Department of Electrical Engineering and Computer Science

Howard University

Washington, DC 20059

Spring 2024

85/100

EECE 401: Senior Design

Sensors

By:

Lamont Syrreal Watson

Timothy Ford

Derrick Boston

Advisor: Dr. Seabron

Instructor: Dr.Kim

Abstract

The Smart Signal Detection project addresses a critical need in modern electronic warfare: real-time classification of radio frequency (RF) signals at the edge. Traditional RF systems rely on centralized processing and post-mission analysis, which introduces delays and limitations in rapidly evolving environments. Our solution leverages machine learning models deployed on a compact Intel NUC device to classify RF signals locally, enabling instant decision-making for use cases including battlefield communication, autonomous drones, and CubeSats.

We implemented a robust pipeline for generating synthetic time-domain signals by combining sine, square, and sawtooth waveforms with randomized amplitude and frequency values, and added Gaussian noise at multiple levels. Signals were transformed into both time and frequency domains and further combined to form hybrid representations. We trained convolutional neural networks (CNNs), recurrent neural networks (RNNs), and Random Forest classifiers to evaluate performance across domains and architectures. The models were tested against conforming and nonconforming datasets to evaluate generalization and robustness under noise. Our results demonstrate reliable classification with edge-optimized speed and power efficiency.

Problem Statement

Modern RF environments demand faster and more adaptive signal intelligence capabilities. Ground units, aircraft, and satellites often lack the infrastructure to send data back to centralized processing centers in time-sensitive scenarios. This creates a need for AI-powered RF classification that runs directly on edge devices.

Our objective was to develop a lightweight, real-time signal classification system capable of detecting and identifying six distinct waveform classes under varying noise conditions. The system must operate autonomously on Intel NUC hardware and maintain accuracy even with nonconforming signals it has never encountered during training.

Design Requirements

-5 Lack of non-technical constraints and compliance

To support edge-based machine learning inference and model testing, our project utilized a hardware setup tailored for field-ready AI deployment. The core component was an Intel NUC, chosen for its ultra-compact form factor and AI-optimized capabilities. This mini-PC enabled local processing of neural networks with minimal power consumption, ideal for edge environments. The NUC was powered by a reliable external supply to ensure stable performance during extended evaluations and real-world deployment scenarios. A monitor was used to provide real-time visualization, aid debugging, and verify system outputs during demonstration sessions. Additionally, an HDMI cable was employed to connect the NUC to the display interface for seamless monitoring and interaction with the deployed models.

From a functionality standpoint, the Smart Signal Detection system was designed to achieve several key performance objectives. It was required to accurately classify six target waveform classes—designated as 11, 12, 13, 22, 32, and 33—each representing a unique combination of sine, square, and sawtooth signals. The system needed to maintain high accuracy across various levels of Gaussian noise, with standard deviations ranging from 0.001 to 1.0, ensuring robustness in noisy environments. To verify the model's ability to generalize beyond its training set, we evaluated performance on nonconforming signal samples that the model had never seen. High training and validation accuracy were expected across all noise conditions. Furthermore, the system was required to deliver low inference latency to support real-time applications, and all

models were deployed to run natively on the Intel NUC using Intel's OpenVINO toolkit for optimized edge inference.

