

Title	ELEVATE: Optimal scheduling of time-sensitive tasks on the heterogeneous reconfigurable edge
Authors	Hoyer, Ingo;Zaarour, Tarek;Khalid, Ahmed;Utz, Alexander;Seidl, Karsten;Brown, Ken;Zahran, Ahmed H.
Publication date	2025-02-04
Original Citation	Hoyer, I., Zaarour, T., Khalid, A., Utz, A., Seidl, K., Brown, K. and Zahran, A. H. (2024) 'ELEVATE: Optimal scheduling of time-sensitive tasks on the heterogeneous reconfigurable Edge', 2024 IEEE 32nd International Conference on Network Protocols (ICNP), Charleroi, Belgium, 28-31 October 2024, pp. 1-6. https://doi.org/10.1109/ICNP61940.2024.10858561
Type of publication	Conference item;proceedings-article
Link to publisher's version	10.1109/icnp61940.2024.10858561
Rights	© 2024, IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Download date	2026-10-31 01:15:23
Item downloaded from	https://hdl.handle.net/10468/17944

ELEVATE: Optimal scheduling of time-sensitive tasks on the heterogeneous reconfigurable Edge

Ingo Hoyer¹, Tarek Zaarour², Ahmed Khalid², Alexander Utz¹, Karsten Seidl^{1,3}, Ken Brown⁴, and Ahmed H. Zahran⁴

¹Fraunhofer IMS, Duisburg, Germany.

²Dell Research, Dell Technologies, Ireland.

³Department of Electronic Components and Circuits, University of Duisburg-Essen, Duisburg, Germany.

⁴School of Computer Science & Information Technology, University College Cork (UCC), Ireland.

Abstract—Edge computing is evolving to include heterogeneous compute nodes with distinct characteristics. Graphic processing units (GPU) and field-programmable gate arrays (FPGA) can execute demanding deep learning (DL) tasks while meeting the deadlines of time-sensitive applications. However, FPGAs require reconfiguration to execute different tasks. In this paper, we first demonstrate that FPGAs can be reconfigured in real-time. Additionally, we propose ELEVATE as a novel scheduling algorithm for reconfigurable heterogeneous edge computing platforms targeting Industry 4.0 post-production quality control. ELEVATE design focusses on optimising the reconfiguration of the FPGA unit for heterogeneous quality inspection tasks. Our simulations indicate that ELEVATE reduces task waiting time by up to two orders of magnitude and achieves energy savings of up to 25 % compared to a statically configured FPGA unit.

Index Terms—Scheduling, FPGA, HLS, RISC-V, offloading

I. INTRODUCTION

Industry 4.0 converges several emerging technologies to facilitate smart, sustainable manufacturing and supply chains [1]. These technologies include advanced precision robotics, artificial intelligence (AI), time-sensitive networking (TSN), and edge computing. Advanced robotics enables the automation of production, while AI enables autonomous decision-making and self-optimising processes within the factory. For example, AI-based computer vision (CV) enables automating decisions for both manufacturing steps and post-production quality control. Additionally, it can be leveraged to optimise production efficiency by analysing sensor data for predictive maintenance. Advanced networking approaches, such as TSN, enable the timely provision of various data streams to ensure automation efficiency. Edge computing enables the economic deployment of smart manufacturing on a large scale by sharing expensive compute resources among simpler end devices [2].

Meeting the strict delay requirements for demanding computing tasks, e.g., computer vision, requires the use of high-performance, powerful hardware. Graphics processing units (GPUs) have proven their ability to deliver such performance for general tasks, but they are known to be energy-

hungry devices. Additionally, GPUs may deliver inconsistent execution performance due to speed throttling to cool the chip [3]. FPGAs have recently evolved as a fast, energy-efficient alternative for executing these demanding tasks [4]. Including an FPGA in a setup can increase the performance by a factor of 2 [5]. However, FPGA-based accelerators lack the general-purpose aspect of GPUs, as they can only perform the task defined by their loaded configuration, called bitstream. Hence, edge platforms are evolving to integrate heterogeneous computing platforms to leverage their distinct benefits.

