mechanism is particularly beneficial in power-constrained environments and is widely applied in energy-critical systems, such as data centers and mobile platforms [14], [15]. By dynamically aligning performance with demand, DVFS helps achieve a more efficient use of energy resources.

C. Pareto Optima Algorithms

Pareto optima is a cornerstone of multi-objective optimization, designed to identify the optimal tradeoffs between conflicting objectives. Unlike single-solution optimization, it generates a Pareto front—a collection of solutions where improving one objective is only possible by compromising another. These Pareto optima solutions represent balanced tradeoffs, ensuring no objective can be enhanced without adversely affecting others.

Recently, Pareto optima has been widely adopted in research to determine optimal solutions across multiple performance metrics [16], [17]. In the context of SPS, latency and power consumption are two critical yet conflicting metrics. Achieving lower latency often requires operating GPUs at higher clock speeds, which leads to increased power consumption. Conversely, reducing GPU clock speeds to conserve power typically results in higher latency. By applying Pareto optima to this tradeoff, it becomes possible to identify frequency configurations that strike an effective balance between these competing objectives.

To effectively identify the Pareto front, various algorithms such as NSGA-II [18], MOEA/D [19], MOPSO [20], and PEA [21] have been developed, leveraging evolutionary algorithms and particle swarm optimization (PSO) techniques. Traditional approaches, like convex hull methods, become computationally infeasible as the number of objectives increases [22]. Moreover, complex tradeoffs—where improvements in one objective require significant sacrifices in another—can lead to non-convex Pareto front. In such cases, convex hull

methods fail to fully explore the Pareto front, limiting their applicability [23].

D. Related Work

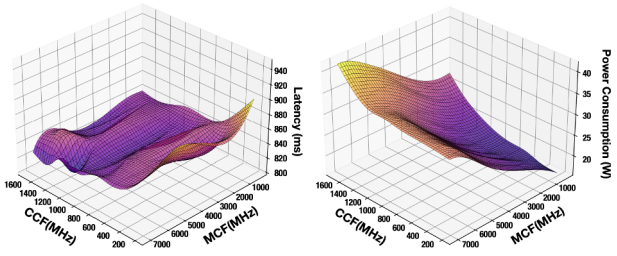
Latency-aware Approach. Fast data processing in SPS is critical for delivering high-quality service to users, making the reduction of end-to-end latency a primary focus of prior research [24]. To achieve this, many approaches have integrated GPUs with SPS to enhance overall latency performance [8], [25], [26], [6], [10].

For example, the state-of-the-art Online ML-based SPS, DYNO [6], leverages a gradient boosting tree (GBT) to learn, in real time, detailed information about queries, traffic, and devices without requiring prior knowledge. Specifically, DYNO dynamically generates optimal device mapping plans. Unlike earlier methods that relied on approximate, cost-based comparisons, DYNO employs an ML model to accurately predict execution times at the operator level. This enables precise end-to-end latency predictions, setting a new standard for latency optimization in SPS environments.

Energy-aware Approach. In recent years, efforts toward sustainable computing have increased, drawing significant attention to energy-efficient design [24]. In SPS, which must continuously process data in real-time, sustained energy consumption accumulates over time, significantly impacting overall power usage. As hardware-level improvements in power efficiency approach their limits, enhancing energy efficiency at the software level has become increasingly critical [27]. This challenge is equally relevant in the IoT domain, where energy consumption directly affects system liveness [28].

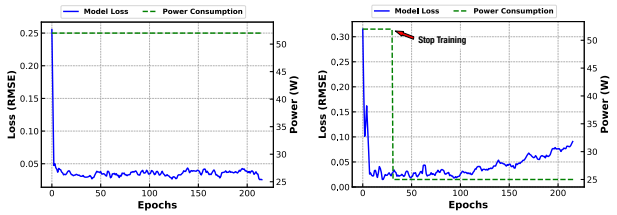
To tackle this challenge, numerous studies have introduced adaptation strategies that prioritize energy consumption as a core metric [29], [30], [7], [9]. For instance, STROME [7] achieved significant energy savings in SPS environments by utilizing application performance data to dynamically adjust power caps while maintaining peak throughput. This method effectively surpasses the traditional constraints of DVFS, representing a major leap forward in energy-efficient computing.

Limitation of Previous Studies. Existing studies on latency-aware and energy-aware optimization encounter inherent limitations. Latency-focused approaches in SPS, which increasingly utilize GPUs for their high computational power, often lead to substantially higher energy consumption than CPU-based systems. This is attributed to the inherently power-intensive nature of GPUs. Therefore, effective latency optimization requires the incorporation of energy efficiency strategies tailored to the unique characteristics of GPUs. For example, while DYNO achieved significant latency improvements through Online ML-based techniques that accurately predict the latency of candidate plans and generate optimal ones, it did not account for the increased energy consumption associated with GPU usage during training or the energy efficiency of the generated plans. This oversight underscores the need for approaches that simultaneously optimize latency and energy efficiency in GPU-accelerated SPS environments.



(a) End-to-End Latency (b) Power Consumption

Fig. 2: Latency (ms) and power consumption (W) based on CCF and MCF. We used a 3D plot to intuitively show correlations between CCF, MCF and the target metric.



(a) Continuous Training (b) Stop Training (epoch = 30)

Fig. 3: Model loss (RMSE) and power consumption (W) by epochs. We compared the patterns of loss and power consumption between continuous training and stopped training.

Energy-aware approaches have predominantly focused on enhancing energy efficiency in conventional CPU-based SPS environments. However, with the growing reliance of SPS systems on GPUs, there is a pressing need for energy-efficient designs specifically optimized for GPU-based environments. For instance, while STROME achieved significant energy savings with minimal impact on throughput, it was designed for CPU-based SPS and did not address the unique challenges associated with GPU-accelerated systems.

