Simulation of an Open-Hardware RISC-V Coprocessor for Low-Power Medical Artificial Intelligence

Cecilia Hernández Serrano, Luna Nirel Jiménez Montiel

Instituto Tecnológico de Pachuca, Hidalgo, México

***Corresponding Author**

DOI: https://doi.org/10.47772/IJRISS.2026.102500040

Received: 29 October 2026; Accepted: 03 November 2026; Published: 24 November 2026

ABSTRACT

The objective of this work was to evaluate, through simulation, whether a simple open-hardware coprocessor coupled to a RISC-V core can run medical artificial intelligence on low-power devices. The EdgeMAC coprocessor was designed; it adds three custom instructions: bias loading, four parallel 8-bit multiply-accumulate operations, and requantization. An instruction-level simulator was developed with a cycle model based on an open microcontroller core and a first-order energy model. A three-layer neural network that classifies breast tumors as benign or malignant from a public dataset was trained, quantized to 8-bit integers, and executed on three configurations: a core without a multiplier, a core with a 32-bit multiplier, and the core with the coprocessor. The quantized model kept an accuracy of 95.6% and a sensitivity of 97.6% on 114 test cases, identical to the floating-point model. The coprocessor reduced the cost of each diagnosis from 17,496 to 2,187 clock cycles, 8.0 times fewer than the core with a multiplier and 102.8 times fewer than the core without a multiplier, and the estimated dynamic energy, a lower bound, fell from 85.3 to 13.1 nanojoules. It is concluded that a minimal instruction-set extension is enough to bring assisted diagnosis to battery-powered portable devices and that, once arithmetic is accelerated, instruction and data access become the next limit. The originality lies in a reproducible methodology, built only with free tools, that enables students and institutions with limited resources to explore hardware-software co-design for socially oriented artificial intelligence.

Keywords: coprocessor, RISC-V, embedded machine learning, quantization, medical artificial intelligence, low power

INTRODUCTION

Artificial intelligence (AI) applied to medicine already achieves, in narrowly defined classification tasks, performance comparable to that of human specialists (Topol, 2019). However, most of these models run on servers or in the cloud, which requires permanent connectivity, moves sensitive clinical data away from where they are generated, and entails infrastructure costs that many rural clinics and underserved communities cannot afford. Edge computing, that is, processing data on the device itself or very close to it, has been proposed as a response to these latency, bandwidth, and privacy problems (Shi et al., 2016). Its most extreme form, known as TinyML, aims to run machine learning models on battery-powered microcontrollers that consume only a few milliwatts (Warden & Situnayake, 2019).

This need is pressing because access to medical technology remains uneven. The World Health Organization (2010) describes a persistent mismatch between the medical devices on the market and the needs of low-resource settings, and frames access in terms of availability, accessibility, appropriateness, and affordability. Likewise, Land et al. (2019) point out that the lack of quality diagnostics is still a major contributor to the health burden in resource-limited settings, and propose that point-of-care tests be affordable, user-friendly, rapid, equipment-free, and connected in real time (the REASSURED criteria). Portable devices with on-board intelligence are one way to meet these requirements, provided that their processors are cheap and efficient enough.

The main obstacle is efficiency. A neural network consists almost entirely of multiply-accumulate (MAC) operations, and a general-purpose processor spends far more energy fetching, decoding, and coordinating each instruction than on the arithmetic operation itself (Horowitz, 2014; Sze et al., 2017). For this reason, computer architecture is moving toward domain-specific architectures, which add specialized units to the processor for the type of computation that dominates an application (Hennessy & Patterson, 2019). Until recently, such specialization was reserved for companies able to license proprietary architectures and manufacture custom chips.

The RISC-V instruction set architecture (ISA) changes this scenario. It is an open, royalty-free standard that reserves encoding space for custom instructions, so any group can extend an existing core without asking for permission or paying licenses (Asanović & Patterson, 2014; Waterman & Asanović, 2019). On this basis, researchers have proposed ultra-low-power cores with digital signal processing extensions (Gautschi et al., 2017), optimized libraries for quantized neural networks on clusters of RISC-V cores (Garofalo et al., 2020), and open-source frameworks for designing custom function units on programmable logic devices (Prakash et al., 2023). On commercial processors, libraries such as CMSIS-NN show that 8-bit integer arithmetic is key to running neural networks on microcontrollers (Lai et al., 2018), in line with quantization schemes that enable integer-only inference (Jacob et al., 2018).

These works, however, usually require development boards, synthesis tools, or specialized toolchains, and most of them target vision or audio workloads. There is a niche for studies that quantify, with accessible tools, how much a minimal instruction-set extension contributes in a concrete medical application, and that can serve as teaching material in engineering programs with limited resources. There is also a humanistic dimension: UNESCO's Recommendation on the Ethics of Artificial Intelligence (UNESCO, 2022) stresses that the benefits of AI should reach all people equitably, which in practice requires inexpensive, open technologies that can operate without costly infrastructure.

The general objective of this work is to evaluate, through simulation, the performance and energy efficiency of an open-hardware coprocessor coupled to a RISC-V core for running a medical classifier on low-power devices. The specific objectives are: (a) to design a minimal coprocessor, named EdgeMAC, that accelerates 8-bit MAC operations through custom instructions; (b) to build a reproducible simulator with documented cycle and energy models; (c) to train and quantize a breast tumor classifier and verify that quantization does not degrade the diagnosis; and (d) to compare the core with the coprocessor against two reference configurations without acceleration.

