An Ultralow-Power Real-Time Machine Learning Based fNIRS Motion Artifacts Detection

Renas Ercan, Yunjia Xia, *Graduate Student Member, IEEE*, Yunyi Zhao, Rui Loureiro, *Member, IEEE*, Shufan Yang, *Senior Member, IEEE*, and Hubin Zhao, *Member, IEEE*

***Abstract*— Due to iterative matrix multiplications or gradient computations, machine learning modules often require a large amount of processing power and memory. As a result, they are often not feasible for use in wearable devices, which have limited processing power and memory. In this study, we propose an ultralow-power and real-time machine learning-based motion artifact detection module for functional near-infrared spectroscopy (fNIRS) systems. We achieved a high classification accuracy of 97.42%, low field-programmable gate array (FPGA) resource utilization of 38 354 lookup tables and 6024 flip-flops, as well as low power consumption of 0.021 W in dynamic power. These results outperform conventional CPU support vector machine (SVM) methods and other state-of-the-art SVM implementations. This study has demonstrated that an FPGA-based fNIRS motion artifact classifier can be exploited while meeting low power and resource constraints, which are crucial in embedded hardware systems while keeping high classification accuracy.**

***Index Terms*— Field-programmable gate array (FPGA), functional near-infrared spectroscopy (fNIRS), low power, machine learning, motion artifact detection, real time, support vector machines (SVMs).**

I. Introduction

FUNCTIONAL near-infrared spectroscopy (fNIRS) is an emerging modality that aims to characterize cortical hemoglobin fluctuations through intensity measurements of diffusely scattered near-infrared light [1], [2]. It can help neuroscientists to determine which brain regions are activated during specific actions. However, preprocessing is essential for fNIRS data, which can be noisy. Due to the participant's motion, nonevoked systemic signal components in recorded fNIRS signals pose a challenge. This challenge is one of the main issues affecting fNIRS applications, as it results in motion artifacts, causing an erroneous detection of functional cortical activity [1].

Conventional motion detection is processed offline using benchtop computers, and these methods are based on peaks or shifts in time-series signals, including spline interpolation, wavelet filtering, and principal component analysis [3], [4], [5]. However, the performance of these methods largely depends on a set of assumptions to describe motion artifacts and the subjective selection of signals with associated tuning of parameters. Hence, the need for a method that eliminates the subjective fine-tuning of parameters and avoids relying on stringent assumptions becomes crucial. To date, the common machine learning method employed for automatically learning with the fine-tuning of parameters is based on a denoised autoencoder architecture which requires the use of high-power graphics processing units (GPUs), such as a Titian Xp GPU card [3]. Implementations using GPUs can achieve high classification accuracies, with [3] quoting a 100% success rate in removing motion artifacts. However, GPU card is not suitable for integration into wearable devices. Moreover, GPUs are not appropriate to power-constrained applications. In the execution of support vector machine (SVM) algorithms, field-programmable gate arrays (FPGAs) are quoted to consume over an order of magnitude less power as compared to GPUs. This makes FPGA feasible to carry out machine learning algorithms in low-power applications.

In this work, we deploy SVM as a machine learning method instead of neural network implements to consider the hardware constraints for standalone devices for fNIRS motion artifact detection. Efficient SVM hardware implementations can be achieved by considering various techniques and optimizations. One approach is to use reduced precision arithmetic, such as fixed-point or low-precision floating-point formats, to perform computations with lower energy consumption [6]. Alternatively, parallel processing units can speed up SVM computations and reduce power consumption or optimize memory access patterns and utilize ON-chip memory resources efficiently. This reduces data transfer and storage requirements, leading to reduced power consumption [7]. However, past attempts have several critical limitations. Several simplification methods were applied when reducing the hardware complexity,

Manuscript received 25 July 2023; revised 2 November 2023 and 26 December 2023; accepted 13 January 2024. This work was supported in part by the Department of Orthopaedics and Musculoskeletal Science; and in part by the Wellcome Trust and Engineering and Physical Sciences Research Council (EPSRC) through the WEISS Center, UCL under Grant 203145Z/16/Z. The work of Shufan Yang was supported in part by the SHED Project Royal Academy of Engineering under Grant IF2223-172 and in part by the Innovate U.K. KTP under Grant 013191. The work of Hubin Zhao was supported in part by the Royal Society Research under Grant RGS\R2\222333 and in part by the Engineering and Physical Sciences Research Council under Grant 13171178 R00287 and Grant EP/W000679/1. *(Renas Ercan and Yunjia Xia are co-first authors.) (Corresponding author: Hubin Zhao.)*

Renas Ercan was with UCL, WC1E 6BT London, U.K. He is now with the Department of Physics, University of Cambridge, CB2 1TN Cambridge, U.K. (e-mail: re378@cam.ac.uk).

Yunjia Xia, Yunyi Zhao, and Hubin Zhao are with HUB of Intelligent Neuro-Engineering (HUBIN), Division of Surgery and Interventional Science, UCL, WC1E 6BT London, U.K. (e-mail: yunjia.xia.18@ucl.ac.uk; yunyi.zhao.21@ucl.ac.uk; hubin.zhao@ucl.ac.uk).

Rui Loureiro is with IOMS, Division of Surgery and Interventional Science, UCL, WC1E 6BT London, U.K. (e-mail: r.loureiro@ucl.ac.uk).

Shufan Yang is with the Institute of Medical and Biological Engineering, School of Mechanical Engineering, University of Leeds, LS2 9JT Leeds, U.K., and also with UCL, WC1E 6BT London, U.K. (e-mail: s.f.yang@leeds.ac.uk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2024.3356161.

Digital Object Identifier 10.1109/TVLSI.2024.3356161

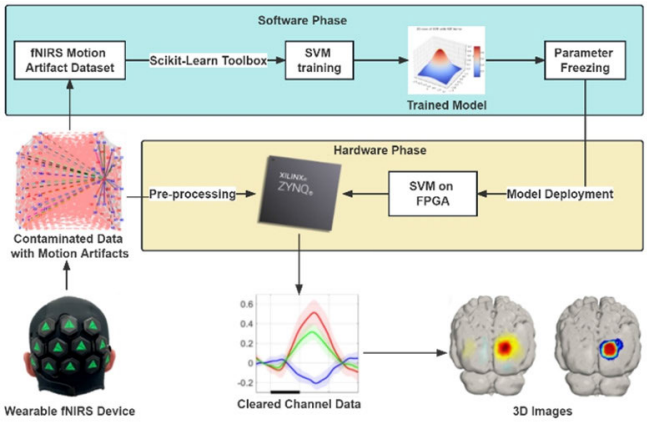

Fig. 1. Co-design workflow for machine learning deployment in fNIRS applications: integrating Scikit-Learn, MATLAB/Simulink, and Vivado for FPGA-based high-performance computing.

consequently sacrificing classification accuracy. Past architectures also lack flexibility and scalability and have exceptionally high power consumption usage; this is problematic due to the challenge of achieving a low-power hardware system [8]. A similar problem also occurs when leveraging FPGAs to achieve a high classification accuracy. Although FPGAs have flexible digital circuital design and extensive parallel computation capabilities, the power consumption is still very high [7]. Overall, past implementations of SVMs struggle to meet important constraints imposed by the FPGAs, such as high classification accuracy, real-time processing, minimal resource utilization, and low-power usage [8].