Moreover, in real-world SPS deployments, managing both latency and energy constraints simultaneously is often required. Therefore, there is a clear need for solutions that consider both factors and dynamically adapt their strategies based on the current context, rather than prioritizing only one aspect.

E. Motivation

To demonstrate the motivation for our work, we conducted two experiments using a general SPS with GPUs and a GBT model implemented in a state-of-the-art framework [6]. For the experiments, we used Spark [31] as the SPS and the Linear Road Benchmark [32] as the input data. The machine configurations used in these experiments are identical to those described in Section IV.

Excessive Energy Usage. To explore the relationship between end-to-end latency and power consumption in Online ML-based SPS, we conducted a series of experiments using an NVIDIA RTX-3070 GPU. As shown in Fig. 2, these experiments evaluated all possible combinations of two clock speeds:

CCF and MCF. Since NVIDIA limits the modifiable values of CCF and MCF, we used these constraints to define the combinations. Fig. 2 presents the experimental results in a 3D graph. As shown in Fig. 2a, the general trend indicates that higher CCF and MCF values result in lower latency, and vice versa. However, the relationship between latency and CCF/MCF is non-monotonic, exhibiting fluctuations instead of a consistent pattern. In contrast, Fig. 2b shows that power consumption trends are monotonic, with higher CCF and MCF values leading to increased power usage, occasionally exceeding a twofold difference.

In general, latency and energy are typically considered to have a trade-off relationship. However, our experiments reveal that this trade-off is not monotonic, indicating that it is possible to achieve the same latency with lower power consumption, or vice versa. Additionally, MCF, which has been less explored in traditional GPU-based DVFS studies, significantly influences power consumption. Existing approaches that focus solely on CCF are insufficient for fully optimizing energy efficiency.

Therefore, as the relationship between latency and power consumption is non-monotonic, it is required to define objective functions that accurately model latency and power consumption as functions of CCF and MCF. Moreover, the creation and inference of these objective functions must be carefully designed to avoid negatively impacting the performance of the existing system. As low latency is critical in SPS, it is essential to quickly and accurately determine the optimal clock frequency solution during runtime. Hence, our goal is to efficiently and precisely identify the optimal solution among hundreds of possible CCF and MCF combinations using specialized algorithms designed specifically for the SPS environment, rather than conventional batch processing systems.

Gradually Decreasing Accuracy. In SPS, the need to handle dynamic and unpredictable traffic in real-time requires continuous adaptation to maintain high accuracy in execution time prediction models. Numerous previous studies have utilized runtime adaptation techniques to address this requirement [33], [6]. However, the constant training and adaptation to real-time conditions during runtime result in significant energy consumption. Fig. 3 illustrates the variations in loss and power consumption when training is stopped at different epochs. As shown in Fig. 3a, the continuous training approach effectively minimizes loss but results in high power consumption. In contrast, Fig. 3b demonstrates the impact of halting training at a specific epoch: power consumption decreases significantly once training is stopped, but the loss gradually increases over time. Thus, a new energy-efficient algorithm is required to dynamically adjust the training process, improving energy efficiency while minimizing the impact on model accuracy.

III. Design and Implementation

In this section, we begin by providing a high-level overview of our EQUILIBRIA system. We then describe the detailed explanations of how the Pareto optima approach for SPS is applied to GPU DVFS to optimize both latency and energy efficiency, along with the methodology for managing the training of the model. Finally, we highlights the integration of these techniques within the SPS environment to achieve low latency and high energy efficiency.

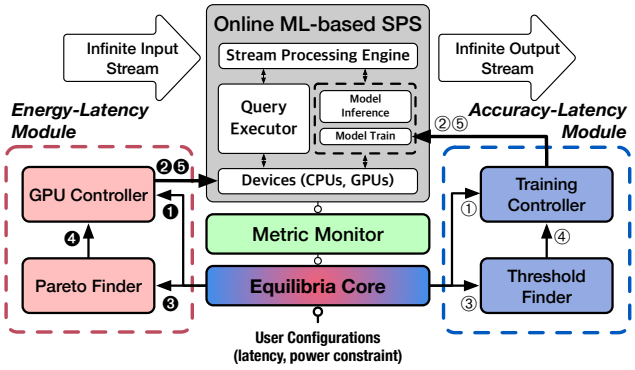


Fig. 4: An overview of EQUILIBRIA. EQUILIBRIA is composed of two modules and a core. A line shown as a white circle rather than an arrow indicates that information is passed in that direction whenever an event occurs, regardless of the flow. The order of each module's actions is indicated by the number of circles.

A. Overview of EQUILIBRIA

Fig. 4 illustrates the overall workflow of EQUILIBRIA, highlighting *Equilibria Core* and two key modules. The *Equilibria Core* receives latency and power constraints from the user, controlling the operation of the two modules to align with the specified goals. Simultaneously, it processes metrics related to latency, model accuracy, and power consumption collected by the metric monitor and provides the processed data to the respective modules. The *Energy-Latency Module* and *Accuracy-Latency Module* modules work collaboratively to identify the optimal configuration under the given constraints, adjusting the GPU frequency and model training time accordingly. During the initialization phase, these modules operate sequentially; afterward, they function concurrently when user constraints are modified.