The methodology is quantitative and experimental: the same C program was compiled for three processor configurations and executed in an instruction-level simulator on 114 test cases. The paper is organized as follows: Section 2 describes the model, the coprocessor, the simulator, and the procedure; Section 3 presents the results; Section 4 discusses them in light of the literature and sets out the limitations; and Section 5 summarizes the conclusions.

METHODOLOGY

Research Design

A quantitative, experimental, and comparative design based on simulation was adopted. The independent variable was the processor configuration, with three levels: (A) an RV32I core, which lacks multiplication instructions and must emulate them in software; (B) an RV32IM core, which includes a 32-bit multiplier; and (C) an RV32IM core with the EdgeMAC coprocessor. The dependent variables were the number of executed instructions, clock cycles per inference, latency, estimated dynamic energy, program size, and the diagnostic quality of the model. The entire process was implemented with free tools: Python and scikit-learn for machine learning (Pedregosa et al., 2011), the Clang 18 compiler to generate RISC-V code, and a custom simulator written in Python.

Application Case and Artificial Intelligence Model

Breast tumor diagnosis from fine-needle aspiration biopsies was chosen as the medical case. The public Wisconsin Diagnostic Breast Cancer dataset was used; it contains 569 cases (357 benign and 212 malignant) described by 30 numerical features of the cell nucleus, such as radius, texture, perimeter, and concavity, obtained through image analysis (Street et al., 1993; Wolberg et al., 1994). The data were split in a stratified manner into 455 training cases and 114 test cases (42 malignant and 72 benign) with a fixed random seed, and they were standardized with the mean and standard deviation of the training set.

A multilayer perceptron with a 30–32–16–1 architecture was trained, with rectified linear unit (ReLU) activation in the hidden layers, L2 regularization with a coefficient of 0.001, and a maximum of 2,000 iterations. The output is a single value whose sign indicates the class: positive for benign and negative for malignant. The model has 1,601 parameters and requires 1,552 MAC operations per inference, including the two padding inputs added so that the input vector has 32 elements, a multiple of four.

To run it with integer arithmetic, symmetric per-tensor quantization was applied following the scheme of Jacob et al. (2018): weights and activations as signed 8-bit integers, biases as 32-bit integers, and scales computed from the 99.9th percentile of the training activations. After each layer, the 32-bit accumulator is converted back to 8 bits through an integer multiplier M and a shift s, according to (1):

$$y = sat_L\left((acc \cdot M + 2^{s-1}) \gg s\right) \qquad (1)$$

where acc is the accumulator, $s = 24$, the term 2^{s-1} performs rounding, and the function sat_L saturates the result to the interval $[L, 127]$, with $L = 0$ in the hidden layers (which implements ReLU) and $L = -128$ in the output layer.

Design of the EdgeMAC Coprocessor

EdgeMAC is a functional unit tightly coupled to the core, in the style of the custom function units of Prakash et al. (2023). It is connected through the *custom-0* opcode, which the RISC-V specification reserves for non-standard extensions (Waterman & Asanović, 2019), so the existing instructions and the compiler toolchain remain unchanged. Its structure is shown in Fig. 1: four signed 8×8-bit multipliers, an adder tree, an internal 32-bit accumulator, and a requantization unit that implements (1).

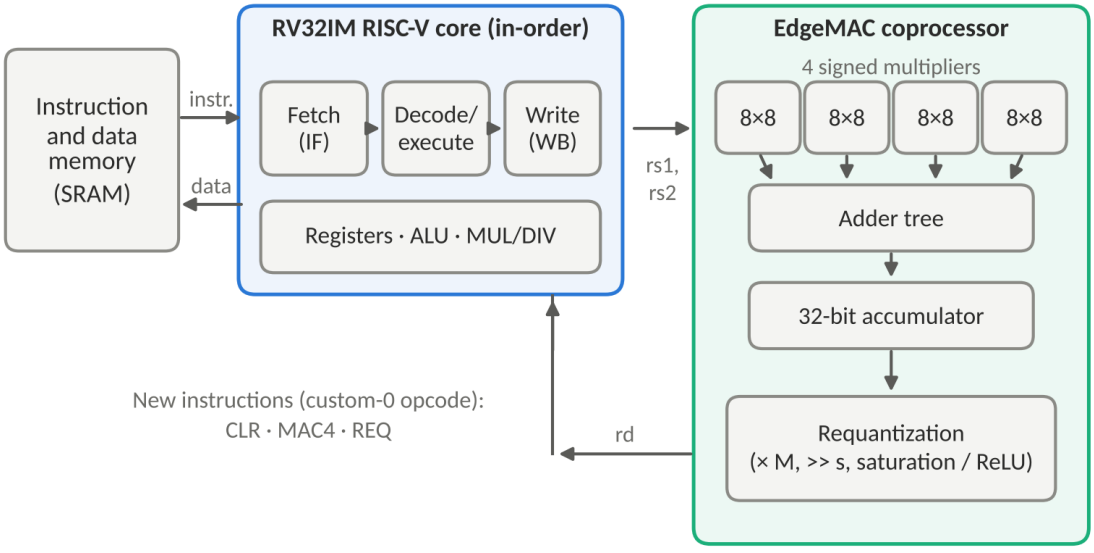

Figure 1. RISC-V core with the EdgeMAC coprocessor and the data path between them (rs1, rs2 → EdgeMAC → rd)

Source: own elaboration.

