Open Source Real-Time Automatic Modulation Classification with Deep Learning for Internet of Things Devices

Simon Boka
Tickle College of Engineering
University of Tennessee
Knoxville, U.S.A
email: sboka@vols.utk.edu

Oladayo Bello
College of Engineering | New Mexico State University
Cape Town, South Africa | Las Cruces, U.S.A
email: oladayo@ieee.org

Innocent Davidson
Cape Peninsula University of Technology
Cape Town, South Africa
email: davidsoni@cput.ac.za

Abstract— **Deep Learning (DL) has redefined Automatic Modulation Classification (AMC) by replacing traditional hand-engineered features with end-to-end neural networks that process raw signal data, thus demonstrating high accuracy at moderate-to-high Signal-to-Noise Ratios (SNRs). While contemporary convolutional and hybrid recurrent network architectures achieve excellent performance, they often incur significant computational costs that hinder deployment on resource-constrained Internet of Things (IoT) edge devices. To address this challenge, this work proposes and presents a low-cost, open-source radio platform that performs signal acquisition and utilizes vector extensions for accelerated inference. The platform integrates a commodity Realtek Software-Defined Radio (RTL-SDR) with Reduced Instruction Set Computer – Five (RISC-V) processors. The workflow methodology for the proposed approach is a reproducible, end-to-end pipeline for deploying signal classification models on resource-constrained devices in IoT networks. The pipeline's primary strength is its deterministic dataset assembly. The workflows process establishes a coherent baseline for embedded classification under strict memory and processing power constraints typical in IoT devices.**

Keywords-Automatic modulation classification; Signal to noise ratio; RISC-V; interference.

I. INTRODUCTION

Deep learning has reshaped AMC by replacing hand-engineered features with end-to-end models that operate directly on raw In-phase and Quadrature (I/Q) sequences, that achieve strong performance at moderate-to-high SNRs. Convolutional Neural Networks (CNNs) and hybrid Convolutional Neural Network–Recurrent Neural Network (CNN–RNN) models trained on datasets such as RadioML have shown much higher accuracy, often above 90–98% at stronger SNRs. However, these models come with significant computational costs during inference, making efficient IoT edge device deployment a big challenge. Though CNN variations and spectrogram-based techniques are continually introduced, showing the clear shift toward deep learning. Yet, persistent issues remain, including performance drops at low SNR, difficulty generalizing beyond synthetic datasets, and the need to sustain real-time processing under hardware limits [1],[2].

Open RISC-V platforms with the RISC-V Vector (RVV) 1.0 vector extension help accelerate vector-heavy signal-processing and inference workloads central to intelligent radio. The acceleration is done via RVV's Vector-Length Agnostic (VLA) programming model, flexible register grouping, and support for mixed-precision arithmetic. These processes enable scalable Single Instruction, Multiple Data (SIMD) style parallelism tuned from embedded to High-Performance Computing (HPC) class implementations. RISC-V vector cores illustrate how RVV-backed designs pair a scalar pipeline with a decoupled vector unit and high-throughput memory subsystems. The pairing facilitates efficient IoT edge device inference with publicly documented configurations touting 512-bit vector registers, BFloat16 (BF16)/16-bit Floating Point (FP16)/8-bit Integer (INT8) support, and Machine Learning (ML) oriented instruction extensions for neural kernels and matrix operations [3],[4],[5],[6],[7],[8].

The RTL-SDR, which is a USB Software-Defined Radio (SDR) derived from Digital Video Broadcasting – Terrestrial (DVB-T) tuner chipsets, provides wide coverage and stable sample rates up to roughly 2.56 Mega Samples per second (MS/s) for reliable demodulation. The wide coverage provided is commonly between 24 MHz and 1766 MHz with popular tuners. These attributes make the RTL-SDR a practical, inexpensive front end for collecting real I/Q datasets to complement synthetic corpora during development and testing. As a commodity device with 8-bit Analog-to-Digital Converter (ADC) samples and ubiquitous host support, it enables rapid, repeatable data capture across bands of interest for model pre-training, augmentation, and validation. These enable it to keep total system cost low enough and allow it to scale benchtop experiments to distributed field measurements [2],[9].

