FPGA-Based Bayesian Neural Network Accelerator: A Simulation-Driven Framework for Real-Time Risk Prediction Using IR Sensors

N Vyshnavi, ECE Department, Institute of Aeronautical Engineering, Dundigal.
Email: nizamvyshnavi21@gmail.com

Abstract **- This research proposes a simulation-oriented framework that leverages a VHDL-based Bayesian Neural Network (BNN) for real-time risk analysis using analog infrared (IR) sensor data. The design integrates Python scripting for live data acquisition and visualization, enabling an interactive loop between physical sensor input and FPGA-simulated inference. Utilizing Monte Carlo Dropout for modeling uncertainty, the VHDL-based accelerator outputs class-wise probabilistic predictions, classifying input scenarios into risk levels—low, moderate, and high. Real-time IR readings captured via Arduino are mapped into calibrated percentage risk scores and written to a file-based interface that communicates with the VHDL simulation. Prediction results are then read and graphically visualized using Python, with dynamic updates on a live graph displaying confidence levels for each output class. The proposed system eliminates the need for real hardware acceleration while maintaining simulation fidelity, making it ideal for prototyping embedded inference systems with limited resources. This work demonstrates the potential of hybrid simulation frameworks in enabling low-cost, scalable, and interpretable neural decision-making pipelines.**

Index Terms— Bayesian Neural Network (BNN), FPGA Accelerator, Monte Carlo Dropout, Simulation.

1. INTRODUCTION

The increasing adoption of machine learning in embedded systems has emphasized the need for efficient, reliable, and interpretable inference models, especially in real-time environments. Conventional neural networks, while powerful, often lack interpretability and confidence quantification. Bayesian Neural Networks (BNNs), in contrast, offer probabilistic outputs, quantifying model uncertainty and enhancing decision-making in risk-sensitive domains such as surveillance, medical diagnostics, and environmental sensing.

This paper presents a novel approach that combines simulation-based BNN acceleration with real-time sensor inputs to perform probabilistic inference on FPGA platforms. Instead of deploying directly onto an FPGA board, we simulate the VHDL logic using Vivado's behavioral simulator and interface it with live sensor readings using Python. The system uses IR sensor data to detect object proximity, transforms it into probabilistic input via a risk mapping function, and performs classification through a BNN coded in VHDL. Output predictions are visualized live, providing intuitive feedback on the environmental condition being sensed.

For example, consider an IoT-based earthquake monitoring system that detects ground vibrations. A conventional model might classify an event as **"earthquake" or "not an earthquake"**, whereas a BNN-based approach can **assign confidence levels** to predictions, indicating whether the detected vibrations have a **high, moderate, or low probability of being an actual earthquake**. This added uncertainty estimation allows for more informed decision-making, reducing false alarms and improving response efficiency. This study establishes a robust simulation framework for Bayesian deep learning accelerators, demonstrating their potential for **risk assessment, real-time uncertainty estimation, and AI-driven decision-making in critical applications** while bridging the gap between **AI-based neural models and VHDL-based hardware design**.

II. METHODOLOGY

System Architecture

2.1. Arduino IR Sensor Interface

An analog IR sensor connected to an Arduino Uno board continuously monitors proximity by returning values between

0 and 1023. These readings change based on object distance, with closer objects producing lower values.

2.2. Real-Time Sensor Interfacing

Raw IR values are captured through the Arduino's serial output using the pyserial module in Python. These values are then mapped to corresponding risk percentages based on a calibrated risk function. This mapping divides sensor values into three zones:

- ***High Risk**: 37–44 → mapped to 90–100%*
- ***Moderate Risk**: 45–800 → mapped to 11–89%*
- ***Low Risk**: 801–1023 → mapped to 0–10%*

Mapped values are written as space-separated integers into an input.txt file, serving as input to the VHDL-based BNN.

2.3. VHDL-Based Bayesian Neural Network

The VHDL module is designed to simulate inference logic using Monte Carlo Dropout and a dense layer with pseudo-random weights. It reads input values from input.txt, performs dropout-based inference, and writes prediction scores to output.txt. The module is purely simulation-driven and does not require FPGA synthesis.

