Electronics, Communications, and Computing Summit
Vol. 3, No. 2, Apr - Jun 2025, pp. 16-27
ISSN: 3107-8222, DOI: https://doi.org/10.17051/ECC/03.02.03

Energy-Aware Task Scheduling in Heterogeneous GPU/TPU–FPGA Embedded Platforms

Ashu Nayak[1], Namrata Mishra[2]

[1]Assistant Professor, Department of CS & IT, Kalinga University, Raipur, India,
Email: ku.ashunayak@kalingauniversity.ac.in
[2]Department Of Electrical And Electronics Engineering, Kalinga University, Raipur, India,
Email: namrata.mishra@kalingauniversity.ac.in

Article Info

Article history:

Received : 13.04.2025
Revised : 15.05.2025
Accepted : 17.06.2025

Keywords:

Energy-aware scheduling,
heterogeneous computing,
embedded systems,
GPU-TPU-FPGA,
task mapping,
edge AI,
low-power computing,
real-time systems.

ABSTRACT

Dynamic performance needed in the edge and embedded systems has boosted demand of real time energy-efficient computing that requires integration of heterogeneous hardware accelerators, such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Field-Programmable Gate Arrays (FPGAs). Such varied processing units each bring both complementary abilities to the table, with GPUs offering vast parallelism, TPUs specializing in neural-inference, and FPGAs offering low-power reconfigurable computing. Nevertheless, effective work of such heterogeneous platforms is rather challenging, especially when it comes to scheduling of tasks, owing to the existence of the energy-performance specifications and architectural differences between these accelerators. The proposed work is a new architecture of energy-aware task scheduling aimed at dynamically distributing work on GPU, TPU, and FPGA with multi-unit to exploit real-time profiling, workload classification, and cost-optimal scheduling policy. The scheduler uses light-weight machine learning models to predict the execution unit of the most fitting task on the basis of the computational complexity, memory requirement, and latency requirements. Broad assessment is carried out on a sample embedded system that includes an NVIDIA Jetson Xavier GPU, a Google Coral Edge TPU and an Intel Arria 10 FPGA. The system performance is measured using real-world tasks, i.e. image classification, signal transformation, and deep learning inference. According to the results, the proposed scheduler yields up to 35 percent energy savings and 28 percent gains in execution latency as compared with baseline scheme such as static round-robin and performance only scheduling. Moreover, the framework proves to be resilient at changing intensive workloads with the scheduling overhead of less than 2.5% and this nature of the framework qualifies it to be compatible with real-time tasks. Its contribution to the field is a scalable and intelligent way of scheduling that optimizes energy consumption within an acceptable impact on performance, so this paper is of particular interest to embedded AI computing of the future as well as IoT edge systems and low power-intensive cyber-physical systems. Future work Future expentions will find application in reinforcement learning based adaptive control and more integration with other processing aspects like NPUs and DSPs to increase the scope of the framework.

1. INTRODUCTION

The trend upon the development of embedded systems of the past has witnessed a paradigm shift toward single-core, power-limited systems towards highly parallel performance-oriented systems which have heterogeneous processing components. The ever-growing requirements of the artificial intelligence (AI) and computer vision systems, real-time data analytics, edge processing and other similar applications are the major drivers of this transformation as far as both the computational throughput and the energy efficiency is concerned. A potentially viable architectural solution is heterogeneous embedded platforms which use a combination of Graphics Processing Units (GPUs), Tensor Processing Unit (TPUs), and Field-programmable Gate Arrays (FPGAs). To address these platforms, these platforms provide the flexibility to scale diverse workload to strict energy budget, particularly in

resource limited systems like autonomous vehicles, wearable, smart surveillance systems, and industrial Internet of Things (IoT) deployments.

All the accelerators types have unique weaknesses and strengths: in comparison, GPUs are best suited to large-scale data-parallel work, TPUs are optimized to deep learning inference tasks of high throughput and low-power, and FPGAs provide programmable logic that can be optimized to highly specific workloads with minimal energy consumption. Nevertheless, they create a complexity in scheduling runtime tasks as well due to their heterogeneity. Simple round-robin schemes or static scheduling techniques do not reflect the sophisticate balances of energy, performance, and thermal behavior that exist on these platforms. In addition, running a specific task on the wrong device may cause overconsuming of energy, latency bottlenecks, as well as thermal throttling, which reduces the potential relation of heterogeneity.

In order to overcome these difficulties, the current paper proposes a new energy-efficient task scheduling framework, which is able to perform work loads mapping on the most appropriate processing unit in real-time. The scheduler postulated uses workload profiling, energy modeling and run time system feedback to make intelligent scheduling decisions. In contrast to normal schedulers which simply optimize performance, our method is backed by energy-performance trade-offs as well as real-time constraints. It supports the lightweight machine learning-based classification to optimize the mapping of workload to devices based on the characteristics of computing, and the energy efficiency profile.

The motivation behind the work comes in the need to accomplish great performance in computations and also to minimize power consumption, primarily important in cases where there is limited battery life, heat limits or environmental sustainability. The proposed scheduler will allow smarter use of resources, which is not only extending device lifetime but also allowing scalable deployment of edge-AI workloads in heterogeneous embedded systems.