Task offloading literature is dominated by solutions focused on homogeneous computing [6]. Solutions that model the complexities of heterogeneous computing platforms are still evolving. Liu et. al. [7] propose resource allocation and task offloading in heterogeneous edge computing settings. Similarly, Cooke and Fahmy [8] propose a graph-based task offloading strategy for heterogeneous computing platforms. However, these solutions do not leverage FPGA reconfiguration, which is considered impractical. Such static configurations may underutilise heterogeneous computing system resources and reduce its capacity.

Evolving FPGA hardware and firmware enable FPGA real-time reconfiguration for DL models. To change the configuration of an FPGA, the bitstream has to be loaded onto the device. Modern devices allow for partial reconfiguration, which is much faster than exchanging the entire bitstream memory. Additionally, the conversion from Python-based DL model into hardware description language (HDL) has become much faster with tools such as tinyHLS and hls4ml [9, 10].

This paper first illustrates FPGA's operational dynamics and processing gain in comparison to GPUs using real-world experimentation. Specifically, we consider convolutional neural network (CNN) models designed for time series analysis of data collected by robotic arms. This study illustrates the feasibility of reconfiguring the FPGA in real time. Second, we propose ELEVATE as a novel task scheduling algorithm for heterogeneous edge industrial robotics systems that include FPGA and GPU units. ELEVATE is presented for quality control after production for a number of products. ELEVATE represents an optimised solution that maximises the utilisation of FPGA units by leveraging FPGA reconfiguration and task

This research has been funded in part by the project CLEVER (Project ID 101097560), that is supported by the Key Digital Technologies Joint Undertaking and its members (including top-up funding by the German Federal Ministry of Education and Research (BMBF) and Enterprise Ireland (EI)).

queuing. Our performance evaluation is based on simulations that compare ELEVATE and another heuristic that does not leverage FPGA reconfiguration. Our results illustrate that ELEVATE reduces the energy consumption by up to 25% at low system load, reduces the median task waiting time by two orders of magnitude at high system load, and increases the system processing capacity.

The rest of this paper is organised as follows. Section II illustrates the timing and performance benefits of FPGA in comparison to GPU. Section IV presents ELEVATE design followed by its performance evaluation in Section V. Finally, we conclude in Section VI.

II. MOTIVATION AND MODEL FOUNDATION

This research aims at a general optimisation for heterogeneous workloads on the edge. For a quantitative analysis towards the benefits of using an FPGA, as well as to collect data for a scheduling model, three different CNNs are used. To demonstrate the advantage of a heterogeneous setup including an FPGA, the inference runtimes for the FPGA, GPU, and CPU are measured. Based on the timing results, a model can be developed in Section III.

The first CNN is from a condition monitoring application that aims to detect tool wear from **vibration** data. It includes five convolutional layers, interspersed with four max-pooling layers, followed by a Global Average Pooling (GAP) layer and six dense layers [11]. This configuration is similar to those used in network designs for spectrogram-based analysis of time-series data [12]. After each convolutional and dense layer, the activation function used is the rectified linear unit (ReLU). In order to minimise hardware resources, the CNN parameters are encoded in a fixed-point datatype using two's complement with a total of 16 bits, with one bit representing the integer part. For training, QKeras was used to overcome the quantisation restrictions [13].

The second CNN is designed to detect particular abnormalities from **voltage** time series data. It has only three convolution layers, GAP, and two dense layers [14, 15]. An 8-bit quantisation with four fractional bits was used and trained with QKeras. It receives 512 samples on each of the two channels as inputs.

The third CNN is designed to interpret a small, scalar vector such as individual **features** from an inspection. It uses a convolution layer, followed by max-pooling and a convolution layer with GAP. The features could be minimum or maximum values, averages, time windows, and any other scalar feature of a time series signal. The final classification in the CNN is performed with two dense layers. The datatype was the same as for the voltage CNN. It receives 32 inputs.

A. FPGA Implementation

In this work, an FPGA (Xilinx Versal VCK190) [16] in an edge microdata centre (EMDC) [17] is used. On the FPGA, both the AIRISC, an open-source processor based on RISC-V, and the studied CNN accelerators are implemented. The AIRISC utilises the RV32IMFC instruction set [18, 19]. It

manages the data flow into and out of the accelerators, along with handling communication via standard digital interfaces such as I2C, SPI, and UART. The CNN accelerators are integrated as a memory-mapped peripheral via the AHB-Lite system bus with 32-bit words, [20]. For the FPGA implementation, a frequency of 100 MHz was assumed. The additional time for communication with the scheduler is not accounted for.