The *Energy-Latency Module* applies a Pareto optima technique to determine the optimal combination of GPU CCF and MCF that balances latency and power consumption under the given constraints. For example, ❶ When the *Equilibria Core* receives notification of the model's initial training completion, it sends all permissible combinations of CCF and MCF to the GPU Frequency Controller in a sequential manner over time. ❷ The GPU Frequency Controller adjusts the GPU's frequency based on the provided CCF and MCF combinations. ❸ The *Equilibria Core* collects the average latency and power consumption metrics for each combination and forwards them along with the constraints to the Pareto Finder. The Pareto Finder applies the Pareto optima technique to identify the Pareto front and determines the optimal solution for CCF and MCF that satisfies the constraints. ❹ The determined solution is sent back to the GPU Frequency Controller. ❺ The GPU

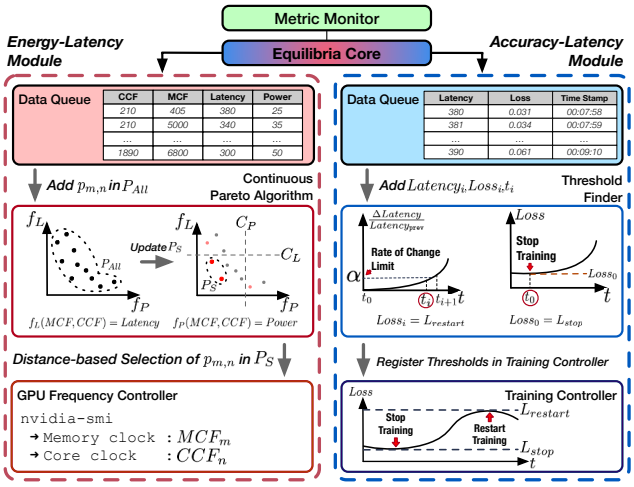


Fig. 5: Overall workflow of *Energy-Latency Module* (red dashed line) and *Accuracy-Latency Module* (blue dashed line) in *Equilibria Core*. This figure shows how steps 3 through 5, as described in Fig. 4, operate within each module.

Frequency Controller adjusts the GPU clock frequencies to match the optimal solution.

The *Accuracy-Latency Module* uses a two-tier threshold technique to dynamically adjust model training while maintaining accuracy at a level that does not impact latency. For example, ① Once *Energy-Latency Module* completes the optimal adjustment of GPU CCF and MCF, the *Equilibria Core* notifies the Training Controller. ② The Training Controller pauses the training of the model. ③ The *Equilibria Core* collects average latency and model accuracy metrics over time and forwards them, along with the latency constraints, to the Threshold Finder. Based on the changes in average latency relative to accuracy, the Threshold Finder sets the two-tier thresholds. ④ These thresholds are sent to the Training Controller. ⑤ The Training Controller adjusts the model training process according to the determined thresholds.

These two modules dynamically adapt during runtime to accommodate changes in user-defined constraints. If only the power constraint is modified or added, the *Energy-Latency Module* re-executes. If the latency constraint is modified or added, the *Accuracy-Latency Module* additionally re-executes. This modular and adaptive design ensures that EQUILIBRIA responds effectively to evolving runtime requirements while maintaining optimal system performance.

B. Pareto Optima Technique for GPU DVFS

The left-hand side of Fig. 5 illustrates the process within *Energy-Latency Module* for determining the optimal clock frequencies (e.g., CCF and MCF) based on metrics received from the *Equilibria Core*. First, the metrics are processed to create objective functions representing latency and power as functions of CCF and MCF. Instead of using learning- or regression-based methods, these objective functions are implemented using a table-based approach. Table-based method is effective for two key reasons. One is that the number of

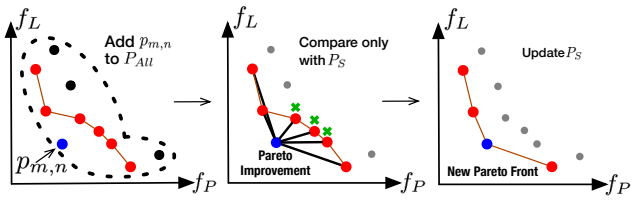


Fig. 6: Pareto optima algorithm of EQUILIBRIA. The red dots are Pareto front. The blue dot is new data point ($p_{m,n}$). The red lines represents P_S. After Pareto improvement, we marked the deleted points in P_S with green **X** marker.

combinations of MCF and CCF on a GPU for DVFS tuning is typically limited to a small number of states. For small size of set, it is relatively easy to directly measure or profile the latency on a real system for each state, so the average latency at each frequency condition can be pre-measured and stored in a table. This avoids the additional complexity and computational resources required to build and train a separate latency objective function(f_L) such as a regression or DNNs. The other is the relationship between clock frequencies and power consumption is monotonic. One notable characteristic of GPU DVFS is the restricted range of configurable MCF and CCF values. Given the monotonic relationship between clock frequencies and power consumption, a table-based method provides a fast and accurate alternative to developing a complex regression-based power objective function(f_P).

$$\begin{aligned} \text{Minimize} \quad & (f_L(p), f_P(p)) \\ \textit{s.t.} \quad & p \in P_{All},\ f_L(p) \le C_L,\ f_P(p) \le C_P \end{aligned} \tag{1}$$

Eq. 1 formulates the Pareto optima problem that *Energy-Latency Module* must solve, highlighting the need to minimize both the latency and power consumption objective functions while satisfying the given constraints. To address this multi-objective optimization challenge in the SPS environment, EQUILIBRIA applies a Pareto optima technique to identify the constrained Pareto front(P_S) of the optimal frequencies.

To find the solution of Eq. 1, the system first identifies the Pareto front from P_{All} that satisfies the Pareto optima condition. Specifically, for a point p^* to belong to the Pareto front, it must satisfy the condition defined in Eq. 2.

$$\begin{aligned} \forall p \in P_{All}, \quad & R(p) \Rightarrow S(p) \\ \textit{s.t.} \quad & R : f_L(p) \le f_L(p^*) \wedge f_P(p) \le f_P(p^*) \\ & S : f_L(p) = f_L(p^*) \wedge f_P(p) = f_P(p^*) \end{aligned} \tag{2}$$

Previously, evolutionary algorithms or PSO methods were commonly used to identify points satisfying Eq. 2 efficiently. These approaches are suitable because the objective space is typically high-dimensional, and the objective functions are complex, making brute force methods inefficient. However, these conventional approaches often require several seconds to execute, rendering them unsuitable for SPS environments, where processing occurs within hundreds of milliseconds.

