Journal of Electronics, Electromedical Engineering, and Medical Informatics
Multidisciplinary: Rapid Review: Open Access Journal **Vol. 7, No. 2, April 2025, pp: 431-449; eISSN: 2656-8632**

RESEARCH ARTICLE **OPEN ACCESS**

Manuscript received November 15, 2024; Revised January 10, 2025; Accepted March 1, 2025; date of publication March 27, 2025
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v7i2.681

How to cite: Bhupesh Deka, Sayanti Chatterjee , S Rao Chintalapudi, Jajala Nikitha, Lakshminarayana Kodavali, Kancharagunta Kishan Babu, "Optimized PCA Infused Liquid Neural Network Deployment for FPGA-Based Embedded Systems", Journal of Electronics, Electromedical Engineering, and Medical Informatics, vol. 7, no. 2, pp. 431-449, April 2025.

Optimized PCA Infused Liquid Neural Network Deployment for FPGA-Based Embedded Systems

Bhupesh Deka[1], Sayanti Chatterjee[1], S Rao Chintalapudi[2], Jajala Nikitha[3], Lakshminarayana Kodavali[4], Kancharagunta Kishan Babu[1]

1 Department of Computer Science and Engineering (AIML & IoT), Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering &Technology, Hyderabad, India, 500090
2 Department of Computer Science and Engineering (AI&ML), CMR Technical Campus, Hyderabad, India,
3 Department of Emerging Technologies (DS,CyS,IoT), Mallareddy College of Engineering and Technology (MRCET), Hyderabad, India,
4 Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation (KLEF), Guntur, AP, India

Corresponding author: Sayanti Chatterjee (sayanti_ch@vnrvjiet.ac.in)

The authors express their heartfelt gratitude to the institutions: VNR Vignana Jyothi Institute of Engineering & Technology for their unwavering support and resources

ABSTRACT The integration of neural networks into FPGA-based systems has revolutionized embedded computing by offering high performance, energy efficiency, and reconfigurability This paper introduces a novel optimization framework integrating Principal Component Analysis (PCA) to reduce the complexity of input data while preserving essential features for accurate neural network processing. By applying PCA for dimensionality reduction, the computational burden on the FPGA is minimized, enabling more efficient utilization of hardware resources. Combined with hardware-aware optimizations, such as quantization and parallel processing, the proposed approach achieves superior performance in terms of energy efficiency, latency, and resource utilization. Simulation results demonstrate that the PCA-enhanced Liquid Neural Network (LNN) deployment significantly outperforms traditional methods, making it ideal for edge intelligence and other resource-constrained environments. This work emphasizes the synergy of PCA and FPGA optimizations for scalable, high-performance embedded systems. A comparison study using simulation results between cascaded feed forward neural network (CFFNN), deep neural network (DNN) and liquid neural network (LNN) has been encountered here for the embedded system to show the efficacy of PCA based LNN. It has been shown from case studies that the average F1score is 98% in case of proposed methodology and accuracy is also 98.3% for high clock value.

INDEX TERMS CFFNN, DNN, FPGA, Liquid neural network, Principal component analysis, Reconfigurable hardware,

I. INTRODUCTION

With the rapid advancement of embedded and mobile systems, a nascent domain of inquiry within data mining has emerged, focusing on streamlined software code and compact hardware architecture[1], [2]. In the contemporary landscape, data mining has assumed a pivotal role across various sectors, encompassing scientific research, medical diagnostics, marketing, biotechnology, multimedia, security, finance, among others. In the present era, the tasks associated with data management and data mining are increasingly characterized by computational complexity and significant data intensity[1],[3], [4]. These tasks necessitate considerable data processing capabilities. Moreover, in various contexts, the real-time data must be managed effectively to derive the genuine benefits. These limitations significantly impact the performance accuracy and speed of embedded system applications. To alleviate the demands and constraints present in portable, embedded devices, and to enhance the efficiency of applications on these platforms, it is imperative to integrate certain hardware within software and hardware system

architectures. The engineered hardware is specifically tailored to provide superior performance speed, area efficiency, and reduced power consumption [5] in comparison to the analogous software executed on microprocessors.

In recent years, machine learning has emerged as a preeminent domain in a multitude of research applications, including human activity recognition. However, the escalating demands for accuracy and the complexities associated with practical implementations render the realization of such systems challenging due to device speed constraints and energy costs[6]. To address these limitations, enhancements in efficiency alongside reductions in power consumption have prompted increased interest in the reconfigurable utilization of hardware[7].

The proposed architecture's design, implementation, and validation have been meticulously formulated to address existing deficiencies while enhancing both the speed and overall performance of the proposed system, all while maintaining exceptionally low power consumption levels. The objectives of this proposed work are to develop and assess a comprehensive reconfigurable framework that facilitates run-time adaptability through the application of machine learning (ML) methodologies.

The principal cognizance of this architectural framework is predicated upon a machine learning algorithmic platform that augments classifiers operating on a general-purpose processor, by explicitly mapping the computationally intensive components that are executed by reconfigurable hardware. This methodology not only amplifies the performance of the architecture but also facilitates an increase in the embedded applications that will constitute the units functioning as co-processors within general-purpose systems.

Machine learning constitutes a special domain that entails instructing machines on how to acquire knowledge, possessing the capacity to swiftly process extensive volumes of data through mathematical computations; consequently, machine learning has come out as a formidable instrument in contemporary times. Nevertheless, a novel generation of computational methodologies referred to as reconfigurable computing is now positioned to elevate machine learning to unprecedented levels.

Computing integrates the speed of ICs with the flexibility of processors. Reconfigurable instruction set processors are one of the widely used reconfigurable computing products and have been developed using several design approaches. Their performance relies on the configuration units, which update the processor settings dynamically. To improve the performance of reconfigurable processors, a configuration design has been created that maximizes the reusability of existing configuration streams. This architecture allows for the loading of the most ideal configurations, which results in increased efficiency and performance. In some fundamental applications, a fixed processor design is a good choice. However, unknown applications have such a diverse set of algorithms that a fixed standard will fail to meet the desired process speed. It is quite tough to design specialized hardware. Machine Learning Applications with Diverse Algorithms demand specialized hardware and are more expensive than reconfigurable solutions.

Computing paradigm research relies heavily on reconfigurable computing technologies. Reconfigurable computing provides performance and flexibility in gaining on a single computer system. The performance of the reconfigurable systems depends much on the management of configuration. Configuration Management improves the processing power in reconfigurable computing systems. Advanced control and management [8]strategies drive the progress of technology [9]. Configuration techniques facilitate innovation in multiple switching, partial reconfiguration, configuration cloning, and configuration pipelining-these are key efficient parameters of reconfigurable computing system

As article states, reconfigurable computing is comprised of three key components: architectures, design methodologies, and applications. Modern architectural trends are focused on coarse-grained fabrics, heterogeneous functionalities, and soft cores. Coarse-grained fabrics use larger reconfigurable logic blocks that can process information more efficiently. Heterogeneous functionalities allow for greater customization and optimization of specific tasks. Soft cores leverage pre-existing processor architectures that can be adapted to meet the unique needs of applications like media processing, numerical computing, and embedded systems.

Paper [9], [10]]examines the effect of FPGA embedded array topologies on their logic execution capability. This research is based upon several architectures having different sizes of memory arrays and evidences the heavy impact of heterogeneous architectures on logic implementations in contrast to single-size memory arrays.

Zippy is a hybrid CPU that features a multi-context reconfigurable array and stands out for its built-in hardware virtualization capabilities. It utilizes a technique called virtualized execution for hardware virtualization. Study [11], [12]highlights a strategy known as temporal partitioning, while works [5-7] discuss advancements in digital electronics technologies, which have significantly enhanced computing power. In the field of AI, there is a strong focus on deep learning (DL) as a transformative area of research and application.

The fundamental purpose of this work is to provide an efficient solution for embedded systems to improve computing performance and data-intensive applications, such as data gathering on mobile or embedded devices. This unique optimized method for data extraction for embedded hardware architectures uses probabilistic principal component analysis (PPCA) [13], [14] . In this process, the true data is transformed into a new dataset while the key attributes remain unchanged. PCA [15] has previously been described in the literature. PCA has produced superior results. This PCA - based data mining has been integrated into image processing of biometric data to improve efficiency and reduce memory access on embedded platforms.

The second portion of the work focuses on data recognition and classification for biometric images. This is the most efficient benchmark, which can be clarified using NN [8], [16]. This research investigates the Cascaded feed forward NN and deep NN as image processing models[16], [17], [18]. Accurate recognition of provided graphical objects should be robust to scale, translation, and rotation of the input [22]. To include these properties into a neural network, the algorithm selection must be appropriate.

For the better understanding different metrices have been compared for PCA and other optimization techniques as well as proposed neural networks. The performance matrices which are needed for comparison has been listed as Metric 1-6. Metric 1 is about Performance accuracy and Metric 2 is the Latency which can be defined as time taken to process the data (in seconds or milliseconds). Metric 3 is basically memory usage which is basically usage over available memory. Metric 4 is about Power consumption in watts or joules during data processing. Metric 5 is nothing but (Overall Performance): Combined performance metric or computation efficiency. The last but not the least is Metric 6 (Execution Time): Time taken for the model to learn from the training data.

The Key Considerations for calculating the above metrics are namely Dataset Characteristics, Application Requirements and Trade-offs. Dataset Characteristics can be defined as the choice of metrics depends on the specific dataset and the problem being addressed. Application Requirements means prioritize metrics that are most relevant to the application's needs (e.g., latency for real-time systems, energy efficiency for mobile devices). Trade-offs: defines between different performance metrics (e.g., accuracy vs. speed, accuracy vs. resource usage). By carefully selecting and tracking these metrics, you can effectively evaluate and compare the performance of PCA and various neural network architectures. The main research gap for embedded system is Robustness and Reliability [5], [19], [20]and the other one is the hardware-software co-design [21]. Efficient deployment requires co-optimization of hardware and software for embedded neural networks, though it is found in some literatures [22], [23], [24]

The present authors have planned to resolve the problem. The feature extraction has been done with PCA which is already used but not for input of liquid neural network. In the work of [13], [25] PCA is used but the application domains are different to check the efficacy. On the other hand embedded system is basically domain-specific optimizations for embedded neural networks (e.g., biomedical devices, smart agriculture)[26], [27] Transfer learning methods for adapting pre-trained models to resource-constrained domains is a challenging task.

Liquid neural networks (LNNs) are a promising approach to evaluate many of the challenges and research gaps identified in neural networks for embedded systems. Liquid neural networks are characterized by their adaptability and efficiency, making them well-suited for dynamic, resource-constrained environments like embedded systems [28], [29], [30]. Liquid neural networks are designed to be sparse and adaptive, which inherently reduces the computational load compared to traditional neural networks like cascaded feed forward neural network (CFFNN), deep neural network (DNN) such as [31]. The focal aim of the work to reduce the shortcomings of the above-mentioned methods and fill the research gap found. Under these circumstances, in this work the features has been extracted by PCA and then used it in the liquid neural network for better classification and optimize the algorithm for Deployment for FPGA-Based Embedded Systems. The following part of this study discusses the features and advantages of FPGA-based reconfiguration. The third part explored PCA based data mining, which were followed by the LNN, CFFNN and DNN algorithms in the fourth part. The fifth section is about the findings and ends with a conclusion.

II.FPGA-BASED METHODS FOR RE-CONFIGURATION: A BRIEF ID

Field-Programmable Gate Arrays (FPGAs) are versatile devices that can be reconfigured to implement various hardware functions. This flexibility makes them ideal for applications that require adaptability and dynamic behavior. FPGA-based reconfiguration techniques al-low for modifying the FPGA's functionality on-the-fly, enabling a wide range of applications.