Therefore, the main contribution of this article is the development of an online machine learning-based motion detection high-level synthesis (HLS) Simulink model and subsequent generation of a model at register transfer level (RTL) level for online motion artifact detection for an fNIRS system with ultralow-power. Our machine learning module uses exponential approximation and overcomes the impact of accuracy degradation when reducing power consumption using a serial channeling method. This method overcomes resource constraints and introduces an online processing technique that can be miniaturized and seamlessly integrated as a standalone device. In addition, we conducted a systematic power comparison, which demonstrates the novelty of the proposed approach and leads to a practical design solution for a fast FPGA-based prototype. A system development flowchart of this study is shown in Fig. 1.

II. Digital Architecture Design and Simulation

In this section, we present a detailed design of the ultralow-power, real-time implementation for detecting motion artifacts in fNIRS. We utilized the Gaussian radial basis function (RBF) kernel, widely recognized as a powerful and popular choice for handling nonlinear data [9]. We employed a software and hardware co-development method. The preprocessing and motion detection integration stages were implemented using MATLAB and Simulink. Subsequent models were then validated through a set of datasets with an in-built Simulink testbench. The testbench enabled the calculation of all miss-classification errors, providing a reliable measurement of the classification rate for each architecture. By adopting a high-level simulation approach early in the design cycle, we facilitated the rapid prototyping of designs. Each solution can be evaluated for its speed, complexity, and accuracy, allowing for a thorough assessment of its performance characteristics.

A. fNIRS Dataset

Raw fNIRS data were obtained from a study wherein subjects wearing the fNIRS device were given tasks including "seated-texting" and "walking-texting." The data was passed through an fNIRS-specific data processing toolbox called Homer3 [10] and a function called hmrMotionArtefact to determine periods of motion artifact [10]. The purpose of finding these periods of motion artifacts was to train the SVM model using labeled data where the classifications have already been identified. Training datasets were created through the downsampling and balancing of a larger dataset with 99 087 instances and two features.

Balanced and unbalanced datasets were used when testing the architectural and HLS generated RTL SVM designs. The tests used balanced datasets so that many segments of data with motion artifacts could be tested. Unbalanced datasets were used to test the SVM model on signals that mimic a naturalistic scenario.

B. Training the SVM Model

The full development cycle of the proposed SVM motion artifact classifier starts by training a model offline in software [9]. The model was trained in Python, and cross validation was applied with a grid search to find the best cost parameter and kernel coefficient. An exhaustive search over various SVM parameters was conducted to fine-tune the model. The cross validation revealed that the best parameters for the model were $\Gamma = 1$ and regularization $\lambda = 10$. Here, $\Gamma = (1/\sigma)$, where σ is the variance and Γ represents how much impact one training point has on its surrounding data points. The regularization term λ was used to prevent overfitting. The support vectors were extracted upon constructing the finished model, of which 55 were generated, and the associated Lagrange multiplier coefficients and bias value were obtained.

C. Digital Architecture Design of Preprocessing

The RBF kernel in the SVM algorithm assumes that incoming data has been centered and scaled. Therefore, the incoming fNIRS signals need to be preprocessed for normalization purpose. This required each feature to have its mean value equal to zero and its standard deviation equal to one. To achieve real-time preprocessing, we calculated an exponentially weighted running mean and standard deviation. In the time-domain the exponentially weighted running mean is a statistic calculation that would consume a large number of FPGA hardware resources, hence the frequency-domain representation which is a first-order infinite impulse response (IIR) filter consisting of a real pole was applied. This is simple to implement in the time domain and uses far fewer hardware resources. We then

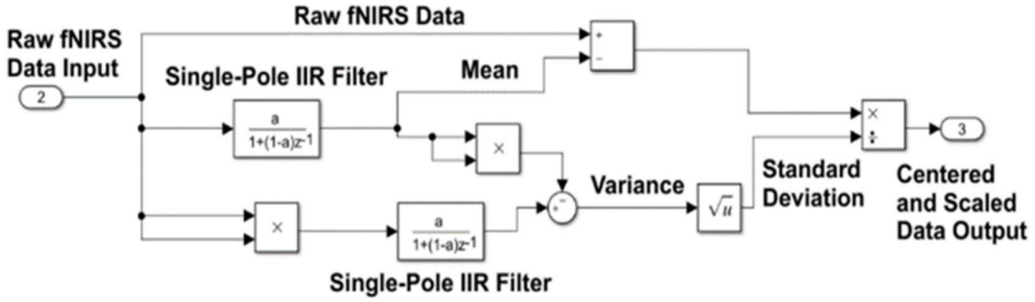

Fig. 2. Simulink-based preprocessing of fNIRS data: IIR filter design using single-pole z transform method.

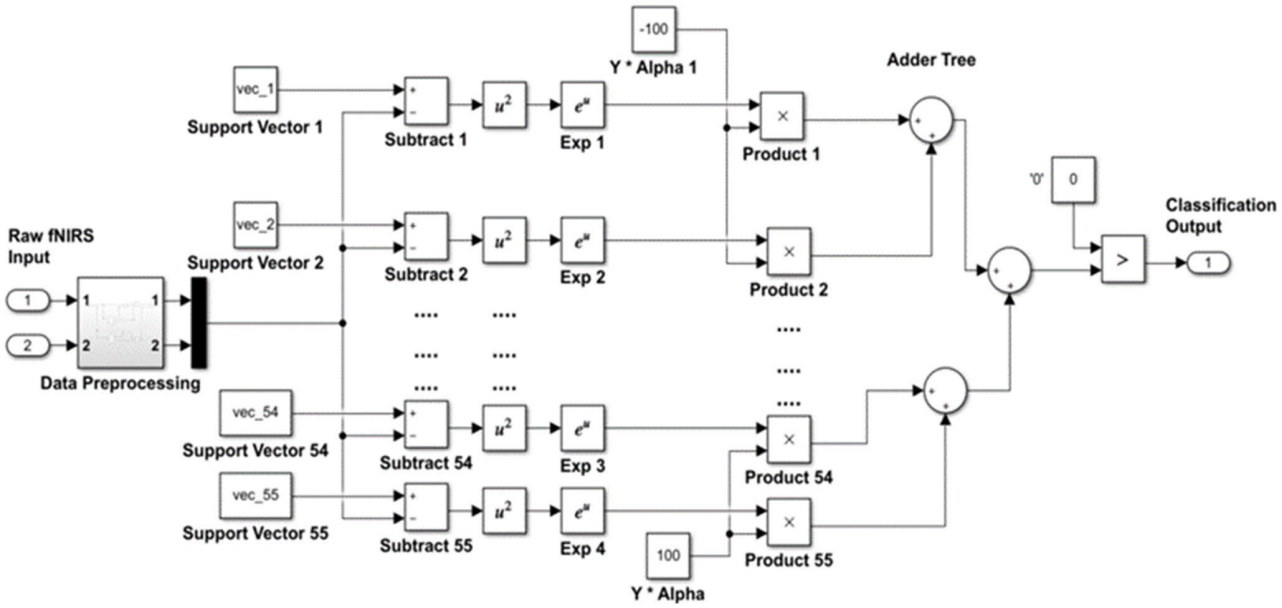

Fig. 3. Advanced Simulink architecture for fNIRS data: integrating SVM with kernel realization, inner product accumulation, and threshold comparison modules for motion artifact detection, yielding binary classification outputs to indicate the presence or absence of motion artifacts.

created an IIR filter circuit, following a z transform, we find the transfer function:

$$H(z) = \frac{a}{1-(1-a)z^{-1}}. \tag{1}$$

Herein, $0 < a < 1$ is a constant that determines the effective length of the running average. To go to the continuous domain, we make the substitution $z = e^{sT}$, where T is the sample time. After solving $1-(1-a)e^{-sT} = 0$, the continuous system has a pole at $s = (1/T)\log(1-a)$, where we set a as $1-\exp(((2\cdot\pi\cdot T)/\tau))$, τ is the averaging time constant. The best value of τ and subsequently a was found through a comprehensive brute force search that evaluates classification accuracy as a result; the final value taken forward was $a = 0.01$. Given that the transfer function (1) calculates the exponentially weighted running mean, the variance and the standard deviation can be efficiently computed. The Simulink architecture used to process a single feature of the input fNIRS signal is shown in Fig. 2.

