BFAMod: FPGA Acceleration of Binary Fuzzy ART for Rapid Inference on Embedded MPSoCs

Jacob L. Schroll
Dept. of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA
jsnph@mst.edu

Jian Liu
Dept. of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA
jliu@mst.edu

Niklas M. Melton
Dept. of Computer Science
Missouri University of Science and Technology
Rolla, MO, USA
niklasmelton@mst.edu

Sasha Petrenko
Dept. of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA
petrenkos@mst.edu

Micah A. Renfrow
Dept. of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA
marhnd@mst.edu

Iwan Sandjaja
Dept. of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA
iskg3@mst.edu

Donald C. Wunsch II
Dept. of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA
dwunsch@mst.edu

***Abstract*—Adaptive Resonance Theory (ART) enables stable learning in non-stationary environments, but deploying ART models on embedded platforms is often limited by floating-point operations and the cost of repeatedly scanning many learned categories. We present BFAMod, a lightweight CPU–FPGA co-design that accelerates *inference* for Binary Fuzzy ART (BFA) on embedded MPSoCs. BFAMod streams magnitude-coded inputs and category prototypes via DMA, performs on-the-fly magnitude-to-thermometer expansion, and implements BFA matching and category selection using word-level logic and popcount accumulation with division-free cross-multiplication. An on-chip input cache reuses the current input across the full category scan, eliminating redundant host–device transfers. Implemented on a Kria KV260 platform at 100 MHz, BFAMod achieves more than an order-of-magnitude higher end-to-end inference rate than a quad-core ARM Cortex-A53 baseline on Fashion-MNIST, HAR, and Mini Speech Commands, and exhibits predictable scaling with category count and encoded dimensionality in controlled sweeps. The small resource footprint indicates feasibility as an embedded accelerator alongside other real-time pipelines, while optional online updates can remain host-managed when enabled.**

***Index Terms*—Adaptive Resonance Theory (ART); binary fuzzy ART (BFA); edge computing; embedded systems; field programmable gate arrays (FPGA); hardware acceleration; logic-based inference; online learning.**

I. Introduction

A fundamental capability for embedded intelligence is online learning under changing dynamics, but deploying such models on resource-constrained hardware requires overcoming tight compute and memory budgets, especially when inference relies heavily on floating-point arithmetic. Adaptive Resonance Theory (ART) provides a reliable framework for online learning because it can learn new information while retaining previous knowledge, but many ART models still rely on floating-point arithmetic and comparisons that are costly on resource-constrained hardware [1].

This paper showcases BFAMod, a hardware accelerator variant for Binary Fuzzy ART [2] (BFA). BFA reformulates core ART mechanisms by using binary data and Boolean algebra. To simplify the underlying arithmetic, floating-point operations are replaced with bitwise logic and population count operations. In our hardware implementation, BFA is able to achieve high throughput and low latency, making it suitable for deployment on resource-constrained hardware.

Although BFA significantly simplifies ART arithmetic, practical deployment on embedded MPSoCs still faces a critical scalability gap. During inference, BFA must repeatedly scan learned categories and compute match/choice statistics over high-dimensional encoded vectors. As the number of learned categories and the encoded dimensionality of inputs grow, the search for the resonating category experiences increased latency, and the embedded platform reaches a relatively low performance ceiling. Moreover, the bottleneck is often not only compute, but also repeated data movement and host–device overhead. These constraints motivate a dedicated accelerator that can sustain bit-level operations efficiently while minimizing unnecessary data traffic in edge and real-time settings.

Prior work has explored FPGA acceleration for machine learning workloads and hardware implementations of ART variants [3]–[5], yet there remains limited system-level guidance on how to translate BFA's end-to-end inference pipeline—including continuous-data encodings, streaming popcount accumulation, and division-free category selection—into an embedded CPU–FPGA co-design with predictable

scaling. BFAMod targets this missing link between BFA's theoretical hardware-friendliness and practical deployment constraints on MPSoC-class platforms.

This work is guided by three related questions: how to express BFA inference using hardware-friendly primitives such as bitwise logic, popcount, and division-free comparison without changing inference behavior; how to represent and transfer inputs and weights in compact magnitude form while preserving the thermometer-coded structure required for continuous inputs; and which factors, including control overhead, DMA bandwidth, category count, and encoded dimensionality, dominate end-to-end scalability on embedded MPSoC platforms.