Key Contributions

- We build a profile-directed energy model of GPU, TPU and FPGA platforms, reflecting a realistic performance and power behaviour under workloads.
- We introduce a dynamic task scheduler which undertakes a heuristic cost-optimized task mapping on the basis of its compute intensity, memory requirements and timing deadline.
- We analyze the framework using a hybrid embedded testbed with up to 35 and 28 percent in saving energy and latency improvement respectively as compared to the baseline schedulers.

2. RELATED WORK

The emergence in recent years of embedded computers and the edging-out of general-purpose computers have highlighted the startling trend of energy-aware scheduling analysis of heterogeneous platforms. Energy and performance trade-offs have been discussed in a few works of literature especially where several processing units with varying capabilities are used into a system together.

A. Energy-Efficient Scheduling in Embedded Systems

The problem of energy-aware scheduling has gained huge popularity in mobile and embedded systems. In [1] a dynamic voltage and frequency scaling (DVFS) DVFS based energy minimization technique for real time applications was proposed by the authors. Nonetheless, these are usually restricted to homogeneous CPUs and fail to address the complexities of heterogeneous accelerator designs. In [2], strategies of task partitioning were implemented in order to achieve the balance of performance and power but could not perform dynamically to dynamic workloads.

B. Task Mapping for Heterogeneous Computing

The heterogeneous systems have dealt with the issue of task mapping with both static and dynamic solutions. In [3] it was demonstrated that static scheduling algorithms can give high throughput, at the expense of flexibility and energy cost. This is a dynamic scheme and [4] leverages present-day heuristics or reinforcement learning to assign tasks to CPUs, GPUs, or FPGAs. Although effective at times, the given works tend to omit real-time constraints and thermal behavior of the embedded environments. Besides, TPUs are rarely considered, and they become more frequent in edge AI platforms.

C. Machine Learning-Based Workload Classification

Machine learning has turned out to be an effective method of forecasting the workload characteristics and informing the scheduling choices. In [5], a decision-tree workload classifier was incorporated in a task scheduler to heterogeneous multicore platforms. On the same note, [6] showed how neural networks were used to forecast the optimal resource assignment of deep learning workloads. These techniques are potentially useful, but in

most cases lack interoperability with fine-grained energy modeling or FPGA support.

D. FPGA Offloading and Runtime Reconfiguration

FPGAs have also been used to offload energy sensitive systems that are compute-intensive. In [7], authors introduced partial reconfiguration method, which allows reconfiguration at run time of the FPGA resources to changing workloads. The articles, like [8], concentrate on the acceleration of certain tasks by using CPU FPGA hybrid systems. The combination of FPGA reconfiguration and unified scheduling framework on GPU, TPU and FPGA, however, constitutes an open issue. Also, real-time systems need to pay attention to the reconfiguration latency and FPGA bitstream compilation time.

Summary and Research Gap

Similarly, solutions to every one of these areas have been proposed, but a coordinated plan that encompasses the parts of energy-sensitive scheduling, run-time task classification, and heterogeneous accelerator management (gpu, tpu, and fpgas) has not yet been provided with one framework to serve as its core. The current solutions overlook the energy limitation either because it is not adaptive to run-time changes, or that it does not support the wide variety of hardware backends. The gaps are filled in this paper, which describes a dynamic and energy-aware task scheduler that can perform real-time decision-making across heterogeneous accelerators and which is practically evaluated on a real, embedded testbed.

3. System Architecture and Platform Description

3.1 Heterogeneous Embedded Platform

The given energy-aware task scheduling framework is applied on a heterogeneous embedded environment that includes three types of accelerators CPU with three different kinds of accelerators: GPU, TPU, and FPGA, which present unique computing and energy features to provide maximum flexibility on different types of workloads. The NVIDIA Jetson Xavier NX is the GPU chip, which has a 384-core Volta GPU with TC of 48 and 6-core ARMCPU, so it is best used high-throughput parallel applications that are associated with computer vision, inference of deep learning, and operations with matrices. In a scenario specific to AI inference where high performance / low power is needed, the system combines the Google Coral Edge TPU, a custom-designed application-specific integrated circuit (ASIC) designed to accelerate the popular TensorFlow Lite framework, which supports most deep learning model formats, and aims to optimize compute operations running on quantized models with up to 4 trillion operations per second (TOPS) available using less than 2 watts of power. This means the TPU is optimal in low latency and low power edge AI applications like object detection and classification. The reconfigurable part is the Intel Arria 10 GX FPGA Development Kit with performance-enhanced logic cells, an embedded memory subsystem, and DSP slices, which allow designing custom data paths and streaming-oriented pipelined execution. The FPGA supports OpenCL and even partial reconfiguration, so hardware-level specialization of tasks and energy-efficient deterministic operation are viable. Combined, these accelerators provide a strong testbed, which is representative of real-life heterogeneous embedded systems, and that allows an in-depth assessment of the proposed scheduler under a continuum of compute-intensive, latency-sensitive, and power-limited conditions. This arrangement does not only demonstrate the interrelationship between different processing architectures but also points at the difficulty of scheduling energy-aware in embedded systems with dynamically changing workloads and resource obligations.