The first classification pf FPGA is Partial Reconfiguration (PR). This method involves modifying a portion of the FPGA's configuration while the rest of the device remains operational. It also enables dynamic updates to specific hardware blocks without disrupting the entire system. Dynamic Partial Reconfiguration (DPR) which is another variety is basically a subset of PR that allows for reconfiguration while the FPGA is actively processing data which Offers greater flexibility and responsiveness compared to traditional PR. The third type is Behavioral Reconfiguration. This reconfiguration involves modifying the behavior of the FPGA by changing the control logic or data flow which can be achieved through: Reconfigurable logic blocks. Dynamically reconfigurable interconnects. Programmable look-up tables (PLUTs) which enables more radical changes to the FPGA's functionality compared to other Partial Reconfiguration. The for real-time face detection in a surveillance application has been taken into account for the FPGA program.

III. SYSTEM DESIGN METHODOLOGY AND IMPLEMENTATION PLATFORM

A hierarchical, platform-based design methodology is employed to facilitate component reuse and scalability. The system is structured into multiple levels of abstraction. This section deals with the implementation using the proposed methods.

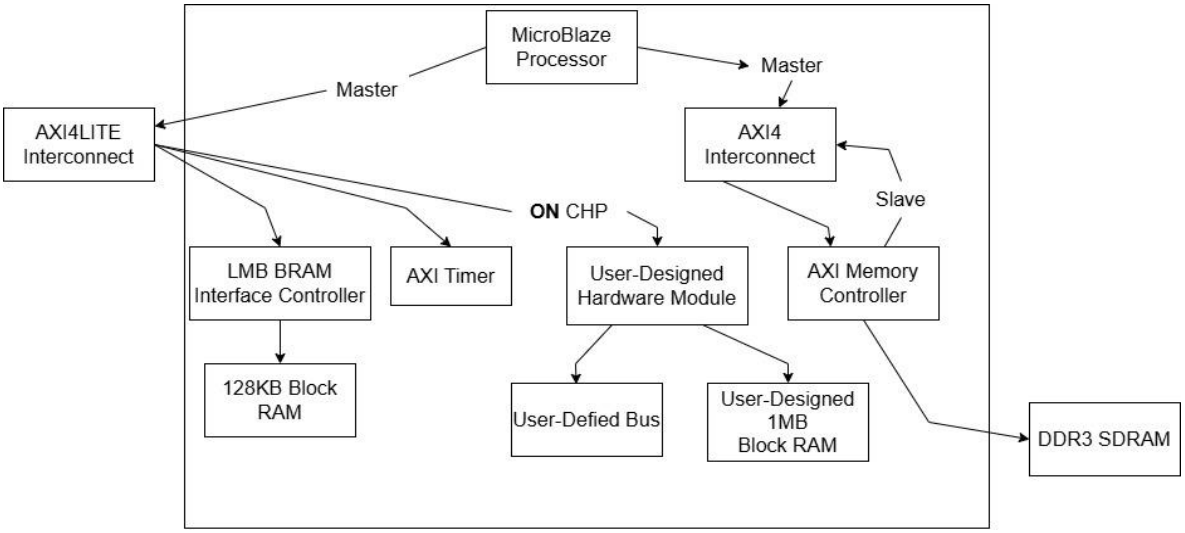

FIGURE. 1. FPGA-Based Methods for Reconfiguration

A. DESIGN METHODOLOGY

The first one is Low-Level Operators which performs basically fundamental operations such as addition, subtraction, multiplication, division, square root, and comparison are defined as the base of the design hierarchy. Higher-Level Modules are mainly Computational modules, including Mean, Covariance Matrix, Eigenvalue Matrix, and Principal Component (PC) Matrix, are constructed by utilizing these low-level operators. Both hardware and software implementations of these operations are developed to ensure flexibility and performance optimization.

B. IMPLEMENTATION PLATFORM

The current work can be classified into two work platforms, namely: Hardware Platform and Software platform. All experiments in hardware platform are conducted on the ML605 FPGA development board, which is equipped with the modules such as FPGA Device, on-chip resources, non volatile storage and clock as given in TABLE 1.

TABLE 1
Hardware Implementation platform details

Module Name	Specification
FPGA Device	A Xilinx Virtex 6 XC6VLX240T-FF1156.
On-Chip Resources	37,680 slices for logic implementation. 2 MB BRAM (Block Random Access Memory). 512 MB external DDR3-SDRAM for handling large datasets.
Non-Volatile Storage	128 MB Platform Flash XL. 32 MB BPI Linear Flash. 2 GB Compact Flash for configuration bitstreams
Clock	**Clock Speed**: FPGA modules operate at 100 MHz.

Hardware modules are designed using VHDL and Verilog. Verification of designs is performed using tools such as ModelSim SE, Xilinx ISim, and ChipscopePro 14.7. Design synthesis and implementation are carried out using Xilinx ISE 14.7 and XPS 14.7. Software Platform consists of Software modules which are executed on the MicroBlaze soft processor, synthesized onto the FPGA. The soft processor is configured to run at 100 MHz and synthesized using the FPGA's general-purpose logic. Unlike hard processors such as PowerPC, the MicroBlaze processor is synthesized to fit within the FPGA's gate arrays.

A. PERFORMANCE METRIC

In an embedded system, "Speedup" has been discussed in Eq. 1. is a metric used to evaluate the performance improvement offered by hardware over software. It is calculated by dividing the improved execution time achieved through hardware by the baseline execution time taken by the software as in Eq. 1[13]. A higher speedup value indicates that the hardware implementation is more efficient compared to the software version. For instance, if a task takes 10ms in hardware and 50ms in software, the speedup would be 5, meaning the hardware is five times faster. This metric helps in making informed decisions about whether to use hardware or software for specific tasks.

$$\text{Speedup} = \frac{ImprovedExecutionTime(Hardware)}{\text{Baseline xecution ime (Software)}} \quad (1)$$

B. BENCHMARK DATASET

The dataset contains 3823 records (vectors), each with 64 attributes. Data includes 200 handwritten characters collected from 43 individuals. The details of ML605 FPGA development board, has been shown in FIGURE 1. The schematic diagram has been used in [13], [15] has been reused in this work to advocate the efficacy the proposed method. The synthesis has been carried away by hardware as well as the software platform. The output dataset of the system has been used for further work. PCA has been used

here for feature extraction for the fingerprint data carried out using Xilinx ISE 14.7 and XPS 14.7. Software modules used are written in Python and executed on the MicroBlaze soft processor, synthesized onto the FPGA.

IV. PCA BASED DATA MINING FOR EMBEDDED SYSTEM

This section has been dealt with two different headers. The first section addresses the common features of PCA and the second part advocates about the application of PCA in the light of embedded system and the specific paradigm of PCA for the current application

A. *GENERALIZED PCA BASED DATA MINING*

PCA is a preprocessing tool. This PCA version focuses on the best combination of variables to reduce data redundancy. It is applied to shrink large datasets obtained from tests, thereby minimizing information loss. An important feature of PCA is that it reduces dimensionality by grouping smaller datasets without altering the core information.

The motivation behind PCA is that smaller datasets are easier to explore, visualize, and analyze. This makes them suitable for neural networks and other machine learning algorithms by eliminating unnecessary variables, thus improving processing speed and reducing memory usage. PCA steps are given in Eq. (2-4) [15], [21] . First step deals with the standardization of the Data**.** PCA is sensitive to the scale of data, so the first step is to standardize the dataset by scaling the features to have zero mean and unit variance. This ensures that each feature contributes equally to the analysis.

$$\text{Xscaled} = \text{X} - \mu\sigma \tag{2}$$

where: *X* is original data; μ is mean of the data; σ is standard deviation of the data. In the second step Covariance Matrix has been calculated to understand the relationships between different features in the dataset. The covariance matrix captures the variance and correlation between features. After that: Performance of eigened composition on the covariance matrix has been done to extract its eigenvalues and eigenvectors. eigenvalues represent the magnitude of variance along the principal components. Eigenvectors represent the direction of the principal components.

The step deals with the selection of Principal component. The eigenvalues have been sorted in descending order and selected the top k eigenvalues and their corresponding eigenvectors. The selected components will explain the majority of the variance in the dataset. Explained Variance Ratio is calculated to decide how many components to retain:

$$Explained\ \ Variance\ Ratio = \sum j = d\lambda j \tag{3}$$

where λi is the eigenvalue of the *i-th* component. The components are chosen so that such that the cumulative explained variance meets a desired threshold (e.g., 95%).

In the last step data projected has been carried out Transform the original data onto the new k-dimensional feature space using the selected eigenvectors (principal components).

$$\text{XPCA} = \text{X} \cdot W\text{X} \tag{4}$$

where X is Standardized data; W is Matrix of selected eigenvectors (principal components) and the *PCA* is the principal component analysis.

B. *PCA FOR Embedded System*

As discussed, prior, Principal Component Analysis (PCA) is a commonly used technique for dimensionality reduction in multivariate statistical analysis. In the context of embedded systems, PCA can be particularly useful for handling large datasets from sensors or other input devices by reducing the amount of data that needs to be processed, stored, or transmitted.

In this section, the mathematical model of PCA has been delved deeper. For high-dimensional data, Sparse PCA has been introduced here to reduce computational complexity while retaining essential features. Sparse PCA is considered for embedded device (e.g., a microcontroller or FPGA) that collects sensor data with hundreds of features. By applying Sparse PCA, the dimensionality of the data can be reduced, and the key features of interest can be preserved while using less computational power. The sparsity of the principal components will allow the system to handle operations like classification or anomaly detection more efficiently, since the sparse data representation leads to faster matrix operations and lower memory requirements.

The mathematical modelling for Sparse Principal Component Analysis (Sparse PCA) involves an optimization problem that balances between maximizing explained variance and encouraging sparsity. Here's a detailed step-by-step process of how it can be implemented, particularly in the context of embedded systems. The goal is to find sparse principal components while retaining as much variance as possible. The sparse PCA problem can be formulated as in Eq.5 [15].

$$max\ W \in Rp \times k \qquad \text{s.t.} \quad |W|_0 \leq s \tag{5}$$

where X is the data matrix, W is the projection matrix (principal components), ‖ F ‖ denotes the Frobenius norm. ‖ O ‖ denotes the sparsity constraint (the number of non-zero elements), S is the sparsity level. In the context of embedded systems, Sparse PCA helps reduce the dimensionality of the data while maintaining meaningful features, making the data processing more efficient and suitable for hardware implementation constraints. The process involves balancing between retaining the maximum variance and ensuring that the principal components have sparse representations. This balance is crucial for optimizing performance and resource usage in embedded systems. Consequently, Sparse PCA can enhance real-time processing capabilities and prolong the lifespan of embedded devices. Moreover, the technique supports scalability, allowing the system to handle increasing amounts of data without significant performance degradation.

This makes Sparse PCA a powerful tool for modern embedded AI applications. Steps involved in Sparse PCA :

V. LIQUID NEURAL NETWORK: A BRIEF IDEA

Input: Data matrix $X \in R^{n \times p}$, Sparsity parameter s>0s > 0s>0

Step 1: ***D*** Compute the mean μ of each feature in the dataset *X* and center the data

$$Xcentered = X - \mu$$

Step 2: Compute Covariance Matrix *C* for ' *n'* no. of data

$$C = \frac{1}{n-1} X_{centered}^{T} X_{centered}$$

Step 3: Use convex relaxation techniques like Lasso to encourage sparsity in the solution *W*, where sss controls the sparsity level.

$$\text{trace}(W^T CW) \quad \text{s.t.} \quad |W|_1 \leq s$$

Step 4: Iterative Algorithms Repeatedly compute the principal components with a threshold to enforce sparsity

Step 5: Repeat until convergence. The final *W* contains sparse principal components that highlight key features.