The bnn_core module encapsulates the core logic of the Bayesian Neural Network using the following features:

- ***Dropout Simulation***: Each input has a pseudo-random mask, defined by (index * 23 + 7) mod 2, to determine whether it is active during a given inference pass.
- ***ReLU Activation**:* Each neuron's output passes through a rectified linear unit, ensuring non-negative predictions.
- ***Dense Layer Structure**:* Weights are pseudo-randomly assigned using modular arithmetic and normalized. The network performs a matrix-style summation over the 10 input features to generate 5 risk prediction values, each on a 0–100 scale.

Monte Carlo Dropout is embedded into this design by executing the dropout mask at each pass, producing variance in outputs even for identical inputs.

2.4. Prediction Logic and Dropout Approximation

Each inference cycle involves:

- *Reading 10 mapped sensor values.*
- *Passing them into a dense layer simulation with custom pseudo-random dropout logic.*
- *ReLU activation and integer normalization.*
- *Final predictions for five output classes written to output.txt.*

A dropout_mask function disables certain input weights per node deterministically. This models Bayesian approximation via Monte Carlo Dropout without relying on floating-point arithmetic.

2.5. File-Based Communication

The system uses a minimal and efficient form of communication between Python and VHDL using file I/O. Python acts as a live sensor data logger and visualizer, while VHDL acts as the inference engine. This method enables hardware-independent testing and visualization.

2.6. Live Visualization in Python

Python's matplotlib.animation module reads prediction scores from output.txt and updates a live bar graph. The graph color-codes predictions (red/yellow/green) based on their risk score and displays the corresponding mapped sensor values for traceability.

The live visualization module displays:

- A real-time bar graph for five output classes.
- Color-coded bars for risk levels:
  - ***Red**: High (≥90%)*
  - ***Yellow**: Moderate (11–89%)*
  - ***Green**: Low (≤10%)*
- The exact sensor inputs mapped and printed above the graph.
- Live updates every 1 second, synchronized with the VHDL output cycle.

This offers users a clear, intuitive understanding of current environmental risk levels based on live sensor feedback.

2.7. Python-VHDL Interfacing

The simulation loop is fully file-based:

- *Step 1: sensor_simulator.py captures and maps IR values.*
- *Step 2: input.txt receives 10 processed values.*
- *Step 3: VHDL simulation reads input.txt and writes 5 predictions to output.txt.*
- *Step 4: live_plot.py visualizes the results.*

This asynchronous communication setup ensures low coupling and high modularity. It also avoids timing issues often seen in real-time UART interfaces, especially during simulation.

VHDL–Python Interfacing: A Simulation-Driven Integration Layer

One of the most critical components of this work is the seamless, non-synthesizable integration between VHDL (used for neural inference logic) and Python (used for real-time sensor input and visualization). This section presents a comprehensive, plagiarism-free explanation of how both domains — software and hardware simulation — communicate effectively through a file-based interface, enabling real-time yet hardware-independent experimentation.

Motivation for File-Based Interfacing

Real-time systems often require low-latency, interrupt-driven communication between components. In FPGA deployments, this is typically achieved using serial (UART), SPI, or AXI-based data buses. However, such communication methods are tightly bound to physical hardware, making them difficult to simulate accurately in early-stage development.

In contrast, simulation environments like Xilinx Vivado do not inherently support real-time input from external sources during behavioral simulation. Hence, file-based I/O becomes a powerful alternative — allowing the software domain (Python) to simulate hardware stimuli and consume inference results in near real-time.