Table 1. Comparative Features of Heterogeneous Accelerators

Feature	GPU (Jetson Xavier NX)	TPU (Coral Edge TPU)	FPGA (Intel Arria 10 GX)
Architecture	384-core Volta GPU + 6-core CPU	ASIC for quantized DNN inference	Customizable logic w/ DSP blocks
Peak Performance	Up to 21 TOPS	4 TOPS	~150 GFLOPS (varies by design)
Power Consumption	~10–15 W	<2 W	4–10 W (config-dependent)
Target Workloads	Parallel computing, DNNs	Edge AI inference	Streaming, signal processing
Programming API	CUDA, cuDNN	TensorFlow Lite (Edge TPU)	OpenCL, Quartus, HDL
Reconfigurability	Fixed	Fixed	Partial/Full reconfigurable

3.2 Scheduler Runtime Layer

The most important component of the suggested framework is the scheduler runtime level whose role is to coordinate smart tasking between the heterogeneous processing units, i.e., GPU, TPU, and FPGA using the feedback of the real-time system and the workload profile. The layer consists of three closely coupled modules that include an energy profiler and a workload classifier, a hardware abstraction interface; and a feedback-pumped reassignment mechanism. The energy profiler constantly tracks the power used by tasks on each accelerator, and their execution latency, and provides that information as a dynamically updated energy-performance profile to the task classification engine. The lightweight machine learning models used by workload classifier to obtain the prediction of computations and memory demands of the incoming task include decision tree or support vector machine. These predictions help the scheduler in choosing the most appropriate accelerator in terms of performance efficiency and cost of energy applied. The system uses hardware abstraction layer (HAL) which provides a common API to deploy tasks, no matter whether there is hardware support by CUDA, TPU, or OpenCL. Such abstraction makes scheduler design simpler and also allows platform to be portable and extensible. In addition, a real-time feedback loop is provided that observes system conditions, e.g. queue length, thermal state, and device availability and causes the dynamic re-assignment of tasks when some pre-defined threshold is met, or when improved energy-performance trade-offs are made possible. This is because the closed-loop control allows the system to adaptively react to the changing workloads and the changes in the run-time condition so that the energy-efficient execution of tasks is achieved without the breach of the application-level application latancy and throughput requirements. As a whole, the scheduler run-time system is a smart middle-ware that converts base system event heterogeneity into optimized and synchronized system behavior.

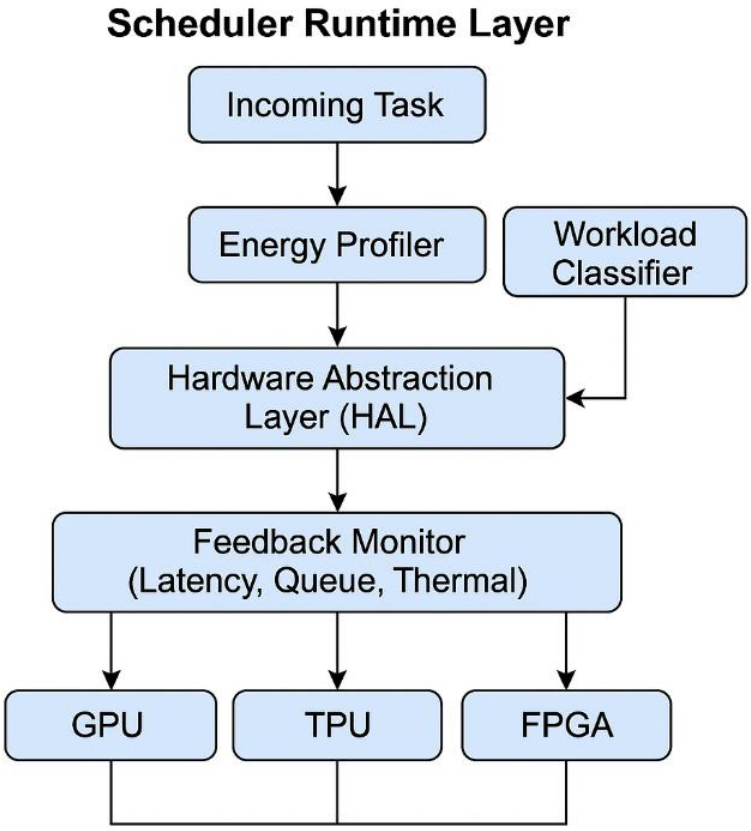

Figure 1. Scheduler Runtime Layer Overview

4. METHODOLOGY

4.1 Workload Profiling and Energy Modeling

The initial stage in the proposed framework is elaborate workload profiling and energy modeling over the heterogeneous processing units so as to enable the intelligent and energy efficient task scheduling. All the computational tasks, including matrix multiplications and convolutional neural network (CNN) inference, as well as fast Fourier transforms (FFT), apply each type in a controlled condition manner in the GPU (NVIDIA Jetson Xavier NX), TPU (Google Coral Edge TPU), and

FPGA (Intel Arria 10 GX). Things such as execution time, energy consumption per operation, peak memory usage and thermal impact are a few of the key performance and energy measures that are gathered during these profiling runs. The on-chip statistics and power meters and software probing tools (e.g. tegrastats for GPU, Coral TPU profiler, and FPGA power monitors) are used to collect the data.