To achieve low latency, EQUILIBRIA applies a Pareto optima algorithm optimized for the SPS environment. Algorithm 1 and Fig. 6 show how the Pareto optima algorithm for SPS

TABLE I: Descriptions of symbols in figures & equations.

Symbol	Description
f_P	Power Objective Function
f_L	Latency Objective Function
C_P	User-defined Power Constraint
C_L	User-defined Latency Constraint
P_S	Set of Constrained Pareto Front
P_{All}	Set of every Power-Latency pairs
MCF_m	m^{th} possible Memory Clock Frequency
CCF_n	n^{th} possible Core Clock Frequency
$p_{m,n}$	Elements in P_{All}, m and n denotes index of each frequency. If m and n are omitted, then it represents a general element.
p^*	The elements in Pareto Front
$\hat{p^*}$	The element selected in Pareto Front
N	Number of points to be searched
l	Window size of Moving Average
L_{stop}	Threshold of stop training
$L_{restart}$	Threshold of restart training
α	Supremum of Latency Increase Rate
η_{obj}^T	Energy Efficiency of objective method, T means Type of power(e.g. Dynamic, Total).
$N_{X,obj}$	Normalize value of X_{obj}. X means measured metric (e.g. Latency, Power Consumption).

Algorithm 1 Pareto Optima Algorithm for SPS

Input : A Continuously added data point, $p_{m,n}$
Output : Pareto Front, P_S
global $finishFlag$

procedure FINDPARETOFRONT
  initialize $P_{All} \leftarrow \{\}$
  while not $finishFlag$ **do**
    get $p_{m,n}$ from *Equilibria Core*
    add $p_{m,n}$ to P_{All}
    for every p^* in P_S **do**
      if $p_{m,n}$ dominates p^* **then**
        delete p^* from P_S
        if $p_{m,n}$ not in P_S **then**
          add $p_{m,n}$ to P_S
        end if
      end if
    end for
  end while
  return P_S
end procedure

in EQUILIBRIA works. *Equilibria Core* monitors the required data from each device and system, and passes it to *Accuracy-Latency Module* for preprocessing as $p_{m,n}$ whenever data is collected. The new P_S is updated by a Pareto improvement process that only compares preprocessed $p_{m,n}$ to the existing P_S. This enables fast exploration by comparing it to P_S, which is typically much smaller than P_{All}, the set of all points, and can actually reduce the time spent on exploration by overlapping the exploration time by operating simultaneously with the processing of data in SPS. As a result, the process of finding Pareto front is quickly and effectively integrated into SPS.

After the Pareto front (P_S) are found, *Accuracy-Latency Module* selects a point in P_S that satisfies the given conditions. There are three cases in total. In each case, we'll explain how we choose the final point ($\hat{p^*}$).

Case 1 : No Constraints on C_L and C_P. If no constraints are given from the user, *Accuracy-Latency Module* will choose $\hat{p^*}$ by distance-based selection between the Ideal Point and each p^*.

$$\text{Ideal Point}(p_I) = (f_P^{\min}, f_L^{\min}) \tag{3}$$

Eq. 3 shows the definition of Ideal Point, where $f_P^{\min}$ and $f_L^{\min}$ are the minimum value of f_P and f_L in P_{All}, respectively. Distance-based selection selects the closest point in P_S based on its distance from p_I. Since f_L and f_P have different ranges of values, they are subjected to min-max normalization in Eq. 4.

$$\tilde{f_P}(p) = \frac{f_P(p) - f_P^{\min}}{f_P^{\max} - f_P^{\min}}, \quad \tilde{f_L}(p) = \frac{f_L(p) - f_L^{\min}}{f_L^{\max} - f_L^{\min}} \tag{4}$$

Eq. 5 is an expression that defines $D(p)$, the distance between p_I and p based on the normalized values. Since the value of p_I is the same as Eq. 3, the values of $\tilde{f_P}(p_I)$ and $\tilde{f_L}(p_I)$ are zero, which finally simplifies to Eq. 5.

$$\begin{aligned} D(p) &= \sqrt{(\tilde{f_P}(p) - \tilde{f_P}(p_I))^2 + (\tilde{f_L}(p) - \tilde{f_L}(p_I))^2} \\ &= \sqrt{\tilde{f_P}(p)^2 + \tilde{f_L}(p)^2} \quad (\because \tilde{f_P}(p_I), \tilde{f_L}(p_I) = 0) \end{aligned} \tag{5}$$

$$\hat{p^*} = \underset{p^* \in P_S}{\arg\min} D(p^*) \tag{6}$$

The *Energy-Latency Module* selects p^* as the final solution ($\hat{p^*}$) that has a minimum over the elements of P_S with respect to $D(p)$ as shown in Eq. 6 and pass it to the *GPU Frequency Controller* to regulate the clock frequency of the GPU.

Case 2 : Either C_L or C_P Exists. Given only one constraint between C_L and C_P, *Energy-Latency Module* finds the p^* that minimizes the metric of the other while satisfying the given constraint according to Eq. 7 and set it to $\hat{p^*}$.

$$\min_{p^* \in P_S} \begin{cases} f_P(p^*) & \text{if } C_L \text{ is given and } f_L(p^*) \le C_L, \\ f_L(p^*) & \text{if } C_P \text{ is given and } f_P(p^*) \le C_P. \end{cases} \tag{7}$$

Case 3 : Both C_L and C_P Exists. Given both C_L and C_P, *Energy-Latency Module* constructs a new P_S' with p^* points in P_S that satisfy both constraints according to Eq. 8. The process is then the same as in Case 1, computing Eq. 6 to find $\hat{p^*}$.

$$P_S' = \{p^* \in P_S \mid C_L(p^*) \le C_L^{\max} \text{ and } C_P(p^*) \le C_P^{\max}\} \tag{8}$$