Fig. 1 also shows how data move between the core and the coprocessor. EdgeMAC has no register file or memory port of its own. The core fetches and decodes every instruction; when the decoder finds the custom-0 opcode, it reads the source registers rs1 and rs2 from its register file and forwards their 32-bit values, together with the funct3 and funct7 fields that select the operation, to EdgeMAC through the same operand path used by the arithmetic logic unit (ALU). The coprocessor completes the operation in the execute stage and returns a 32-bit result, which the core writes into the destination register rd in the write-back stage, exactly as it would for an ALU result. The only state inside EdgeMAC is the 32-bit accumulator, which persists from one instruction to the next: CLR initializes it with the bias, each MAC4 adds four products to it, and REQ reads it, requantizes it, and returns the resulting 8-bit activation through rd. Activations and weights reach the coprocessor only through registers: the core loads them from memory with ordinary 32-bit load instructions (lw), each of which brings four packed 8-bit values. For a neuron with n inputs, the sequence is therefore one CLR with the bias, n/4 repetitions of two lw instructions followed by one MAC4, one REQ, and one byte store (sb) of the result. In a multitasking system, the accumulator would have to be saved and restored on context switches; this is unnecessary in the bare-metal programs used here.

The coprocessor adds the three instructions listed in Table 1. The most important one, MAC4, receives four packed 8-bit values in each of its two source registers (four activations and four weights) and updates the accumulator according to (2):

$$acc \leftarrow acc + \sum_{k=0}^{3} \quad a_k \cdot w_k \qquad \textbf{(2)}$$

so that a single instruction replaces the four multiplications, four additions, and byte extractions that the base core must perform with separate instructions.

Table 1. EdgeMAC Coprocessor Instructions

Instruction	funct7	funct3	Operation	Latency (cycles)
CLR	0	0	acc ← rs1 (loads the neuron bias)	1
MAC4	1	0	acc ← acc + Σ $a_k \cdot w_k$ with four 8-bit pairs packed in rs1 and rs2; rd ← acc	1
REQ	2	0 / 1	rd ← requantization of acc according to (1) with M = rs1 and s = rs2; saturation to [−128, 127] or, with funct3 = 1, to [0, 127] (ReLU)	2

Source: own elaboration. Latencies are design assumptions.

With these instructions, computing each neuron reduces to one CLR instruction, n/4 MAC4 instructions (where n is the number of inputs), and one REQ instruction. As a first-order estimate, the area of an array multiplier grows with the square of its operand width, so four 8×8-bit multipliers amount to about one quarter of a 32×32-bit multiplier such as the one already included in configuration B.

Simulator and Cycle Model

An instruction-level simulator was developed that decodes and executes the complete RV32IM instruction set plus the three EdgeMAC instructions, starting from the binary image produced by the compiler. The programs were written in bare-metal C and compiled with Clang 18 at optimization level -O2 for the rv32i and rv32im architectures; in configuration C, the compute kernel of each dense layer invokes the custom instructions through inline assembly. All three configurations share the same model source code, the same weights, and the same sequence of operations, so the measured differences are due only to the available hardware.

The cycle cost of each instruction class was taken from the manual of the open-source CV32E40P core (OpenHW Group, n.d.), a descendant of the ultra-low-power core described by Gautschi et al. (2017), assuming zero-wait-state memories. A one-cycle penalty was also modeled when an instruction uses the result of the immediately preceding load. In configuration A, which lacks a multiplier, the compiler replaces each product with a shift-and-add software routine whose cost depends on the operand values. The parameters are summarized in Table 2.

Energy Model

A first-order dynamic energy model was adopted, using the per-operation values reported by Horowitz (2014) for a 45 nm technology at 0.9 V. The energy per inference was estimated with (3):

$$E = N_{inst}E_{fetch} + N_{mem}E_{mem} + \sum_i \quad N_iE_i + C \cdot E_{cycle} \qquad \textbf{(3)}$$

where N_{inst} is the number of executed instructions and E_{fetch} = 5 pJ is the energy of fetching each 32-bit instruction from an 8 kB memory; N_{mem} is the number of data loads and stores, with E_{mem} = 5 pJ; N_i and E_i are the count and energy of each type of arithmetic operation in Table 2; C is the number of cycles; and E_{cycle} represents the per-cycle energy of the clock, the pipeline registers, and static leakage. Because this last term depends strongly on the physical implementation, it was set to 0 pJ in the main analysis. Consequently, every energy value reported with E_{cycle} = 0 is a lower bound: a physical implementation would consume more, since clock distribution, pipeline registers, and leakage are always present. To show how much higher the values could be, a sensitivity analysis was performed with 2, 5, and 10 pJ per cycle. This choice does not favor the coprocessor, because the configuration that needs the fewest cycles is the one that gains the most as E_{cycle} grows.

Table 2. Cycle and Energy Model Parameters

Operation class	Cycles	Operation energy (pJ)
Fetch of each instruction	—	5.0
Integer arithmetic and logic	1	0.1
Load and store	1 (+1 on immediate dependency)	0.1 + 5.0
Branch not taken / taken	1 / 3	0.1
Jump (JAL, JALR)	2	0.1
Multiply MUL / MULH	1 / 5	3.1
EdgeMAC: CLR	1	0.1
EdgeMAC: MAC4	1	1.2 (4 × 0.2 + 4 × 0.1)
EdgeMAC: REQ	2	3.2

Source: own elaboration with data from OpenHW Group (n.d.) for cycles and Horowitz (2014) for energy.