Therefore, this work proposes and presents a new approach for real time automatic modulation classification using open-source platforms. The method utilizes inexpensive RTL-SDR USB dongles for capturing signals and RISC-V vector chips for fast speed running of Artificial Intelligence (AI) models on miniature, power-constraint devices. The contribution and significance of this approach is fourfold. First, it equips IoT nodes and gateways with on-device spectrum intelligence. Such capability allows distributed IoT devices to monitor, classify, and react to the radio frequency environment in real time without relying on centralized cloud processing. Second, it shortens the path from simulated data to real over-the-air recordings, by improving real-time speed and reliability for AMC.

Particularly, this reduction in time supports IoT use cases such as local interference detection on smart-city lampposts, factory floor coexistence monitoring, and edge device anomaly alert transmission without the need for constant cloud backhaul. Third, implementing AMC at the ultra-edge also facilitates the adaptation of radio in situ for IoT device deployments. For example, selecting robust modulation techniques under congestion, flagging unauthorized emitters near industrial assets, or triaging spectrum events in environmental sensor networks are possible. All these capabilities reduce latency, bandwidth, and power while maintaining service quality. Lastly, the approach makes wide-area spectrum monitoring become feasible due to the adoption of distributed receivers on battery-powered or solar-powered IoT gateways. These gateways classify signals locally and share only compact summaries, in order to improve scalability and privacy while preserving situational awareness. IoT applications that benefit from this capability include utility metering, telehealth, telemetry backhaul, and campus-scale asset tracking.

Overall, the proposed approach makes spectrum intelligence more accessible by pairing modern deep learning with vectorized execution on widely available RISC-V hardware. Both concepts have been explored separately, but until now, have not been paired together as explored in this work. The methodology leverages the growing adoption of RISC-V in IoT edge devices due to cost, openness, and efficiency. Specifically, the aforementioned capabilities support IoT deployments in smart cities, industrial environments, and environmental monitoring sensor networks. They facilitate localized decision-making, latency reduction, bandwidth conservation and network reliability.

The remainder of this paper is organized as follows: Section II surveys and motivates the selection of hardware platforms. A comparison of RISC-V compute modules and SDR front-ends is given to highlight trade-offs in cost, performance, and suitability for edge deployment. Section III presents the end-to-end workflow of the proposed approach, which includes data collection, signal preprocessing, spectrogram-based CNN training, and compilation to kmodel for execution on constrained K210 microcontrollers. Section IV outlines directions for future work, including implementation in advanced and alternative architectures, utilizing expanded over-the-air datasets, and signal intelligence testing in broader applications. Section V concludes the paper by highlighting the work’s contributions to accessible edge spectrum intelligence for typical miniature IoT devices.

II. Hardware Survey And Selection

There are potentially different types of hardware that can be used as the platform for this work. Thus, it is important to evaluate candidate platforms along both performance and integration dimensions. For an IoT-oriented pipeline, cost, power consumption, and form factor are just as critical as raw computational throughput. Accordingly, two categories of hardware are reviewed which are RISC-V and software-defined radio (SDR). RISC-V compute platforms are capable of running machine learning inference at the edge, while SDRs are front-ends for signal capture.

A. *RISC-V compute platforms*

These are low-cost platforms that facilitate efficient on-device inference for edge Digital Signal Processing (DSP) classification. Their key differentiators include the CPU microarchitecture, availability of vector or Neural Processing Unit (NPU) acceleration, memory capacity, and indicative pricing for Bill Of Materials (BOM) planning. These platforms are particularly well-suited for IoT nodes because they balance affordability with power efficiency, making it feasible to deploy spectrum-aware intelligence across a large number of distributed IoT devices. By handling feature extraction and inference locally, such platforms reduce the need for continuous backhaul to the cloud, improving both scalability and responsiveness. The results of this survey are summarized in Table I.

B. *SDR front-ends (RX/TX)*

On the RF side, SDR front-ends were surveyed to identify capture devices that complement lightweight RISC-V compute platforms. Available SDRs span ultra-low-cost USB dongles through to higher-end lab-grade radios. Selection criteria included frequency coverage, converter depth and sampling rate, frequency stability, front-end filtering, duplex capability (receive-only or full transmit/receive), and cost trade-offs for system integration. For IoT deployments, receive-only devices often suffice, since the primary task is passive spectrum monitoring and classification rather than active transmission. Low-cost SDRs with stable frequency control and sufficient bandwidth can therefore enable practical large-scale sensing deployments while keeping per-node costs minimal. Table II compares candidate SDR devices.