C. Two-tier Threshold Training Management

The right-hand side of Fig. 5 illustrates how *Accuracy-Latency Module* performs two-tier threshold training management, corresponding to steps ③ through ⑤ in Fig. 4. First, after the model's training is stopped by the *Equilibria Core*, the *Equilibria Core* collects data on loss and latency over time. Based on the constructed table, *Accuracy-Latency Module* dynamically generates the two-tier thresholds, L_{stop} and $L_{restart}$, at runtime without prior knowledge.

$$(\text{latency}_t^{\text{MA}}, \text{loss}_t^{\text{MA}}) = \frac{1}{N} \sum_{k=t-l+1}^{t} (\text{latency}_k, \text{loss}_k) \quad (9)$$

Loss of Stop (L_{stop}). L_{stop} threshold is determined by monitoring the loss after training is paused. Since the loss at which the model converges varies based on factors such as the query type [6], runtime determination is necessary for accurate threshold setting. When the model's loss stabilizes (i.e., no further reduction), it is appropriate to pause training at this point. However, the model's loss typically oscillates slightly around the convergence value. To address this, *Accuracy-Latency Module* applies a moving average method, as shown Eq. 9, to minimize the errors caused by these oscillations.

Loss of Restart ($L_{restart}$). $L_{restart}$ threshold is selected by analyzing how the model's accuracy affects overall latency. The threshold is based on the loss value at which the latency impact exceeds a defined limit.

$$L_{restart} = loss_t^{\text{MA}} \text{ if } r_t > \alpha,\ r_t = \frac{\text{latency}_t^{\text{MA}} - \text{latency}_{t-1}^{\text{MA}}}{\text{latency}_{t-1}^{\text{MA}}} \quad (10)$$

Eq. 10 indicates how $L_{restart}$ is determined. r_t represents the rate of change of the moving average of latency over time, and α is the user-set threshold for the rate of increase. The *Accuracy-Latency Module* continuously monitors real-time latency changes and adjusts the $L_{restart}$ value based on a comparison with α. This approach is effective because the loss of a model that ceases training during the threshold setting process tends to increase gradually, as shown in Fig. 3.

As a result, the model gradually loses the ability to accurately predict the execution time of the query, which leads to the generation of non-optimal plans, which in turn leads to an increase in latency. *Accuracy-Latency Module* recognizes the tendency by monitoring the growth rate of latency. When the growth rate of latency exceeds a set limit (α), it determines that the model's loss ($loss_t^{\text{MA}}$) at that point has become inaccurate enough to exceed the range of latency growth rates set by the user, and sets the threshold to restart training, $L_{restart}$.

Through this process, as depicted in the Training Controller section of Fig. 5, the model's loss fluctuates between L_{stop} and $L_{restart}$. Training is paused between L_{stop} and $L_{restart}$, significantly reducing the GPU's power consumption for training while maintaining efficiency.

IV. Evaluation

A. Evaluation Setup

Configurations. All of our experiments were conducted on a single machine equipped with one Intel i7-13700F 16-core 2.1GHz CPU with 32GB memory and two NVIDIA RTX 3070 GPUs, each with 8GB of GPU device memory and 220W as thermal design power (TDP). We used one master and two workers, each running a single executor. Apache Kafka [34] was configured as the message broker, and Spark version 3.2.3 was used as the stream processing engine. The experiments aimed to validate the superiority of EQUILIBRIA in scenarios with minimal nodes, where maximizing energy efficiency, as often done in other comparative studies, was the primary focus. Default values were used for the Spark settings.

TABLE II: Query details of real-world streaming workloads.

Query	Description
Q_1	SELECT L.timestamp, L.vehicle, L.speed, L.highway, L.lane, L.direction, L.segment FROM SegSpeedStr (range 30 slide 1) as A, SegSpeedStr as L WHERE (A.vehicle == L.vehicle)
Q_2	SELECT timestamp, highway, direction, segment, AVG(speed) as avgSpeed FROM SegSpeedStr (range 300 slide 1) GROUPBY (highway, direction, segment) HAVING (avgSpeed $\leq$ 40.0)
Q_3	SELECT timestamp, highway, direction, segment, COUNT(vehicle) as numVehicle FROM SegSpeedStr (range 30 slide 1) GROUPBY (highway, direction, segment)

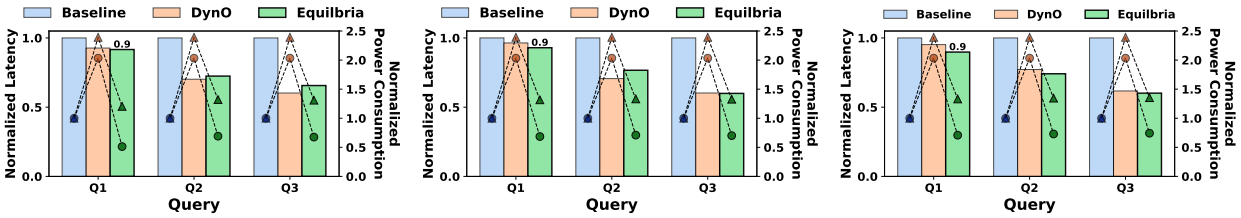
Comparisons. We evaluated the latency and energy efficiency of RAPIDS (baseline) [35], DYNO, and EQUILIBRIA in various scenarios. The baseline represents a GPU library running on Spark, utilizing a single GPU to process input data. DYNO, a state-of-the-art Online ML-based SPS, minimizes latency in SPS leveraging GPUs by generating optimal plans using Online ML and operates with two GPUs. We compared the performance of EQUILIBRIA against these methods in terms of latency and power consumption. To rigorously validate EQUILIBRIA's superiority, we ensured identical experimental setups, using no additional devices beyond the configurations employed by DYNO.

Workloads and Stream Traffic Types. For these experiments, we utilized the linear road benchmark (LRB) [32], a real-world streaming workload commonly used in recent studies to evaluate SPS performance [6], [8]. LRB is suitable for performance evaluation in realistic environments and includes a variety of query operations such as join, projection, aggregate, group-by, and selection. To compare performance with DYNO, we used the same set of queries employed in their study. Each query corresponds to Q_1, Q_2, and Q_3 as labeled in Table II.