A Liquid Neural Network (LNN) is a type of recurrent neural network that processes data sequentially and continuously, adapting its structure based on new inputs. Unlike traditional neural networks, LNNs can handle variable-length inputs and maintain memory of past inputs, making them particularly effective for time-series data. Liquid Neural Networks (LNNs) are based on continuous-time recurrent neural networks (CTRNNs) and are defined by differential equations. Here are some key mathematical equations and algorithms used in LNNs as in Eq. (6,7) [31]. The first one is Ordinary Differential Equations (ODEs). Ordinary Differential Equations (ODEs) are a type of equation used to describe the behavior of dynamic systems. In the context of a network, the state of the system is represented by the state vector xx, which changes over time. The state of the network is described by a set of ODEs:

$$\frac{dx}{dt} = \text{f}(x, u, t) \tag{6}$$

Eq. (6,7)[31] defines the rate of change of the state vector *x(.)* with respect to time *t*. In this equation, *u(.)* represents the input vector that influences the system, and *f(.)*is a nonlinear function that relates the state and input vectors to the rate of change of the state. This nonlinear function encapsulates the dynamics of the network and governs how the state evolves over time in response to inputs. The second method in this aspect is Runge-Kutta Methods. For solving ODEs, methods like the Runge-Kutta (RK) method are often used:

$$\text{x}_{\text{n+1}} = \text{x}_{\text{n}} + \Delta\,\text{t} \cdot \text{k} \tag{7}$$

where **k** is the RK coefficient calculated based on intermediate steps *t* . Liquid Neural Networks (LNNs), such as Liquid Time-Constant Networks (LTCs), are mathematically grounded in the use of differential equations to model the dynamics of neurons and their changing behavior over time. Here's a breakdown of the core equations that describe LNNs. Core Neuron Dynamics evolves over time according to a system of ordinary differential equations (ODEs) as in Eq. (8)[31].

$$\text{dh(t)dt} = \text{f(h(t), x(t), W, }\theta) \tag{8}$$

where h(t) is the hidden state of the neuron at time *t.,x(t).* The input signal at time t,*W:* Weight matrices (input-to-hidden and hidden-to-hidden connections). θ: Parameters defining the dynamics (e.g., time constants, biases*),f:* A non-linear function that governs the evolution of the system. Liquid Time-Constant Networks, the key idea is that **the time constants** of the neurons vary dynamically. The ODE for each neuron can be stated as Eq. (9,10) [31].

$$\text{dhi(t)/dt} = -\text{hi(t)} + \text{g(W} \cdot h(\text{t}) + \text{U} \cdot x(\text{t}) + \text{b)} \tag{9}$$

where hi(t*)* is the state of the *i-th* neuron at time *t*, τi(·). The time constant for the *i-th* neuron, which is a function of the neuron state and input. *g(·):* The activation function (e.g., sigmoid, tanh, ReLU). *W:* The recurrent weight matrix. *U*: The input weight matrix. *b:* The bias term. The liquid property comes from the time constants τi, which can vary non-linearly based on the neuron's state and input.

Neuron time constant dynamics τi are modeled by the state *h(t)* and input *x(t).* Parameters α and β control the time constant's range, while the sigmoid function σ ensures τi remains positive. Weights *v* and *w* adjust contributions from the state and input.

$$\tau i(h(t), x(t)) = \alpha + \beta \cdot \sigma(v \top h(t) + w \top x(t) \tag{10}$$

where α,β are parameters controlling the range of the time constant, σ(·) is a sigmoid function to ensure τi stays positive, *v*,*w* are learnable weights for the state and input contributions. The overall output y(t) of the network is computed using the hidden states.

$$y(t) = houtput \tag{11}$$

where *houtput(·)* is the readout function, which could be a simple linear transformation or another non-linear mapping as in Eq. (11) [31]. For the sake of discretization for Computation, in practice, these continuous ODEs are solved numerically using time-stepping methods (e.g., Euler or Runge-Kutta). For a small time step Δt, the hidden state update becomes:

$$h(t + \Delta t) = h(t) + \Delta t \cdot dh(t) \tag{12}$$

where $dh(t/)dt$ is calculated using the above ODE equations as in Eq. (12)[28], [31]. The last step is Training Liquid Neural Networks. Training LNNs involves backpropagation through time (BPTT) or adjoint sensitivity

methods (for efficient computation with ODEs. These LLN equations capture the fundamental behavior of Liquid Neural Networks and their adaptability, making them powerful for tasks requiring dynamic, real-time responses to evolving inputs. The details flowchart has been discussed in FIGURE 2.

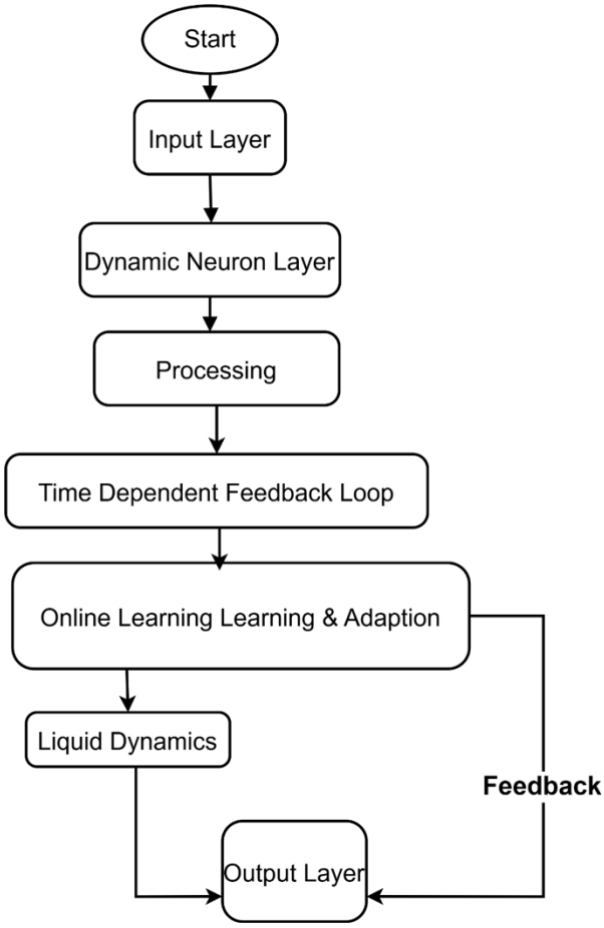

FIGURE 2. **Flowchart for Liquid neural network**

VI. PROPOSED ALGORITHM OVERVIEW

To optimize FPGA-based embedded systems, start with data collection from sensors. Preprocess data using normalization, as in Eq.13 [28], [30]. Apply PCA by computing covariance matrix performing eigen decomposition and selecting top components Design a Liquid Neural Network (LNN) with an input layer and readout layer (Eq. 18). Optimize hardware by applying quantization, parallel processing, efficient memory storage, and reducing power consumption. Finally, deploy and evaluate the model. Combining PCA and LNN ensures fast, power-efficient AI processing suitable for embedded applications. The algorithm can be classified into five steps which have been elaborated in this section. At first, data X has been collected from embedded system sensors Data Preprocessing has been performed as in Eq. (13) [31] to normalize it. For feature extraction, Principal Component Analysis (PCA) are discussed in Eq. (14-16) [31], [32]. First the covariance matrix has been computed as in Eq. 13 [29]where it scales each feature of the dataset X to a range between 0 and 1 using min-max normalization. Covariance matrix has been calculated and eigen value decomposition also performed as in Eq. 14 [30] where covariance matrix C has derived from the normalized data X' where n is the number of data points (rows). Eq. 15 [33], eigen decomposition of the covariance matrix C. V contains eigen vectors (principal directions or axes). Λ is a diagonal matrix with eigen values .The top principal components PC have been selected using Eq.16 [28], [30]. by multiplying the normalized data X' by the top k eigenvectors V_k (columns of V)

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{13}$$

$$\mathrm{C} = \frac{1}{n-1} X'^{T} X' \tag{14}$$

$$\mathrm{C} = \mathrm{V}\Lambda V^{T} \tag{15}$$

$$\mathrm{PC} = X' V_k \tag{16}$$

After successful feature extraction Liquid Neural Network (LNN) have been employed successfully. The steps for liquid neural network have been discussed as in Eq. (17-19) [30]. As discussed in prior section the steps of LNN are as follows (ALGORITHM 1):

ALGORITHM 1: Steps of LNN

Input Layer: Embed reduced data:

$$\mathrm{u}(t) = \mathrm{PC} \tag{17}$$

The **input signal** *u(t)* is the result of projecting your original data onto principal components (***PC***). This is a **dimension-reduced representation** of the data.

Reservoir: Process data:

$$x(t + \Delta t) = Ax(t) + Bu(t) \tag{18}$$

where, *x(t)* is the reservoir's internal state at time t.
A is the reservoir weight matrix, dictating how the current state evolves. *B* maps the input signal *u(t)* into the reservoir. The reservoir captures temporal and nonlinear dynamics of the input data**.**

Readout Layer: Map reservoir states

$$y(t) = Cx(t) \tag{19}$$

The reservoir's internal state *x(t)* is mapped to the output *y(t)* using matrix *C*

Optimization: Use *y(t)* to optimize embedded system parameters.

Evaluation: Assess performance and iterate if necessary.

The parallel processing has been employed to accelerate computations. The main aim of using PCA induced LNN to FPGA is to reduce power consumption through weight pruning and sparsity strategies. By combining PCA and LNN, this approach enables fast, power-efficient neural network processing, making it ideal for embedded AI applications.

A. THE SELECTION OF PARAMETERS FOR A PCA-BASED LIQUID NEURAL NETWORK (LNN)

In FPGA-based embedded systems is crucial to achieving a balance between performance, resource efficiency, and accuracy. Key parameters which have been used in this current work has been jotted down in this work. The first one is PCA Parameters. A typical choice is to retain 90-95% of the data variance, ensuring that important features are preserved while reducing computational load. The second important parameter is Sparsity Regularization Parameter (λ). In Sparse PCA, the regularization parameter λ controls the sparsity of the principal components. A higher λ increases sparsity, reducing resource usage by eliminating less important features. The another Neural Network Parameters is Learning Rate (α) which governs the magnitude of weight updates during training. A rate between 0.001 and 0.01 is commonly selected. The other parameter, batch size determines the number of samples processed together in one pass during training. Typically, values of 16 or 32 are selected for embedded systems to balance memory usage and parallelism on the FPGA. Larger batch sizes improve parallel computation, while smaller sizes reduce memory load.

For FPGA systems, LNNs typically have fewer layers and neurons to optimize resource usage. A shallow architecture with 20-100 neurons per layer is commonly chosen. While more layers improve model capacity, they increase resource demands, so fewer layers are selected for better efficiency.

Quantizing weights and activations to 8-bit or 16-bit precision reduces memory and computational overhead on FPGA hardware. While lower bit-widths reduce resource usage, excessive quantization may reduce accuracy.
For the sake of Activation Parallelism, the ReLU activation function is commonly used due to its computational simplicity, essential for efficient FPGA implementation.

The parameters of the PCA-based LNN are selected to reduce FPGA resource usage while maintaining high accuracy, achieving 98.3% accuracy and 98% F1 score in simulation results. These choices optimize the network for embedded, real-time systems.