C. *Selection rationale*

For cost-effective, edge-deployed classification, the MaixCAM provides enough integer SIMD and a small NPU. These features accelerate lightweight DSP and inference under tight power and memory budgets, while maintaining a compact BOM and integrated camera-oriented I/O for data capture. The RTL-SDR Blog V4 pairs well by offering stable frequency control, improved high frequency performance, and integrated filtering at a fraction of the cost of wideband Transmit/Receive (TX/RX) radios whose transmit capability is unnecessary for receive-only classification pipelines. Together, these devices can be used to create a compact, IoT-ready sensing node capable of autonomous spectrum monitoring, which is critical for distributed edge applications where network connectivity may be intermittent or bandwidth-limited. IoT application examples include Health Internet of Things where devices are miniature and resource constrained.

TABLE I - COMPARISON OF RISC-V COMPUTE PLATFORMS

Device	SoC / cores	Vector / NPU	RAM	Storage	I/O highlights	Notes
Sipeed MaixCAM	Sophgo SG2002, dual T-Head C906 (1.0 GHz + 0.7 GHz)	Legacy RVV 0.7.x; 1 TOPS NPU	256 MB DDR3	microSD	MIPI CSI, DVP cam, USB-C	Compact module for edge vision/DSP; selected compute node
StarFive VisionFive 2	StarFive JH7110, quad SiFive U74 (up to 1.5 GHz)	No RVV; RV64GC	2–8 GB LPDDR4	microSD	GbE, HDMI, M.2 (PCIe 2.0)	Mature RISC-V SBC with broad Linux support
Milk-V Duo S	Sophgo SG2000, dual C906 + 1× Cortex-A53	Legacy RVV 0.7.x; vendor NPU	512 MB SIP DRAM	microSD	MIPI CSI/DSI, USB	Tiny hybrid RISC-V + ARM for I/O flexibility
Sipeed LicheePi 4A	T-Head TH1520, quad C910	Vendor vector ext; NPU present	4–16 GB LPDDR4	microSD/eMMC	PCIe 3.0, HDMI, MIPI	Higher-end RISC-V SBC for heavier workloads
Milk-V Mars	T-Head TH1520, quad C910	Vendor vector ext; NPU present	4–16 GB	microSD/eMMC	PCIe, HDMI, MIPI	Dev board variant around TH1520
Pine64 Star64	StarFive JH7110, quad U74	No RVV; RV64GC	4–8 GB LPDDR4	microSD	GbE, PCIe, HDMI	JH7110 platform in Pine64 ecosystem
Banana Pi BPI-F3	SpacemiT K1 (multi-core RISC-V)	Vendor vector/NPU (SoC-dependent)	up to 8 GB	microSD/eMMC	GbE, PCIe, HDMI	Newer RISC-V SBC line; specs evolving
MangoPi MQ-Pro (D1)	Allwinner D1, single XuanTie C906 (~1 GHz)	Legacy vector ext	512 MB DDR3	microSD	GPIO, USB OTG	Ultra-low-cost entry RISC-V Linux
HiFive Unmatched	SiFive FU740 (quad U74 + S7)	No RVV; RV64GC	8 GB DDR4	M.2 NVMe	PCIe x8 (x4 elec), GbE	High-end dev board; limited availability
BeagleV Ahead	T-Head TH1520	Vendor vector ext; NPU present	4–8 GB	microSD/eMMC	PCIe, HDMI, MIPI	Community SBC with TH1520
StarFive VisionFive (v1)	StarFive JH7100 (dual U74)	No RVV	up to 8 GB	microSD	GbE, HDMI	First-gen predecessor to VF2
Milk-V Meles	CVITEK CV1800B (RISC-V C906)	Vendor vector; ISP/NPU (SoC)	512 MB	microSD	Dual MIPI CSI, Ethernet	Camera-centric edge module