Experimental procedure

The experiment followed six steps. First, the floating-point model was trained, and accuracy, sensitivity (proportion of malignant tumors detected), specificity, precision, and F1 score were measured on the test set. Second, the model was quantized and a Python reference implementation with exact integer arithmetic was obtained. Third, the three RISC-V programs were generated. Fourth, the 114 test cases were simulated in each

configuration, and the output of each case was verified to match the reference bit for bit. Fifth, the instructions, cycles, and instruction mix per inference were recorded, and the latency at a 50 MHz clock and the energy with (3) were computed. Sixth, to study scaling, the simulation was repeated with an isolated dense layer ranging from 16×16 to 256×256 neurons, with random integer weights and an input vector written into memory at run time, so that the compiler could not exploit known input values; the compiled code was also inspected to determine whether the compiler had fully unrolled the inner loop. Speedup was defined according to (4):

$$S = \frac{C_{ref}}{C_{EdgeMAC}} \quad \textbf{(4)}$$

where C_{ref} and C_{EdgeMAC} are the cycles per inference of the reference configuration and of the configuration with the coprocessor, respectively.

RESULTS

Diagnostic Quality of the Model

Table 3 shows the classifier metrics on the 114 test cases. The model quantized to 8 bits obtained exactly the same values as the floating-point model: 95.6% accuracy and 97.6% sensitivity, with only one malignant tumor classified as benign and four benign tumors classified as malignant. The predictions of both models matched in 100% of the cases, while the memory needed for the parameters fell from 6,148 to 1,748 bytes, a 3.5-fold reduction.

Table 3. Classifier Performance on the Test Set

Metric	Floating point (32-bit)	Integer (8-bit)
Accuracy	95.6%	95.6%
Sensitivity (malignant tumor)	97.6%	97.6%
Specificity	94.4%	94.4%
Precision	91.1%	91.1%
F1 score	94.3%	94.3%
Malignant detected / missed	41 / 1	41 / 1
Benign correct / false alarms	68 / 4	68 / 4
Parameter memory	6,148 B	1,748 B

Source: own elaboration with data from Street et al. (1993).

Table 4 presents the confusion matrix, which was identical for both models. The single false negative, a malignant tumor classified as benign, corresponds to a false-negative rate of 2.4% (1 of 42 malignant cases), while the four false positives, benign tumors flagged as malignant, correspond to a false-positive rate of 5.6% (4 of 72 benign cases). In a screening context, these two errors do not weigh the same: a false positive leads to additional tests, whereas a false negative can delay treatment.

Table 4. Confusion matrix on the test set (identical for the 32-bit and 8-bit models)

Actual class	Predicted malignant	Predicted benign	Total
Malignant	41 (true positives)	1 (false negative)	42
Benign	4 (false positives)	68 (true negatives)	72
Total	45	69	114

Source: own elaboration with data from Street et al. (1993). The malignant class is treated as positive.

Computational performance by configuration

In all three configurations, the 114 inferences produced outputs bit-identical to the reference implementation. Table 5 and Fig. 2 summarize the computational cost. The core with EdgeMAC required 2,187 cycles per inference, compared with 17,496 for the core with a multiplier and an average of 224,746 for the core without a multiplier, which corresponds to speedups of 8.0 and 102.8, respectively. At a 50 MHz clock, these figures equal 43.7 µs, 349.9 µs, and 4.49 ms per diagnosis.

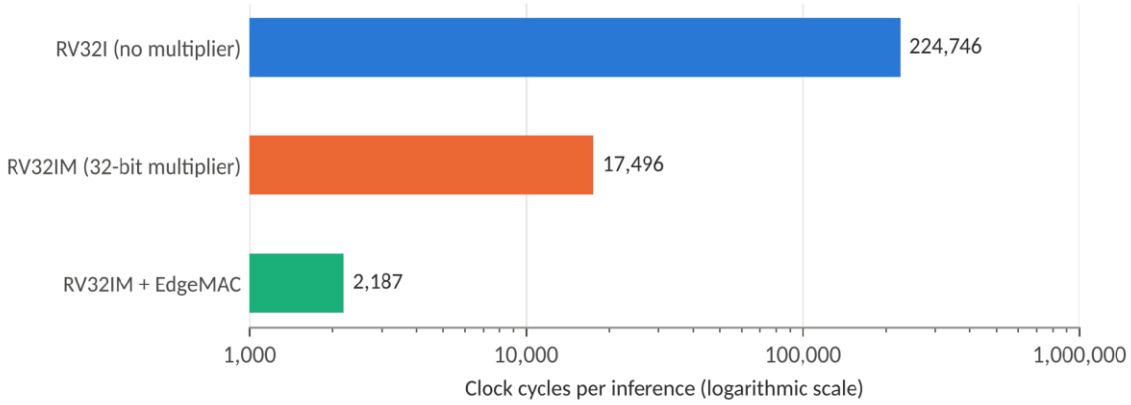

Figure 2. Clock cycles per inference in each configuration (mean of the 114 test cases; logarithmic scale)

Source: own elaboration.

The number of executed instructions fell from 12,582 to 1,612 (7.8 times) when moving from configuration B to configuration C. In configuration C, the inference consisted of 388 MAC4 instructions, 49 CLR instructions, 49 REQ instructions, and 820 memory loads, and 432 load-dependency penalty cycles were recorded. The execution time of configuration A ranged from 101,855 to 331,873 cycles depending on the input values, whereas that of configuration C was constant in all cases.