D. Digital Architecture of SVM Inference

The underlying theory of the SVM architecture builds a streaming architecture model based on the functional decomposition of the SVM kernel [9]. The fundamental arithmetic operations of the Gaussian RBF kernel $K(\vec{x}i, \vec{x}) = e^{-(\|\vec{x}i,\vec{x}\|^2/\sigma)}$ were directly mapped to Simulink arithmetic blocks, where $\vec{x}i$ and $\vec{x}$ are two input test points and Gamma $\Gamma = (1/\sigma)$ is defined as a new variable as a metric for how much impact one training point has on its surrounding data points. The proposed SVM hardware design was segmented into three principal blocks: a kernel realization (A), inner product addition with an adder tree (B), and a threshold comparison (C) [11]. The support vector values and Lagrange multiplier coefficients were taken from the trained Python SVM model. Fig. 3 shows the data preprocessing block on the left-hand side feeding preprocessed fNIRS signal into the SVM algorithm architecture. The preprocessed fNIRS data were streamed into square difference units with 55 support vectors where the square difference between the fNIRS signal and the support vectors were calculated and then passed to exponential function units to achieve the RBF kernel function. The adder tree and multipliers construct the classification function as follows:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha i \cdot K\left(\vec{x}i, \vec{x}\right) + b\right). \tag{2}$$

The α term represents the Lagrange multipliers, and b is a bias parameter. The bias parameter was not included as it

TABLE I
OVERVIEW OF KEY HARDWARE RESOURCES ON THE TARGET FPGA FOR SYSTEM IMPLEMENTATION

FPGA Resource	Quantity Available
Logic Cells	154,000
Flip-Flops	141,000
Look-up-tables	71,000
Block RAM (BRAM)	240kB
Clock Management Units (CMTs)	3
18x25 MACCs (Multiply, Addition and Multiply- Accumulate Blocks)	360

was found to increase the usage of FPGA resources while offering no improvement in classification accuracy. In the last step, the classification results were forecast using the output of the adder tree and a relational operator compared to "0"—the classification that indicates the absence of a motion artifact.

The combined fNIRS data preprocessing and RBF kernel SVM algorithm architecture were designed and simulated within the Simulink environment. This work utilized MATLAB and Simulink's automatic HDL code generation to convert the digital system architecture to HDL code. HDL code creation methods generally produce Verilog code that is highly optimized and efficient whilst requiring minimal changes. These overall methods allow for a fast development time.

III. HLS GENERATED RTL DIGITAL DESIGN AND SIMULATION

The HLS generated RTL digital design was evaluated using Xilinx Vivado [12] and then generated bitstream to download into Xilinx Zynq Ultrascale + MPSoC ("sfvc784-1-i" family). The FPGA resources are shown in Table I [13].

Simulink tools generated arithmetic modules and captured the digital design in Verilog code. RTL was generated in the IEEE754 32-bit single-precision floating-point format. The overall architecture utilized a differential clock to run the RTL; this form of differential signaling employs two complementary clock signals to transmit one information signal [13]. This signaling system enhances noise resistance and enables reduced voltage fluctuations, leading to decreased power consumption in FPGAs. A unified software/hardware codesign method was then developed. The HLS generated RTL design was to replace critical blocks designed and tested in the Simulink architecture with synthesizable Verilog blocks to provide the same function with less resource requirement. The entire RTL design employs a streaming architecture where the output of a subsystem is fed directly to the input of the next subsystem. The streaming architecture enables a subsystem to initiate computation once sufficient data has been accumulated. This approach led to reduced latency, as we directly utilized the results from each subsystem without storing them in OFF-chip memory. The only source of latency is the interim time between starting the device setup and the initial feeding of the first model's layer, after which all computations proceed concurrently.

Data preprocessing of the fNIRS signals principally revolved around using a single real pole IIR filter. This was implemented in RTL by breaking the filter into core floating-point arithmetic operations. A multiplier using a "part multiplier, part add-shift" mantissa multiplication architecture was designed, which allows the filter's functions execution while preserving accuracy. This architecture revolved around splitting the 32-bit inputs into their sign, exponent, and mantissa, then performing simple assignments, binary bit switching, and shifting of the two inputs. Constants were fed into this particular multiplication module to create a gain, and a single input was given twice to create squaring operations. Addition and subtraction modules were designed using similar RTL architectures. The preprocessing data section was completed with standalone floating-point square root and division modules, which were instantiated to calculate the input data's normalization.

A principal component analysis of the fNIRS data was used to identify which input features contributed the most variation in the data and, thus, which features best captured the data's structure. This revealed only two features that were required. Each feature of these two input features has a preprocessing RTL channel following the architecture given in Fig. 2. The two preprocessing channels were operating in parallel, and for any dataset, with more features, these can be easily extended to include more channels.

The underlying principle of the SVM classifier architecture was to exploit the FPGA's parallel computational power and resources to execute the decision function (2) most efficiently; computation of this function involves highly parallelizable vector operations. Consequently, the RBF function was partitioned into small arithmetic blocks that form parallel Support Vector channels. The proposed FPGA architecture for the SVM classifier at RTL-level HDL design follows the digital Simulink architecture given in Fig. 3. The RBF kernel to FPGA architecture mapping allows each of the 55 support vector channels to run synergistically, achieving a parallelized classification system. Internal FPGA memory was employed solely for the support vectors and Lagrange multiplier coefficients. The raw fNIRS signal was streamed into the FPGA and fed into the data-preprocessing units. Upon preprocessing, we tackled the kernel calculation, the most fundamental part of the SVM algorithm's RTL, which would be the most resource and power-intensive part of the RTL design. The RTL model's compact size offered the advantage of accommodating all SVM parameters, including support vectors, within ON-chip memory. This eliminates the need for slower OFF-chip memory access, leading to improved overall efficiency.

The problem of the RBF kernel calculation in RTL mimics that seen in the digital architecture as it is distributed into smaller parallel arithmetic units that are executed in larger blocks as modules. These processing units employ the inherent parallelism of the FPGA to accelerate any computation of the decision function substantially. The parallel implementation of the RBF kernel with 32-bit single precision across all 55 support vector channels can provide fast processing speed. However, it simultaneously overused FPGA hardware resources. Given the target FPGA board hardware resource limitations, we considered utilizing the oversampled channel method in a serialized fashion.