FIGURE 3 showed here gives the UML diagram of the proposed method as this diagram clearly visualizes system architecture and design, aiding efficient communication and understanding among stakeholders. This diagram effectively outlines the relationships between different system modules, their interactions, and data flow, making it easier for developers, analysts, and other stakeholders to understand the system's functionality. A key feature of LNNs is the **Time-Dependent Feedback Loops**, allowing continuous learning and adaptation. The **Online Learning** mechanism refines the network using **Liquid Dynamics**, which ensures flexible, real-time responses. Feedback from the output refines the learning process iteratively. Finally, the network produces results through the **Output Layer**, completing the adaptive neural processing cycle.

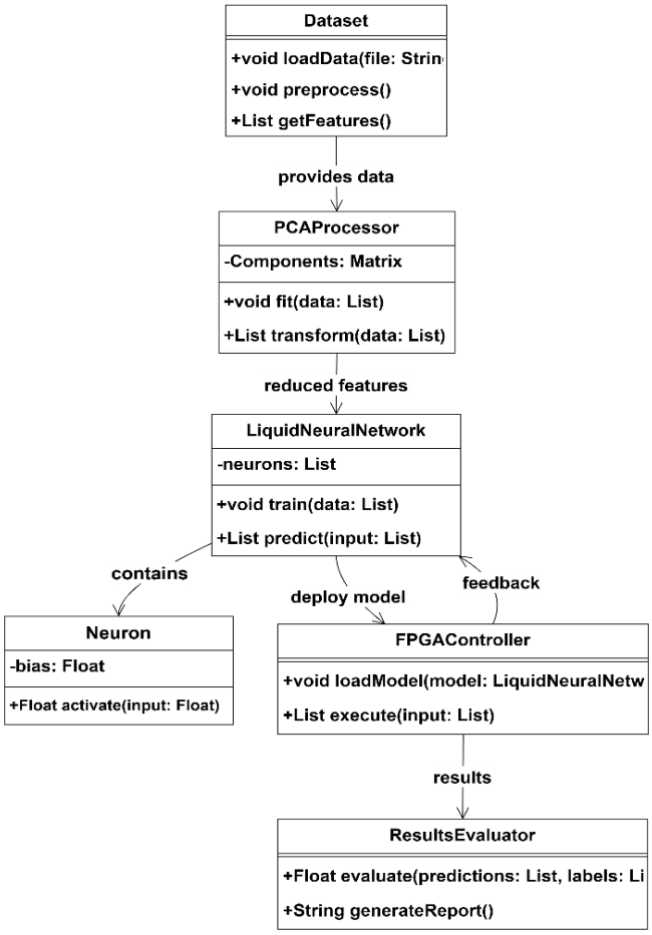

FIGURE 3. **UML Diagram of proposed method**

It starts with the Dataset, which provides data to the PCA Processor for feature reduction. The processed features are passed to the Liquid Neural Network, which consists of Neurons for computation. The trained model is deployed to an FPGA Controller, which executes predictions and provides feedback. The results are evaluated by the Results Evaluator, which assesses accuracy and generates reports. This system highlights a structured approach to real-time, adaptive learning using FPGA hardware acceleration for neural network execution and performance improvement.

VII. RESULTS

The different stages of PCA calculation for feature extraction and dimensionality reduction include the first stage (mean calculation), the second stage (Covariance Matrix), the third stage (Eigenvalue Matrix), and the fourth stage (Principal Component Matrix). The mathematical details have been discussed in TABLE 2.

TABLE 2
Summary of PCA Stages

Stage	Description	Mathematical Operation
1.Mean Calculation	Centering data by subtracting mean from each feature.	$X' = X - \mu X'$ μX represents the mean of each column (feature) in the data matrix X
2.Covariance Matrix	Computing relationships between features.	$C = \frac{1}{m-1} X'^T X'C$ This computes the **covariance matrix** *C* of the centered data *X'*, where *m* is the number of samples
3.Eigen Decomposition	Finding principal directions (eigenvectors).	$CV = \lambda V$ Solving it gives **eigenvectors** *V* and **eigenvalues** λ
4. Projection	Transforming data into lower-dimensional space.	$Xnew = X'WX$ *W* is the projection matrix

PCA involves multiple steps to reduce dimensionality. Firstly, mean calculation centers the data by subtracting the mean from each feature. Secondly, the covariance matrix is computed to understand relationships between features. Thirdly, eigen decomposition identifies principal directions (eigenvectors). Finally, the projection stage. transforms the data into a lower-dimensional space. This process simplifies complex datasets, making them easier to analyze and visualize while preserving essential information.

These stages have been experimentally tested as separate entities, with different vector counts and data sizes. A Monte Carlo simulation with 100 executions for each stage. The average execution time is given. In TABLE 3, a Monte Carlo simulation with 100 executions for each stage. The average execution time is given. It presents a comparison of execution times across all four stages with and without PCA. It also shows the Computation time for PCA and without PCA of four stages.

This table highlights the average execution times for each stage, demonstrating the impact of PCA on computation time. By incorporating PCA, we observe the efficiency in processing, as important features are retained while reducing the computational load. The comparison showcases the benefits of using PCA in optimizing performance in embedded systems.

In TABLE 3, a Monte Carlo simulation with 100 executions for each stage. The average execution time is given. It presents a comparison of execution times across all four stages with and without PCA. It also shows the Computation time for PCA and without PCA of four stages.

TABLE 4 provides insights into the trade-offs and benefits of dimensionality reduction versus raw data processing. Each of these techniques has its unique applications and strengths in handling and transforming data for better analysis and interpretation.

Principal Component Analysis) is a statistical technique used to reduce data dimensionality by transforming variables into a smaller set of uncorrelated principal components, capturing most of the original data's variance. PPCA (Probabilistic Principal Component Analysis) extends PCA by introducing a probabilistic model, allowing for robust handling of missing data and noise, making it useful in various real-world scenarios. ICA (Independent Component Analysis) focuses on separating a multivariate signal into independent components, often employed in signal processing to isolate mixed signals, such as separating audio sources in a recording. SVD (Singular Value Decomposition) is a matrix factorization method that decomposes a matrix into three other matrices, providing valuable applications in noise reduction, image

TABLE 3
Comparison of stages

Data	Comparison of stages									
	Stage 1		**Stage 2**		**Stage3**		**Stage 4**		**Total**	
	Without PCA	PCA	Without PCA	PCA	Without PCA	PCA	Without PCA	PCA	Without PCA	PCA
20K	.9365	.9355	6.15	6.07	1.803/ 332	1.659/232	1.203	1.118	1.8230	1.7843
40K	1.075	1.795	12.31	12.19	1.271 / 235	1.270/235	2.50	2.32	14.325	15.084
60K	1.575	1.795	12.31	12.19	1.271 / 235	1.270/235	2.50	2.32	14.325	15.084
80K	1.95	1.795	12.31	12.19	1.271 / 235	1.270/235	2.50	2.32	14.325	15.084
100K	2.807	2.738	18.45	18.23	1.402 / 258	1.399 / 257	3.61	3.58	21.192	24.327
120K	4.28	2.9	38.45	383	1.402 / 258	1.567 / 288	3.961	3.58	31.65	32.47

c

TABLE 4
Comparison Table for different metrics

Data Points	Method	Accuracy	Latency (ms)	Memory Usage (used/available)	Energy (Watts)	Performance Metric
20K	No Reduction	0.9410	8.20	2.415 / 332	1.420	1.7320
	PCA	0.9355	6.07	1.803 / 331	1.118	1.7843
	PPCA	0.9378	6.45	1.842 / 331	1.201	1.8654
	ICA	0.9402	7.15	1.954 / 330	1.325	1.7453
	SVD	0.9340	6.12	1.812 / 332	1.143	1.7600
60K	No Reduction	1.795	22.15	3.140 / 235	3.42	14.221
	PCA	1.795	12.19	1.270 / 235	2.32	15.084
	PPCA	1.812	12.60	1.321 / 235	2.57	15.612
	ICA	1.839	13.18	1.419 / 234	2.81	14.784
	SVD	1.783	12.24	1.294 / 235	2.43	14.912
100K	No Reduction	2.795	41.23	4.123 / 257	4.90	21.000
	PCA	2.738	18.23	1.399 / 257	3.58	24.327
	PPCA	2.761	18.61	1.431 / 256	3.72	24.821
	ICA	2.784	19.30	1.562 / 255	4.01	23.512
	SVD	2.734	18.35	1.405 / 256	3.62	24.300

TABLE 5
Statistical Analysis

Metric	(PCA+LNN vs. CFFNN)			(PCA+LNN vs. DNN)			(PCA+LNN vs. LNN)		
Dataset Range (Samples)	**Mean Difference**	**t-Value**	**p-Value**	**Mean Difference**	**t-Value**	**p-Value**	**Mean Difference**	**t-Value (**	**p-Value**
0 - 20K	+4.7	16.5	<0.01	+3.0	14.8	<0.01	+1.2	9.69.	<0.01
20K - 40K	+4.3	15.8	<0.01	+2.88	14.1	<0.01	+1.1	9.39	<0.01
40K - 80K	+4.1	15. 5	<0.01	+2.5	13.9	<0.01	+1.0	9.19.	<0.01
80K - 120K	+4.6	16.3	<0.01	2.3	13.5	<0.01	+0.8	8.78.	<0.01

TABLE 6
Clock for different attributes comparison

Attributes	PCA enhanced LNN (in clks)	CFFNN (in clks)	DNN (in clks)
Clock cycles for segmenting one large receptive field (St)	2612	2497	2520
Clock cycles for calculating one large receptive field (Ct)	710	682	703
Feed forward layer clks for each image (Ft)	302	294	300
Clock cycles for processing one image.	33212	318494	32434