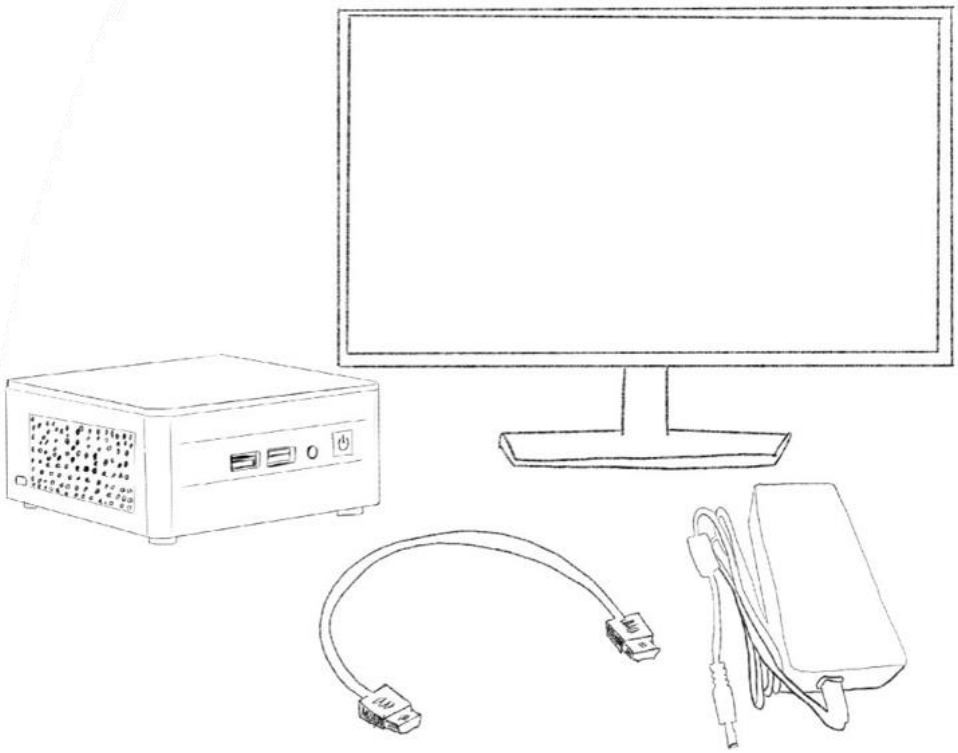

Figure 1: Shows requirements we used for our project

Your **Solution Design** section is mostly strong, well-structured, and communicates the workflow effectively. However, to align with formal senior design report expectations (full sentences, ~~no~~ bullet-style formatting), and improve clarity and flow, here's an improved version that stays true to your structure while polishing it for academic submission:

Why this unnecessary description here?

-5

-5 Deficiency in patent-style description

Solution Design

Our Smart Signal Detection system was engineered to autonomously classify RF signals using a custom end-to-end pipeline optimized for edge deployment on the Intel NUC. The system follows five key operational stages, beginning with signal generation and ending in real-time performance evaluation.

The signal generation process was implemented in Python and involved synthesizing randomized RF signals using combinations of sine, square, and sawtooth waveforms. Each signal comprised 512 samples and was constructed with random amplitudes ranging from 0.1 to 1.0 and frequencies between 100 kHz and 500 kHz. To simulate real-world interference, Gaussian noise was added at five levels, ranging from 0 to 1.0 standard deviation. To ensure uniqueness and prevent duplicate samples, all signals were hashed using the MD5 algorithm.

Following generation, each signal was transformed into three domain representations. In the time domain, the raw waveform was preserved. For frequency-domain analysis, signals were converted using the Fast Fourier Transform (FFT), log-scaled, and normalized. The hybrid domain combined both time and frequency signals into a two-channel array of shape (512, 2), which enabled multi-modal learning. An example of a hybrid sample combining sine and sawtooth waveforms with no noise is shown in Figure 3 below.

These preprocessed signals were then input into trained neural network models deployed on the Intel NUC. The networks were tasked with classifying each signal into one of six waveform classes: 11, 12, 13, 22, 32, or 33, which represented distinct combinations of base waveforms.

All classification occurred locally on the Intel NUC, which utilized the OpenVINO toolkit for inference acceleration. This allowed the system to operate autonomously in low-power edge environments, without the need for cloud-based processing. By leveraging OpenVINO, we reduced inference time while maintaining high classification accuracy.

Once classification was complete, results were logged and analyzed. This included tracking accuracy, inference time in milliseconds per sample, and performance on nonconforming signals not seen during training. These benchmarks were used to compare model performance across domains and hardware platforms. Figure 2 illustrates the complete system pipeline, from signal generation to inference and evaluation.