Common approaches for high-level synthesis (HLS) use a combination of open-source preprocessing and proprietary HLS tools [10, 21]. A major disadvantage is that these tools cannot target diverse hardware platforms, such as an EMDC with more than one type of FPGA. Hence, in this work, a novel open-source hardware compiler (tinyHLS) is used [9]. It uses a template-based approach for CNN implementation in platform-independent Verilog hardware description language code.

B. Timing Results

To implement an accurate mathematical model of the scheduler, runtime measurements are required for both the recompilation and the configuration of the FPGA (Tab. II). The best case recompilation time is measured for the feature classification CNN, while the worst case is measured for the vibration classification CNN. The time for bitstream modification and FPGA configuration is also considered; the worst case (configuration swapping with a software specialiser) and best case (dynamic reconfiguration at 100 MHz) values are adapted from [22]. Additionally, the inference runtime needs to be measured (Tab. I), including values for CPU (Intel i9-10940X) and GPU (NVIDIA RTX A6000) runtime to ensure an accurate comparison [23, 24]. The FPGA is much faster than both the GPU and CPU, even if the configuration time is calculated in.

III. SYSTEM MODEL

We consider an edge-based robotic quality inspection after production in a smart factory (Fig. 1). The factory has S robotic stations that inspect P different products. Each station is assumed to handle only one product. s_p represents a subset of stations that are dedicated to product p . The robotic arm

TABLE I: Runtime and required resources of the different implementations and CNNs on the used hardware.

Algorithm / Impl.	Timing results			FPGA Utilization
	CPU	GPU	FPGA	LUT in %
Vibrations	153 ms	118 ms	1.1 ms	2.4
Voltages	128 ms	96.3 ms	6.8 ms	14.9
Features	69.7 ms	20.4 ms	4.14 μ s	2.04

TABLE II: Time that a certain step takes.

Step	Time		Description
	best	worst	
Recompilation	6.35 ms	72 ms	Python to Verilog
Bitstream mod.	2.1 ns	398 μ s	Verilog to Bitstream
FPGA configuration	3.49 μ s	1.08 ms	Loading the bitstream

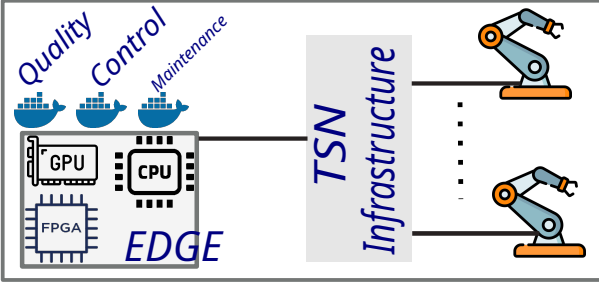


Fig. 1: Edge-based post-production quality control.

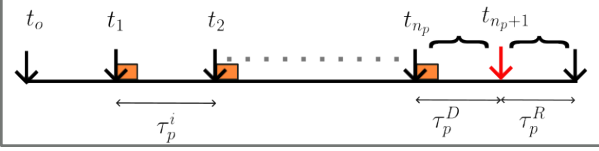


Fig. 2: Product inspection dynamics.

runs different modules at the edge, including mobility control, predictive maintenance, and quality inspection modules. We assume that the robotic arms communicate with the edge servers via TSN infrastructure, which ensures reliable and timely transmission of control and inspection data.

Figure 2 illustrates the dynamics of the quality inspection process. Each product is inspected by capturing n_p checkpoints, where $p \in \{1, \dots, P\}$, using a sensory device, such as a camera. The product arrives at time t_o , after which the robot manoeuvres to reach inspection point $i \in \{1, \dots, n_p\}$ at t_i and sends D_p bytes to the quality inspection module at the edge for inspection. The inspection decision should be sent from the edge back to the arm within τ_p^D s from t_{n_p} . The arm then applies the control action and replaces the currently inspected sample with the following one within τ_p^R s. We consider fixed, product-specific time parameters, i.e., τ_p^i , τ_p^D , and τ_p^R , between inspection points to be fixed to ensure that the worst case delay is covered. This assumption also facilitates the design of the underlying TSN infrastructure.