To answer these questions, we make three main contributions. First, we design a CPU–FPGA co-processing system that separates the control plane from the high-throughput data plane, enabling practical BFA deployment on resource-constrained edge devices. Second, we map BFA inference to a streaming hardware pipeline with on-the-fly magnitude-to-thermometer decoding, word-level AND and popcount accumulation for match computation, and division-free category selection via cross-multiplication, thereby avoiding floating-point division in the critical path. Third, we evaluate BFAMod against an embedded ARM CPU baseline on real datasets and controlled scaling sweeps over category count and encoded dimensionality, and we report both performance trends and hardware cost indicators to characterize the resulting cost–performance tradeoff.

The remainder of this paper is organized as follows. Section II reviews the BFA formulation and encoding choices, Section III presents the BFAMod architecture, Section IV describes the evaluation setup, and Section V reports the results.

To support reproducibility, the code and scripts used in this work are publicly available at https://github.com/jacoblschroll/BFAModExperiments.

II. Background

A. Binary Fuzzy ART

Binary Fuzzy ART is a variant of Fuzzy ART [6] that operates on binary input vectors I and binary weight vectors W. BFA relies on the match criterion and the choice function $T(I, W)$, shown below [2]:

$$M(I, W_c) = H(I \wedge_b W_c) \tag{1}$$

Here, $\wedge_b$ denotes element-wise logical AND and $H(\cdot)$ denotes Hamming weight (popcount), so $M(I, W_c)$ counts the overlap between the input and category prototype. In hardware, this maps directly to word-level AND followed by popcount accumulation across streamed packets.

$$T(I, W_c) = \frac{M(I, W_c)}{H(W_c)} \tag{2}$$

Here, $H(W_c)$ is the number of active bits in the category prototype and acts as a normalization term. In hardware, a direct implementation of Eq. (2) would require division, which is expensive and can dominate latency and area.

As shown in prior work [2], this operation can be eliminated through cross-multiplication:

$$M(I, W_{c1}) \cdot H(W_{c2}) > M(I, W_{c2}) \cdot H(W_{c1}) \tag{3}$$

Eq. (3) preserves the ordering implied by (2) when all terms are nonnegative. In hardware, this transforms category selection into: (i) compute (M, H) pairs for each category, (ii) multiply M by the other category's H, and (iii) compare products using an integer comparator. The multiplications can be mapped to FPGA DSP slices (or LUT multipliers), enabling a compact and fast selection stage.

B. Thermometer Encoding

To maintain the performance of ART when handling continuous data, BFA utilizes thermometer encoding. This encoding scheme maps a continuous value to a bit string, where the number of active bits represents the magnitude of the value. Unlike standard base-2 binary or Gray code representations, thermometer encoding is monotonic and preserves the ℓ_1 distance metric required for ART stability [2]. In hardware, this encoding can be implemented efficiently using shift registers or lookup tables (LUTs), serving as a preprocessing step before the data enters the BFA core.

Thermometer encoding turns a scalar feature into a structured bit-vector so that similarity can be assessed using overlap (AND) and popcount, as in Eq. (1). The encoding bit-depth determines the length of this bit-vector: larger bit-depth gives finer quantization but increases the encoded dimensionality, which directly increases (i) the number of streamed words, (ii) popcount accumulation cost, and (iii) memory bandwidth requirements. This motivates decoding only when needed and reusing decoded inputs across the scan of many categories.

C. Magnitude Encoding

While thermometer encoding allows for an accurate representation of continuous data, it is an inefficient method of data storage. BFA only requires that weights and inputs be formatted as thermometer encoding at compute time. When samples or weights are not in use for compute, they can be efficiently stored in a base-2 representation of their magnitudes. Both weights and samples can be decomposed into feature-level components [2]:

$$\mathbf{I} = \Big\Vert_{f \in \mathcal{F}} \mathbf{I}_f \tag{4}$$

Eq.(4) indicates that the full input vector $\mathbf{I}$ is formed by concatenating (denoted by $\Vert$) per-feature blocks $\mathbf{I}_f$. This structure is useful for hardware because the input can be streamed and processed block-by-block (or word-by-word) without needing the entire vector at once.

$$\mathbf{W}_c = \Big\Vert_{f \in \mathcal{F}} \mathbf{W}_{c,f} \tag{5}$$

Similarly, Eq.(5) shows that each category prototype $\mathbf{W}_c$ is a concatenation of per-feature blocks $\mathbf{W}_{c,f}$. This supports a

streaming implementation where BFAMod processes the same feature block index for $\mathbf{I}$ and $\mathbf{W}_c$ concurrently, accumulating partial popcounts to build $M(I, W_c)$ and $H(W_c)$ across all blocks.