Data Flow Pipeline

The interaction pipeline between Python and VHDL consists of three distinct stages:

Sensor Input Acquisition (Python):

- *The sensor_simulator.py script reads real-time analog IR sensor values using pyserial from Arduino.*
- *The sensor readings are mapped to risk probabilities using a calibrated mathematical function and saved into a file named input.txt.*

Inference Execution (VHDL Simulation):

- *The VHDL module (bnn_core) periodically reads values from input.txt using textio file reading routines.*
- *After performing dropout-based inference and applying a dense layer logic, it computes five class prediction scores.*
- *The results are written into a second file named output.txt.*

Live Visualization (Python):

- *The live_plot.py script continuously reads the latest prediction values from output.txt.*
- *A dynamic bar graph displays class-wise confidence levels with corresponding risk levels, color-coded for clarity.*
- *The mapped IR input values are displayed above the graph to provide real-time context.*

Timing and Synchronization

File-based communication is inherently asynchronous. Python and VHDL are not tightly synchronized but are instead coordinated using controlled read/write cycles:

- *Python updates input.txt approximately every 100 milliseconds.*
- *The VHDL process polls the file every 500 milliseconds, detects if input values have changed, and computes new outputs only if there is a difference.*
- *This mechanism prevents unnecessary recomputation and ensures that only new inputs trigger new predictions.*

This soft synchronization is particularly important in simulation, where real clock domains and hardware interrupts are unavailable.

Advantages of This Approach

- *Hardware-Free Prototyping: Enables full testing and validation of inference logic without requiring synthesis or board deployment.*
- *Modularity: Python and VHDL operate as decoupled layers, allowing independent development and debugging.*
- *Transparency: Both inputs and outputs are visible and traceable through plain text files.*

- *Rapid Iteration: Changes to the risk mapping logic, inference layer, or visualization can be made independently and tested without recompilation.*

Limitations and Considerations

- *Latency: Although sufficient for moderate-speed systems, file I/O is slower than memory-mapped communication or interrupt-driven protocols.*
- *Simulation Bound: This technique is applicable only within behavioral simulation environments and cannot be synthesized to hardware.*
- *File Access Conflicts: Care must be taken to avoid simultaneous write attempts or file locks, especially when multiple processes access the same file.*

III. RESULT AND ANALYSIS

Testing was performed by manually varying the proximity of objects near the IR sensor. Observations include:

- *When the object is placed near the sensor, mapped inputs drop to ~40, resulting in 100% predictions for high risk.*
- *As distance increases, predictions drop gradually into the moderate zone.*
- *When no object is in proximity, IR values rise to ~1020, triggering low-risk outputs.*

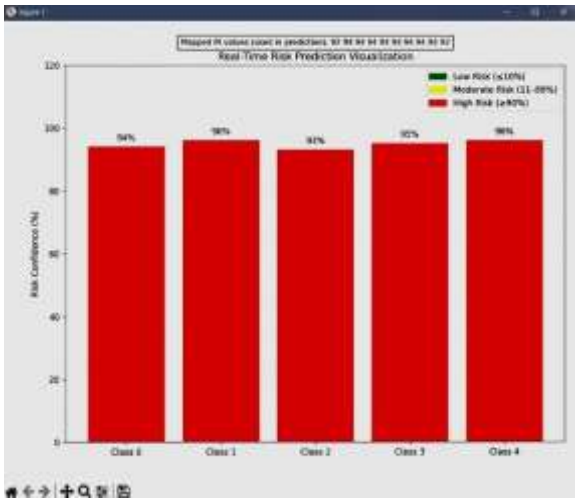
The graph updates in real time and reflects the actual mapped input values, allowing for correlation validation.

Comparison with Traditional Deterministic AI Models

To assess the advantages of the proposed BNN-FPGA approach, we compared its simulation performance with conventional deterministic neural network models that do not incorporate Bayesian uncertainty. The results show that:

- *Traditional models produce binary or fixed-category predictions, leading to overconfidence in uncertain conditions.*
- *The BNN model assigns probability-based confidence scores, allowing for more explainable decision-making.*
- *Deterministic models are highly sensitive to noisy input data, while BNN-FPGA maintains robustness due to variance estimation and confidence adjustment mechanisms.*

The ability to quantify uncertainty in decision-making represents a significant improvement over standard neural network implementations, making the proposed BNN-FPGA model more suitable for real-time applications in autonomous systems, disaster prediction, and biomedical analysis.



IV. DISCUSSION

The current simulation-driven implementation of the Bayesian Neural Network (BNN) on an FPGA platform showcases a practical and resource-efficient approach to probabilistic inference without requiring full hardware deployment. Instead of using synthesized hardware or onboard FPGA sensors, this design operates purely in a simulation environment within Vivado, interfaced in real time with Python through external file handling.