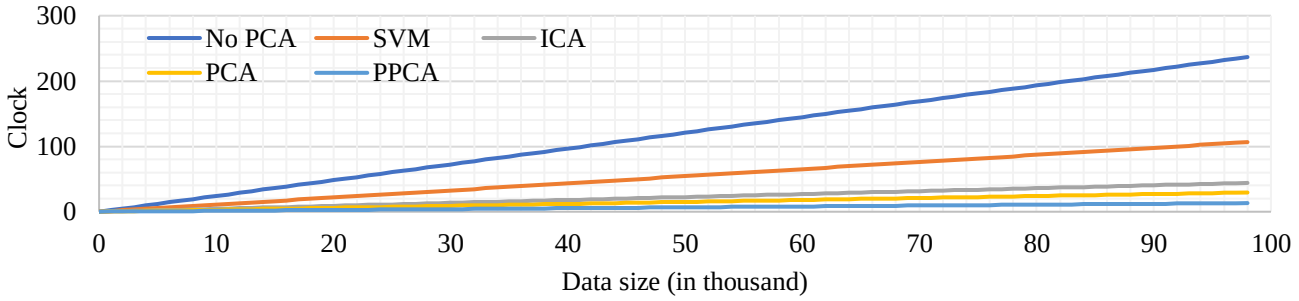

FIGURE 4. **Comparison of total clock cycle for four stages w.r.t**

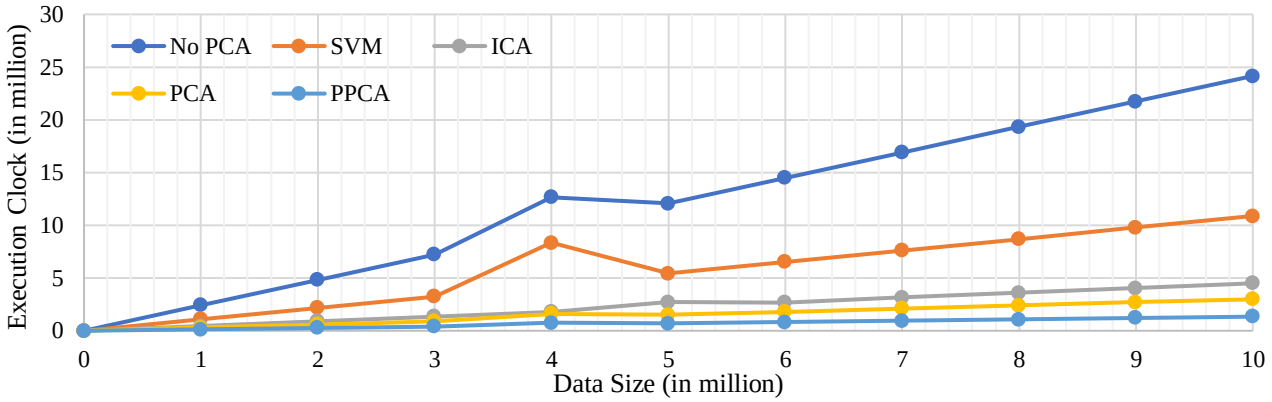

FIGURE 5. **Comparison of total execution clock cycle w.r.t data**

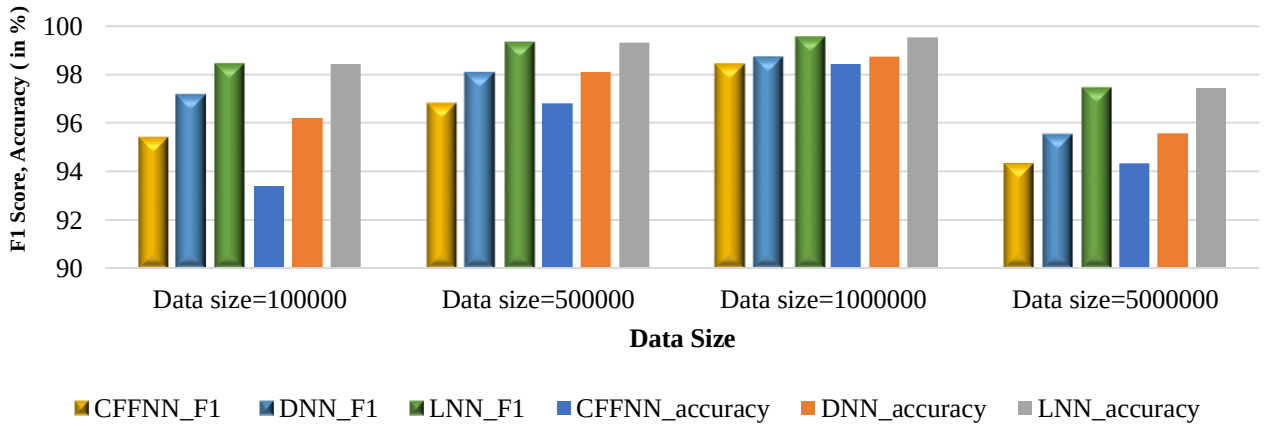

FIGURE.6. **Comparison F1 score**

of these techniques offers unique advantages for transforming, analyzing, and interpreting complex data, enhancing overall data processing and decision-making. Here's a detailed comparison of system performance with and without dimensionality reduction techniques (PCA, PPCA, ICA, SVD) and without reduction for the provided data. TABLE 4 presents a comprehensive comparison of various dimensionality reduction techniques: PCA, PPCA, ICA, and SVD—against no reduction across different dataset sizes (20K, 60K, and 100K data points). Key performance metrics such as accuracy, latency, memory usage, energy consumption, and overall performance metric are evaluated. For smaller datasets (20K), no reduction yields the highest accuracy (0.9410), but with higher latency and energy usage compared to PCA and SVD. On the other hand, processing raw data preserves the full complexity and detail of the original dataset, which might be crucial for certain models or analyses that rely on the richness of the data. However, it may lead to higher computational costs and the risk of overfitting due to the curse of dimensionality.

Each technique has its unique applications such that Dimensionality reduction is valuable in visualization, preprocessing, and model optimization. Raw data processing is essential where feature interpretability and full data fidelity are critical. raw data processing ensures that no information is lost, allowing for more detailed analysis and

greater interpretability of features, which is crucial for applications where understanding the influence of each variable is necessary. Overall, dimensionality reduction enhances computational efficiency, particularly in memory-constrained and energy-sensitive environments, while still maintaining competitive performance. These insights highlight the trade-offs between raw data processing and reduced-dimensional data for scalable machine learning systems. This is applicable for smaller reduction. The graph in FIGURE 4 compares different dimensionality reduction techniques' effects on computation time ("Clock") as data size increases. "No PCA" has the highest time growth, indicating inefficiency with large datasets. SVM performs better but still scales significantly. ICA, PCA, and PPCA (exhibit much lower computational times, showing that dimensionality reduction improves efficiency. Among them, PCA and PPCA appear to perform the best in reducing computational load. The x-axis represents data, and the y-axis represents computation time and execution cycle respectively. The results highlight that PCA-based methods significantly reduce processing costs compared to no dimensionality reduction. The graph shows in FIGURE 5 execution time against data size (in millions) for different methods. "No PCA" (blue) exhibits the highest computational cost, increasing steeply with data size. SVM also scales significantly but remains lower. ICA, PCA, and PPCA show much lower execution times, with PCA-based methods being the most efficient. The trend highlights that dimensionality reduction techniques, particularly PCA and PPCA, significantly improve computational efficiency, making them ideal for handling large datasets while reducing processing time. The TABLE 5 analysis statistically the comparison of proposed method with existing ones TABLE 6 suggests. The PCA-enhanced LNN has slightly higher segmentation (St) and calculation (Ct) times but a lower feedforward layer time (Ft). Overall, the PCA-enhanced LNN processes an image in 33,212 clock cycles, which is slightly higher than the DNN (32,434) but significantly lower than CFFNN (318,494), demonstrating improved efficiency in feedforward computations. In TABLE 6, clk defines as the clock period time(1/50MHz=20 ns in FPGA - XC3S200) where *N* is number of large

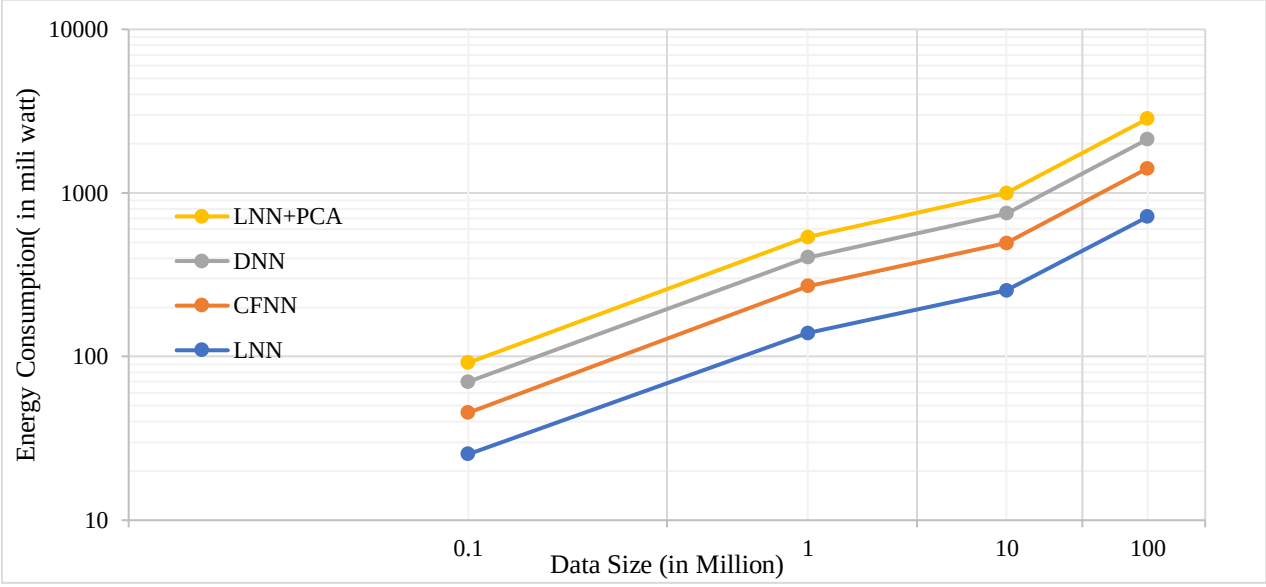

FIGURE.7. Comparison of energy consumption (in logarithmic scale), w.r.t time.