Because these representations are complement coded, both can be further decomposed into an upper and lower-bound complement [2]:

$$\mathbf{I}_f = \left[\mathbf{I}_f^{\text{lower}} \,\middle\|\, \mathbf{1} - \mathbf{I}_f^{\text{upper}} \right] \tag{6}$$

Eq. (6) expresses each feature block in a complement-coded form: the first half encodes a lower-bound component $\mathbf{I}_f^{lower}$ and the second half encodes the complement of an upper-bound component $(1 - \mathbf{I}_f^{upper})$. Complement coding is commonly used in ART to stabilize learning and matching by representing both presence and absence information. In hardware, it implies that each feature contributes two structured sub-blocks, which can still be streamed and processed with the same AND+popcount pipeline used in Eq.(1).

$$\mathbf{W}_{c,f} = \left[\mathbf{W}_{c,f}^{\text{lower}} \,\middle\|\, \mathbf{1} - \mathbf{W}_{c,f}^{\text{upper}} \right] \tag{7}$$

Eq. (7) applies the same complement-coded structure to the category prototype blocks. For BFAMod, this means the weight stream naturally aligns with the input stream: the datapath processes corresponding sub-blocks, accumulates overlap counts for $M(I, W_c)$, and accumulates $H(W_c)$ from the prototype bits.

In the case of input samples, the upper or lower component can be inferred from its complement, further reducing the storage required to represent the input. This observation motivates storing/transferring compact magnitude-coded representations (base-2) via DMA, and performing on-the-fly expansion to thermometer-coded and complement-coded bit-vectors only inside the FPGA compute path. This reduces off-chip bandwidth while preserving the exact computation required by Eqs.(1)–(3).

III. Device Design and Considerations

BFAMod targets fast inference of trained BFA models on embedded MPSoCs, where repeated category scanning becomes the dominant cost as the number of learned categories and encoded dimensionality increase. The design uses a heterogeneous CPU–FPGA architecture in which the host handles preprocessing and control, while the FPGA executes the streaming BFA datapath on magnitude-coded data transferred via DMA and returns the best category index and associated statistics through a memory-mapped interface.

The architecture in Fig. 1 implements the BFA equations from Section II as a streaming datapath: Eq. (1) maps to word-level AND plus popcount accumulation to compute $M(I, W_c)$, Eq. (2) motivates normalization by $H(W_c)$, and Eq. (3) enables division-free selection through cross-multiplication and comparison. This mapping ensures that the critical-path operations are limited to bitwise logic, integer accumulation, and integer multiply/compare, which are well-suited to FPGA fabric and DSP resources.

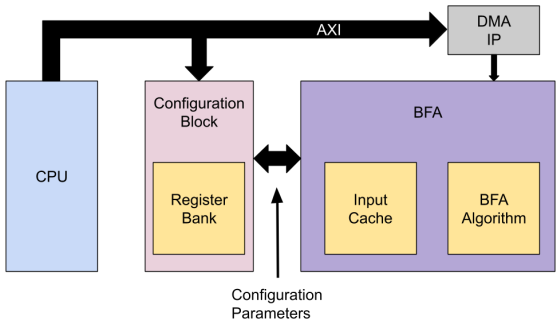

Fig. 1. BFAMod system architecture on an embedded MPSoC. The host CPU configures the accelerator via a memory-mapped AXI control interface and streams magnitude-coded inputs and category weights through DMA. On the FPGA, BFAMod performs on-the-fly magnitude-to-thermometer decoding followed by a streaming BFA inference pipeline, and returns the best category index and associated statistics through the same memory-mapped interface.

A. CPU-FPGA Interface

In a practically sized BFA model, the input and weight vectors are substantially wider than the streaming bus width. Thus, performance is constrained by sustained streaming throughput and the ability to reuse data (e.g., the input) across many category comparisons.

To meet the throughput requirements of this device, the architecture separates the data and control planes with DMA and AXI interconnects. An AXI interconnect is used to set and manage control signals, while DMA is employed for high-throughput transfer of bulk data, such as weights and inputs. In any practically sized BFA model, input and weight vectors exceed the 32-bit data bus width of the DMA interface, requiring vectors to be divided into 32-bit packets.