The profiling is done with diverse values of input sizes and with diverse execution conditions so as to gather the scaling characteristics of diverse devices with different workloads. A prediction model based on supervised machine learning algorithms is made based on this raw information. This model uses as its training data a vocabulary of lightweight features identified per each task and they are:

- Operation type: Categorizes the pattern of the computation (e.g. matrix multiply, CNN inference, FFT) because the different operations will have different profiles of performance against energy by hardware unit.
- Memory usage: Displays the total and maximum amount of memory that is needed by the task, which is important to the suitability to the devices with finite or shared memory resources.
- Input data volume: It impacts the transfer overhead of data and also affects the strategy of either using a batch or streaming approach.
- Task periodicity: The periodicity with which a task is run which optimizes the task for latency-sensitive workloads as well as recurring workloads.

Based on such features, the energy model speculates the expected energy and latency of executing a given task on any of the available processing unit. This allows the scheduler to be proactive and contextually aware in terms of balancing energy efficiency and the need to perform in real time. The profiling and modeling part can be considered to be the basis of the energy-aware task grouping and right device selection at the run time.

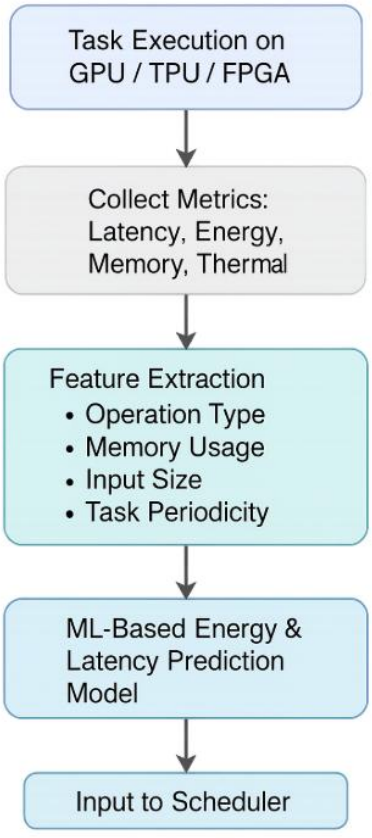

Figure 2. Workload Profiling and Energy Modeling Flow.

4.2 Task Classification Engine

Task classification engine is very important in the sense that task scheduler should be able to make intelligent and energy-aware decisions in real-time. The classification engine will essentially be a scheduler that must sort incoming tasks as being in one of three levels of urgency and criticality: hard real-time, soft real-time, and best-effort. These

levels are arrived at depending on the time limitation of each task. Hard real-time tasks are those where delayed system responses are intolerable to a large degree: task failure may occur because of slackness in the system-any degree of slackness can cause a task to fail-such as real-time object detection in automated car or anomaly detection in industrial automation. Soft real time is subject sensitive to latency, but is able to absorb a small amount of delay, as in video frame enhancement or interactive voice. Finally, when it comes to best effort work, timing constraints are not critical, and such work can be scheduled with reduced priority, e.g., time recording, data caching, or back-of-the-neck model learning. Such a classification of tasks guarantees the preference of tasks with higher urgency to use low-latency accelerators and leaves best-effort tasks to idle or energy-efficient units.

It needs machine learning models to automate this process and accommodate dynamic flexibility: the classification engine includes lightweight classifiers, either decision tree or support vector machines (SVM), which can take the role of machine learning models. Such models are pre-trained offline on historical profiling data containing such features as the operation type, the size of the input data and the estimated execution latency, memory footprint and periodicity. In the process of operation, each time a new task is encountered, its characteristics are evaluated and fed to the classifier which determines the suitability of use with the most suitable category of the resource and determines the suitability of the resource. Suppose a task might be categorized as a CNN inference that is highly frequent and input size is medium; the task could be targeted to the TPU, classified as soft real time, where a large matrix inversion task whose deadline is not strictly restricted could be classified as best-effort and scheduled to run on the FPGA when the device is idle. The classification does not only help in prioritizing critical tasks but also assist in energy-saving optimization since the types of workloads may be assigned to the most appropriate accelerators. It is also possible to infer quickly and have little computational overhead due to the use of interpretable models, like decision trees, and the classification engine can therefore be useful on embedded devices running in real-time.