IV. ELEVATE DESIGN

Scheduling different quality inspection tasks to heterogeneous computing platforms can be classified as a generalised assignment problem [25], which is known to be NP-hard. Hence, the ELEVATE scheduler splits the scheduling decision into two components, including the optimisation of FPGA assigned tasks and the running processes to schedule the available computational resources. This design is motivated by the speed and energy efficiency of FPGAs. Hence, ELEVATE first identifies a schedule for FPGA tasks while leveraging reconfiguration and task queuing. ELEVATE then assigns other tasks to the GPU.

A. Optimizing FPGA Utilization

Figure 3 illustrates a typical ELEVATE schedule for an FPGA unit. The FPGA unit time will be divided into sub-cycles dedicated for processing compute tasks for a subset

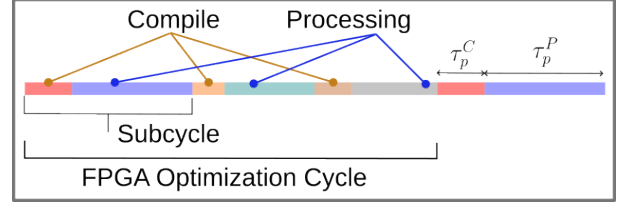


Fig. 3: FPGA Cycle

of products. Each sub-cycle consists of two parts, including configuration and processing. Once a sub-cycle ends, the FPGA is reconfigured for the following sub-cycle for the next product. These sub-cycles form a cyclic configuration of the FPGA unit that repeats as long as the factory setup remains fixed, i.e., S , P , and $s_p \forall p \in \{1, \dots, P\}$.

Part of the FPGA time is typically wasted in the configuration phases (Fig. 3). ELEVATE design is centred on maximising FPGA processing and minimising the configuration overhead. This design is envisioned to maximise both system utilisation and energy efficiency. To achieve this goal, the FPGA cycle is identified by solving problem (P1), which is shown below. Additionally, ELEVATE reduces the FPGA reconfiguration overhead by queuing tasks for product p before the start of its sub-cycle. Any enqueued task should meet its deadline, t_{n_p+1} in Fig. 2, when processed on the FPGA unit. This constraint can be expressed as

$$t_{n_p+1} - t_o^P - \bar{q}_p \tau_p^F > \tau_p^F, \quad (1)$$

where t_o^P represents the starting time of the processing phase of product p subcycle, $\bar{q}_p$ represents the number of enqueued tasks that have earlier deadlines, and τ_p^F denotes the duration needed by the FPGA to process an inspection task for product p . Eq. 1 states that the task is authorized if its time limit minus the waiting time until the start of the processing of p products on the FPGA is sufficient to process all tasks with earlier deadlines and the incoming task. Note that we assume that the earliest deadline of the waiting tasks is granted the highest priority in the queuing mechanism.

The utilization of every subcycle is calculated as the ratio between the processing portion for product p , denoted as τ_p^P , to the FPGA cycle duration, denoted as T . τ_p^P , can be calculated as

$$\tau_p^P = (q_p + s_p \lambda_p \tau_p^P) \tau_p^F, \quad (2)$$

where q_p represents the number of enqueued tasks for product p at the start of its processing phase, λ_p represents the arrival rate of product p tasks during the processing phase. While the processing phase include sensing, decision, and action phases, the task arrival is associated with the sensing part only. Hence, λ_p can be expressed as

$$\lambda_p = \frac{n_p}{n_p \tau_p^i + \tau_p^D + \tau_p^R}.$$

By simplifying Eq. 2, τ_p^P can be expressed as

$$\tau_p^P = c_p q_p,$$

where $c_p = 1/(1/\tau_p^F - s_p\lambda_p)$. Hence, product p FPGA utilization, denoted as v_p , can be calculated as

$$v_p = \frac{\tau_p^P}{\sum_p \gamma_p (\tau_p^P + \tau_p^C)}, \quad (3)$$

where γ_p represent a binary decision variable set to 1 if product p is allocated a subcycle on the FPGA and τ_p^C represent the FPGA configuration time for product p . Hence, the total FPGA utilization, denoted as v , can be expressed as

$$v = \sum_{p \in P} \frac{\gamma_p \tau_p^P}{\sum_k \gamma_k (\tau_k^P + \tau_k^C)}. \quad (4)$$