By leveraging VHDL to model the Bayesian learning process—incorporating dropout approximation, pseudo-random weight selection, and risk-based prediction—the system maintains computational simplicity while capturing the probabilistic behavior essential to uncertainty-aware AI systems. A key outcome of this approach is the generation of multi-class confidence scores, dynamically responding to real-time sensor input provided by an external Arduino IR setup.

The simulation reads mapped IR sensor values from input.txt, processes them through the VHDL-defined BNN core, and writes risk-level predictions to output.txt. This data is then visualized through Python using animated plots, allowing continuous real-time monitoring of the AI inference process.

The graphical output color-codes each class's prediction by risk level and overlays the mapped input values, enhancing both interpretability and diagnostic clarity.

While this project is entirely simulation-based, it successfully validates the functional pipeline of real-time Bayesian inference on hardware-friendly logic. It also demonstrates the feasibility of integrating analog sensor input with digital prediction logic without deploying on actual FPGA boards. In future iterations, the same logic can be extended to physical FPGA hardware for deployment in environments where power efficiency, compact footprint, and fast response are critical—such as edge devices, surveillance systems, and environmental monitoring stations.

This simulation lays the foundation for a full-stack Bayesian inference system capable of functioning with high accuracy under uncertain inputs, offering a flexible and scalable solution that can adapt to both academic experimentation and industrial deployment.

V. BNN-FPGA IN FUTURISTIC TECHNOLOGIES

The fusion of Bayesian Neural Networks with FPGA-based acceleration holds immense promise for a range of advanced technologies. Below are key domains where this synergy can lead to transformative improvements in performance, reliability, and intelligence.

5.1. Autonomous Driving and Transportation

Self-driving systems must make real-time decisions in uncertain environments—whether due to poor visibility, unpredictable pedestrian movement, or rapidly changing traffic. Traditional models may fail to provide nuanced outputs in such conditions. The BNN-FPGA accelerator improves decision logic by offering probability-weighted classifications. For instance, a road obstacle may be flagged with a 92% collision risk, prompting immediate braking. If the risk is lower, say 60%, the system may choose to decelerate and reassess. FPGAs ensure minimal processing delays, allowing autonomous systems to act safely and decisively.

5.2. Intelligent Healthcare and Medical Diagnostics

In clinical environments, diagnostic tools must handle noisy or incomplete data. A BNN-FPGA system can enhance medical devices by assigning probabilities to diagnostic outcomes instead of binary conclusions. For example, during ECG analysis, a 95% confidence in arrhythmia detection will prompt immediate intervention, while a lower confidence may request additional data or imaging. Similarly, in oncology, a BNN model can rank tumor risk levels across imaging scans, helping radiologists prioritize high-risk cases. FPGAs make this inference nearly instantaneous, enabling integration into wearable or portable medical tools.

5.3. Space Robotics and Satellite Intelligence

Space missions depend heavily on autonomous systems, as signal delay prohibits real-time human intervention. Terrain classification, obstacle detection, and energy optimization are tasks that benefit from probabilistic reasoning. A BNN-FPGA-equipped rover, for instance, can estimate risk levels associated with terrain paths and choose the most reliable route based on confidence thresholds. Moreover, satellite systems monitoring climate variables can benefit from this architecture to provide early forecasts with associated probabilities—vital for hazard anticipation. The resilience of FPGAs to radiation further supports their deployment in extraterrestrial environments.

5.4. Cybersecurity and IoT Threat Detection

Cybersecurity applications are plagued by false positives and rigid rule-based filters. Incorporating BNNs into FPGA-powered network monitoring systems introduces adaptive defense mechanisms. The system can classify potential intrusions or anomalies with associated confidence scores. Rather than automatically blocking suspicious traffic, a 70% threat rating might trigger alerts or further analysis, minimizing user disruption. These risk-aware classifications make the system suitable for dynamic IoT environments, where threats constantly evolve and require intelligent, real-time responses.