TABLE II
COMPARISON OF KEY FPGA RESOURCE UTILIZATION AND CLASSIFICATION ACCURACY FOR FOUR SVM ALGORITHMS

	1) Floating-Point Model	2) Fixed-Point Model	3) New Exponential Function	4) Single Oversampled Channel
LUT as Logic	232,069	220,785	86,751	38,354
Total Registers	309,666	248,770	149,107	3,592
Total Block Memory Bits	187,816	172,612	93,935	6,024
Total DSP Blocks	342	342	342	12
Total Pins	70	70	70	70
Classification Accuracy	94.34%	50.10%	93.60%	97.42%

IV. IMPLEMENTATION

Several methods were attempted to reduce the RTL FPGA area usage. First, a common technique of applying a fixed-point numerical representation was trialed. However, the model uses an immense scale of numbers, and therefore, this approach severely reduced the classification accuracy due to a restriction of numbers representable by fixed-point notation. Any 32-bit fixed-point data with a value higher than $2^{31} - 1 = 2\,147\,483\,647$ cannot be processed and expressed precisely and often the internal RTL values did exceed this precision. In contrast, the 32-bit floating-point number can accurately represent values up to $\approx 3.4028235 \times 10^{38}$. Next, it was found that the SVM's most computationally demanding task was the kernel's exponential function. This function used different approximations taking advantage of its mathematical relationships. We employed a trigonometric calculation approach that utilized a Coordinate Rotation Digital Computer (CORDIC) for hardware-efficient trigonometric calculations, including a Taylor series approximation and experimented with a table-driven calculation module. The most efficient technique found for our design was to use the autogenerated HDL code that Simulink gives for the exponential function. This method, similar to the addition and subtraction modules, breaks the exponential function down into a long series of simplistic bit operations on the floating-point input's mantissa and exponent. To evaluate the effectiveness of various techniques employed to reduce FPGA resource utilization, we conducted an analysis of different SVM algorithms and their respective resource consumption. Table II summarizes the resource consumption and respective classification accuracy of four distinct SVM implementations: 1) an initial floating-point model; 2) a fixed-point model; 3) a new exponential function using an LUT stored in memory rather than a mathematical implementation of the exponential function; and 4) a single oversampled channel. This comparison highlights the development trajectory of the final SVM algorithm, where the fixed-point model compromised classification accuracy and the single oversampled channel gave a lower resource utilization compared to the new exponential function. Hence, algorithm 4) was opted for as it achieved the best resource-efficient FPGA implementation of the four implementations.

The final digital design that we adopted incorporates a resource-aware scheme, which translates the initial fully parallel design into a hybrid architecture that combines both parallel and serial processing. In order to optimize FPGA area utilization, we focused on the 55 support vector channels, running them in an oversampled channel, enabling the operations of multiple channels to a single hardware unit. We first converted the parallel support vectors and processed fNIRS signals into a singular stream of samples time-multiplexed onto a singular channel. Through this method, we optimized the hardware of the resource-costly kernel and inner product accumulation RTL. A singular subtraction, squaring, and exponential function RTL were written for the kernel. Lagrange coefficients were still stored in FPGA memory; however, their multiplication operation was included within the shared FPGA RTL. In the streaming design, the timing of each channel is critical, a synchronization between the serial and parallel sections of the design and channels is needed for accurate operation of the RTL. To share these resources without adding significant cycles of latency, the RTL of the singular shared channel was oversampled at 55 times at the base clock rate of the overall model. Consequently, the model only has one extra cycle of latency of the base rate. The RTL operated at the maximum power-optimized clock speed of 2.5 MHz. However, with this new architecture, the base rate was significantly reduced to 45.45 kHz. This adjustment accommodated the 55-fold increase in clock speed for the oversampling channel method avoiding this with only an additional latency of approximately 330 ns. Considering that neural activities function on a timescale of tens of seconds, coupled with the response activation times for measuring oxygenation and deoxygenation, an internal clock is necessary for serial data processing. The new architecture of the SVM kernel and inner product accumulation RTL with their singular channel is shown in Fig. 4.

The proposed streaming architecture alongside the partially parallel, partially serial model offers many advantages and is crucial for a low-power design desired by neuroimaging technologies. However, it demands meticulous design to avoid bottlenecks that could impair the entire system's performance. Fig. 5 shows the timing diagram of the architecture that ensures there is no such bottleneck whilst giving further explanation to the operation of the parallel-serial model. A pseudocode showing the general flow and logic of the HDL realization and subsequent HLS generated RTL design for the combined data preprocessing and SVM algorithm is shown in Fig. 6. The block diagram showing the FPGA module layout is shown in Fig. 7.

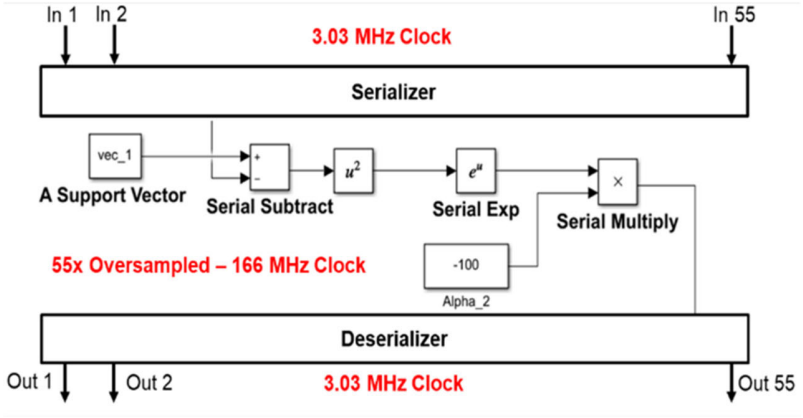

Fig. 4. Refined RTL architecture of the SVM for serial singular and oversampled channel. This block diagram details the serialization and deserialization processes in a single-channel implementation of the SVM, incorporating key components as depicted in Fig. 3.

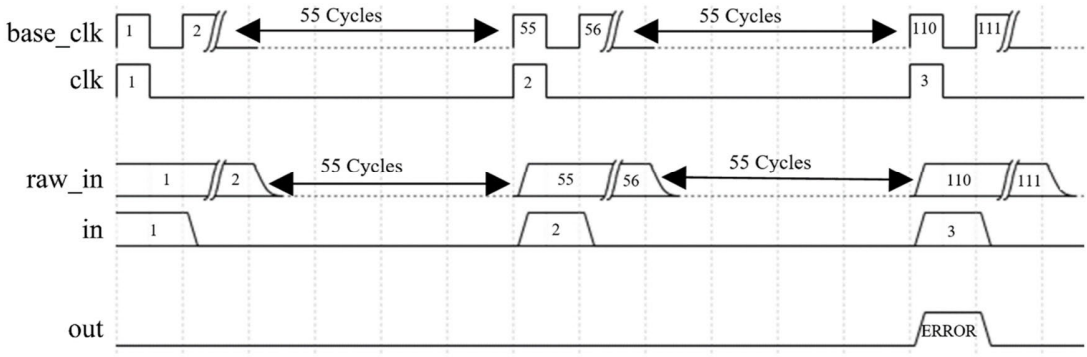

Fig. 5. Timing diagram of low-level design highlighting bottleneck avoidance. The diagram illustrates the parallel-serial model's operation, showing the base clock and raw input data along with their oversampled equivalents. It emphasizes the oversampled clock's speed, which is 55 times slower than the base clock, and includes both oversampled input data and an example of the resultant clean output signal, which signifies the detection of a motion artifact, labeled as an "error."