The AXI interconnect attaches to two modules, the configuration block and the Xilinx DMA IP core. The configuration block manages control signals and model parameters such as the resonance threshold (ρ), system state, and cluster results. These need to be read and written infrequently and thus do not warrant the use of DMA, unlike the loading of inputs and weights.

The separation of control and data planes is chosen to maximize system performance, which is constrained by the rate at which data is delivered to the accelerator. By partitioning the low- and high-throughput data streams, efficient coordination with the host processor is achieved. Specifically, the host configures ρ and input-dimensionality, initiates DMA transfers for the input and category weights, and then polls/reads back the best category index and associated statistics once the scan completes. *This host-driven protocol keeps the FPGA datapath fully utilized during the category scan while minimizing control-plane overhead.*

B. Implementation of BFA

Due to the design of the BFA algorithm, the resources required for implementing it in hardware are minimal (see

Table I). The computation is organized as a streaming pipeline that processes one 32-bit packet per cycle (after initial handshake), enabling sustained throughput while accumulating global statistics across packets.

For each packet of vectors that is passed, all BFA arithmetic occurs in a single clock cycle. Data is initially decoded from its magnitude-encoded form into a thermometer encoding. This on-the-fly decoding avoids storing full thermometer-coded vectors in off-chip memory while still providing the bit-vector structure needed for Eq. (1).

To calculate the Hamming weight, an adder tree is used to allow for rapid calculation of large sums. The resulting Hamming weight is accumulated until the input and weight vector have been completely passed to the device. Concretely, for each category c, BFAMod accumulates (i) $M(I, W_c) = H(I \wedge_b W_c)$ by popcounting the AND result and (ii) $H(W_c)$ by popcounting the weight stream. These two integers form the sufficient statistics required for selection in Eq. (3).

The accumulated Hamming weight grows alongside bit-depth and input size. To account for this, the accumulated results are stored as a 32-bit register. In practice, accumulator width should be selected based on the maximum encoded vector length; in our implementation, a fixed-width accumulator is used consistent with the largest evaluated configuration.

Resonance is checked using Eq. 3. To reduce the FPGA resources demanded by this device, the multiplication and comparison are performed by DSP slices rather than LUTs. During the scan, BFAMod maintains the current best category by comparing the new candidate's cross-multiplied score against the stored best score, updating the best index and associated statistics when the new candidate wins.

If the current weight achieves a higher resonance than previously occurred, it is written to registers storing the index of the weight, as well as $|I|$ and $|I \wedge W_C|$. Here, $|I|$ corresponds to $H(I)$ and $|I \wedge W_c|$ corresponds to $M(I, W_c)$.

Once all weights have been processed, the best index can be read from the device. In the case that the input did not achieve sufficient resonance with any weights, a sentinel value (e.g., -1) is returned. If an online-update mode is enabled, the host may then perform a BFA learning/update step and stream the updated weight(s) back to BFAMod for subsequent inferences; otherwise, BFAMod operates purely as an inference accelerator.

The datapath in Fig. 2 implements the BFA inference flow using on-the-fly decoding, streaming overlap accumulation, and division-free category selection.

C. Caching Input Data

To reduce the data required for transfer to the device, the architecture includes a cache to hold the current input vector. Due to both device and algorithm constraints, there is little opportunity for caching. Weights are accessed with uniform frequency and are likely to exceed the available on-chip block memory. While the weights of smaller models could potentially be cached on the FPGA's block memory, this would consume significant resources that may be needed by other

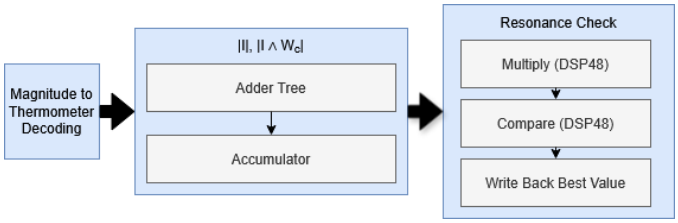

Fig. 2. BFAMod arithmetic stages and streaming datapath. Magnitude-coded values are expanded to thermometer-coded bit-vectors, then the match statistic $M(I, W_c) = H(I \wedge_b W_c)$ is computed via word-level AND and popcount accumulation. Division in the choice function is avoided by cross-multiplication, and category selection is implemented with integer multiply/compare (e.g., DSP slices) to realize Eq. (3).

modules in a real-world application. Because of this, caching weights provides minimal benefit to the performance of the device.