TABLE II - COMPARISON OF SOFTWARE DEFINED RADIO FRONT-ENDS

Device	Frequency coverage	ADC / sample rate	TCXO	Preselection / filters	Notes
RTL-SDR Blog V4	~0.5–30 MHz (direct) + ~24/28–1766 MHz	8-bit (RTL2832U), up to ~2.4–3.2 Msps	1 ppm	Improved HF path, FM notch	Bias-T; RX only; Selected RF frontend; stable, low cost
RTL-SDR Blog V3	~0.5–30 MHz (direct) + 24–1766 MHz	8-bit, up to ~2.4 Msps	1 ppm	Basic, optional FM notch	Bias-T; RX only; Proven baseline dongle
HackRF One	~1 MHz–6 GHz	8-bit, up to 20 Msps	~20 ppm	Minimal onboard filtering	No Bias-T; Half-duplex TX/RX Wideband, experimental TX
Airspy Mini	~24–1800 MHz	12-bit, up to 6–10 Msps	0.5 ppm	Moderate front-end filtering	No Bias-T; RX only; High dynamic range for VHF/UHF
Airspy R2	~24–1800 MHz	12-bit, up to 10 Msps	0.5 ppm	Improved linearity/filtering	No Bias-T; RX only; Performance-oriented dongle
Airspy HF+ Discovery	~0.5 kHz–31 MHz + 60–260 MHz	16-bit MF stages, high effective ENOB	0.5 ppm	Strong HF preselection	No Bias-T; RX only; Elite HF sensitivity and selectivity
SDRplay RSP1A	~1 kHz–2 GHz	12–14-bit, up to 10 Msps	0.5 ppm	Multi-band preselection	No Bias-T; RX only; Versatile coverage with filtering
SDRplay RSPdx	~1 kHz–2 GHz	12–14-bit, up to 10 Msps	0.5 ppm	Enhanced HF front-end	No Bias-T; RX only; Improved LF/MF/HF robustness
SDRplay RSPduo	~1 kHz–2 GHz (dual tuners)	12–14-bit, up to 10 Msps	0.5 ppm	Preselection per tuner	No Bias-T; RX only (dual coherent) Diversity/DF use cases
LimeSDR Mini 2.0	~10 MHz–3.5 GHz	12-bit, up to ~30.72 Msps	1 ppm	Basic, external filtering advised	No Bias-T; Full-duplex Compact TX/RX platform
ADALM-Pluto (PlutoSDR)	~325 MHz–3.8 GHz (70 MHz–6 GHz mod)	12-bit, up to ~61.44 Msps RX	1 ppm	Minimal onboard filtering	No Bias-T; Full-duplex Flexible teaching/experimental SDR
USRP B200mini-i	~70 MHz–6 GHz	12-bit, up to ~56 Msps	2.5 ppm OCXO (-i)	External filtering recommended	No Bias-T; Full-duplex; Lab-grade, UHD ecosystem
KrakenSDR (coherent)	~24–1766 MHz (5 coherent tuners)	8-bit, per-tuner ~2.4 Msps	0.5–1 ppm	FM notch options	Bias-T; RX only; DoA/beamforming with 5-way phase-coherence
KerberosSDR (coherent)	~24–1766 MHz (4 tuners)	8-bit, per-tuner ~2.4 Msps	0.5–1 ppm	Optional filtering	No Bias-T; RX only; Earlier 4-tuner coherent array

III. WORKFLOW- METHODOLOGY

In this section, the proof-of-concept implementation of the proposed approach is presented. The flowchart in Figure 1 illustrates the workflow methodology. The end-to-end pipeline runs on a desktop host and outputs a compact kmodel artifact for the K210's Kendryte Processing Unit (KPU). It starts with raw I/Q captures and finishes with a compiled model that adheres to the memory and operator constraints documented for Maix/MaixPy deployments on the K210. Although development occurs on a desktop host, the resulting models are fully compatible with IoT edge devices. Consequently, the models facilitate the implementation of autonomous spectrum classification in IoT devices deployed in remote or power-constrained environments. The workflow uses SDR# as shown in Figure 2 with an RTL-SDR to collect labeled I/Q recordings, Scientific Python (SciPy) to generate time-frequency spectrograms from complex baseband arrays, TensorFlow/Keras to train a CNN on those images, and TensorFlow Lite plus *nncase*/KPU tooling to export and compile an embedded-ready kmodel which is standard practice supported by [1],[2],[3],[4],[5].