Table 5. Cost of one inference in each configuration

Metric	RV32I	RV32IM	RV32IM + EdgeMAC
Executed instructions	177,461	12,582	1,612
Cycles (mean)	224,746	17,496	2,187
Cycles (minimum – maximum)	101,855 – 331,873	17,495 – 17,501	2,187 – 2,187
Cycles per MAC operation	144.8	11.3	1.4

Latency at 50 MHz	4,494.9 µs	349.9 µs	43.7 µs
EdgeMAC speedup (S)	102.8	8.0	1.0
Dynamic energy, lower bound (E_{cycle} = 0)	921.2 nJ	85.3 nJ	13.1 nJ
Energy per MAC operation	593.6 pJ	55.0 pJ	8.5 pJ
Binary image including weights	2,984 B	2,820 B	2,384 B
Outputs identical to reference	114 / 114	114 / 114	114 / 114

Source: own elaboration.

Estimated energy

Using the model in (3) with E_{cycle} = 0, the lower bound of the dynamic energy per inference was 921.2 nJ in configuration A, 85.3 nJ in configuration B, and 13.1 nJ in configuration C, that is, 6.5 times less with EdgeMAC than with the conventional multiplier. Fig. 3 and Table 6 show how this energy is distributed among instruction fetch, data access, and arithmetic operations. In configuration A, instruction fetch accounted for 96.3% of the total, data access for 1.8%, and arithmetic for 1.9%. In configuration B, the shares were 73.7%, 18.7%, and 7.6%, and in configuration C they were 61.3%, 33.1%, and 5.6%, respectively. Thus, as the configurations become more efficient, the share of instruction fetch falls and that of data access grows.

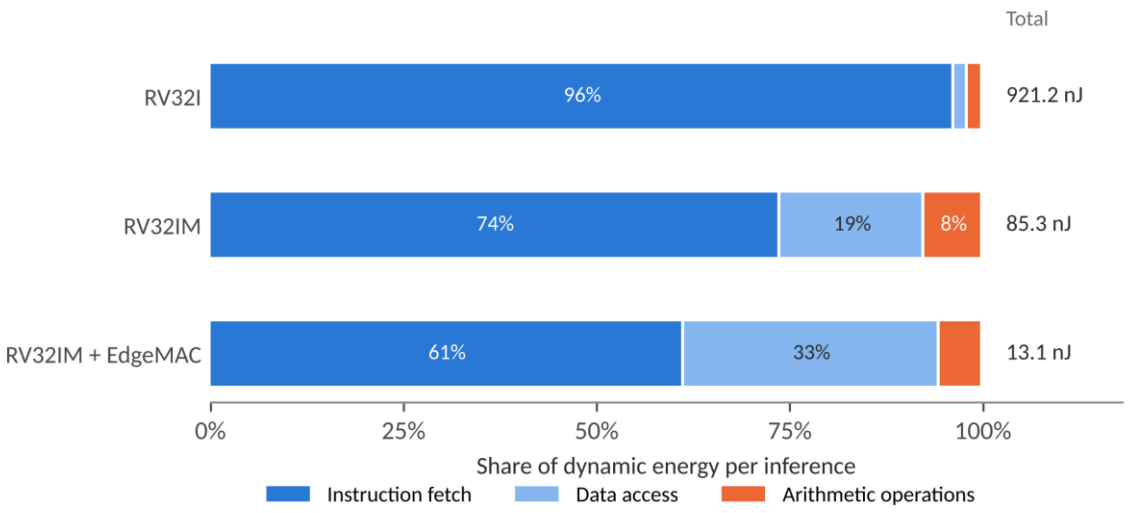

Figure 3. Share of the dynamic energy per inference spent on instruction fetch, data access, and arithmetic operations (lower bound, E_{cycle} = 0)

Source: own elaboration with the energy model in (3).

Table 6. Breakdown of the dynamic energy per inference (lower bound, E_{cycle} = 0)

Component	RV32I	RV32IM	RV32IM + EdgeMAC
Instruction fetch	887.3 nJ (96.3%)	62.9 nJ (73.7%)	8.1 nJ (61.3%)
Data access	16.1 nJ (1.8%)	15.9 nJ (18.7%)	4.3 nJ (33.1%)
Arithmetic operations	17.7 nJ (1.9%)	6.4 nJ (7.6%)	0.7 nJ (5.6%)

Total	921.2 nJ	85.3 nJ	13.1 nJ

Source: own elaboration with the energy model in (3).

The sensitivity analysis in Table 7 indicates that, when a per-cycle cost between 2 and 10 pJ is included, the energy reduction of EdgeMAC relative to configuration B increased from 6.5 to 7.4 times, and relative to configuration A it rose from 70.1 to 90.5 times.

Table 7. Sensitivity of energy per inference to the per-cycle cost

E_{cycle} (pJ)	RV32I (nJ)	RV32IM (nJ)	EdgeMAC (nJ)	Reduction vs. RV32IM	Reduction vs. RV32I
0	921.2	85.3	13.1	6.5	70.1
2	1,370.7	120.3	17.5	6.9	78.2
5	2,044.9	172.8	24.1	7.2	84.9
10	3,168.7	260.3	35.0	7.4	90.5

Source: own elaboration.