In contrast, the input vector is reused across every category comparison within a single inference and often has a memory footprint that fits in available on-chip BRAM/LUTRAM. By caching the current input on the FPGA, BFAMod avoids redundant host–device transfers: the input is streamed once per inference and then reused throughout the full category scan. This design choice reduces host–FPGA traffic and provided up to a $2\times$ throughput improvement in our tests.

This benefit grows with the number of categories because the cached input amortizes the host–device transfer cost across many category comparisons.

D. Resource and Power Utilization

BFAMod makes minimal use of the FPGA's resources, as summarized in Table I. Low utilization is important in embedded deployments where the FPGA fabric may be shared with other real-time pipelines (e.g., perception or signal processing).

TABLE I
FPGA RESOURCE UTILIZATION (KV260 ZYNQ ULTRASCALE+ MPSOC)

Resource	Used	Available	Utilization (%)
Lookup Tables (LUT)	10,033	117,120	8.57%
Flip-Flops (FF)	11,782	234,240	4.15%
Block RAM (BRAM)	3.5	144	2.43%
DSP Slices	6	1,248	0.48%

While data was not collected in real-world conditions, we collected power usage metrics from a power utilization report generated by Vivado 2025.1. The total on-chip power demanded is 2.799 W under load; however, the majority demand comes from the CPU. BFAMod consumes 301 mW when idle with an increase to 377 mW when under load. These values are tool-reported estimates rather than external power measurements; nevertheless, they indicate that BFAMod contributes a relatively small incremental power overhead compared to the host subsystem while providing substantial throughput gains.

E. Software

The front end of this device is implemented and managed by the PYNQ package [7]. This package provides a Python front-end for interfacing with the FPGA. PYNQ was chosen based on the use case of this device, as Python is a common language choice for data science and machine learning problems. In our workflow, the host script configures control registers (e.g., ρ and run state), allocates contiguous DMA buffers, initiates DMA transfers for input/weights, and reads back the result registers containing the best category index and related statistics.

IV. Experimental Methodology

The purpose of these experiments is to make a side-by-side comparison of a CPU versus FPGA implementation of BFA. We compare implementations using inference rate as tested for BFA models trained on the Fashion-MNIST [9] and HAR [10] datasets, as well as the Mini Speech Commands dataset, a subset of the Speech Commands Dataset [11]. We measure only the duration of prediction/inference operations. All input preprocessing is performed before timing and is excluded from the reported inference rate.

All experiments were performed on the Kria KV-260 Vision development platform, which integrates a Zynq UltraScale+ MPSoC (quad-core ARM Cortex-A53 + FPGA fabric). All CPU experiments were conducted on the included processor, a quad-core ARM Cortex-A53 with a maximum frequency of 1.3 GHz. This processor is representative of edge and embedded processing use cases and thus serves as our comparison baseline. All models were trained, and CPU metrics collected, using the single-threaded C++/pybind11 implementations in [8]. FPGA inference was conducted using the FPGA fabric on the KV260 MPSoC operating at 100 MHz, with the PYNQ package managing the device from a Jupyter notebook.

On CPU, inference executes the same BFA match/choice logic (AND/popcount and selection) in software. On FPGA, BFAMod executes the streaming datapath described in Section III and returns the best category index via memory-mapped registers. Unless otherwise stated, we report end-to-end inference rate measured at the host level, including host-side orchestration and data movement (DMA transfers) required to complete one inference over the full category scan.

For each experiment, we run inference on $N = 1000$ input samples and repeat the measurement for 5 trials. We record the wall-clock duration of each trial and report the mean inference rate (samples/s). A short warm-up run is performed before timing to avoid first-run initialization overhead.

A. Clustering Tasks

To show the differences in device performance when handling real-world data, we selected three datasets and trained a BFA model for each. These models were all trained with the same parameters, using a vigilance threshold of $\rho = 0.9$ and thermometer encoding bit-depth of 15. TTo verify correctness, we first processed the samples using the CPU implementation and treated its predicted category indices as the reference outputs.