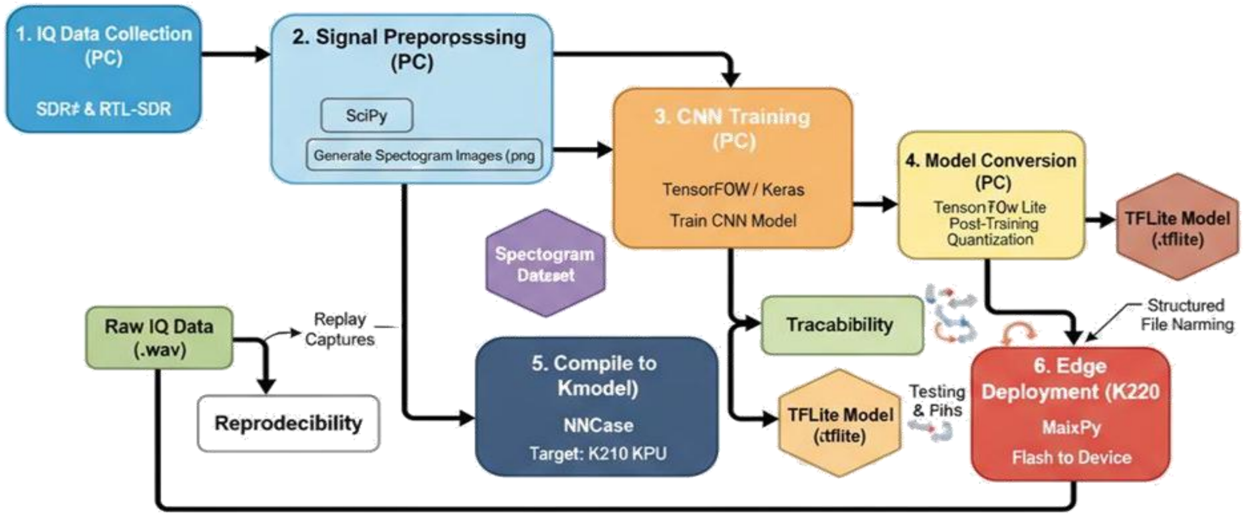

Figure 1. Workflow Approach.

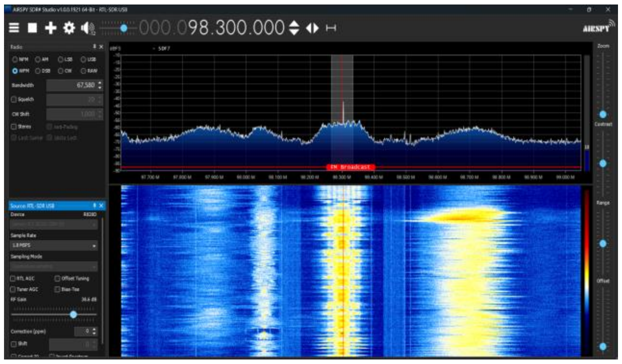

Figure 2. SDR Sharp (SDR#) Interface.

A. Data collection on PC

During this process, raw complex baseband streams are recorded interactively in SDR# via the Recording tab, which supports baseband I/Q capture to Waveform Audio File Format (WAV). The WAV file is used for later offline processing and precise replay in SDR# to enable deterministic dataset curation for downstream steps. To assemble a minimal labeled corpus aligned with target classes, one can capture a strong local FM broadcast segment, record a National Oceanic and Atmospheric Administration (NOAA) Weather Radio transmission within the 162.400–162.550 MHz Very High Frequency (VHF) allocation. Then gather a clip from an unoccupied channel to form a noise baseline, with files organized into class-named directories and metadata preserved in filenames to aid traceability. Alternate SDR ecosystems and guidance on I/Q data handling reinforce the objective of producing contiguous, timestamped baseband data suitable for reproducible post-processing and later validation, independent of the specific GUI tool used [1],[6],[7].

B. *Signal preprocessing and spectrograms*

At this stage, complex I/Q arrays are transformed into two-dimensional time–frequency images using a Short-Time Fourier Transform (STFT) spectrogram. The SciPy's documentation specifies outputs as frequency bins, time-frames, and a non-negative spectral representation suitable for learning and visualization. A practical implementation loads each I/Q recording, computes spectrograms with fixed window and overlap for consistent resolution. Log scaling and normalization are performed, and standardized images are written to per-class folders. These align with established spectrogram practice and tutorials [4],[8].

C. *CNN training on spectrograms*

During this task, with a directory of labeled spectrogram images, a compact CNN is defined and trained using Keras. The task follows TensorFlow's canonical model creation and training patterns that interoperate cleanly with subsequent TensorFlow Lite conversion. In addition, the training routine uses consistent image dimensions and a simple stack of convolutional and pooling layers ending in a softmax head. Then, it saves a validated host-side model artifact before any edge-oriented conversion, which cleanly separates algorithm development from deployment concerns as suggested [5].