Algorithm 1 SVM Module

1: **Input**: Differential clock, reset and clock enable lines, raw NIRS inputs
2: **Output**: Clock enable, classification output
3: **Initialize**: Registers and wires for intermediate SVM variables
4: Assign Support Vector and Lagrange multipliers values to registers or wires
5: Instantiate floating-point arithmetic operations for the data pre-processing for each input feature of the raw NIRS data
6: Instantiate svm_tc module to define the oversampled clock and enable lines
7: Instantiate Subsystem.v to perform the single channel oversampling of the 55 Support Vector channels
8: Instantiate 55 addition modules to create the adder tree
9: Instantiate the final relational operator to compare the adder tree output to 0, assign the output to the SVM.v module output

Algorithm 2 svm_tc Submodule

1: **Input**: SVM.v clock, reset, enable
2: **Output**: Base enable, two oversampling enable lines
3: On high input clock enable, start a counter
4: If counter ≥ 54, reset to zero; else increment the counter
5: Define first output enable as high when the counter $= 54$, and second output enable as high when the counter $= 0$
6: Set output base enable equal to the input clock enable

Algorithm 3 Subsystem.v Submodule

1: **Input**: Base clock, reset, base enable, 55 Support Vectors, processed NIRS data
2: **Output**: 55 SVM kernel algorithm outputs
3: Instantiate counter to count to 55
4: **for** counter from 0 to 55 **do**
5: Assign support vector inputs to the first variable in the intermediate array
6: Assign NIRS input to the second variable of the array for each feature of the NIRS data
7: Create a delay line of length 55
8: Instantiate subtraction module, input NIRS data with the Support Vector input that matches the counter value
9: On each high of oversampled enable line, output subtraction result into delay line and shift previous outputs
10: Feed delay line output to input of the next module
11: Repeat steps for multiplication (squaring) and exponential function modules
12: Instantiate final multiplication module, exchange Support Vectors for Lagrange Multipliers and fNIRS input for exponential function delay line output
13: Feed output of multiplication into delay lines to deserialise into 55 parallel outputs
14: **end for**

Fig. 6. Pseudocode for HDL and RTL logic flow in fNIRS data preprocessing and SVM algorithm. This figure presents a structured overview of the process, divided into three algorithms: the top-level SVM module (Algorithm 1), a submodule for oversampling (Algorithm 2), and the submodule for executing SVM arithmetic operations (Algorithm 3).

V. Results and Discussion

After validating the functionality of the SVM design through behavioral software simulation, the next phase involves the translation of functional hardware description language (HDL) code to an operational FPGA, specifically, the Zynq system-on-chip (SoC). This transition typically occurs in several sequential stages, with the most crucial phases encompassing synthesis, place and route, and the generation of the programming file. All of these processes were executed within the Vivado tool.

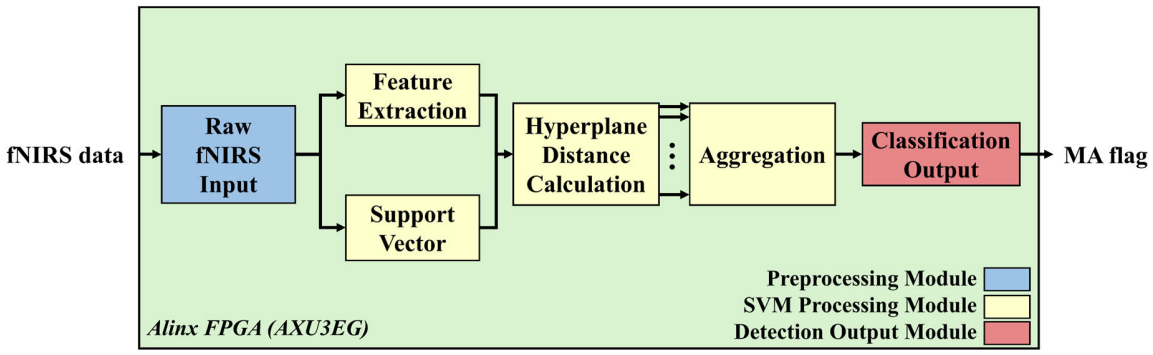

Fig. 7. Block diagram of the proposed FPGA system.

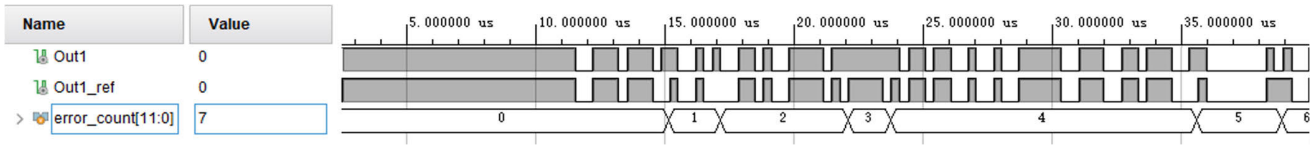

Fig. 8. RTL behavioral simulation outputs and classification accuracy evaluation. "Out1" indicates the detected motion artifact signal from SVM RTL, and its comparison with "Out1_ref" from traditional motion artifact detection software Homer3, demonstrating an accuracy rate of 97.42%.

In this context, the constraints file plays a pivotal role. It defined a 6-ns clock cycle with a 3-ns switching period and maps the register-transfer level (RTL) general-purpose input–output (GPIO) to FPGA universal asynchronous receiver–transmitter (UART) pins, which, in this configuration, are configured as peak-to-peak 3.3-V low-voltage complementary metal–oxide–semiconductor (LVCMOS) pins. The mapping, place and route, and static timing analysis were automated procedures within Vivado. These operations took place during the synthesis process as integral components of the implementation phase.

Following successful navigation through these stages, the place and route operation assessed the final resource utilization and generated a netlist tailored for the FPGA. Finally, the RTL design was exported to a bitstream for configuration within the FPGA hardware.

A. Results

The primary evaluation criterion employed for assessing the models was classification accuracy. Each of the four training datasets was applied to the models, and the quantification of motion artifacts, as determined by the digital architectural or RTL error counter (depending on the model's stage of development), was used to evaluate their performance. A low number of incorrectly identified motion artifacts indicate superior model performance, as shown in Table II. In addition, complementary metrics, such as FPGA resource allocation and power consumption, were utilized to evaluate the models, especially concerning the objectives of low-power operation and real-time capabilities, as demonstrated in Fig. 5. Nevertheless, given the medical application of the SVM models, classification accuracy was considered the paramount metric of importance.

The single-channel oversampled model illustrated in Fig. 4 was taken to the hardware implementation of the tested models. Table II illustrates the stark decrease in the FPGA resource utilization between the initial 1) and final 4) models, where the resource utilization includes the data preprocessing and kernel implementation circuitry.

Using the "LUT as a logic" metric as the most critical indicator of resource utilization, a 151.49% decrease in the area from 232 069 to 32 026 LUTs can be seen—due to the application of the resource-cutting methods described in the design methodology section. The investigation did not record the change in FPGA power consumption. We analyzed the classification accuracy further by looking at the output signals produced by the RTL behavioral simulations, as shown in Fig. 8. This primarily shows the "error_count" counter that was instantiated in the testbench. Here, "Out1" was the output motion artifact signal of the SVM RTL, where 1 (high) indicated the presence of a motion artifact and 0 (low) indicated the absence of a motion artifact. "Out1_ref" was an ideal real-world classification of the fNIRS input as labeled by the Homer3 software. Importantly, the overall data points for the testbench demonstrated a remarkably high accuracy rate of 97.42%. This level of accuracy underscores the reliability and effectiveness of the RTL simulations in classifying motion artifacts in fNIRS data, leveraging the robustness of the SVM algorithm.