Each dataset was assessed and modified to better fit constraints imposed by the CPU. Fashion-MNIST was reduced from 70,000 to 30,000 samples to reduce the number of clusters formed and avoid an out-of-memory error in the CPU implementation. For Mini Speech Commands, a 10-class subset of the Speech Commands dataset, rather than passing raw audio we convert each waveform into a magnitude spectrogram with 33 frequency bins and 125 STFT frames. Converting waveforms to time–frequency features is standard practice in keyword spotting and audio classification; here, the specific spectrogram parameters were selected to produce an input vector size that both the CPU and FPGA implementations can handle reliably [11], [12]. The HAR dataset required no modification.

For each dataset, we additionally record (i) the final encoded input dimensionality after preprocessing/encoding and (ii) the number of learned clusters/categories, since both quantities directly affect inference cost through the category scan and the number of streamed packets.

B. Variable Cluster and Dimension Testing

In order to demonstrate how device performance is impacted by model parameters, we performed inference across an array of *encoded* input dimensionalities and cluster counts. Rather than finding real-world datasets to match each test condition, inputs and weights were initialized with random noise. We also tested two thermometer-encoding bit-depths for continuous inputs, denoted *1-bit* and *15-bit* (as labeled in Fig. 4). In both cases, inputs are transferred as packed magnitudes over a 32-bit DMA stream and are expanded on the FPGA into thermometer-coded bit-vectors for BFA inference.

For the *1-bit* setting, each magnitude is a single bit, so one 32-bit DMA word packs 32 magnitudes and corresponds to 32 decoded thermometer bits (i.e., decoding is the identity mapping). For the *15-bit* setting, each magnitude is stored as a 4-bit integer in $\{0, \ldots, 15\}$, so one 32-bit DMA word packs 8 magnitudes; after on-the-fly decoding, each magnitude expands to a 15-bit thermometer code, so a single DMA word produces $8 \times 15 = 120$ decoded thermometer bits. Therefore, for a fixed decoded dimensionality $\dim(I)$, the number of 32-bit DMA transfers per sample scales as $\dim(I)/32$ for *1-bit* and $\dim(I)/120$ for *15-bit*, which explains why the *1-bit* configuration can require more transfers even though its thermometer bit-depth is lower.

In these tests, we sweep (i) the encoded input dimensionality and (ii) the number of clusters/categories to isolate how each factor affects runtime. This synthetic setup removes dataset-specific effects and highlights the dominant scaling trends expected from the streaming scan: inference time increases approximately linearly with both vector length (number of 32-bit packets) and the number of categories evaluated.

TABLE II
TRAINED MODEL CHARACTERISTICS (CLUSTER COUNT DENOTES THE NUMBER OF LEARNED CATEGORIES AFTER TRAINING WITH $\rho = 0.9$).

Dataset	Enc. dim. (bits)	Clusters
Fashion-MNIST	23,520	15,869
Mini Speech Commands	123,840	2,066
HAR	17,040	13,774

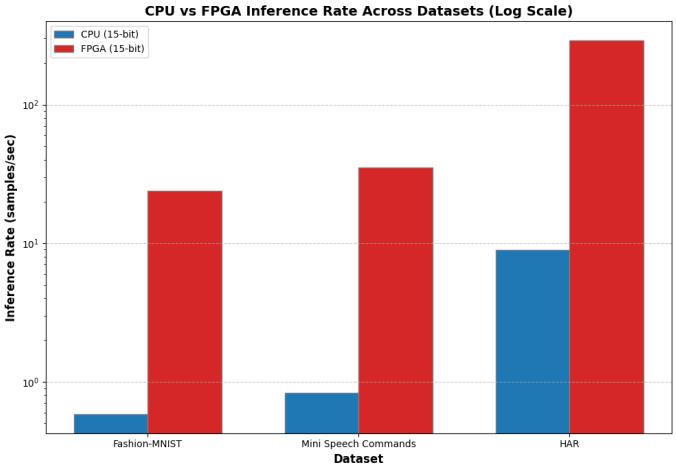

Fig. 3. Inference rate comparison on real datasets (CPU vs. BFAMod). Bars show mean inference rate over repeated trials; the y-axis is log-scaled to visualize results spanning multiple orders of magnitude.