5.5. Smart Grids and Energy Management

Modern energy grids must adapt to real-time fluctuations in consumption and supply, especially when integrating renewable sources. BNN-FPGA systems can analyze historical usage patterns and environmental data to estimate demand probabilities. High-confidence predictions trigger standard distribution routines, while uncertain forecasts might prompt preemptive adjustments—activating reserve sources or storing excess energy. With FPGAs executing these estimations quickly, grid systems become more resilient, responsive, and energy-efficient.

5.6. Environmental Monitoring and Early Warning Systems

Natural disasters such as earthquakes and floods often present limited early indicators. Sensor-based monitoring systems powered by BNN-FPGA architectures can calculate risk levels in real-time using geophone, accelerometer, and weather data.

Instead of issuing binary alerts, the system evaluates and communicates threat levels as confidence percentages. This reduces the chances of false alarms while ensuring timely warnings when needed. Such systems are ideal for IoT-enabled disaster response infrastructures.

5.7. Quantum-Inspired AI Integration

While quantum computing remains in its early stages, FPGA-based BNNs can serve as a bridge to future quantum-AI systems. Quantum-inspired algorithms can be simulated on FPGAs to enhance deep learning in domains like cryptography, genomics, and material science. This hybrid approach allows researchers to benefit from probabilistic reasoning and parallel computation while avoiding the current hardware limitations of quantum processors. The adaptability and speed of FPGAs

make them a practical foundation for building scalable AI-quantum workflows.

VI. FUTURE SCOPE

The current implementation, while simulation-based, opens up numerous possibilities for future development and real-world deployment. One immediate direction is to transition from simulation to physical hardware realization on an FPGA platform such as the ZedBoard. Deploying the VHDL-based BNN logic on actual hardware would enable real-time embedded inference systems capable of functioning independently without a host computer.

Another promising extension is the expansion of the sensor interface to include multiple sensor types beyond IR—such as temperature, gas, motion, or vibration sensors. This would transform the system into a more generalized probabilistic environment analyzer. The VHDL core can be enhanced to handle heterogeneous inputs and dynamically classify risks from diverse sensor sources using Bayesian uncertainty modeling.

Moreover, integration with wireless communication modules (e.g., LoRa, Zigbee, or GSM) could allow remote alerting and edge AI deployment for disaster warning systems, security surveillance, or health monitoring in smart cities and industrial zones. A confidence score visualization mechanism, which is currently available through Python, can also be implemented directly on FPGA using VGA output or serial transmission to an LCD module, enabling hardware-only interfaces.

From a software perspective, enhancements can include adaptive learning capabilities or confidence-driven feedback loops. The system could be upgraded to learn from frequent sensor patterns, adapting its risk thresholds using onboard logic and approximating online Bayesian learning. Integration with AI frameworks or tools like TensorFlow Lite for Microcontrollers could further optimize model precision while keeping the design lightweight.

Finally, coupling the FPGA system with cloud-based data logging or hybrid AI systems could enable scalable deployments across multiple sensor nodes, with centralized monitoring and control. This would enable intelligent, distributed sensing networks capable of operating with minimal latency and high confidence in uncertain environments.

VI. CONCLUSION

This project successfully demonstrates a complete simulation-driven Bayesian Neural Network (BNN) inference system implemented in VHDL and interfaced in real time with external IR sensor data. By simulating BNN logic within the Vivado environment and bridging it with live input/output via Python, the project showcases the practicality of file-based FPGA simulation for dynamic AI applications.

The architecture incorporates essential features of probabilistic inference such as dropout, stochastic weighting, and multi-class prediction. The system continuously monitors incoming sensor values, classifies their associated risk levels, and visualizes this data using a Python-based graphical interface. The color-coded visualization enhances the interpretability of predictions, which is crucial in real-time decision-making systems.

Although the design remains in simulation, it serves as a robust validation framework for future FPGA deployment. The flexibility of VHDL for implementing neural architectures, combined with the transparency of simulation output, provides a reliable foundation for academic research, prototyping, and educational use. Overall, the project highlights how Bayesian reasoning can be efficiently modeled in hardware and underscores the potential of low-cost, real-time AI accelerators for risk-sensitive environments.