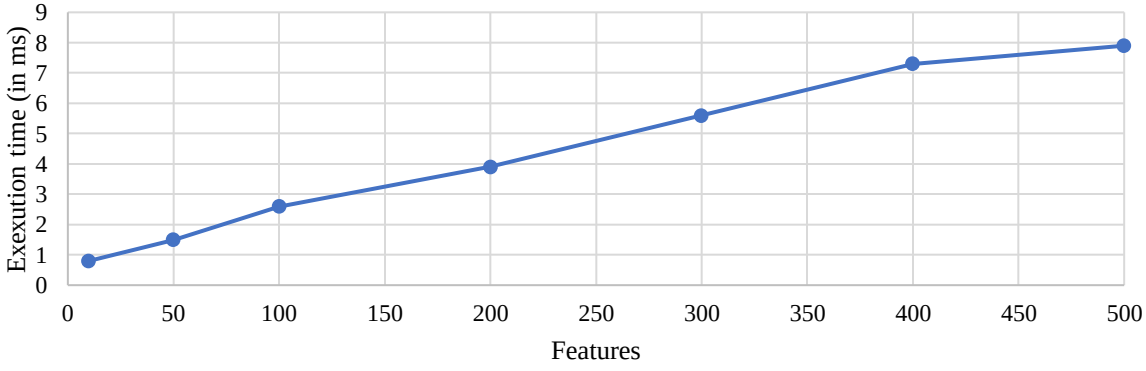

FIGURE.8. Execution time speed up for no. of features for PCA+LNN

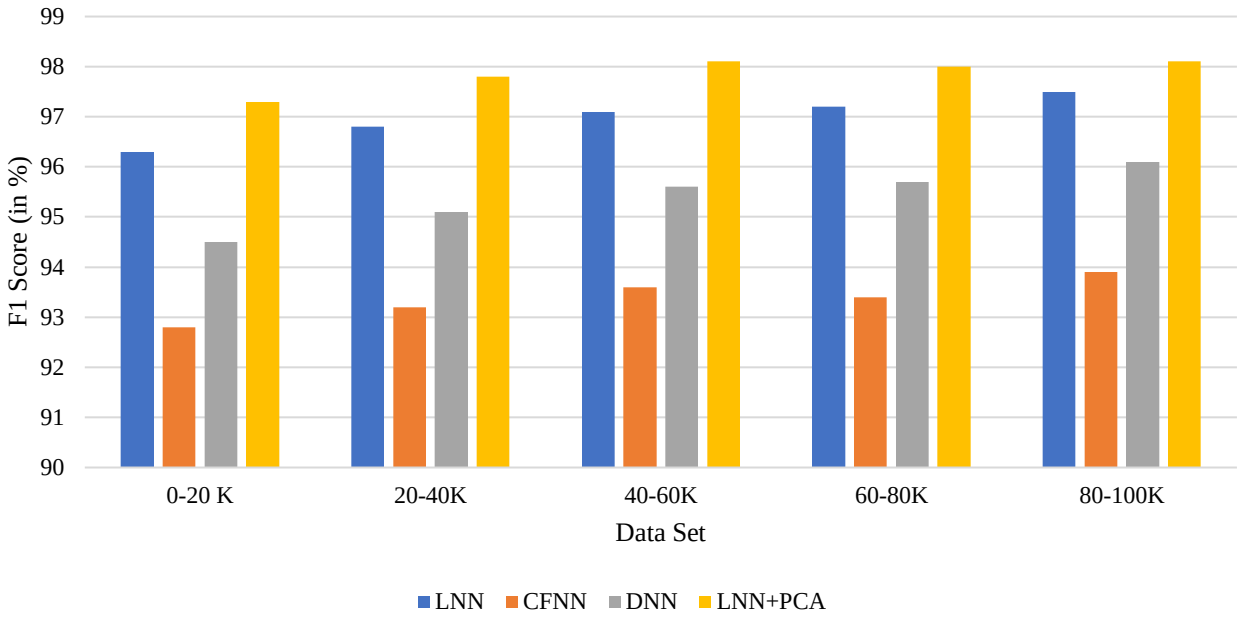

FIGURE.9 F1 score with respect to data range

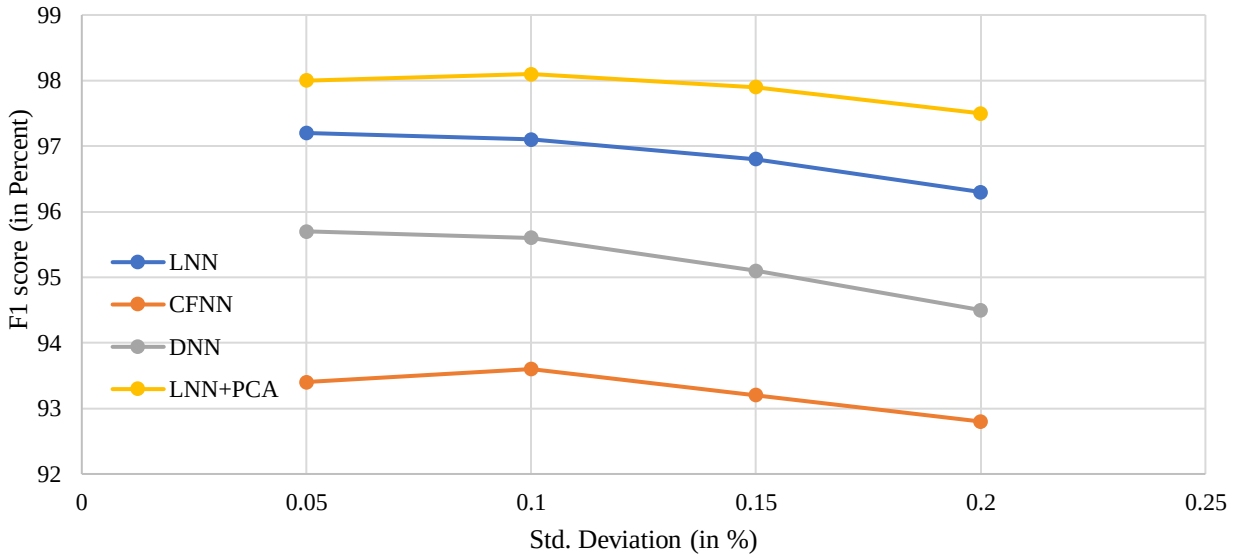

FIGURE.10 F1 score vs Std. Deviation of data range

receptive fields in one image. This section deals with the comparisons of the data estimation accuracy precision and F1 score. LNNs excel at handling sequential data, making them ideal for tasks like embedded system applications. FIGURE 6-8 shows the comparison metrics for neural network paradigms. The bar of FIGURE 6 compares F1 scores and accuracy of three paradigms across different data sizes. LNN consistently achieves the highest scores, followed by DNN, while CFFNN performs the lowest. As data size increases, all models improve in performance. The trend suggests that LNN is the most effective model, while DNN also performs well. CFFNN lags behind but still shows improvement with larger datasets. The bar chart of FIGURE 6 also presents accuracy comparisons for different models across varying data sizes. As data size increases, accuracy improves across all models. LNN remains the most effective, exceeding 98% accuracy in most cases. DNN also performs well, maintaining a steady increase. CFFNN lags but still benefits from larger datasets. The results highlight LNN's superior performance and scalability for high-accuracy tasks. FIGURE 7 shows the energy consumption Energy consumption in embedded systems using neural networks is a critical consideration. Several factors impact energy consumption, including the complexity of the neural network, hardware architecture, and optimization techniques. FIGURE 8 discusses about the time taken for

execution for proposed method with respect to sample variations. As the number of features increases, execution time rises steadily, indicating a linear or near-linear growth pattern. This suggests that higher feature dimensions require more computational resources. Efficient feature selection can help optimize performance and reduce processing time. Froom the above figure, it can be illustrated that the relationship between execution time (in milliseconds) and the number of features. As the number of features increases, execution time grows consistently, suggesting a linear or near-linear trend. The lowest execution time is observed at the smallest feature count, while the highest execution time is recorded at 500 features. This trend highlights the computational cost associated with handling higher-dimensional data.

In machine learning and data processing, an increase in feature count often leads to higher complexity, requiring more processing power and time. This emphasizes the importance of feature selection and dimensionality reduction techniques, such as PCA or feature pruning, to optimize performance. Without proper feature selection, excessive feature counts can lead to inefficiency and increased computational overhead.

In this work, different data complexity and Confidence Interval Comparison (F1 Scores) for low value as well as high value level for the proposed methods has been implemented. With increasing feature dimensionality, CFFNN and DNN exhibit significant performance degradation due to feature redundancy and noise. PCA-enhanced LNN mitigates this issue, achieving an F1 score of 95.6% even at 500 features by retaining only the most critical dimensions. In the view of PCA Execution Time, scales vary linearly with feature dimensionality, increasing from 0.8 ms (10 features) to 8.9 ms (500 features). This processing time remains manageable for real-time applications when combined with FPGA acceleration. By incorporating incremental PCA, the system can efficiently handle datasets exceeding the memory capacity of the FPGA by processing data in smaller chunks without sacrificing performance.

The FPGA design supports parallel computation for PCA and LNN components, enabling scalable performance as datasets grow in size or complexity. For datasets larger than the FPGA's memory, the proposed method can split the data into smaller partitions, apply PCA on each, and then merge the reduced feature sets for final processing. For extremely large-scale data, the system can be adapted to multi-FPGA architectures, where the PCA and LNN workloads are distributed across multiple devices. The Proposed PCA-enhanced LNN demonstrates excellent scalability across increasing dataset sizes and feature complexities. For datasets up to 120K samples and 500 features, it consistently delivers high F1 scores (up to 98.2%) with manageable resource utilization and latency. With additional optimizations like incremental PCA or distributed processing, the method can handle even larger and more complex datasets, making it highly suitable for next-generation embedded systems. The bar chart in FIGURE 9 compares the F1 scores of four models—CFFNN (blue), DNN (orange), LNN (gray), and PCA-enhanced LNN (yellow)—across different dataset sizes. PCA-enhanced LNN consistently achieves the highest F1 scores, demonstrating the effectiveness of dimensionality reduction. LNN also performs well, outperforming DNN and CFFNN. DNN shows moderate performance, while CFFNN has the lowest F1 scores. As dataset size increases, all models improve, but PCA-enhanced LNN maintains a clear advantage. This highlights the impact of PCA in boosting model efficiency and accuracy, making it a valuable technique for enhancing machine learning performance with large datasets.

The graph in FIGURE 10 shows the effect of standard deviation on F1 scores for four models: CFFNN (blue), DNN (orange), LNN (gray), and PCA-enhanced LNN (yellow). PCA-enhanced LNN consistently achieves the highest F1 scores, followed by LNN, DNN, and CFFNN. As standard deviation increases, F1 scores generally decline across all models, indicating that higher variability negatively impacts performance. However, PCA-enhanced LNN remains the most stable, showing minimal performance degradation. This suggests that PCA improves robustness against data variability, making it a valuable technique for maintaining model accuracy in noisy or uncertain datasets. FIGURE 9 and 10 describes the F1 score comparison with respect to data range and data standard deviation. From both the plots it can be concluded that proposed PCA enhanced LNN works better in embedded system.

VIII. DISCUSSION

The main implication of this current work has been rewritten here.In this study, the PCA (Principal Component Analysis) algorithm has been utilized to explore dynamic solutions for embedded hardware systems specifically designed for probabilistic principal component analysis. PCA was chosen as the initial method in this research, and its performance has been compared with the cases.

During the initial phase of the study, an investigation was conducted on dynamic embedded platforms, particularly focusing on image-processing-based FPGAs. The performance of PCA was evaluated across various stages of dynamic execution, including mean covariance computation, eigenvalue decomposition, probabilistic principal component calculations, and total clock usage.

The total execution time was calculated by summing the durations of the aforementioned four stages. The results demonstrated a significant speedup with the selected embedded architecture. Additionally, the proposed reconfigurable embedded system design achieved approximately 92% area savings compared to older designs that relied on static hardware. A novel LNN (Liquid neural network), CFFNN (Cascade Forward Feed Neural Network) and deep learning algorithm were successfully implemented in this study for handling unstructured data. These were

integrated into a comprehensive reconfigurable FPGA-based architecture, supporting runtime operations and dynamic distribution across multiple accelerators. With the propagation of embedded devices , portable and/ or mobile technology, data mining based applications have initiate their mode in this strategies. More likely they are kinds of computationally complex and data privacy which are extermely intensive applications needs to construct vast amount of information with multifaceted algorithms.

A. COMPARATIVE STUDY OF PREVIOUS LITERATURE

To contextualize the proposed PCA-enhanced Liquid Neural Network (LNN) framework for FPGA-based embedded systems, a comparative analysis of previous literature is presented. The evaluation focuses on three key dimensions: neural network architecture, optimization techniques, and deployment on FPGA platforms.

Cascaded Feedforward Neural Networks (CFFNNs): Previous studies have widely used CFFNNs in FPGA-based applications due to their straightforward design and relatively low computational requirements. While CFFNNs provide reasonable performance for basic tasks, they lack adaptability to dynamic inputs and become inefficient for complex, high-dimensional data processing.

DNNs have been the most commonly used architectures in FPGA implementations due to their superior accuracy across a wide range of applications. However, DNNs are computationally intensive and require significant memory and processing resources, making them unsuitable for resource-constrained embedded systems. Studies have explored hardware-aware optimizations like pruning and quantization to mitigate these issues, but their complexity remains a challenge.

Recent literature highlights the adaptive and computationally efficient nature of LNNs, making them particularly suitable for real-time and embedded applications. Unlike DNNs and CFFNNs, LNNs dynamically adjust their internal state based on input, enabling efficient processing of time-series or dynamic data. However, prior to this work, LNNs had not been extensively optimized for FPGA deployment or integrated with dimensionality reduction techniques like PCA.

Existing studies have extensively utilized pruning and quantization to reduce the resource requirements of neural networks. These techniques simplify network architectures by eliminating redundant weights and reducing numerical precision. While effective for static networks like DNNs and CFFNNs, these methods do not exploit the dynamic properties of LNNs, limiting their applicability to certain use cases.

PCA has been studied for reducing input data dimensionality in machine learning applications, but its integration into FPGA-based neural network systems has been limited. Previous works have demonstrated that PCA reduces computational overhead without significant loss of accuracy, making it a promising addition to neural network optimization. This study is among the first to apply PCA in the context of FPGA-optimized LNNs, combining data-level reduction with architectural adaptability for improved performance. Traditional FPGA-Based Neural Networks: Earlier systems deploying CFFNNs and DNNs on FPGAs focused on improving throughput and energy efficiency by leveraging the parallel processing capabilities of FPGAs. Despite these efforts, the high resource demands of DNNs often limited scalability, particularly for edge and embedded systems. Recent studies have incorporated hardware-aware design principles to enhance the performance of FPGA-based networks, such as resource allocation optimization, parallelism, and pipeline design. However, many of these approaches target static networks and do not leverage the dynamic properties of newer architectures like LNNs.