V. RESULTS & DISCUSSION

A. Clustering Tasks

Fig. 3 compares the inference rate of the CPU and BFAMod across three real-world workloads (Fashion-MNIST, HAR, and Mini Speech Commands). The results show that BFAMod consistently outperforms the CPU baseline, achieving more than an order-of-magnitude higher inference rate across all three datasets. This consistent speedup aligns with the expected hardware mapping described in Sections II–III: the dominant computations in BFA inference reduce to bitwise operations and popcount accumulation (Eq. (1)), while division is avoided through cross-multiplication (Eq. (3)). As a result, BFAMod can sustain a high-throughput streaming pipeline over category scans, whereas the CPU baseline incurs higher per-category overhead for the same operations.

The relative performance across datasets depends primarily on (i) the encoded input dimensionality after preprocessing/encoding and (ii) the number of learned clusters/categories, since both factors increase the amount of data streamed and the number of category evaluations per input. Table II summarizes these trained-model characteristics for the three real workloads in Fig. 3.

B. Variable Cluster and Dimension Testing

Fig. 4 characterizes how inference rate changes as model size increases, sweeping both encoded input dimensionality and the number of clusters/categories. These experiments reveal key characteristics of both approaches.

For models with a low-dimensional input vector or few clusters running on BFAMod, the inference rate did not change significantly. This indicates that there is a constant overhead in the device's operation that limits the maximum inference rate achievable for very small models. In practice, this overhead is dominated by control-plane interactions (register writes/polls) and fixed per-inference orchestration costs, which do not scale with vector length or category count.

Additionally, in models with very few clusters, the benefit of input caching diminishes. When only a small number of categories are evaluated, reusing the cached input amortizes relatively little work, so the fixed overhead becomes more visible in the measured end-to-end time. This overhead is more pronounced in the 15-bit decoding implementation, as less data needs to be transferred to BFAMod for the same input dimension. Consequently, transfer time decreases, but control-plane overhead remains, making the constant component a larger fraction of total time.

Once this bottleneck is surpassed with a sufficiently large model, the FPGA scales at a predictable rate as both cluster count and input dimensionality increase. This is consistent with a streaming scan cost: total work grows approximately linearly with (i) the number of 32-bit packets required to represent the encoded input and (ii) the number of categories evaluated. In this regime, BFAMod's datapath approaches steady-state utilization and DMA becomes the dominant contributor to end-to-end runtime.

The CPU implementation outperformed BFAMod on models with low input dimensionality or few clusters while scaling predictably; however, models of these sizes are unlikely to be effective in real-world use cases. For practical deployments where a larger category set is required, BFAMod achieves substantially higher throughput, matching the trend observed in the real-dataset experiments.

In addition, we observe that for the same amount of data (in terms of feature magnitudes), inference rate can vary. This is a result of magnitude encoding, which can encode varying amounts of information within the same 32-bit packet. Specifically, different magnitude patterns can lead to different effective decoding workloads and memory access patterns, introducing secondary variation around the main scaling trend.

Overall, these results demonstrate that BFAMod is capable of outperforming CPU implementations for practically sized BFA models, reinforcing the results of our clustering experiments.

VI. CONCLUSION

BFAMod is a CPU–FPGA accelerator for rapid BFA inference on embedded MPSoCs, designed to address the dominant cost in practical BFA workloads: repeated category scanning over high-dimensional encoded inputs. By streaming magnitude-coded data via DMA, performing on-the-fly magnitude-to-thermometer decoding, computing match statistics with word-level AND and popcount accumulation, and selecting categories using division-free cross-multiplication, BFAMod achieves more than an order-of-magnitude higher