Scaling with layer size

Fig. 4 and Table 8 present the cost per MAC operation of an isolated dense layer. In all cases, the outputs matched the reference implementation. The core without a multiplier required between 169 and 204 cycles per MAC, and the core with a multiplier between 4.8 and 11.0. With EdgeMAC, the cost was 1.82, 1.39, and 1.19 cycles per MAC in the 16-, 32-, and 64-neuron layers, and it rose to 2.38 and 2.31 in the 128- and 256-neuron layers. The speedup relative to the core with a multiplier was 2.7 in the smallest layer, peaked at 8.8 in the 64-neuron layer, and stabilized at about 4.4 in the larger layers. The inspection of the compiled code showed that the compiler fully unrolled the inner loop of the RV32IM kernel only in the 16-neuron layer, and that of the EdgeMAC kernel up to the 64-neuron layer; this information is included in Table 8.

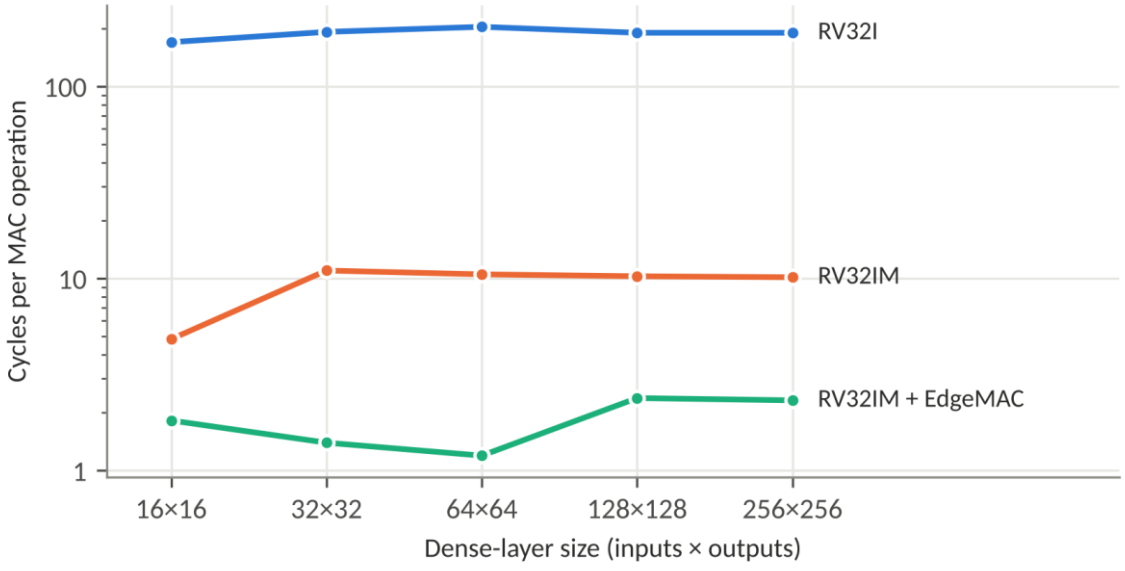

Figure 4. Cycles per MAC operation of an isolated dense layer as a function of its size (logarithmic scales)

Source: own elaboration.

Table 8. Scaling with dense-layer size

Layer	Cycles/MAC RV32I	Cycles/MAC RV32IM	Cycles/MAC EdgeMAC	Inner loop fully unrolled (RV32IM / EdgeMAC)	Speedup vs. RV32IM	Energy reduction vs. RV32IM
16 × 16	169.5	4.8	1.82	Yes / Yes	2.7	3.1
32 × 32	191.6	11.0	1.39	No / Yes	7.9	6.4
64 × 64	203.7	10.5	1.19	No / Yes	8.8	6.8
128 × 128	189.6	10.2	2.38	No / No	4.3	4.5
256 × 256	189.6	10.1	2.31	No / No	4.4	4.6

Source: own elaboration. Unrolling was determined by inspecting the compiled code; energy reduction computed with (3) and $E_{cycle} = 0$.

DISCUSSION

The first relevant result is that 8-bit quantization did not change any diagnosis. This agrees with what Jacob et al. (2018) reported for much larger models and confirms that, in tabular problems of this size, numerical precision is not an obstacle to deploying the model on a microcontroller. The 97.6% sensitivity is especially important in the clinical context, because the costliest error is missing a malignant tumor. In addition, reducing memory to 1,748 bytes allows the model to fit comfortably in the internal memory of the smallest microcontrollers.

Because a false negative is the costliest error, the decision rule deserves attention. The classifier labels a case as benign when its output exceeds zero, but this threshold can be moved toward the benign side so that doubtful cases are flagged as malignant. In the test set, raising the threshold until the only missed tumor was detected increased the false positives from 4 to 19 of the 72 benign cases. This trade-off is illustrative only, since the threshold was explored on the test set itself; in practice it should be fixed on independent validation data together with clinicians. In any case, changing it costs nothing in hardware, because it is a single comparison of the final output.

The 8.0-fold speedup over a core that already has a multiplier, and the 102.8-fold speedup over one that lacks it, are of the same order as those reported by works that combine instruction-set extensions with software optimization: Garofalo et al. (2020) report improvements of up to 63 times over a baseline RISC-V core by exploiting signal processing extensions and eight-core parallelism, and Prakash et al. (2023) achieve speedups of 55 to 75 times with custom function units on programmable devices. The difference is that here the gain is obtained with a single core and only three instructions, which suggests that a substantial part of the benefit is within reach of very modest designs.