Specifically, the cycle can be defined by solving the following optimisation problem (P1).

$$\max \quad v \quad (5)$$

$$\text{st.} \quad v \leq 1 \quad (6)$$

$$\gamma_p \in \{0, 1\}$$

$$q_p \in Z$$

Eq. 6 ensures that the FPGA unit utilisation is less than 100%. The program represents a MINLP but can be transformed into a Mixed-Integer Quadratically Constrained Programming (MIQCP) by performing a couple of variable transformations. Specifically, introducing a variable (u) to linearise the objective and introducing new variables (x_p) to convert cubic terms (e.g., $u\gamma q_k$) in 6 into quadrature terms (e.g., $x_k q_k$). The resultant program P2 can be expressed as follows:

$$\max u \quad (7)$$

$$\text{st.} \quad \sum_k x_k (c_k q_k + \tau_k^C) \leq \sum_{p \in P} \gamma_p c_p q_p \quad (8)$$

$$x_k \leq u \quad (9)$$

$$x_k \leq \gamma_k \quad (10)$$

$$x_k \geq 0 \quad (11)$$

$$x_k \geq u + \gamma_k - 1 \quad (12)$$

$$u \leq 1 \quad (13)$$

$$\gamma_p \in \{0, 1\}$$

$$q_p \in Z^+$$

B. Task Scheduling

With the FPGA cycle optimized, incoming jobs would be scheduled by applying the following rules

- 1) Enqueue the job for FPGA if it can satisfy the scheduling constraint; i.e., eq. (1).
 - a) If the queue is full, send the head of FPGA task queue to the GPU.
- 2) Otherwise, Send the job to the GPU.

TABLE III: Simulation parameters

P	4 to 8	τ_p^i	$U[100, 500]$	ρ	$U[1, 80]$
S	$U[5, 10]$	τ_p^C	$U[50, 250]$	τ_p^R	$U[250, 400]$
n_p	$U[5, 10]$	τ_p^D	$U[250, 400]$	β	0.04

V. EVALUATION

A. Evaluation Setup

We evaluate the performance of the proposed scheme using simulations that are implemented with Python. Table III highlights the simulation parameter details. The GPU processing time of product p task is estimated as $\beta\tau_i$ while the FPGA processing time is calculated as τ_p^G/ρ . In this evaluation we scale the system load by increasing the number of products from 4 to 8.

We compare the following scheduling algorithms

- ELEVATE: We solve P2 using Gurobi solver to identify the dynamics of FPGA reconfiguration.
- MaxUtil: a static configuration (SC) scheme in which the FPGA is configured to process the product that has the highest FPGA utilization; i.e., $\arg\max_p (s_p\lambda_p\tau_p^F)$. Both FPGA and GPU tasks are scheduled using earliest deadline first algorithm.

Our key performance metrics include task waiting time at the assigned compute (FPGA or GPU), task success ratio with the success is based on satisfying the task deadline, and system energy consumption estimated as

$$\text{Energy} = T^F + T^G * \delta,$$

where T^F and T^G represent the FPGA and GPU processing time for all assigned tasks during the simulation, and δ represents a scaling factor for excess energy needed to process a task in the GPU in comparison to FPGA. δ is typically 5 to 8 depending on task complexity and used hardware [26]. The reported results are based on 10 runs with every run lasting for 10 cycle durations.

B. Performance Results

Figure 4 compares the success ratio of ELEVATE and maxUtil strategies when the number of products ranges from 4 to 8. The figure shows that the dynamic configuration of FPGA enables the system to achieve higher success ratio as the compute load increases. While both scheduling approaches achieve the highest success for a light load, maxUtil started to fail with a number of 6 or more products. In contrast, ELEVATE achieved a success ratio of 1 and 98.2% for $P = 6$ and $P = 7$ respectively. This shows that the use of ELEVATE can increase the system capacity.