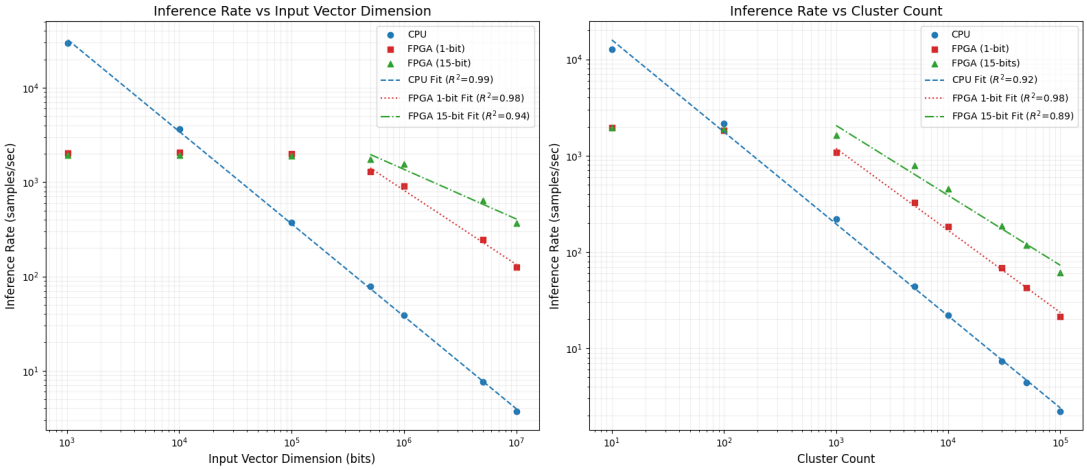

Fig. 4. Scaling results over encoded input dimensionality and cluster count (CPU vs. BFAMod). Each point shows the mean inference rate across repeated trials. Lines indicate trend fits in the steady-state regime; selected FPGA points at very small model sizes are excluded from the fit where fixed control-plane overhead dominates (see discussion).

end-to-end inference rate than an ARM Cortex-A53 baseline on three real datasets. Controlled scaling sweeps further show predictable throughput trends as category count and encoded dimensionality increase, supporting BFAMod's suitability for larger, practically sized models.

The current evaluation focuses on inference throughput on a single MPSoC platform. Reported power values are tool-estimated rather than externally measured, and we do not yet provide a full latency breakdown separating control-plane overhead, DMA transfer time, and FPGA compute time. In addition, online model updates are host-managed and have not been quantified end-to-end.

We will (i) perform board-level power/energy measurements under representative workloads, (ii) report detailed latency breakdowns and optimize the control path for small-model regimes, and (iii) evaluate the complete online-learning loop by measuring host-side update costs and weight write-back/streaming overhead. We also plan to explore portability to other MPSoCs and architectural extensions such as partial on-chip weight caching and multi-engine parallelism for very large category sets.

Acknowledgment

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-22-2-0209. This research was supported by NSF grant 2420248 and by the Kummer Institute, Mary Finley Endowment, and Intelligent Systems Center of the Missouri University of Science and Technology.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

The computation for this work was performed on the high-performance computing infrastructure provided by Research Support Solutions at Missouri University of Science and Technology https://doi.org/10.71674/PH64-N397

References

[1] J. Dongarra, J. Gunnels, H. Bayraktar, A. Haidar, and D. Ernst, "Hardware Trends Impacting Floating-Point Computations In Scientific Applications" *arXiv* [math.NA]. 2024. [Online]. Available: https://arxiv.org/abs/2411.12090

[2] N. Melton, S. Petrenko, J. Liu, J. Schroll, M. Renfrow, I. Sandjaja, and D. Wunsch, "A lightweight binary ART model for resource-constrained online learning," accepted for publication in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, IEEE WCCI, 2026.

[3] Y. Hao, "A general neural network hardware architecture on FPGA," *CoRR*, vol. abs/1711.05860, 2017.

[4] O. I. Bureneva, M. S. Prasad, and S. Verma, "FPGA-based Hardware Implementation of the ART-1 classifier," in *2023 XXVI International Conference on Soft Computing and Measurements (SCM)*, 2023, pp. 171–174.

[5] O. I. Bureneva, Y. B. Ibrahim, S. Verma, and M. S. Prasad, "Hardware Implementation of the ART-2 Classifier," in *Proc. V Int. Conf. Neural Networks and Neurotechnologies (NeuroNT)*, St. Petersburg, Russia, 2024.

[6] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, no. 6, pp. 759–771, 1991.

[7] Advanced Micro Devices, Inc., "PYNQ: Python Productivity for Zynq", 2025. [Online]. Available: https://github.com/Xilinx/PYNQ.

[8] N. M. Melton, D. Tanksley, and D. C. Wunsch, "Adaptive resonance lib: A Python package for adaptive resonance theory (ART) models," *Journal of Open Source Software*, vol. 10, no. 114, p. 7764, 2025.

[9] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST," Zalando Research, Berlin, Germany, 2017. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist

[10] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human Activity Recognition Using Smartphones Data Set," UCI Machine Learning Repository, 2013. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

[11] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *arXiv preprint arXiv:1804.03209*, Apr. 2018. [Online]. Available: http://arxiv.org/abs/1804.03209

[12] K. J. Piczak, "Environmental sound classification with convolutional neural networks" in *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2015, pp. 1–6.