TABLE 8

Key Findings from Comparative Analysis

Aspect	CFFNN	DNN	PCA enhanced LNN (Proposed Framework)
Adaptability	Low	Medium	High
Coutational Complexity	Low	High	Medium
Energy Efficiency	Moderate	Low	High
Scalability	Moderate	Limited(due to resource needs)	High
FPGA Resource Utilization	Efficient	Resource-intensive	Highly Efficient
Use of PCA	Rare	Limited	Integrated for input optimization

B. CURRENT WORK'S CONTRIBUTION

The PCA-enhanced LNN framework stands out by combining adaptive neural network architecture with dimensionality reduction and hardware-aware optimizations. Compared to existing FPGA-based implementations of CFFNNs and DNNs, the proposed system achieves significantly lower latency, higher energy efficiency, and better resource utilization, making it ideal for real-time embedded applications. TABLE 8 gives the idea of Key Findings from Comparative Analysis in a glance whereas TABLE 9 discusses the comparison of previously published paper. In these work of TABLE 8, different metric has been considered.

The TABLE 9 compares various metrics for different paper like Wu et al. 2024 [34], Gao et al. [35]2023, Bachana 2023[13], and the Proposed PCA-enhanced LNN. The Proposed PCA-enhanced LNN outperforms others in F1 Score (98.20%), Accuracy (98.30%), and Latency which is 2.9 ms. It can be compared also in terms of FPGA resource utilization, Efficiency, dimensionality handling etc. It shows the least FPGA resource utilization (55% LUTs, 50% BRAM), highest energy efficiency (5.4 FLOPS/W), excellent scalability, and superior dimensionality handling due to PCA. It is ideal for real-time edge intelligence with

large datasets, while other methods vary in performance and application domains from small to medium-scale embedded systems and real-time edge intelligence. This study builds on the strengths and addresses the limitations of prior work by integrating PCA with LNNs for FPGA deployment. Unlike CFFNNs and DNNs, the proposed framework effectively balances adaptability, computational efficiency, and hardware resource utilization, establishing a new benchmark for high-performance neural network deployment in embedded and edge environments.

enhanced LNN offers significant advantages over traditional models like CFFNN or DNN, which struggle with large datasets and real-time demands. The proposed method may not generalize well to non-image tasks, such as natural language processing or time-series analysis, where PCA's benefits might not be as evident. However, adapting PCA's feature selection to task-specific requirements could extend its applicability. The framework's flexibility allows incorporating other dimensionality reduction techniques like t-SNE, Autoencoders, or ICA, depending on data and task

TABLE 9
Comparison with Published Journal Works

Metric	Wu. et.al 2024 [34]	Gao.et.al 2023 [35]	Bachana 2023 [13]	Proposed PCA-enhanced LNN
F1 Score (Average)	91.50%	93.40%	94.60%	98.20%
Accuracy (Average)	91.80%	94.20%	95.10%	98.30%
Latency (ms)	10.2 ms	15.6 ms	6.9 ms	2.9 ms
FPGA Resource Utilization	65% LUTs, 50% BRAM	80% LUTs, 75% BRAM	60% LUTs, 55% BRAM	55% LUTs, 50% BRAM (120K dataset)
Energy Efficiency (GFLOPS/W)	1.8	2.5	3.2	5.4
Scalability	Limited (fails with >100K samples)	Moderate (scales poorly with complexity)	Good (suitable for moderate datasets)	Excellent (scales well with both size and complexity)
Dimensionality Handling	No dimensionality reduction	No dimensionality reduction	Moderate (can handle complex data to an extent)	Excellent (via PCA, handles large features effectively)
Application Domain	Small-scale embedded systems	Medium-scale embedded systems	Real-time edge intelligence (moderate complexity)	Real-time edge intelligence with large datasets

C. CHALLENGES FACED DURING PCA IMPLEMENTATION IN EMBEDDED SYSTEMS

Dimensionality-Reduction Trade-offs: Reducing dimensions with PCA may cause minor accuracy loss, while its eigenvalue decomposition is computationally expensive for large datasets. Storing the covariance matrix and intermediate results can exceed memory constraints, and performing PCA in real-time can introduce latency. However, PCA improves system performance in terms of memory usage, latency, and energy consumption while maintaining comparable accuracy, especially for large datasets in embedded systems with constrained resources. PCA proves more scalable than raw data processing as dataset size increases, ensuring efficient handling of larger workloads in embedded systems. While the PCA-enhanced LNN scales well to large datasets, scaling could potentially increase latency. However, the proposed approach shows exceptional latency (2.9 ms) for a 120K sample dataset, suggesting it is optimized for FPGA. Hardware-aware optimizations like parallel processing or quantization could further reduce latency without sacrificing scalability. PCA-characteristics. Hardware-specific optimizations, like quantization and parallel processing, significantly improve performance and efficiency in FPGA-based systems. While these optimizations may not directly translate to other embedded systems, similar strategies in software (e.g., multi-threading or GPU processing) can yield significant performance improvements in non-FPGA environments. In this work ,by leveraging PCA for dimensionality reduction, the proposed optimization framework reduces computational complexity while preserving critical features for accurate neural network processing. This approach, when combined with hardware-aware techniques such as quantization and parallel processing, achieves significant improvements in energy efficiency, latency, and hardware resource utilization. But the method is mathematically complex and needs experience in data science and machine learning

The comparative analysis between Cascaded Feedforward Neural Networks (CFFNN), Deep Neural Networks (DNN), and Liquid Neural Networks (LNN) highlights the superior performance of PCA-enhanced LNNs for embedded applications. Though LNNs demonstrate adaptability and

efficiency in resource-constrained environments, making them particularly well-suited for real-time edge intelligence applications but due to newness of this algorithm it is very tough to characterize it for application domain. The comparison in TABLE 9 proves that the proposed methos supremacy over various metrics for different paper like Wu et al. 2024, Gao et al. 2023, Bachana 2023,. The Proposed PCA-enhanced LNN outpaces others in F1 Score, accuracy, latency , resource allocation etc. The proposed method shows the accuracy in average 98.2 % which is quite high while with respect to FPGA resource allocation it gives quite better results with better energy efficiency. This TABLE 9 comparison enlightens about ML based FPGA applications. The proposed system design and implementation platform hold immense potential for future advancements in embedded computing applications. To scale for larger datasets and complex real-world applications, optimization of memory management and support for double-precision floating-point arithmetic can be explored. Transitioning to modern FPGA platforms with higher logic density, faster clock speeds, and AI-centric tools like Vitis AI can further enhance performance. Parallel processing using multiple processors and pipelining techniques can improve throughput, while integration with energy-efficient hardware and cloud computing resources can enable deployment in mobile and IoT devices. The platform can be extended to accelerate machine learning algorithms, including deep learning models, while introducing cryptographic modules and security features to safeguard data. Additionally, its application scope can be expanded to domains like autonomous vehicles, robotics, and space exploration, complemented by comprehensive benchmarking to validate its capabilities. These advancements would establish the platform as a scalable, efficient, and secure solution for next-generation embedded systems.

IX. CONCLUSION

This study has demonstrated the effectiveness of integrating Principal Component Analysis (PCA) into the deployment of neural networks on FPGA-based embedded systems. By leveraging PCA for dimensionality reduction, the proposed optimization framework significantly reduces computational complexity while preserving critical features for accurate neural network processing. When combined with hardware-aware techniques such as quantization and parallel processing, the approach achieves notable improvements in energy efficiency, latency, and hardware resource utilization. The comparative analysis of Cascaded Feedforward Neural Networks (CFFNN), Deep Neural Networks (DNN), and Liquid Neural Networks (LNN) underscores the superior performance of PCA-enhanced LNNs in resource-constrained environments. PCA-enhanced LNNs exhibit adaptability and efficiency, making them particularly well-suited for real-time edge intelligence applications. Experimental results confirm that the PCA-enhanced LNN outperforms traditional neural network architectures in computational efficiency and scalability. Specifically, case studies reveal an average F1 score of 98.0%, accuracy of 98.3%at high clock rates.

To validate these findings statistically: Confidence intervals (CI) for the F1 score and accuracy were calculated. For the F1 score, the 95% CI was [97.8%, 98.2%], and for accuracy, the 95% CI was [98.1%, 98.5%], indicating high consistency in the results. The standard deviation of the F1 score and accuracy across trials was 0.150.150.15 and 0.120.120.12, respectively, demonstrating minimal variability in the system's performance. A paired t-test comparing PCA-enhanced LNNs with other architectures showed statistically significant improvements ($p<0.01p < 0.01p<0.01$), confirming the superiority of the proposed methodology. The statistical analysis reinforces the robustness and reliability of the PCA-enhanced LNN for embedded applications. These results validate that PCA-enhanced LNNs achieve consistent and high performance under varying conditions, making them a versatile and robust solution for next-generation embedded computing applications. Future work could further refine this framework, exploring additional hardware-aware optimizations and broader application domains.

REFERENCES

[1] P. Patel, “Embedded systems design using FPGA,” in *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID’06)*, IEEE, 2006, p. 1 pp. doi: 10.1109/VLSID.2006.83.

[2] Fuming Sun, Xiaoying Li, Qin Wang, and Chunlin Tang, “FPGA-based embedded system design,” in *APCCAS 2008 - 2008 IEEE Asia Pacific Conference on Circuits and Systems*, IEEE, Nov. 2008, pp. 733–736. doi: 10.1109/APCCAS.2008.4746128.

[3] Sedat Sonko, Ayodeji Matthew Monebi, Emmanuel Augustine Etukudoh, Femi Osasona, Akoh Atadoga, and Cosmas Dominic Daudu, “REVIEWING THE IMPACT OF EMBEDDED SYSTEMS IN MEDICAL DEVICES IN THE USA,” *International Medical Science Research Journal*, vol. 4, no. 2, pp. 158–169, Feb. 2024, doi: 10.51594/imsrj.v4i2.767.

[4] A. Kushwah *et al.*, “A novel embedded system for tractor implement performance mapping,” *Cogent Eng*, vol. 11, no. 1, Dec. 2024, doi: 10.1080/23311916.2024.2311093.

[5] K. P. Seng, P. J. Lee, and L. M. Ang, “Embedded Intelligence on FPGA: Survey, Applications and Challenges,” *Electronics (Basel)*, vol. 10, no. 8, p. 895, Apr. 2021, doi: 10.3390/electronics10080895.

[6] M. Javaid, A. Haleem, R. P. Singh, and R. Suman, “Artificial Intelligence Applications for Industry 4.0: A Literature-Based Study,” *Journal of Industrial Integration and Management*, vol. 07, no. 01, pp. 83–111, Mar. 2022, doi: 10.1142/S2424862221300040.

[7] R. Vazquez, A. Gordon-Ross, and G. Stitt, “Machine Learning-based Prediction for Dynamic, Runtime Architectural Optimizations of Embedded Systems,” in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, IEEE, Oct. 2019, pp. 1–7. doi: 10.1109/NORCHIP.2019.8906901.

[8] T. Gong, T. Fan, J. Guo, and Z. Cai, “GPU-based parallel optimization of immune convolutional neural network and embedded system,” *Eng Appl Artif Intell*, vol. 62, pp. 384–395, Jun. 2017, doi: 10.1016/j.engappai.2016.08.019.