Within resource utilization, the most valuable resource can vary between digital designs and project requirements. Based on the neuroimaging FPGA's real-time and low-power project objectives, the use of specific resources could be minimized. Utilization of memory logic was intentionally very low and was split relatively evenly across LUT RAM (1.20%), Flip-Flops (4.27%), and BRAM (1.85%) to achieve the hardware objectives. As LUT RAM and Flip-Flops used for memory are fast, and a value can be obtained immediately instead of waiting for the next clock edge. However, it would use more power than the BRAM, which has higher latency. A similar issue was found for the DSP blocks as they may allow an RTL design to employ the parallel architecture of an

TABLE III
FPGA RESOURCE UTILIZATION OF THE PROPOSED DESIGN

Resource	Utilization	Availability	Utilization%
LUT	38,354	70,560	54.36
LUTRAM	347	28,800	1.20
Flip-Flops	6024	141,120	4.27
BRAM	4	216	1.85
DSP	12	360	3.33
I/Os	70	252	27.78
BUFG(Global Clock Buffer)	4	196	2.04

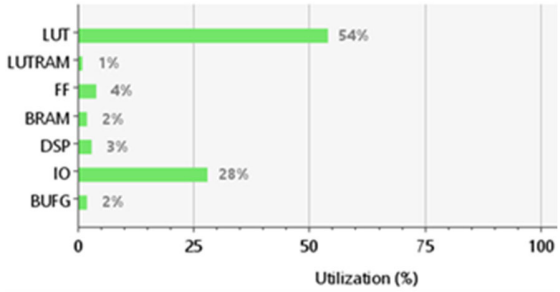

Fig. 9. Visual depiction of the key FPGA resource utilization of the proposed design.

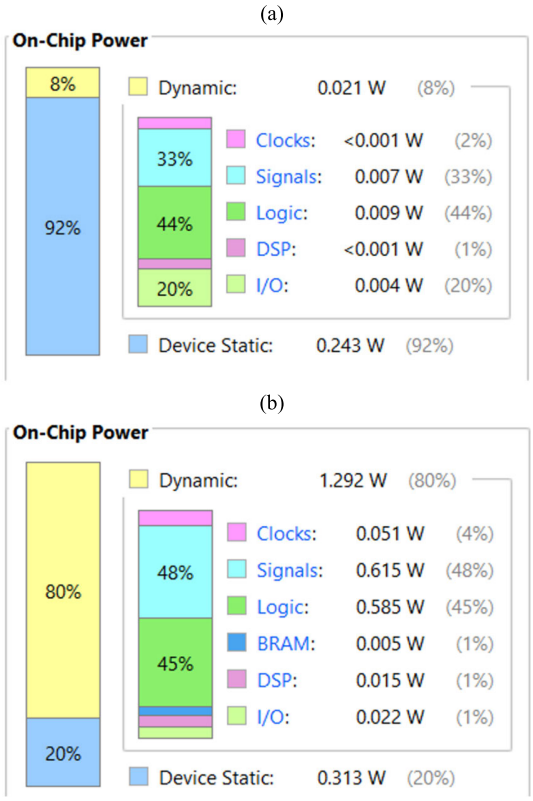

Fig. 10. Detailed representation of the SVM RTL power consumption, both dynamic and static. (a) Initial and final optimized power consumption at a clock frequency of 2.5 MHz. (b) Initial and optimized power consumption after the clock frequency was increased to 166.67 MHz.

FPGA better, hence meeting the real-time objective; however, this method utilized more power. Given the broadly serial nature of the FPGA design and its medical fNIRS application, we have opted to limit the utilization of DSP modules. These modules were specialized and complex components designed for intricate signal processing, and they often consume more resources and power and can increase both the RTL synthesis time and operating time of the FPGA. Hence, they have been limited to a utilization of 3.3%. This was decided based on the recognition that the inherent advantages of DSP modules would not be fully harnessed in this RTL model. Moreover, restricting DSP module usage allows for future enhancements and features to be added ensuring that the current design is scalable and flexible. The overall FPGA resource utilization is summarized in Table III and depicted visually in Fig. 9. It shows that only around 50% of resources were utilized.

The power consumption of the final HLS generated RTL model implementation was synthesized and reported using the Vivado software to provide benchmark tests of the design. The power simulation was run in a "worst-case" scenario to generate the highest estimated power consumption. This scenario includes a high ambient temperature of 40 °C, an airflow of 250 linear feet per minute, and a maximum process intensity.

Power is divided into two categories and governed by the sum of its static (fixed) and dynamic (variable) power consumptions. Static power originates from the FPGA technology silicon design and dynamic power is derived from the digital designs' distinctive utilization. Initial power consumption values wherein the clock frequency was set to 166.67 MHz gave a total power of 1.605 W, as seen in Fig. 10(b). Although such a power can be regarded as low and rivals that of similar devices seen in the literature. This research aimed to prioritize energy efficiency as it becomes vital in low-power applications of the fNIRS technology. A tradeoff between energy efficiency with throughput existed, so by lowering the baseline clocking frequency to 2.5 MHz, the power consumption was drastically reduced. Post the clock frequency reduction our targeted device consumed 0.264 W, of which 0.243 W (92%) was reported for the static power and 0.021 W (8%) for the dynamic power consumption, all dissipating through the PL side of the Zynq. The most power-hungry sections of the RTL design were the signals at 0.007 W (33%) and the logic, which consumed 0.009 W (44%) of the total dynamic power. This is expected because the core power rail consumes most of the power, which uses RTL signals to drive the logic, a central aspect of FPGA design. A more detailed breakdown of these power figures is shown in Fig. 10(b) and gives a clear picture of one of the more fundamental contributions of this work. The FPGA underwent simulation at two different clock frequencies 166.67 and 2.5 MHz. The final lower frequency of 2.5 MHz assumed typical conditions where the ambient temperature was 28.6 °C, while at 166.67 MHz, maximum power simulation settings were employed with an ambient temperature of 33.8 °C. These simulations yielded insights into how the design performs under different operational conditions. Our study uncovers insights that are often overlooked in fNIRS and FPGA-SVM artifact rectification approaches. In addition, the literature on fNIRS lacks substantial focus on real-time, low-power hardware implementations using machine learning algorithms. Hence, a comparison between the SVM digital

TABLE IV
COMPARISON OF DIFFERENT FPGA SVM IMPLEMENTATION

SVM Kernel	Power Use(W)	Number of SVs	Frequency/Processing Speed	Resource Utilization in LUTs	Classification Accuracy	Reference
Linear	1.686	61	56.60 µs	2870	97.92%	[14]
Linear	1.756	248	11.26 µs	2566	80.85%	[14]
Gaussian RBF	15	16	12.5 MHz / 80 ns	122,637	N/A	[15]
Linear Polynomial	4.9	122	70 MHz	35,532	80%	[16]
Linear Polynomial	3.2	254	84 MHz	31,854	84%	[17]
Multiclass Polynomial	2.021	192	146 MHz	461	N/A	[18]
Gaussian RBF	0.264	55	2.5 MHz	38,354	97.42%	This Design

design with other fNIRS-based approaches was not able to be conducted. Instead, the successes and limitations of the FPGA SVM design presented with other available hardware SVM models found in the literature were compared to the proposed design. One notable criticism of the FPGA SVM models in existing literature is the lack of comprehensive reporting on key metrics such as power consumption, resource utilization, and classification accuracy. Within the scope of this work, several papers failed to address one or more of these important factors. Six studies were selected with a thorough report on power consumption and resource utilization and were detailed in Table IV [14], [15], [16], [17], [18].