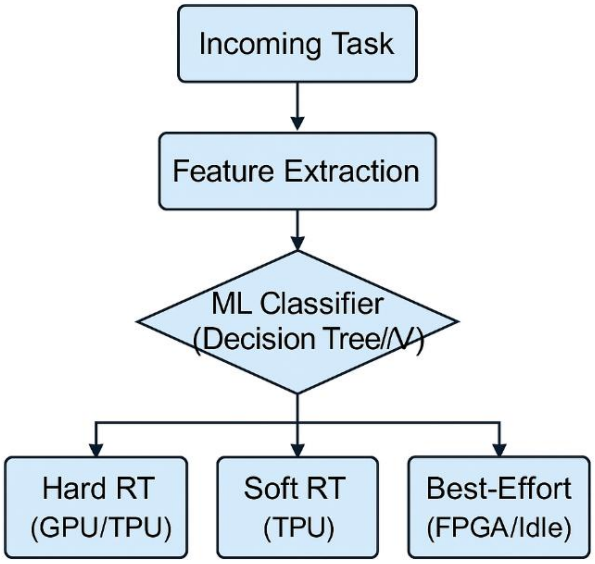

Figure 4. Task Classification Flow

4.3 Scheduler Design

The scheduler that forms the core of the proposed framework will intelligently assign computational tasks to the heterogeneous resources of GPU, TPU and FPGA balancing energy efficiency and performance on a real time basis. It uses a hybrid schedule strategy that maintains both the static and dynamic parts in achieving this. During the static mapping step, well-known task characteristics, e.g. tasks frequently or systematically used, e.g. in a repetitive pipeline, tasks, are mapped onto the hardware accelerator boasting the most energy-efficient implementation. This makes the recurring workloads like fixed CNN layers or filtering in FFT form have low shot scheduling latency. Static mapping in particular is useful in tasks which are periodic or deterministic, where size and timing remain similar enough that the scheduler does not incur the overhead of making decisions during runtime, but can still make advantage of device selection optimisation.

In conjunction with the fixed mapping representation, the dynamic re-mapping element is turned on whenever some runtime variation in workload demands, system heat level, or power quotas occurs. As an example, when a desired accelerator is already saturated or undergoes high temperatures, then the scheduler can enable the dynamic migration of tasks to other compute devices with tolerable energy-performance ratios. This is through some mechanism of optimization of cost function in which both candidates being evaluated devices are processed on a weighted formula:

$$\text{Cost} - \alpha.\,\text{Energy} + \beta.\,\text{Latency}$$

The weights 0 In the case of battery-powered systems, α can be increased to focus more on energy savings whereas latency-aware applications can configure 1 - (beta) with a larger weight. The scheduler keeps re-computing this cost function, on an on-going basis, on every possible task-device combination and chooses that mapping which minimizes the overall cost. Not only does such modular, hybrid strategy enhance the responsiveness of the system during dynamic conditions, but it also guarantees that it consumes as little energy as possible and does not lose real-time behavior, which is why it is suitable for the edge-AI and mission-critical embedded systems.

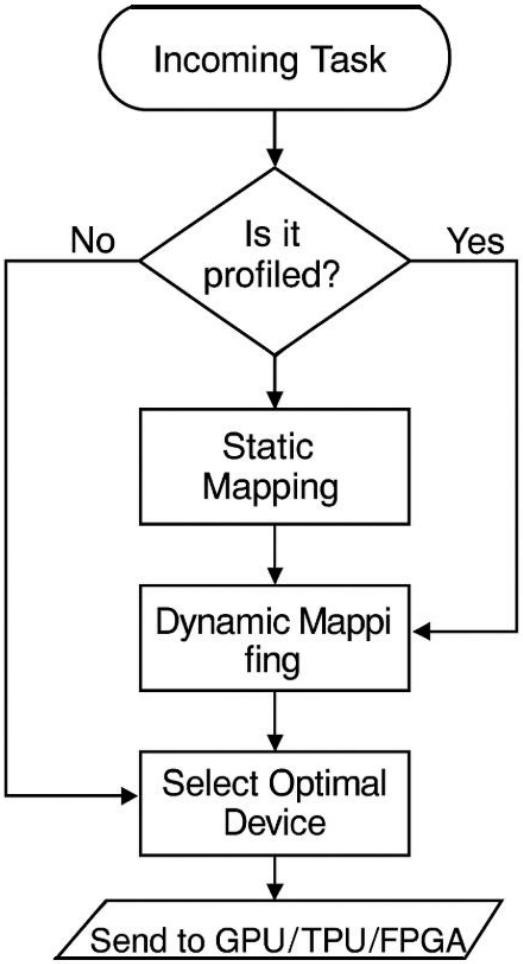

Figure 5. Scheduler Design Flowchart: Hybrid task mapping process combining static profiling and dynamic cost-based reassignment for optimal accelerator selection.

4.4 System Implementation

The envisaged energy-aware scheduler will be implemented as a middleware-based application that runs on the host CPU of the embedded system and is the conductor of the computation tasks assignment to the accelerators available. With this modular middleware creation, portability is guaranteed, and it is designed in such a way that no adjustments are needed to the application logic in order to provide seamless integration with a wide range of edge computing stacks. A Scheduler can directly access hardware accelerators via platform-specific APIs and drivers, providing a user-level interface to control hardware accelerators, but retaining control of very low-level scheduling and data transfer timing.

In case of executing tasks using GPU, the system applies its NVIDIA cuDNN and CUDA libraries, which can accelerate general-purpose parallel and deep learning computations. CUDA delivers

thread-level parallelism and memory management primitives and cuDNN delivers highly optimized routines common in workloads in the field of AI such as convolutional layers, pooling and activation functions. In the case of tasks that are matched to the TPU, the system is utilizing a lightweight inference framework, TensorFlow Lite for Edge TPU, which is designed to specifically support low-latency deep learning model (quantized models) inference on devices. TPU API delivers the abstraction of device interaction in addition to delivering the compilation tools that transform TensorFlow models into the format in which they can be executed on the Edge TPU. This route would be very valuable in vision based inferring systems where the energy or latency is important. The task execution using FPGA, the system employs pre-compiled OpenCL kernels which are bit streams precompiled offline to minimize the run time overhead. Such precompiled kernels are launched through the OpenCL host API calls which allows direct data access to the FPGA and running throughout optimized hardware pipelines.