In real-world scenarios, input traffic is typically fluctuating rather than constant. To demonstrate that EQUILIBRIA operates effectively under realistic traffic conditions, we tested each query under various traffic scenarios within the manageable range of the system's default configuration and experimental settings. Details of the traffic configurations are presented in Table III. We also conducted experiments as shown in Sec. IV-B and Sec. IV-C, focusing on the conditions described for Case 1 and Case 2 in Sec. III-B. Although Case 3 requires updating the existing Pareto front to a new one based on user constraints, the remaining process mirrors that of Case 1. To highlight the superior performance of EQUILIBRIA, we chose Case 1 and Case 2 as representative scenarios.

B. Overall Performance Analysis

Fig. 7 presents normalized graphs of latency and power consumption for DYNO and EQUILIBRIA, using RAPIDS with a single GPU as the reference. The graph combines the power consumption of all GPUs used in the system. We categorized power consumption into two distinct types: total and dynamic.



(a) Low Traffic (b) Mid Traffic (c) High Traffic

Fig. 7: The graph shows latency and power consumption, normalized relative to the baseline values for each query. Latency is represented by the bar graph, while power consumption is illustrated using a dotted line graph. Triangle markers indicate total power consumption, and circle markers represent dynamic power consumption.

TABLE III: Query types and traffics. Each second, a random number of data tuples is generated, following a normal distribution with the average value serving as the mean. Each tuple has a fixed data size of approximately 0.1 KB.

Query	Q_1	Q_2, Q_3
Traffics (Low, Mid, High)	10, 50, 100	100, 500, 1000

TABLE IV: Comparison of energy efficiency. Energy efficiency is determined by multiplying the normalized latency and power consumption, relative to the baseline, with the energy efficiency metric defined in Eq. 11. A lower value signifies greater energy efficiency.

Cases		**Energy Efficiency** ($\eta_{obj}^{\text{Total}}$ / $\eta_{obj}^{\text{Dynamic}}$)		
Traffic	**Query**	**Baseline**	**DynO**	**Equilibria**
Low	Q_1	**1.0** / 1.0	2.16 / 1.84	1.12 / **0.45**
	Q_2	1.0 / 1.0	1.68 / 1.43	**0.97** / **0.52**
	Q_3	1.0 / 1.0	1.44 / 1.23	**0.84** / **0.42**
Mid	Q_1	**1.0** / 1.0	2.28 / 1.94	1.17 / **0.63**
	Q_2	1.0 / 1.0	1.75 / 1.49	**0.99** / **0.54**
	Q_3	1.0 / 1.0	1.44 / 1.23	**0.78** / **0.42**
High	Q_1	**1.0** / 1.0	2.19 / 1.93	1.17 / **0.68**
	Q_2	1.0 / 1.0	1.73 / 1.54	**0.91** / **0.56**
	Q_3	1.0 / 1.0	1.38 / 1.23	**0.78** / **0.45**

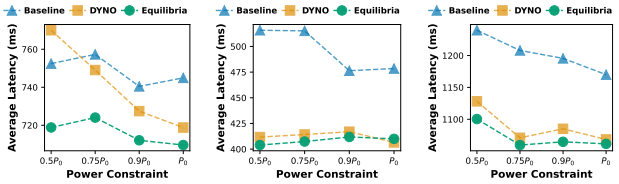
Total power consumption consists of both the static and dynamic components. Here, static power consumption refers to the energy used when the device is idle, performing no work. In contrast, dynamic power consumption represents the additional energy drawn when the device is actively carrying out tasks. Across all queries, the results clearly demonstrate EQUILIBRIA's superior performance in balancing latency and energy efficiency compared to both the baseline and DYNO.

By selecting appropriate Pareto front to adjust the GPU's MCF and CCF, EQUILIBRIA achieves latency that is either lower than or comparable to DYNO's while reducing power consumption by up to 70%. Even compared to the Baseline, which uses only one GPU, EQUILIBRIA improves latency by up to 50% while reducing 30% dynamic power consumption. While EQUILIBRIA's latency improvements vary depending on the query characteristics, its power consumption consistently decreases significantly regardless of query type.

$$\eta_{obj}^{T} = \frac{E_{obj}^{T}}{E_{Base}^{T}} = \frac{L_{obj} * P_{obj}^{T}}{L_{Base} * P_{Base}^{T}} = N_{L,obj} * N_{P,obj}^{T} \quad (11)$$

Table IV shows the calculated energy efficiency for each method. For all queries, EQUILIBRIA outperforms the other two methods, improving energy efficiency by up to 58% compared to Baseline, and 76% compared to DYNO. Although DYNO achieves lower latency compared to the baseline, its high power consumption—resulting from the use of two GPUs—leads to lower energy efficiency. In contrast, EQUILIBRIA, while also utilizing two GPUs, dynamically adjusts clock frequencies and training operations, significantly reducing power consumption and improving energy efficiency.

C. Performance under Various Constraints

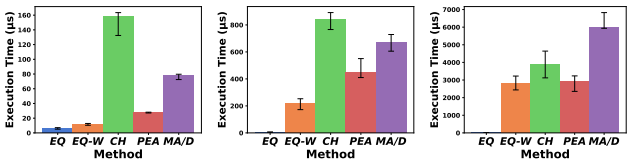


(a) Q_1 (b) Q_2 (c) Q_3

Fig. 8: The graph illustrates the variation in average latency for each query when different power constraints are applied to the GPU in EQUILIBRIA. Here, P_0 denotes the baseline power consumption for each query without any constraints.

Fig. 8 illustrates the average latency of different methods under varying power constraints. Regardless of the method or query type, the overall latency tends to increase as the power constraint decreases. Notably, EQUILIBRIA maintains lower latency compared to other methods, regardless of query type. Furthermore, EQUILIBRIA consistently sustains low latency even under stringent power constraints.