D. *Model conversion to TensorFlow Lite*

After host-side validation, the Keras model is converted into a *.tflite* FlatBuffer using the TensorFlow Lite Converter API, which converts Keras models to byte buffer for exporting and saving the result. To meet embedded constraints, post-training quantization can be enabled during conversion to reduce model size and improve inference efficiency. These steps establish both float and quantized TFLite variants for rapid A/B checks prior to device-specific compilation as proven in [5],[9].

E. *Compilation to kmodel for K210*

At this level, the Kendryte K210's KPU executes models in the vendor-specific kmodel format generated by *nncase*. The Sipeed's Maix/MaixPy documentation outlines KPU loading modes, typical memory ceilings by firmware variant, and the expectations for kmodel artifacts compiled from TFLite. In practice, the TFLite model is compiled with *nncase* to produce a kmodel and then verified against the host TFLite baseline on representative inputs. The process ensures operator support and adherence to KPU memory limits described in Sipeed guidance for C software development kit (C SDK) or MaixPy runtimes. Community implementation notes also emphasize using a compatible *nncase* release for K210 workflows and cross-checking inference numerics between TFLite and kmodel before flashing or SD-card deployment [2],[3],[10].

F. *Setting up the edge device to run the converted model*

The sub-steps for this task are 1) on the edge device, a suitable MaixPy firmware or a C SDK–based firmware is installed, 2) on either the SD card or in on-board flash, the kmodel is provisioned, as supported by MaixPy's KPU loader and the Kendryte flashing utility, 3) loading models are placed on an SD card with the MaixPy KPU API using a filesystem path; models flashed to a designated offset can be loaded from flash, with Sipeed documentation to describe memory limits and firmware variants, 4) modules are deployed, which typically involves flashing firmware with kflash.py or its GUI, copying the kmodel to SD or embedding it in flash, and writing a minimal runtime that initializes the sensor or input pipeline, 5) frames are pre-processed to the model's input shape and data layout; this invokes the KPU inference, and emits results over serial, display, or General-Purpose Input/Output (GPIO) as applicable to scenarios discussed in [2],[3],[11].

G. *Reproducibility considerations*

Replaying recordings to validate labeling and preprocessing is facilitated by SDR#'s ability to open baseband I/Q WAV files in order to enable confirmation of tuned stations, SNR, and channel occupancy prior to batch spectrogram generation and training. Moreover, informative file naming, directory schemes, and general I/Q data management best practices support traceability across data collection, preprocessing, and inference stages without changing the core methods described here. Retaining both original I/Q archives and derived spectrograms ensures experiments can be reconstructed or extended. The maintenance of float and quantized TFLite baselines provides stable references for evaluating compiler effects prior to kmodel flashing and device trials. By producing compact, portable models, this workflow allows IoT devices to perform on-site classification, anomaly detection, and local interference management while maintaining minimal power and memory usage.

IV. CONCLUSION AND FUTURE WORK

This work has aimed to present an end-to-end, reproducible pipeline for converting raw SDR# baseband recordings into an optimized model for inference on resource-constrained Kendryte K210 microcontrollers. The methodology integrates four key stages and a total of six tasks including the key stages. These key stages are standardized spectrogram preprocessing, compact CNN training, post-training quantization via TensorFlow Lite, and final compilation using *nncase*. The resulting workflow establishes a tractable and verifiable pathway from RF signal acquisition to on-device classification, explicitly addressing the memory and operator limitations inherent to edge hardware. This approach makes spectrum intelligence accessible to a wide range of IoT deployments, enhances local decision-making, reduces latency, and conserves bandwidth. However, areas for future work include implementing formal dataset quality assurance beyond manual replay, empirically justifying spectrogram parameters, enforcing numerical parity between the TFLite and kmodel outputs, and conducting instrumented on-device profiling to measure real-world performance.