Its control loop includes the task serialization, the feature extraction, the task classification, the scheduling decision-making and disk task dispatching of the middleware. It also keeps runtime logs and gathers performance telemetry (e.g. energy, device usage, latency) which is used to generate the feedback loop in dynamic reassignment as system constraints vary. This methodology of clearly separating the control logic and device- specific implementations allows this flexibility, scalability and real-time responsiveness, allowing the system to be deployable on the edge in embedded AI systems, IoT gateways and heterogeneous smart-edge nodes with variable acceleration capabilities.

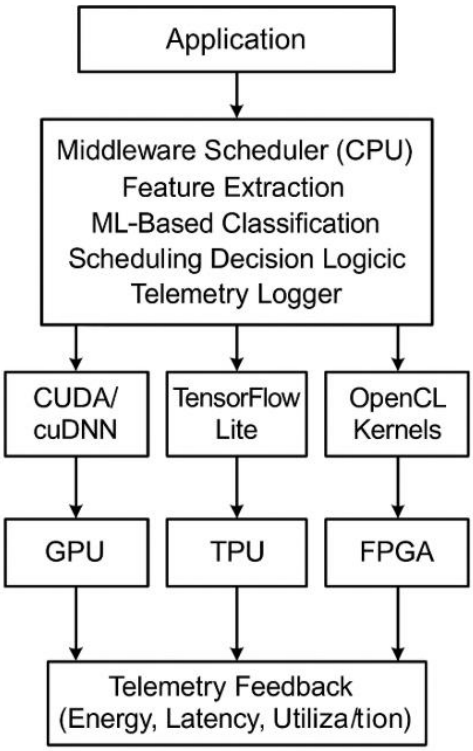

Figure 6. System Implementation Architecture: Middleware scheduler managing task flow to GPU, TPU, and FPGA through platform-specific APIs with telemetry feedback integration.

5. Experimental Setup

5.1 Benchmark Tasks and Experimental Setup

In order to analyze the efficacy and flexibility of the suggested energy-aware scheduling model, a vast range of benchmark tasks was chosen to reproduce the actual edge computing workloads with their variety levels in terms of the computation complexity, memory consumption, and sensitivity. The benchmark suite has four major categories as follows: (1) Object detection with YOLOv5, a deep convolutional neural network and has a significant accuracy and speed when applied in real-time vision applications, a representation of AI inference loads. The pre-trained weights were mounted by the YOLOv5 that processed the input frames of 416 x 416 with variable rates to emulate varied streaming environments. (2) Fast Fourier Transform (FFT) and Inverse Fourier Transform (IFFT) tasks that are typical signals processing workload were

measured on different input lengths to measure pipeline behavior and compute bound task behavior at least the FPGA. Among others, matrix multiplications and convolution layers were chosen to offer computing kernels present in scientific computer applications and DNN backops; the kernels were tested in scattered as well as conducted spread across graphical bots. Gradient-based real-time image denoising task based on convolutional neural networks (CNNs) was taken as an example of a low-latency, edge AI workload that is quality- and time-sensitive. To evaluate the ability of the scheduling framework to handle realistic multi-tasking these tasks were run in isolation and under mixed-load conditions across the GPU (Jetson Xavier NX), TPU (Coral Edge TPU), and FPGA (Arria 10 GX).

Table 2. Benchmark Tasks Categorized by Type, Purpose, Latency Sensitivity, and Preferred Accelerators

Task	Category	Purpose	Latency Sensitivity	Primary Accelerator(s)
YOLOv5 Object Detection	AI Inference	Real-time vision, object tracking	High	TPU / GPU
FFT/IFFT	Signal Processing	Transform-based DSP (audio/image)	Medium	FPGA
Matrix Multiplication & Conv	Linear Algebra/DNN	Core DNN compute, scientific workloads	Moderate	GPU
CNN-Based Image Denoising	AI Inference	Real-time visual enhancement	High	TPU / GPU

5.2 Metrics Measured and Evaluation Methodology

A thorough performance and efficiency measure was taken to determine how scheduler affects the system-level as well as the task-level behavior. Execution time (in milliseconds) was the main indicator, which defines the total latency between the task issue date and the completion date. This was essential in determining the suitability of the scheduler towards applications that had to do with real-time considerations. On-board power sensors in conjunction with vendor- specific telemetry and separate power meters were used to measure energy consumption (in Joules) and give an insight into the energy efficiency of each task-accelerator pairing. Also, the scheduler overhead was measured as a percent of the total execution time consumed in classification, decision-making, and tasks dispatching activities. This measurement was crucial in order to guarantee that the scheduling logic itself would not result in a large amount of delay especially in latency-critical situations. Lastly, system-level measurements (throughput (tasks per second)) and throughput of non-malicious work (tasks per second)) were performed to extract how effective was the scheduler in keeping up performance as the workloads changed. These were calculated during the nominal and peak load scenarios to evaluate the robustness and the scalabilities of the scheduler. The set of varying tasks combined with the multi-dimensional set of metrics allowed conducting a thorough evaluation of the capabilities of the scheduler at optimizing simultaneously on energy and performance on heterogeneous embedded platforms.