Figure 5 show the boxplot of the task waiting time of both ELEVATE and maxUtil against the number of processed products. For all tested scenarios, ELEVATE consistently achieved a lower queuing delay until they are processed. The median ELEVATE delay for the case of 7 products is two orders of magnitude smaller in comparison to maxUtil.

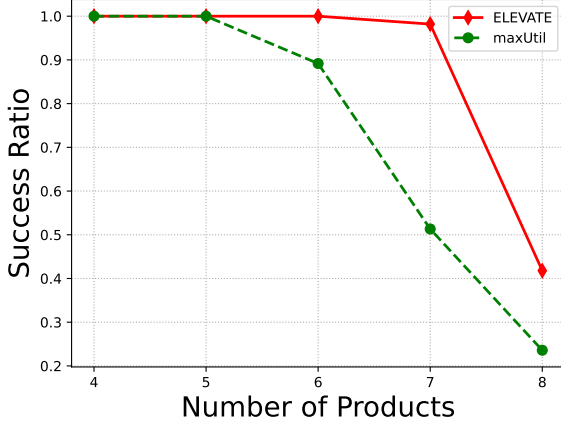


Fig. 4: Task Success Ratio under different loads

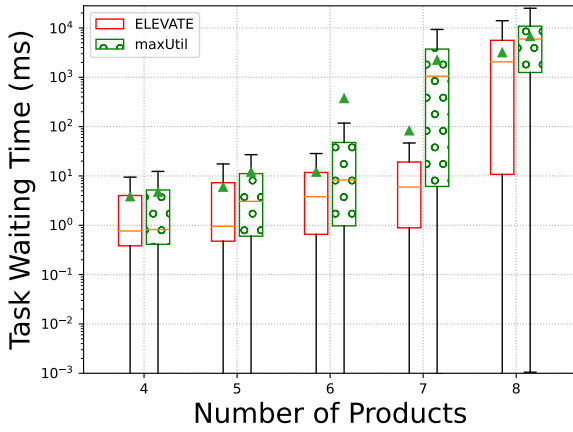


Fig. 5: Task waiting time under different loads

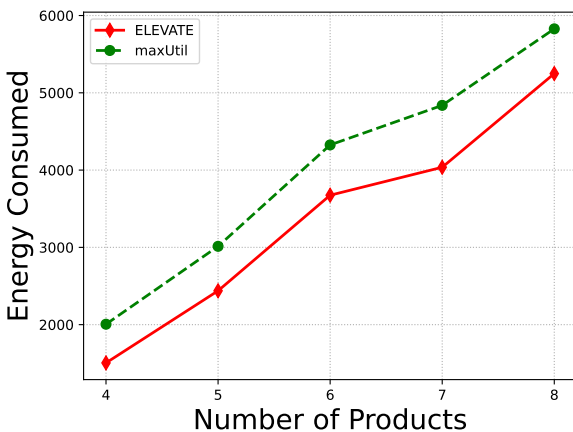


Fig. 6: Consumed energy vs. system load.

Figure 6 compares the energy consumption of maxUtil and ELEVATE strategies for various system loads. ELEVATE consistently consume less energy than maxUtil for all simulated scenarios. At light system load ($P = 4$), the system achieves 25% less energy consumption. This saving diminishes to 16% at high load ($P = 7$).

VI. CONCLUSIONS AND OUTLOOK

FPGA hardware and tools are continuously evolving to support fast, energy-efficient compute platforms for DL models. These benefits motivate the use of heterogeneous computing platforms for edge systems. Our experimentation confirms these benefits and proves the feasibility of fast reconfiguration for industrial DL models. We also propose ELEVATE as a novel scheduling algorithm that leverages FPGA reconfigurability in heterogeneous edge computing systems. Our performance evaluation shows that ELEVATE can increase the system's computing capacity, reduce system delay, and save system energy. In our future work, we will consider the real-world implementation of the simulated system.