A qualitative comparison with TinyML software on commercial microcontrollers helps to situate this gain. On Arm Cortex-M processors, the CMSIS-NN library expands 8-bit data to 16 bits with a sign-extension instruction and then uses the dual 16-bit multiply-accumulate instruction SMLAD, so each arithmetic instruction performs two MAC operations; with these kernels, Lai et al. (2018) reported improvements of 4.6 times in throughput and 4.9 times in energy efficiency over baseline functions for a convolutional network. TensorFlow Lite Micro, in turn, runs models through a portable interpreter that allows optimized kernels to be used on each platform (David et al., 2021). EdgeMAC follows the same principle of packed operations on 8-bit data but goes one step further: it operates directly on 8-bit values without a sign-extension step, performs four MAC operations per instruction instead of two, and fuses requantization and ReLU into a single instruction. The figures are not directly comparable, because the baselines and workloads differ (a convolutional network measured on hardware versus a small dense network in simulation), but they indicate that the gain of EdgeMAC is of the same order as, and somewhat above, what optimized software achieves on commercial cores with digital signal processing extensions.

The energy breakdown in Fig. 3 illustrates the argument of Horowitz (2014): in a programmable processor, energy is spent managing instructions rather than on arithmetic. EdgeMAC saves energy mainly because it executes fewer instructions, not because its multipliers are more efficient. For the same reason, once arithmetic is accelerated, instruction fetch and data reads become the dominant cost. The scaling in Fig. 4 confirms this: in layers of 128 neurons or more, the compiler stops unrolling the inner loop, and each iteration executes six instructions (two loads, the MAC4 instruction, two address increments, and a branch) for only four MAC operations. This is a typical case of Amdahl's law, and it points to the next natural improvements: hardware loops, post-increment addressing, and direct memory access by the coprocessor, techniques already incorporated in the cores of Gautschi et al. (2017). The smallest layer shows the opposite effect: there, the baseline loop is fully unrolled, while the fixed cost of EdgeMAC per neuron (CLR, REQ, and the store of the result) is spread over only four MAC4 instructions, so the speedup falls to 2.7. In the classifier, whose layers have 32, 32, and 16 inputs, the compiler fully unrolled the inner loops, which explains why its cost per MAC (1.4 cycles) is close to the best case of the scaling study.

In absolute terms, 13.1 nJ and 43.7 µs per diagnosis are very small figures compared with the consumption of a sensor or a wireless radio. The practical value of the coprocessor therefore lies not only in direct energy savings but in what it makes possible: processing data on the device itself without relying on connectivity, keeping patient information where it is generated, leaving the processor idle most of the time, and running larger models within the same energy budget. Moreover, the execution time with EdgeMAC was constant, whereas that of the core without a multiplier varied by more than a factor of three depending on the data. A predictable response time makes it easier to guarantee real-time operation, a desirable property in medical equipment.

From a humanistic perspective, this work shows that hardware specialization is no longer exclusive to large corporations. With an open instruction set, a free compiler, and a simulator written in a widely used language, a student can design, program, and evaluate a processor extension in a few days at no licensing cost. This has value for engineering education, because it makes it possible to teach hardware-software co-design through a case with social impact, and for the technological sovereignty of countries that depend on proprietary architectures, in line with the call of UNESCO (2022) to distribute the benefits of AI equitably.

This study has limitations that should be considered. The results come from simulation rather than from a real circuit: the coprocessor was not described in a hardware description language or synthesized, so its area and maximum frequency are estimates. The energy model is first-order, is based on a 45 nm technology, and omits the clock, leakage, and peripherals. The dataset is small, and its features are obtained from images processed off the device, so the evaluated system covers only the classification stage.

In addition, the cycle model assumes memories without wait states, which is realistic for tightly coupled SRAM at low frequency but not for flash memories or systems with caches. A first-order estimate of this effect is obtained by adding W_i cycles to every instruction fetch and W_d cycles to every data access, according to (5):

$$C(W_i, W_d) = C_0 + W_i \cdot N_{inst} + W_d \cdot N_{mem} \qquad \textbf{(5)}$$

where C_0 is the cycle count with ideal memory and N_{inst} and N_{mem} are as defined in (3). The estimate ignores any overlap between stalls, so it is pessimistic. Table 9 shows that the speedup of EdgeMAC over RV32IM drops from 8.0 to 6.8 with one wait state on data accesses and to 6.1 with two, because loads make up a larger share of the accelerated program, but it remains above 6 in every case considered; with one wait state on both instruction and data accesses (no cache), it is 7.1. An instruction and data cache would bring the behavior close to the ideal case when the loops and weights fit in it, as would be expected for a program of less than 2.5 kB such as this one, but cache misses would add latency that this model does not capture.

Table 9. First-order effect of memory wait states on the cost of one inference, according to (5)

Wait states (W_i / W_d)	RV32I (cycles)	RV32IM (cycles)	EdgeMAC (cycles)	Speedup vs. RV32IM	Speedup vs. RV32I	EdgeMAC latency at 50 MHz
0 / 0 (ideal)	224,746	17,496	2,187	8.0	102.8	43.7 µs

0 / 1	227,975	20,686	3,057	6.8	74.6	61.1 µs
0 / 2	231,204	23,876	3,927	6.1	58.9	78.5 µs
1 / 1	405,436	33,268	4,669	7.1	86.8	93.4 µs
2 / 2	586,126	49,041	7,151	6.9	82.0	143.0 µs

Source: own elaboration with the instruction counts obtained in the simulation.