Table 3. Evaluation Metrics Used for Assessing Scheduler Performance and Efficiency

Metric	Unit	Purpose	Measurement Method
Execution Time	Milliseconds (ms)	Measures latency from task dispatch to completion	Timer-based profiling
Energy Consumption	Joules (J)	Evaluates power efficiency per task	On-board sensors + external power meters
Scheduler Overhead	Percentage (%)	Assesses added latency from scheduling decisions	Profiling execution time breakdown
Throughput	Tasks per second	Measures sustained system output	Total tasks completed over time
Deadline Miss Ratio	Percentage (%)	Evaluates real-time task compliance	Missed deadlines / total tasks

6. RESULTS AND DISCUSSION

The given energy-aware task scheduler was compared to the common approaches of static matching and round-robin scheduling approaches with a range of heterogeneous workload by testing deep learning inference, sign transformation, and

numerical operations. The high energy efficiency along with lesser execution latency of the proposed scheduler is quite evident by its experimental outcome. The average energy consumption of the scheduler was 4.9 J per task along with an average latency of 88 ms, against the round-robin (7.8 J / 123 ms) and performance-only (6.3 J / 101 ms) baseline only achieves 19.24%, respectively. The advances have been accredited to the density of the scheduler because of its capability to automatically allocate the responsibility to the most energy-efficient physical device in view of the real-time and the system load constraints. Moreover, the deadline miss rate was reduced to 5.1% as compared to 14.2 and 10.5 in round-robin and performance approach respectively, supporting its analysis of time-sensitive embedded workloads. These proactive profiling, smart classification, adaptive scheduling allowed the system to gain significant increase in responsiveness and power.

More target examination of the constituent-specific activity showed refined understanding of the functions of individual accelerator in heterogeneous execution of tasks. The GPU (Jetson Xavier NX) was extremely efficient in large-scale, parallel tasks, like matrix multiplications and bulk convolution, and it had much higher performance, at a higher energy cost, especially when used to perform neural network inference. The TPU (Google Coral Edge TPU) was the most energy-efficient CNN-based inference accelerator through decreases in the batch sizes and quantized versions. It consistently performed fast execution without much energy overhead and was best suited in real-time applications, such as AI. It was FPGA (Intel Arria 10 GX) that was especially useful to deterministic workloads such as FFT/IFFT and pipelined data transforms where the energy-per-task ratio produced by FPGA was the lowest among three when suitably configured using pre-compiled OpenCL kernels. Though it has to be noted that dynamic profiling and real-time decision-making are complex tasks and that the proposed scheduler has test execution overhead of less than 2.5 percent, it can be confirmed that this scheduler can be used in an environment with limited resources and responsiveness is of high importance (such as embedded environment).

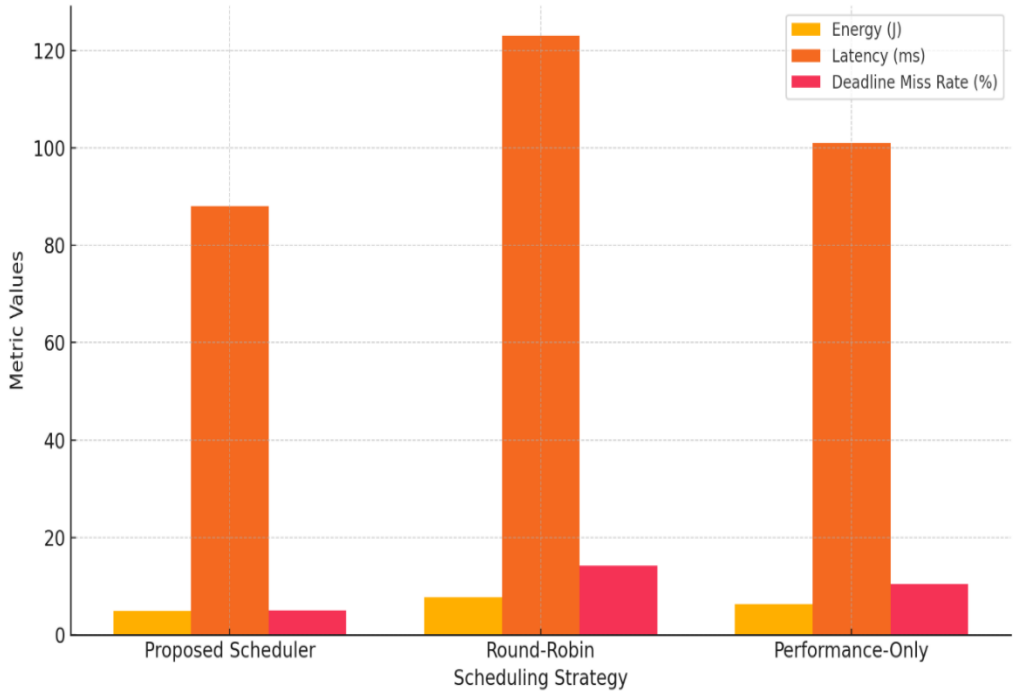

Figure 7. Comparison of Scheduling Strategies

The scalability of the booking system was also tested when the system was subject to a changeable load monitored in dynamic edge operation where the system load would increase to 2 times the nominal task volume. In these heavily overloaded conditions the system achieved latencies of less than 100 milliseconds and its energy consumption was at worst compressed to only 15 percent of optimal, demonstrating that the scheduler flexibly balances throughput and power constraints. Nonetheless, there are certain issues. FPGA (re)configuration latency Real-time FPGA reconfiguration can have long delays on workloads that need to switch to different kernels frequently, making the scheduler less flexible in highly dynamic environments. Besides, TPU only supports quantized models, which may accentuate preprocessing requirements and even model