REFERENCES

- [1] Alejandro Germán Frank, Lucas Santos Dalenogare, and Néstor Fabián Ayala. "Industry 4.0 technologies: Implementation patterns in manufacturing companies". In: *International Journal of Production Economics* 210 (2019), pp. 15–26.
- [2] Kuanishbay Sadatdiynov et al. "A review of optimization methods for computation offloading in edge computing networks". In: *Digital Communications and Networks* 9.2 (2023), pp. 450–461.
- [3] Ian Colbert et al. "A Competitive Edge: Can FPGAs Beat GPUs at DCNN Inference Acceleration in Resource-Limited Edge Computing Applications?" In: *CoRR* abs/2102.00294 (2021).
- [4] Chenren Xu et al. "The case for fpga-based edge computing". In: *IEEE Transactions on Mobile Computing* 21.7 (2020), pp. 2610–2619.
- [5] Anuj Vaishnav, Khoa Dang Pham, and Dirk Koch. "Heterogeneous resource-elastic scheduling for CPU+ FPGA architectures". In: *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. 2019, pp. 1–6.
- [6] Hai Lin et al. "A survey on computation offloading modeling for edge computing". In: *Journal of Network and Computer Applications* 169 (2020), p. 102781.
- [7] Yu Liu et al. "Joint task offloading and resource allocation in heterogeneous edge environments". In: *IEEE Transactions on Mobile Computing* (2023).
- [8] Ryan A. Cooke and Suhaib A. Fahmy. "A model for distributed in-network and near-edge computing with heterogeneous hardware". In: *Future Generation Computer Systems* 105 (2020), pp. 395–409.

- [9] *tinyHLS*. <https://github.com/Fraunhofer-IMS/tinyhls>. 2024.
- [10] Farah Fahim et al. “hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices”. In: *arXiv preprint arXiv:2103.05579* (2021).
- [11] Ingo Hoyer et al. “Hardware Accelerators for a Convolutional Neural Network in Condition Monitoring of CNC Machines”. In: *2023 IEEE SENSORS*. IEEE. 2023, pp. 1–4.
- [12] Sena Yağmur Şen and Nalan Özkurt. “ECG arrhythmia classification by using convolutional neural network and spectrogram”. In: *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE. 2019, pp. 1–6.
- [13] *QKeras*. <https://github.com/google/qkeras>. [Online; accessed 23-Mai-2023]. 2021.
- [14] Renee Lerch et al. “Design of an Artificial Neural Network Circuit for detecting Atrial Fibrillation in ECG Signals”. In: *2021 IEEE Sensors*. IEEE. 2021, pp. 1–4.
- [15] Ingo Hoyer et al. “tinyHLS: Novel Python Package for hardware accelerator generation targeting time series analysis”. In: (). Submitted.
- [16] AMD/Xilinx. *VCK190 Evaluation User Guide (UG1366)*. Rev. 1.1. 2024.
- [17] Enzo Minazzo et al. “A New Edge Micro Data Center and its Passive Thermosyphon Cooling System at TUE: Cooling System Thermal Performance Tests”. In: *2024 40th Semiconductor Thermal Measurement, Modeling and Management Symposium (SEMI-THERM)*. 2024, pp. 1–5.
- [18] *AIRISC Family*. <https://www.airisc.de>. [Online; accessed 24-January-2023]. 2021.
- [19] *AIRISC - The RISC-V Processor for Embedded AI*. https://github.com/Fraunhofer-IMS/airisc_core_complex. [Online; accessed 24-January-2023]. 2021.
- [20] ARM. *AMBA 3 AHB-Lite Protocol*. V1.0. 2006.
- [21] *Vivado Design Suite User Guide*. v2020.1. Xilinx, May 2021.
- [22] Ronnie Rikus Le Roux. *Bitstream specialisation for dynamic reconfiguration of real-time applications*. 2015.
- [23] Intel. *Intel Core i9-10940X*. <https://www.intel.de/content/www/de/de/products/sku/198014/intel-core-i910940x-xseries-processor-19-25m-cache-3-30-ghz/specifications.html>. [Online; accessed 23-June-2024]. 2024.
- [24] NVIDIA. *RTX A6000 Data Sheet*. Rev. JAN21. 2021.
- [25] Temel Öncan. “A Survey of the Generalized Assignment Problem and Its Applications”. English. In: *INFOR* 45.3 (2007), pp. 123–141.
- [26] Xing Liu et al. “Collaborative edge computing with FPGA-based CNN accelerators for energy-efficient and time-aware face tracking system”. In: *IEEE*