In Fig. 8a, the latency of DYNO increases sharply as the power constraint tightens. This is attributed to the characteristics of the queries, particularly Q_1, which has higher GPU core utilization compared to other queries. As the maximum allowable power consumption decreases, the GPU usage for the ML model has a relatively greater impact on overall query processing, leading to a more pronounced increase in latency.



(a) $N = 10$ (b) $N = 100$ (c) $N = 1000$

Fig. 9: The graph compares the execution time of the Pareto front algorithm applied in EQUILIBRIA. EQ represents the method implemented in EQUILIBRIA, while EQ-W shows the execution time under the worst-case scenario for EQUILIBRIA's algorithm. CH and MA/D denote the convex hull and MOEA/D methods, respectively.

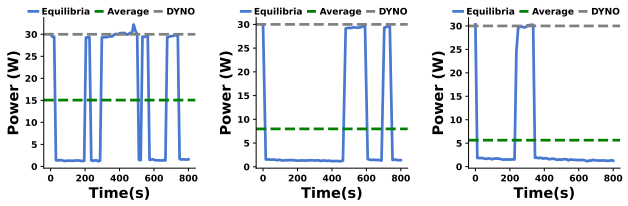
Fig. 8b and Fig. 8c demonstrate that EQUILIBRIA achieves similar or lower latency compared to DYNO. However, considering that EQUILIBRIA reduces the dynamic power consumption of training GPUs by approximately 83% compared to DYNO, it is evident that EQUILIBRIA operates very efficiently across various query types and power constraint scenarios. Additionally, while EQUILIBRIA exhibits dynamic power consumption similar to the Baseline, it achieves significantly superior latency performance compared to the Baseline.

D. Comparison with Other Pareto Optima Algorithms

We compared the execution time of EQUILIBRIA's Pareto front algorithm with other methods under GPU DVFS constraints, where the number of feasible CCF and MCF combinations is limited. For practical performance comparison, we measured the execution time to find the Pareto front from sets of 10, 100, and 1000 points.

We set up three algorithms for comparison, which are convex hull (CH), PEA, and MA/D (MOEA/D). PSO algorithm was not compared because preliminary experiments showed that it has a very long running time when N is 1000 or less compared to other methods. EQ-W indicates the running time in the worst case of the algorithm to find the Pareto front of EQUILIBRIA. By worst case, we mean the case where all the given N points satisfy the Pareto front. Fig. 9 shows that the running time of EQ, the Pareto algorithm of EQUILIBRIA, is very fast, on the order of μs. This does not change as the number of points increases and is relatively very fast as the size of N increases. This shows that the method of EQUILIBRIA, which is to incrementally find the Pareto front during the monitoring process, overlaps with each other and works very well in the SPS environment.

We added and compared EQ-W to show that this approach works better than other traditional Pareto front algorithms even in the worst case. Since our method incrementally updates the Pareto front and only compares newly added points to the Pareto front rather than all points, the worst case is when all points are Pareto front, which requires the most comparisons to be performed. However, as shown in Fig. 9c, it still shows fast performance when N is 1000 or less. This is significant considering that there are less than 1000 possible combinations of CCF and MCF on GPUs. In other words, we show very fast performance independent of the size of N in the general case, and significantly faster performance compared to other algorithms even in the worst case.



(a) $\alpha = 0.001$ (b) $\alpha = 0.003$ (c) $\alpha = 0.01$

Fig. 10: The graph illustrates the dynamic power consumption of the training GPU in EQUILIBRIA as a function of α. The green dashed line represents the average power consumption in EQUILIBRIA, while the gray dashed line indicates the average power consumption in DYNO.

E. Training GPU Power Consumption

Fig. 10 shows the dynamic power consumption of GPUs based on α, a parameter used to determine $L_{restart}$, one of the two-tier thresholds. α reflects the tolerance for latency increases caused by inaccurate query execution plans due to rising model loss. For example, an α of 0.01 indicates tolerance for a 1% latency increase. A higher α increases $L_{restart}$, widening the interval between L_{stop} and $L_{restart}$. This interval represents the period during which the GPU remains idle and the training resumes to reduce the increased loss.

Experimental results show that as α increases, average power consumption decreases, and the frequency and duration of training intervals reduce. This is because the wider interval between L_{stop} and $L_{restart}$ enhances the effect of keeping the GPU idle. The model's loss decreases faster during training restarts than it increases during pauses, resulting in overall lower power consumption.

As shown in Fig. 10c for $\alpha = 0.01$, the dynamic power consumption of the training GPU in EQUILIBRIA averages only 5W, a remarkably low level. EQUILIBRIA selects $\alpha = 0.01$ because further increasing α does not yield sufficient reductions in power consumption to offset the latency increases. Specifically, while higher α values may slightly reduce average power consumption, as shown in Fig. 10, the degree of reduction diminishes and converges to a certain point, making additional latency increases unjustifiable.

V. CONCLUSION

In this paper, we propose EQUILIBRIA, a novel framework designed to simultaneously optimize energy efficiency and latency performance in Online ML-based SPS environments. We develop EQUILIBRIA to suit Online ML-based SPS, accommodating diverse performance improvement goals by utilizing DVFS and Pareto optima algorithm specialized for SPS environments to reduce latency and improve energy efficiency. Furthermore, EQUILIBRIA dynamically manages the ML training process by monitoring model loss in real-time.

Our co-optimization technique is applicable to a wide range of SPSs, including but not limited to GPU-based and Online ML-based SPS. Through extensive evaluations from multiple perspectives, we demonstrate that by properly configuring the GPU's CCF and MCF using DVFS techniques and managing ML model training, the performance and sustainability of Online ML-based SPS can be enhanced, achieving both low latency and low power consumption.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (RS-2024-00453929) (RS-2024-00416666).