Finally, any clinical use would require validation with independent data and medical supervision, since such a system should be understood as diagnostic support and not as a substitute for the specialist. As future work, we propose describing EdgeMAC in Verilog, implementing it on a programmable logic device to measure actual power consumption, and extending the evaluation to other low-power workloads, such as cardiac signal analysis or crop disease detection from agricultural sensors.

CONCLUSIONS

This work demonstrates, through reproducible simulation, that a minimal open-hardware coprocessor is enough to efficiently run a medical classifier on a low-power RISC-V core. With only three custom instructions, EdgeMAC reduces the cost of each diagnosis by 8.0 times compared with a core with a multiplier and by more than 100 times compared with one without a multiplier, and it lowers the estimated dynamic energy by 6.5 to 7.4 times, without changing the predictions of the model. A first-order analysis of memory wait states indicates that the advantage over the core with a multiplier remains above six times when memory is slower than ideal.

Quantization to 8-bit integers fully preserves the diagnostic quality of the breast tumor classifier, with 95.6% accuracy, 97.6% sensitivity, a single false negative, and four false positives, and it reduces parameter memory by 3.5 times, which makes it an enabling condition for medical AI on small devices. In turn, the energy and scaling analysis shows that, once arithmetic is accelerated, the next limit lies in instruction fetch and data movement, which guides the next design improvements.

Finally, the proposed methodology, based solely on free tools, is an accessible resource for engineering education and for research at institutions with limited resources. It brings hardware specialization closer to those who develop technology to solve concrete social problems, such as access to diagnosis in communities without medical infrastructure or connectivity.

REFERENCES

1. Asanović, K., & Patterson, D. A. (2014). *Instruction sets should be free: The case for RISC-V* (Technical Report No. UCB/EECS-2014-146). University of California, Berkeley.
2. David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T., Warden, P., & Rhodes, R. (2021). TensorFlow Lite Micro: Embedded machine learning for TinyML systems. *Proceedings of Machine Learning and Systems, 3*. https://proceedings.mlsys.org/paper_files/paper/2021/hash/6c44dc73014d66ba49b28d483a8f8b0d-Abstract.html
3. Garofalo, A., Rusci, M., Conti, F., Rossi, D., & Benini, L. (2020). PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors. *Philosophical Transactions of the Royal Society A, 378*(2164), 20190155. https://doi.org/10.1098/rsta.2019.0155
4. Gautschi, M., Schiavone, P. D., Traber, A., Loi, I., Pullini, A., Rossi, D., Flamand, E., Gürkaynak, F. K., & Benini, L. (2017). Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25*(10), 2700–2713. https://doi.org/10.1109/TVLSI.2017.2654506
5. Hennessy, J. L., & Patterson, D. A. (2019). A new golden age for computer architecture. *Communications of the ACM, 62*(2), 48–60. https://doi.org/10.1145/3282307

6. Horowitz, M. (2014). Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers* (pp. 10–14). IEEE.
7. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2704–2713). https://doi.org/10.1109/CVPR.2018.00286
8. Lai, L., Suda, N., & Chandra, V. (2018). *CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1801.06601
9. Land, K. J., Boeras, D. I., Chen, X.-S., Ramsay, A. R., & Peeling, R. W. (2019). REASSURED diagnostics to inform disease control strategies, strengthen health systems and improve patient outcomes. *Nature Microbiology, 4*(1), 46–54. https://doi.org/10.1038/s41564-018-0295-3
10. OpenHW Group. (n.d.). *CV32E40P user manual: Pipeline details*. Retrieved October 28, 2026, from https://cv32e40p.readthedocs.io/en/latest/pipeline.html
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.
12. Prakash, S., Callahan, T., Bushagour, J., Banbury, C., Green, A. V., Warden, P., Ansell, T., & Janapa Reddi, V. (2023). CFU Playground: Full-stack open-source framework for tiny machine learning (TinyML) acceleration on FPGAs. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software* (pp. 157–167). IEEE. https://doi.org/10.1109/ISPASS57527.2023.00024
13. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal, 3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198
14. Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. In *Biomedical Image Processing and Biomedical Visualization* (Proceedings of SPIE, Vol. 1905, pp. 861–870). https://doi.org/10.1117/12.148698
15. Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE, 105*(12), 2295–2329. https://doi.org/10.1109/JPROC.2017.2761740
16. Topol, E. J. (2019). High-performance medicine: The convergence of human and artificial intelligence. *Nature Medicine, 25*(1), 44–56. https://doi.org/10.1038/s41591-018-0300-7
17. UNESCO. (2022). *Recommendation on the ethics of artificial intelligence*. United Nations Educational, Scientific and Cultural Organization.
18. Warden, P., & Situnayake, D. (2019). *TinyML: Machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*. O'Reilly Media.
19. Waterman, A., & Asanović, K. (Eds.). (2019). *The RISC-V instruction set manual, Volume I: Unprivileged ISA* (Document Version 20191213). RISC-V Foundation.
20. Wolberg, W. H., Street, W. N., & Mangasarian, O. L. (1994). Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer Letters, 77*(2), 163–171. https://doi.org/10.1016/0304-3835(94)90099-X
21. World Health Organization. (2010). *Medical devices: Managing the mismatch: An outcome of the Priority Medical Devices project*. World Health Organization.