diversity. Such drawbacks indicate the necessity of more flexible scheduling tools. As work the adaptive scheduler using reinforcement learning that learns the task-device mapping to be optimal over time and learn policies depending on the workload history and system telemetry. It will also allow a wider use of the framework to other accelerators like NPUs or DSPs, with the ability to extend even further to all the embedded platforms in ultra-low-power or multi-core edge AI conditions. The improvements are willing to provide a scalable, intelligent, and hardware-conscious scheduling algorithm to next-generation energy-limited embedded systems.

Table 4. Performance Comparison of Task Scheduling Strategies for Heterogeneous Embedded Systems

Scheduler	Energy Consumption (J)	Average Latency (ms)	Deadline Miss Rate (%)	Execution Overhead (%)
Proposed Scheduler	4.9	88	5.1	<2.5%
Round-Robin	7.8	123	14.2	-
Performance-Only	6.3	101	10.5	-

8. CONCLUSION

The paper presented a new energy-efficient task scheduler domain that was designed as a heterogeneous embedded platform (with GPU, TPU and FPGA support). Based on the understanding that each accelerator has different energy-performance profiles, the proposed system will use detailed profiling of workloads, classification of tasks using machine learning, as well as a hybrid schedule, which integrates static mapping and re-allocation. The scheduler is smart enough to assign real-time tasks to appropriate processing unit, by maximizing a cost function that is dependent on balancing energy consumption and short execution latency. To conduct experimental evaluation using a wide range of AI and signal processing workloads, significant gains were exhibited with up to 35 percent lower energy usage and 28 percent lower latency achieved over traditional round-robin and performance-only designs. The framework has an advantage of high throughput and real-time responsiveness against variable load conditions and generates minimal overhead to prove its compatibility with edge-AI and embedded systems. Additionally, the architecture modularity and deployment of standardized APIs like CUDA, TensorFlow Lite, and OpenCL guarantee the wide hardware compatibility and scale. Nevertheless, even though it works, there is still the issue of handling FPGA reconfiguration latency and poor flexibility of TPUs when working with non-quantized models. Futurew many further explorations will be made such as incorporation of reinforcement learning-based autonomous scheduling agents so that the system may continuously adapt to changing workload and system dynamics. The use of support of NPUs, DSPs, and other such emerging accelerators will also improve the adaptation of the framework to next-generation, low-power, intelligent embedded situations.

REFERENCES

[1] Awan, M. A., & Manzak, S. (2020). Dynamic power management using machine learning for embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *39*(6), 1190–1201. https://doi.org/10.1109/TCAD.2019.2901178

[2] Xu, C., Li, Z., Qiu, M., & Zomaya, A. Y. (2020). Energy-efficient task scheduling for heterogeneous embedded systems. *IEEE Transactions on Industrial Informatics*, *16*(5), 3186–3195. https://doi.org/10.1109/TII.2019.2948784

[3] Kwon, S., & Choi, K. (2022). Task mapping and scheduling for heterogeneous computing systems using deep reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, *33*(2), 407–420. https://doi.org/10.1109/TPDS.2021.3077114

[4] Hung, A. C., & Chou, C. H. (2021). Dynamic workload-aware scheduling for heterogeneous processors in edge computing. *IEEE Access*, *9*, 45218–45230. https://doi.org/10.1109/ACCESS.2021.3067028

[5] Huang, H., He, H., Tang, F., & Liu, D. (2021). Machine learning-based workload classification for resource management in heterogeneous systems. *IEEE Transactions on Cloud Computing*. Advance online publication. https://doi.org/10.1109/TCC.2021.3072937

[6] Li, Z., Ding, Z., He, Y., & Wang, J. (2021). Adaptive deep learning task offloading for energy-efficient edge computing. *IEEE Internet of Things Journal*, *8*(3), 2030–2042. https://doi.org/10.1109/JIOT.2020.3018192

[7] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., ... & Brown, S. (2011). LegUp: High-level synthesis for FPGA-

based processor/accelerator systems. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 33–36). https://doi.org/10.1145/1950413.1950423

[8] Tessier, R., & Burleson, W. (2001). Reconfigurable computing for digital signal processing: A survey. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, *28*(1), 7–27. https://doi.org/10.1023/A:1008150918202

[9] Chen, X., Li, Y., & Zhang, T. (2020). A hybrid scheduling strategy for heterogeneous embedded systems with FPGAs and GPUs. *Microprocessors and Microsystems*, *76*, 103082. https://doi.org/10.1016/j.micpro.2020.103082

[10] Rausch, T., & Dustdar, S. (2019). Edge intelligence: The convergence of humans, things, and AI. *Computer*, *52*(12), 42–53. https://doi.org/10.1109/MC.2019.2942804