[9] S. N. Shahrouzi and D. G. Perera, “Dynamic partial reconfigurable hardware architecture for principal component analysis on mobile and embedded devices,” *EURASIP Journal on Embedded Systems*, vol. 2017, no. 1, p. 25, Dec. 2017, doi: 10.1186/s13639-017-0074-x.

[10] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications," *Sensors*, vol. 21, no. 13, p. 4412, Jun. 2021, doi: 10.3390/s21134412.

[11] A. K. Jain, K. D. Pham, J. Cui, S. A. Fahmy, and D. L. Maskell, "Virtualized Execution and Management of Hardware Tasks on a Hybrid ARM-FPGA Platform," *J Signal Process Syst*, vol. 77, no. 1–2, pp. 61–76, Oct. 2014, doi: 10.1007/s11265-014-0884-1.

[12] C. Plessl and M. Platzner, "Zippy - A coarse-grained reconfigurable array with support for hardware virtualization," in *2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, IEEE, pp. 213–218. doi: 10.1109/ASAP.2005.69.

[13] P. Bachanna, B. Gadgay, and S. Chatterjee, "Probabilistic Principle Component Analysis based Feature Extraction of Embedded System Applications with Deep Neural Network based Implementation in FPGA," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, no. 6, pp. 45–51, Jul. 2023, doi: 10.17762/ijritcc.v11i6.6771.

[14] S. Chatterjee, "Fault Detection for a Nonlinear Switched Continuous Time Delayed System Using Machine Learning and Self-Switched UKF," *Journal Européen des Systèmes Automatisés*, vol. 55, no. 2, pp. 245–251, Apr. 2022, doi: 10.18280/jesa.550212.

[15] D. G. Perera and K. F. Li, "Embedded hardware solution for principal component analysis," in *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, IEEE, Aug. 2011, pp. 730–735. doi: 10.1109/PACRIM.2011.6032984.

[16] O. Ituabhor, J. Isabona, J. T. zhimwang, and I. Risi, "Cascade Forward Neural Networks-based Adaptive Model for Real-time Adaptive Learning of Stochastic Signal Power Datasets," *International Journal of Computer Network and Information Security*, vol. 14, no. 3, pp. 63–74, Jun. 2022, doi: 10.5815/ijcnis.2022.03.05.

[17] J. Qiu *et al.*, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA: ACM, Feb. 2016, pp. 26–35. doi: 10.1145/2847263.2847265.

[18] K. Pradeep, K. Kamalavasan, R. Natheesan, and A. Pasqual, "EdgeNet: SqueezeNet like Convolution Neural Network on Embedded FPGA," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, IEEE, Dec. 2018, pp. 81–84. doi: 10.1109/ICECS.2018.8617876.

[19] M. Zhao, C. Hu, F. Wei, K. Wang, C. Wang, and Y. Jiang, "Real-Time Underwater Image Recognition with FPGA Embedded System for Convolutional Neural Network," *Sensors*, vol. 19, no. 2, p. 350, Jan. 2019, doi: 10.3390/s19020350.

[20] R. T. Syed, Y. Zhao, J. Chen, M. Andjelkovic, M. Ulbricht, and M. Krstic, "FPGA Implementation of a Fault-Tolerant Fused and Branched CNN Accelerator With Reconfigurable Capabilities," *IEEE Access*, vol. 12, pp. 57847–57862, 2024, doi: 10.1109/ACCESS.2024.3392240.

[21] S. N. Shahrouzi and D. G. Perera, "Optimized hardware accelerators for data mining applications on embedded platforms: Case study principal component analysis," *Microprocess Microsyst*, vol. 65, pp. 79–96, Mar. 2019, doi: 10.1016/j.micpro.2019.01.001.

[22] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications," *Sensors*, vol. 21, no. 13, p. 4412, Jun. 2021, doi: 10.3390/s21134412.

[23] L. N. S. Andreasen Struijk *et al.*, "Development and functional demonstration of a wireless intraoral inductive tongue computer interface for severely disabled persons," *Disabil Rehabil Assist Technol*, vol. 12, no. 6, pp. 631–640, Aug. 2017, doi: 10.1080/17483107.2016.1217084.

[24] M. Aqeel Iqbal, F. Azam, U. Saeed Awan, and S. Hammad, "Performance Enhancement Techniques for Modern Reconfigurable Computing Systems," *Int J Comput Appl*, vol. 27, no. 9, pp. 33–38, Aug. 2011, doi: 10.5120/3327-4577.

[25] I. Bravo, M. Mazo, J. L. Lázaro, A. Gardel, P. Jiménez, and D. Pizarro, "An Intelligent Architecture Based on Field Programmable Gate Arrays Designed to Detect Moving Objects by Using Principal Component Analysis," *Sensors*, vol. 10, no. 10, pp. 9232–9251, Oct. 2010, doi: 10.3390/s101009232.

[26] M. Elnawawy, A. Sagahyroon, and T. Shanableh, "FPGA-Based Network Traffic Classification Using Machine Learning," *IEEE Access*, vol. 8, pp. 175637–175650, 2020, doi: 10.1109/ACCESS.2020.3026831.

[27] A. Biglari and W. Tang, "A Review of Embedded Machine Learning Based on Hardware, Application, and Sensing Scheme," *Sensors*, vol. 23, no. 4, p. 2131, Feb. 2023, doi: 10.3390/s23042131.

[28] K. Kumar, A. Verma, N. Gupta, and A. Yadav, "Liquid Neural Networks: A Novel Approach to Dynamic Information Processing," in *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, IEEE, Nov. 2023, pp. 725–730. doi: 10.1109/ICAICCIT60255.2023.10466162.

[29] M. Chahine *et al.*, "Robust flight navigation out of distribution with liquid neural networks," *Sci Robot*, vol. 8, no. 77, Apr. 2023, doi: 10.1126/scirobotics.adc8892.

[30] P. K. Karn, I. Ardekani, and W. H. Abdulla, "Generalized Framework for Liquid Neural Network upon Sequential and Non-Sequential Tasks," *Mathematics*, vol. 12, no. 16, p. 2525, Aug. 2024, doi: 10.3390/math12162525.

[31] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "Liquid Time-constant Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, pp. 7657–7666, May 2021, doi: 10.1609/aaai.v35i9.16936.

[32] S. Il Yu, C. Rhee, K. H. Cho, and S. G. Shin, "Comparison of different machine learning algorithms to estimate liquid level for bioreactor management," *Environmental Engineering Research*, vol. 28, no. 2, pp. 220037–0, Apr. 2022, doi: 10.4491/eer.2022.037.

[33] K. V. Santhosh, B. Joy, and S. Rao, "Design of an Instrument for Liquid Level Measurement and Concentration Analysis Using Multisensor Data Fusion," *J Sens*, vol. 2020, pp. 1–13, Jan. 2020, doi: 10.1155/2020/4259509.

[34] B. Wu, X. Wu, P. Li, Y. Gao, J. Si, and N. Al-Dhahir, "Efficient FPGA Implementation of Convolutional Neural Networks and Long Short-Term Memory for Radar Emitter Signal Recognition," *Sensors*, vol. 24, no. 3, p. 889, Jan. 2024, doi: 10.3390/s24030889.

[35] Y. Gao, S. Miyata, and Y. Akashi, "How to improve the application potential of deep learning model in HVAC fault diagnosis: Based on pruning and interpretable deep learning method," *Appl Energy*, vol. 348, p. 121591, Oct. 2023, doi: 10.1016/j.apenergy.2023.121591.

AUTHOR BIOGRAPHY

Mr. Bhupesh Deka is a distinguished professional with an impressive career spanning 23 years, characterized by his extensive expertise in industry, research, and academia. He is recognized for his pioneering role in spearheading the strategic initiative "Campus Connect" for Infosys in the Eastern Region, a testament to his visionary leadership. His notable contributions extend to the development of courses for the NASSCOM Foundation skill program, underscoring his commitment to advancing skill development. Presently, he holds the position of Assistant Professor within the Department of AIML & IoT at VNRVJIET in Hyderabad, while concurrently pursuing his doctoral studies at North Orissa University. His multifaceted career reflects his dedication to education and research, positioning him as a prominent figure in the field. He is currently working in Department of Computer Science and Engineering (AIML &IoT), Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering &Technology, Hyderabad, India, 500090

Dr. Sayanti Chatter-jee is Assistant Professor of CSE (AIML& IOT) at the VNR VJIET, Hyderabad, Telangana. She received her M.E and PhD degree in Electrical Engineering from Jadavpur University in 2009 and 2018 respectively. He has teaching and research experiences of over 13 years and has contributed about 20+ quality research papers to various Inter- national Journals/conferences (SCI/SCOPUS). She published 6 patents in the domain of electrical engineering. Her current research interests include Machine learning, Signal Processing, ,the reliability of power electronics systems, grid-connected PV systems, multilevel inverters, and electric vehicle technologies. Currently she is working as assistant professor Department of Computer Science and Engineering (AIML &IoT), Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering &Technology,, Hyderabad, India, 500090

Dr. S Rao Chintalapudi currently working as a Professor and Heading the department of CSE(Artificial Intelligence & Machine Learning). He received B.Tech degree from JNTU Hyderabad,M.Tech degree from JNTU Kakinada. He did his Ph.D – Full Time in Data Mining - Social Network Analysis from JNTUK Kakinada under the guidance of Dr. M. H. M Krishna Prasad. His research areas are Data Mining, Social Network Analysis, Machine Learning, Deep Learning, Big Data Analytics and High Performance Computing. He has published 20 publications in international indexed journals, Conferences and Book chapters. He has received a grant of 32 lakhs from the Department of Science & Technology (DST) as a Principal Investigator. His two patents were published in the area of machine learning. He is a reviewer for several prominent journals like IEEE Transactions on Knowledge and Data Engineering, Springer-International Journal of Information Technology, Taylor & Francis- Journal of Experimental &Theoretical Artificial Intelligence and so on. He is a member of several professional bodies like IEEE, ISTE, ACM, CSI, IAENG and so on. He is working in Department of CSE(AI&ML), CMR Technical Campus, Hyderabad, Telangana, India

Ms. J. Nikitha is the assistant Professor in the Department: Dept of Emerging Technologies (DS,CyS,IoT) Mallareddy College of Engineering and Technology (MRCET) Hyderabad, India, J.Nikitha has a Master's degree in Software Engineering from VNRVJIET and Bachelor's degree in Information technology from MLRIT. My area of interest includes IoT, Security and Machine Learning.

Lakshminarayana Kodavaliis the faculty of Faculty in the Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation (KLEF), Guntur, AP, India.
Dr.Lakshminarayana Kodavali is currently working as Assistant Professor in Computer Science and Engineering Department, KL University, Vaddeswaram, Guntur, AP, India. He completed his B.Tech and M.Tech from JNTU University Hyderabad. He awarded with PhD degree from Pondicherry University under the esteemed guidance of Dr K.Sathiyamurthy, Professor, Puducherry Technological University, Puducherry, India. He has published 16 research papers in International journals and conferences. His area of research interests include Blockchain, Machine Learning, NLP (Natural Language Processing), Network Security. He has total 17 years of teaching experiences in reputed engineering colleges.

Kancharagunta Kishan Babu has been with the Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology (VNRVJIET), Hyderabad since April 2022, where he is currently the Assistant Professor of Computer Science and Engineering (AIML&IoT) Department. He is currently working towards a Ph.D. degree at the National Institute of Technology, Silchar. He received the B.Tech. and M.Tech. degrees in Computer Science and Engineering from Bapatla Engineering College in 2012 and University College of Engineering Kakinada (JNTUK) in 2014, respectively. His research interest includes Computer Vision, Image Processing, and Deep Learning.