To validate the efficacy of the proposed module, extensive simulations have been designed and conducted, evaluating the accuracy, timing, power, and resource utilization of the algorithm within a controlled environment. These simulations laid a strong foundation for both the theoretical and practical aspects of the proposed work. It should be noted that the results were primarily a reflection of simulated performance evaluations. The validation of these results through a physical implementation on an FPGA board is a crucial next step and is planned as the future work.

B. Discussion

Taking advantage of the single-channel oversampled core, we saw vast hardware resource saving in the FPGA's computing resources while preserving its classification accuracy. However, a more detailed analysis of the partially serial architecture's effects supports the earlier theory that only a single cycle of latency will be added to preserve the real-time objective need for further investigation. Usually, an fNIRS signal that represents a task performed by a patient is produced over a 2–7-s window [1]. Thus, an added latency is unlikely to be detrimental to our real-time goal.

The second objective, which suffered at a higher cost, was the goal of creating a low-power hardware accelerator. Power is a fundamental cost directly linked to FPGA resource utilization. Hence, effort dedicated to the project in reducing resource utilization was also actively decreasing power consumption. The two objectives are linked as many transistors used in the configurable logic blocks (CLBs) that enact the logic of the RTL all require power to operate. The more CLBs, the greater the power consumption. Hence, the dynamic power is a product of each CLB depending on the number utilized and their individual use within the design. Consequently, more densely utilized FPGA designs will consume more power. Utilizing the spread of dynamic power shown in Fig. 10, we can hypothesize that the resources and power of this RTL design can be further reduced if the computational load of the digital circuit is taken off from the dynamic memories found in the "Logic" component of dynamic power and redistributed to the "BRAM" resources.

There are tens of seconds of delay in neural signals and measurement activations due to the oxygenation and deoxygenation of neural activities. The latency is a major concern when using fNIRS as brain–computer interface. Portability is a key point for a wearable device, and therefore, the device may operate on batteries. While FPGAs are typically not ideal for battery-powered devices, we aim for low power consumption to maximize battery life. The low power consumption, in the range of milliwatts as reported in this work, represents a significant contribution. It addresses the challenge of power consumption, which is particularly important when considering that the motion artifact classifier is just one subsystem within the larger neuroimaging device.

In the evaluation of various FPGA-based SVM designs from the literature, it is evident that the final RTL model presented here exhibits lower power consumption compared to other implementations. When examining studies that reported power consumption on the lower end of the spectrum, it was observed that these SVM designs all utilized a Xilinx Zynq board.

The RTL design presented here utilizes lower resource utilization in comparison to many models found in the literature. However, as seen in Table IV, there is room for further reduction in resource utilization without necessarily sacrificing classification accuracy. It is important to note that studies achieving high classification accuracy with low power and resource usage employed more expensive FPGAs that offered larger and more sophisticated CLBs and faster clocks.

Finally, to our knowledge, the hardware-embedded system using an FPGA-based SVM classifier of motion artifacts is considered the first in the literature for fNIRS technology. In addition, the implementation presented here effectively overcame the challenges previously listed in the literature of satisfying FPGA low power and area restrictions while providing effective classification accuracies.

Prior research reports have demonstrated a propensity for superficial application of offline and software-based techniques

would yield an improvement in the accuracy of fNIRS signal processing [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29]. However, it has been noted that attempts to implement real-time hardware-based approaches, incorporating additional hardware components such as acceleration sensors, often lack comprehensive discussions regarding hardware and power constraints. These metrics are of utmost importance in the realm of hardware design, particularly in the context of the technology's practical application in the field of medical science.

Moreover, an examination of previous endeavors revealed instances where fNIRS data, including motion artifacts employed for model training and testing, had been solely generated through simulation, rather than being derived from real-world scenarios [21]. This practice is notably detrimental, given the intricate, variable, and challenging nature of accurately simulating motion artifacts.

VI. Conclusion and Future Work

A. Conclusion

Numerous methodologies have been proposed to address the challenge of motion artifacts in fNIRS signals. This research marks a pioneering effort in introducing a novel solution—an FPGA-based machine learning platform—specifically designed and tested for the fNIRS modality, with a primary focus on achieving ultralow power consumption. Our approach involves a seamless integration of software and hardware, offering a practical and efficient means to incorporate machine learning algorithms into FPGA hardware for fNIRS applications. At the model training stage, we initiated the process with high-level model training using Python. Subsequently, we transitioned to high-level MATLAB and Simulink architectural designs. After a diverse set of architectures underwent rapid prototyping with rigorous performance evaluations, assessing classification accuracy and FPGA resource usage, the architecture that emerged as the "best-performing" option involved data preprocessing through single real-pole IIR filters, followed by the execution of the SVM RBF kernel in a singular oversampled channel. The post-synthesis hardware system demonstrated high performance, achieving this without compromising the core objectives of maintaining a low area footprint, minimal power consumption, and low latency. Our results demonstrated that we overcome the persistent challenge of motion artifacts in fNIRS signals by introducing an innovative FPGA-based machine learning platform. This platform represents a significant step toward enhancing the utility and practicality of fNIRS technology in various domains, including neuroscience and clinical applications. Further exploration and development of this platform holds the potential to revolutionize fNIRS data processing and analysis.

B. Future Work

A critical evaluation considering the literature assessed in this study demonstrates that advancements in research must be made before we could implement a real-time, low-power neuroimaging motion artifact detector. The first would be to evaluate segments of the RTL design to look for possible power consumption-saving improvements. This would serve the study's real-time and low-power objectives. A potential method would be to evaluate FPGA design methods that link and harness the PL and PS sides of the board simultaneously. The second area of required future work would be to complete the hardware integration. This would involve functional RTL simulations, implementing the bitstream onto the FPGA and testing the SVM model on the FPGA in real life. Nevertheless, the quality of the simulation results validated this investigation's utility and established its potential use in a real experimental scenario. Moreover, this research has achieved innovations to modernize and drive research trends for wearable fNIRS and FPGA-based machine learning implementations in the right direction, which could have wider implications for neuroscience and clinical applications.

References

[1] D. Perpetuini, D. Cardone, C. Filippini, A. M. Chiarelli, and A. Merla, "A motion artifact correction procedure for fNIRS signals based on wavelet transform and infrared thermography video tracking," *Sensors*, vol. 21, no. 15, p. 5117, Jul. 2021, doi: 10.3390/s21155117.

[2] R. J. Cooper et al., "A systematic comparison of motion artifact correction techniques for functional near-infrared spectroscopy," *Frontiers Neurosci.*, vol. 6, p. 147, Oct. 2012, Art. no. 041406, doi: 10.3389/fnins.2012.00147.

[3] Y. Gao et al., "Deep learning-based motion artifact removal in functional near-infrared spectroscopy," *Neurophotonics*, vol. 9, no. 4, Apr. 2022, Art. no. 041406, doi: 10.1117/1.nph.9.4.041406.

[4] F. C. Robertson, T. S. Douglas, and E. M. Meintjes, "Motion artifact removal for functional near infrared spectroscopy: A comparison of methods," *IEEE Trans. Biomed. Eng.*, vol. 57, no. 6, pp. 1377–1387, Jun. 2010, doi: 10.1109/TBME.2009.2038667.