REFERENCES

[1] E. Mehmood and T. Anees, "Challenges and solutions for processing real-time big data stream: a systematic literature review," *IEEE Access*, vol. 8, pp. 119123–119143, 2020.

[2] Y. Sasaki, "A survey on iot big data analytic systems: Current and future," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1024–1036, 2021.

[3] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, 2021.

[4] Y. Liu, A. Andhare, and K.-D. Kang, "Corun: Concurrent inference and continuous training at the edge for cost-efficient ai-based mobile image sensing," *Sensors*, vol. 24, no. 16, p. 5262, 2024.

[5] P. Han, S. Wang, Y. Jiao, and J. Huang, "Federated learning while providing model as a service: Joint training and inference optimization," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pp. 631–640, IEEE, 2024.

[6] S. Oh, G. E. Moon, and S. Park, "Ml-based dynamic operator-level query mapping for stream processing systems in heterogeneous computing environments," in *2024 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 226–237, 2024.

[7] C. Eibel, C. Gulden, W. Schröder-Preikschat, and T. Distler, "S trome: Energy-aware data-stream processing," in *Distributed Applications and Interoperable Systems: 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings 18*, pp. 40–57, Springer, 2018.

[8] F. Zhang, L. Yang, S. Zhang, B. He, W. Lu, and X. Du, "Finestream : Fine-grained window-based stream processing on cpu-gpu integrated architectures," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 633–647, 2020.

[9] X. Wei, L. Li, X. Li, X. Wang, S. Gao, and H. Li, "Pec: Proactive elastic collaborative resource scheduling in data stream processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1628–1642, 2019.

[10] G. Jung, Y. Jeong, K. Park, D. Lee, H. Byun, S. Lee, and S. Park, "dstream: An online-based dynamic operator-level query mapping scheme on discrete cpu-gpu architectures," *IEEE Access*, vol. 13, pp. 8239–8256, 2025.

[11] F. Magoulès, A.-K. C. Ahamed, A. Desmaison, J. C. Léchenet, F. Mayer, H. B. Salem, and T. Zhu, "Power consumption analysis of parallel algorithms on gpus," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*, pp. 304–311, IEEE, 2014.

[12] X. Liu and R. Buyya, "Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–41, 2020.

[13] Z. Tang, Y. Wang, Q. Wang, and X. Chu, "The impact of gpu dvfs on the energy and performance of deep learning: An empirical study," in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, pp. 315–325, 2019.

[14] P. J. Kuehn and M. Mashaly, "Dvfs-power management and performance engineering of data center server clusters," in *2019 15th annual conference on wireless on-demand network systems and services (WONS)*, pp. 91–98, IEEE, 2019.

[15] R. Begum, D. Werner, M. Hempstead, G. Prasad, and G. Challen, "Energy-performance trade-offs on energy-constrained devices with multi-component dvfs," in *2015 IEEE International Symposium on Workload Characterization*, pp. 34–43, IEEE, 2015.

[16] H. Zhang, L. Jia, L. Wang, X. Xu, and F. Dou, "Energy-efficient timetable optimization empowered by time-energy pareto solution under actual line conditions," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[17] A. K. Kakolyris, D. Masouros, S. Xydis, and D. Soudris, "Slo-aware gpu dvfs for energy-efficient llm inference serving," *IEEE Computer Architecture Letters*, 2024.

[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[19] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on evolutionary computation*, vol. 11, no. 6, pp. 712–731, 2007.

[20] C. C. Coello and M. S. Lechuga, "Mopso: A proposal for multiple objective particle swarm optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 2, pp. 1051–1056, IEEE, 2002.

[21] D. W. Corne, J. D. Knowles, and M. J. Oates, "The pareto envelope-based selection algorithm for multiobjective optimization," in *International conference on parallel problem solving from nature*, pp. 839–848, Springer, 2000.

[22] H. Ishibuchi, N. Tsukamoto, and Y. Nojima, "Evolutionary many-objective optimization: A short review," in *2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence)*, pp. 2419–2426, IEEE, 2008.

[23] I. Das and J. E. Dennis, "A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems," *Structural optimization*, vol. 14, pp. 63–69, 1997.

[24] V. Cardellini, F. Lo Presti, M. Nardelli, and G. R. Russo, "Runtime adaptation of data stream processing systems: The state of the art," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–36, 2022.

[25] A. Koliousis, M. Weidlich, R. Castro Fernandez, A. L. Wolf, P. Costa, and P. Pietzuch, "Saber: Window-based hybrid stream processing for heterogeneous architectures," in *Proceedings of the 2016 International Conference on Management of Data*, pp. 555–569, 2016.

[26] C. Chen, K. Li, A. Ouyang, Z. Zeng, and K. Li, "Gflink: An in-memory computing architecture on heterogeneous cpu-gpu clusters for big data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1275–1288, 2018.

[27] G. Kp, G. Pierre, and R. Rouvoy, "Studying the energy consumption of stream processing engines in the cloud," in *2023 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 99–106, 2023.

[28] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Low-cost adaptive monitoring techniques for the internet of things," *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 487–501, 2018.

[29] M. Chao, C. Yang, Y. Zeng, and R. Stoleru, "F-mstorm: Feedback-based online distributed mobile stream processing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 273–285, IEEE, 2018.

[30] T. De Matteis and G. Mencagli, "Elastic scaling for distributed latency-sensitive data stream operators," in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 61–68, IEEE, 2017.

[31] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[32] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, "Linear road: a stream data management benchmark," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pp. 480–491, 2004.

[33] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, "Bao: Making learned query optimization practical," in *Proceedings of the 2021 International Conference on Management of Data*, pp. 1275–1288, 2021.

[34] K. M. M. Thein, "Apache kafka: Next generation distributed messaging system," *International Journal of Scientific Engineering and Technology Research*, vol. 3, no. 47, pp. 9478–9483, 2014.

[35] NVIDIA. https://github.com/NVIDIA/spark-rapids, 2021.