The next steps of this proof of concept will focus on the following activities:

- **Porting to a Ratified RVV 1.0 Platform:** The highest priority is to migrate the entire workflow to a newer RISC-V platform that implements the fully ratified version 1.0 of the RVV Extension. Migration will enable a quantitative analysis of the performance gains achievable using the more powerful and flexible VLA programming model rather than the legacy vector implementation used in this work. This step is crucial for demonstrating the full potential of standardized RISC-V vector processing for ML workloads. Porting to RISC-V platforms could further enhance IoT nodes by enabling larger or more sophisticated models to execute on small, distributed devices at the network edge.
- **Comparative Benchmarking:** A performance benchmark will be conducted to compare RISC-V-based platform with a low-cost edge AI accelerator, such as the Raspberry Pi with a Google Coral Tensor Processing Unit (TPU) or the NVIDIA Jetson Nano. This will provide insightful trade-offs between performance, power consumption, cost, and openness across different edge computing paradigms.
- **Advanced Model Architectures:** Models like MobileNets, SqueezeNets, quantization-aware networks could potentially improve classification accuracy on more complex modulation schemes while maintaining or even reducing inference latency. Therefore, more sophisticated and computationally efficient neural network architectures will be explored.
- **Expanded Over-the-Air (OTA) Dataset and Classification Tasks:** Training on a larger, more diverse dataset, which includes various modulation techniques like QPSK, GMSK, 16-QAM, and 64-QAM, will enhance the platform's ability to operate reliably in several RF conditions. This will improve the performance of IoT applications such as smart city infrastructure, telehealth, telemetry, and environmental monitoring.
- **Exploration of New Applications:** The validated platform serves as a foundation for exploring other signal intelligence tasks beyond AMC. The platform can also support IoT-specific applications such as autonomous anomaly detection, RF fingerprinting for device authentication, interference localization in distributed sensor networks, and automated spectrum-aware control of edge devices, or automated signal protocol identification.

ACKNOWLEDGMENT

The authors would like to thank the French South African Institute of Technology (F'SATI), Cape Peninsula University of Technology, Cape Town, South Africa for funding this research.

REFERENCES

[1] O. F. Abd-Elaziz, M. Abdalla, and R. A. Elsayed, "Deep learning-based automatic modulation classification using robust CNN architecture for cognitive radio networks," Sensors, vol. 23, art. no. 9467, Nov. 2023, doi: 10.3390/s23239467.

[2] Ashishware, "Creating a CNN to classify cats and dogs for Kendryte K210 boards," Ashishware.com, Aug. 28, 2024. [Online]. Available: https://ashishware.com/2024/08/29/k210CatDog/.

[3] Android Open Source Project, "TensorFlow Lite converter," GoogleSource, n.d. [Online]. Available: https://android.googlesource.com/platform/external/tensorflow/+/HEAD/tensorflow/lite/g3doc/convert/index.md. (Accessed: Sep. 3, 2025).

[4] RTL-SDR.com, SDRSharp User's Guide, May 7, 2018. [Online]. Available: https://www.rtl-sdr.com/sdrsharp-users-guide/.

[5] A. Frame, "SiFive intelligence X280: Optimized efficiency and control for the modern workload" [Product Brief], presented at the Reduced Instruction Set Computer-Five In Space Conference, ESA, Dec. 2022. [Online]. Available: http://microelectronics.esa.int/riscv/rvws2022/presentations/04-SiFive_Intelligence_X280_for_Space_Exploration_v2.0_Dec_22.pdf.

[6] Google AI Edge, "Convert TensorFlow models," Aug. 29, 2024. [Online]. Available: https://ai.google.dev/edge/litert/models/convert_tf.

[7] Kendryte, kflash.py: A Python-based Kendryte K210 UART ISP utility, 2018. [Online]. Available: https://github.com/kendryte/kflash.py.

[8] Osmocom, "rtl-sdr wiki," n.d. [Online]. Available: https://osmocom.org/projects/rtl-sdr/wiki. (Accessed: Sep. 3, 2025).

[9] H. Ouamna, A. Kharbouche, Z. Madini, and Y. Zouine, "Deep learning-assisted automatic modulation classification using spectrograms," Eng. Technol. Appl. Sci. Res., vol. 15, no. 1, pp. 19925–19932, Feb. 2025, doi: 10.48084/etasr.9334.

[10] Π Node, "RTL-SDR," n.d. [Online]. Available: https://p-node.org/documentation/ressources/rtl-sdr. (Accessed: Sep. 3, 2025).

[11] RISC-V International, RISC-V "V" vector extension, Version 1.0, 2021. [Online]. Available: https://github.com/riscv/riscv-v-spec/releases/tag/v1.0.