[5] L. Gagnon, M. A. Yucel, D. A. Boas, and R. J. Cooper, "Further improvement in reducing superficial contamination in NIRS using double short separation measurements," *NeuroImage*, vol. 85, pp. 127–135, Jan. 2014, doi: 10.1016/j.neuroimage.2013.01.073.

[6] B. Lesser, M. Mücke, and W. N. Gansterer, "Effects of reduced precision on floating-point SVM classification accuracy," *Proc. Comput. Sci.*, vol. 4, pp. 508–517, Jan. 2011, doi: 10.1016/j.procs.2011.04.053.

[7] P. Garcia, D. Bhowmik, R. Stewart, G. Michaelson, and A. Wallace, "Optimized memory allocation and power minimization for FPGA-based image processing," *J. Imag.*, vol. 5, no. 1, p. 7, Jan. 2019, doi: 10.3390/jimaging5010007.

[8] S. Afifi, H. GholamHosseini, and R. Sinha, "FPGA implementations of SVM classifiers: A review," *Social Netw. Comput. Sci.*, vol. 1, no. 3, p. 133, May 2020, doi: 10.1007/s42979-020-00128-9.

[9] B. Scholkopf et al., "Comparing support vector machines with Gaussian kernels to radial basis function classifiers," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2758–2765, Jun. 1997, doi: 10.1109/78.650102.

[10] T. J. Huppert, S. G. Diamond, M. A. Franceschini, and D. A. Boas, "HomER: A review of time-series analysis methods for near-infrared spectroscopy of the brain," *Appl. Opt.*, vol. 48, no. 10, p. 280, Apr. 2009, doi: 10.1364/ao.48.00d280.

[11] X. Song, H. Wang, and L. Wang, "FPGA implementation of a support vector machine based classification system and its potential application in smart grid," in *Proc. 11th Int. Conf. Inf. Technol., New Generat.*, Apr. 2014, pp. 397–402, doi: 10.1109/ITNG.2014.45.

[12] *Vivado ML Overview*. Accessed: Jul. 10, 2023. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[13] *ALINX AXU3EG or AXU3EGB: Xilinx Zynq UltraScale+ MPSOC ZU3EG Ethernet FPGA Development Board*. Accessed: Jun. 14, 2023. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-1cm64x4.html

[14] S. Afifi, H. GholamHosseini, and R. Sinha, "A system on chip for melanoma detection using FPGA-based SVM classifier," *Microprocess. Microsyst.*, vol. 65, pp. 57–68, Mar. 2019, doi: 10.1016/j.micpro.2018.12.005.

[15] M. Pietron, M. Wielgosz, D. Zurek, E. Jamro, and K. Wiatr, "Comparison of GPU and FPGA implementation of SVM algorithm for fast image segmentation," in *Proc. Int. Conf. Architecture Comput. Syst.*, 2013, pp. 292–302, doi: 10.1007/978-3-642-36424-2_25.

[16] C. Kyrkou, C.-S. Bouganis, T. Theocharides, and M. M. Polycarpou, "Embedded hardware-efficient real-time classification with cascade support vector machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 1, pp. 99–112, Jan. 2016, doi: 10.1109/TNNLS.2015.2428738.

[17] C. Kyrkou, T. Theocharides, and C.-S. Bouganis, "An embedded hardware-efficient architecture for real-time cascade support vector machine classification," in *Proc. Int. Conf. Embedded Comput. Syst., Architectures, Model., Simul. (SAMOS)*, Jul. 2013, pp. 129–136, doi: 10.1109/SAMOS.2013.6621115.

[18] R. A. Patil, G. Gupta, V. Sahula, and A. S. Mandal, "Power aware hardware prototyping of multiclass SVM classifier through reconfiguration," in *Proc. 25th Int. Conf. VLSI Design*, Jan. 2012, pp. 62–67, doi: 10.1109/VLSID.2012.47.

[19] Y. Zhang, D. H. Brooks, M. A. Franceschini, and D. A. Boas, "Eigenvector-based spatial filtering for reduction of physiological interference in diffuse optical imaging," *J. Biomed. Opt.*, vol. 10, no. 1, 2005, Art. no. 011014, doi: 10.1117/1.1852552.

[20] M. A. Yücel, J. Selb, R. J. Cooper, and D. A. Boas, "Targeted principle component analysis: A new motion artifact correction approach for near-infrared spectroscopy," *J. Innov. Opt. Health Sci.*, vol. 7, no. 2, Mar. 2014, Art. no. 1350066, doi: 10.1142/s1793545813500661.

[21] F. Scholkmann, S. Spichtig, T. Muehlemann, and M. Wolf, "How to detect and reduce movement artifacts in near-infrared imaging using moving standard deviation and spline interpolation," *Physiological Meas.*, vol. 31, no. 5, pp. 649–662, May 2010, doi: 10.1088/0967-3334/31/5/004.

[22] B. Molavi and G. A. Dumont, "Wavelet-based motion artifact removal for functional near-infrared spectroscopy," *Physiological Meas.*, vol. 33, no. 2, pp. 259–270, Feb. 2012, doi: 10.1088/0967-3334/33/2/259.

[23] M. R. Siddiquee, J. S. Marquez, R. Atri, R. Ramon, R. Perry Mayrand, and O. Bai, "Movement artefact removal from NIRS signal using multi-channel IMU data," *Biomed. Eng. OnLine*, vol. 17, no. 1, p. 120, Dec. 2018, doi: 10.1186/s12938-018-0554-9.

[24] H. Zhao et al., "A wide field-of-view, modular, high-density diffuse optical tomography system for minimally constrained three-dimensional functional neuroimaging," *Biomed. Opt. Exp.*, vol. 11, no. 8, p. 4110, Aug. 2020, doi: 10.1364/boe.394914.

[25] A. Blasi, D. Phillips, S. Lloyd-Fox, P. H. Koh, and C. E. Elwell, "Automatic detection of motion artifacts in infant functional optical topography studies," in *Oxygen Transport to Tissue XXXI* (Advances in Experimental Medicine and Biology), vol 662. Boston, MA, USA: Springer, 2010, pp. 279–284, doi: 10.1007/978-1-4419-1241-1_40.

[26] J. Virtanen, T. Noponen, K. Kotilahti, J. Virtanen, and R. J. Ilmoniemi, "Accelerometer-based method for correcting signal baseline changes caused by motion artifacts in medical near-infrared spectroscopy," *J. Biomed. Opt.*, vol. 16, no. 8, 2011, Art. no. 087005, doi: 10.1117/1.3606576.

[27] A. Metz, M. Wolf, P. Achermann, and F. Scholkmann, "A new approach for automatic removal of movement artifacts in near-infrared spectroscopy time series by means of acceleration data," *Algorithms*, vol. 8, no. 4, pp. 1052–1075, Nov. 2015, doi: 10.3390/a8041052.

[28] X. Cui, J. M. Baker, N. Liu, and A. L. Reiss, "Sensitivity of fNIRS measurement to head motion: An applied use of smartphones in the lab," *J. Neurosci. Methods*, vol. 245, pp. 37–43, Apr. 2015, doi: 10.1016/j.jneumeth.2015.02.006.

[29] M. R. Siddiquee et al., "Sensor fusion in human cyber sensor system for motion artifact removal from NIRS signal," in *Proc. 12th Int. Conf. Human Syst. Interact. (HSI)*, Jun. 2019, pp. 192–196, doi: 10.1109/HSI47298.2019.8942617.