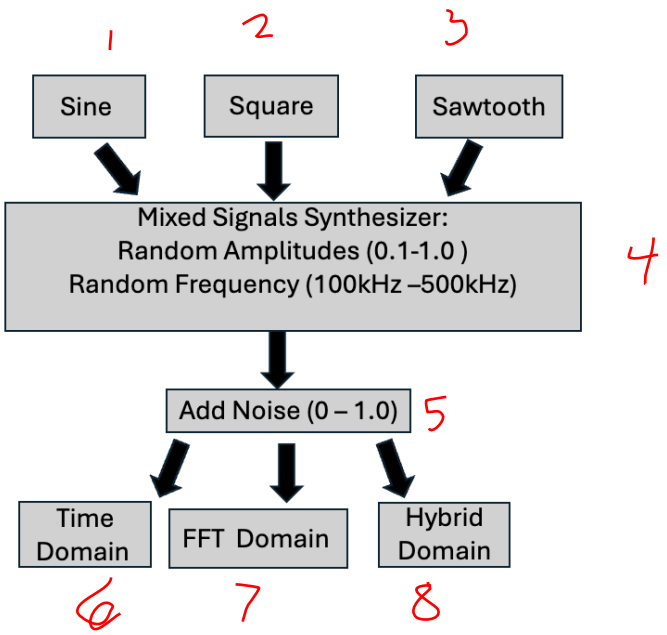

Figure 2: Diagram of Signal Dataset Generation and Processing

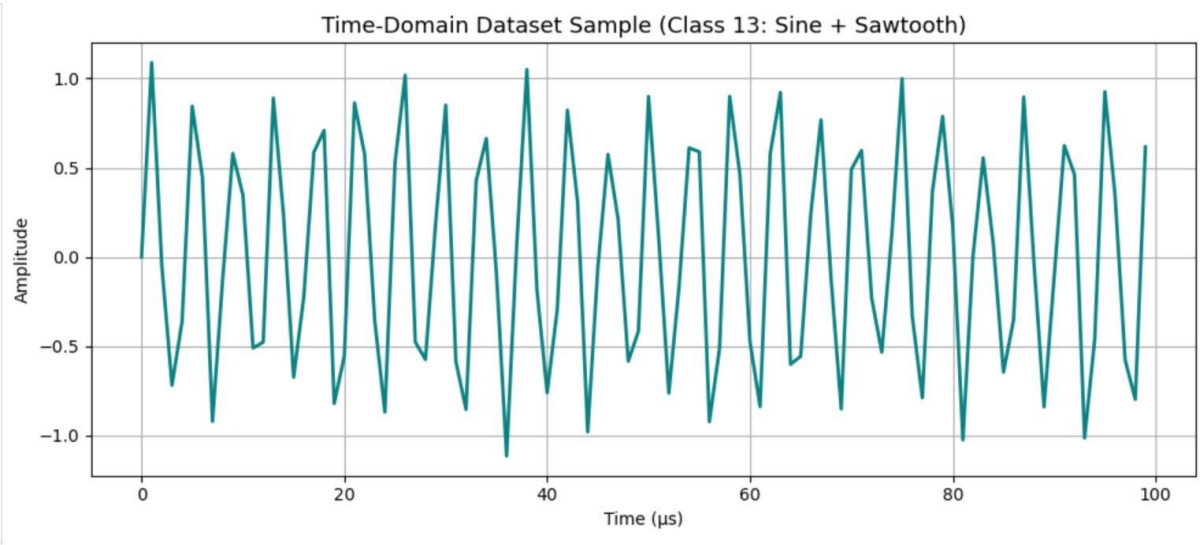

Figure 3: Example of a Sine+Sawtooth Time Domain Sample with no noise

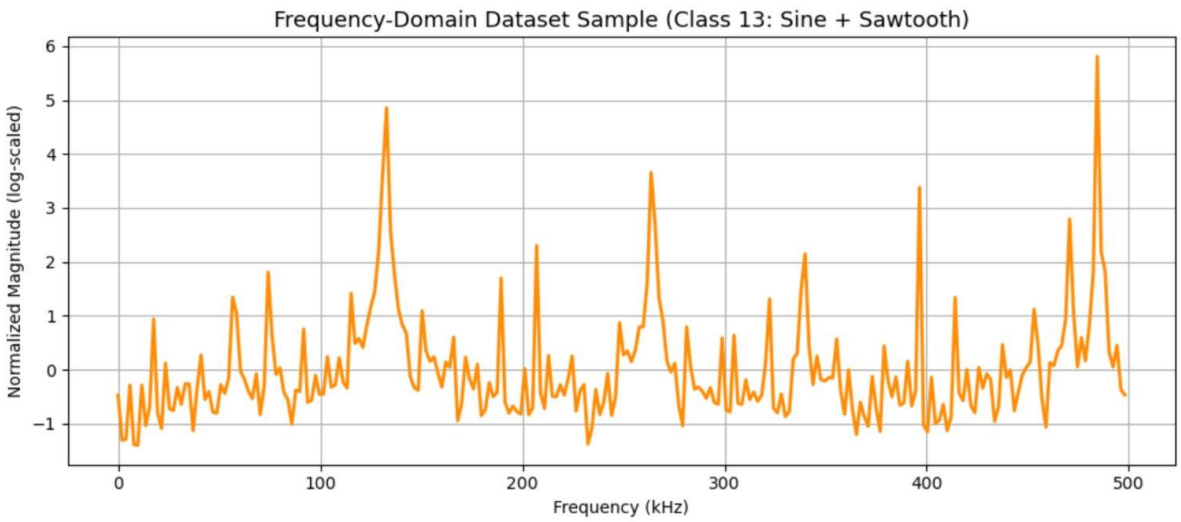

Figure 4: Example of a Sine+Sawtooth Frequency Domain Sample with no noise

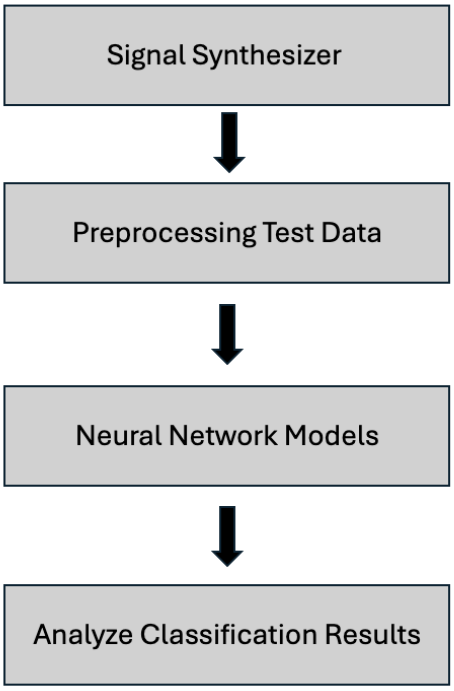

Figure 5: illustrates the full data flow from signal synthesis to classification and analysis:

Hardware Benchmarking: Why Intel NUC?

To assess edge-deployment feasibility, we compared training and inference performance across two platforms: a **MacBook Pro (M1)** and an **ASUS NUC Pro 14**. While the MacBook proved effective for development, the NUC was selected as the deployment platform due to several operational advantages.

Benchmarking and Hardware Analysis

To evaluate model performance across different platforms, we conducted benchmarking on two distinct hardware configurations: the MacBook Pro (M1) and the ASUS NUC Pro. Both systems supported our machine learning workflows but served different purposes in our project lifecycle.

The MacBook Pro, powered by Apple's M1 chip with an integrated 8-core GPU and 8 GB of unified memory, offered exceptional performance during model development. It delivered high baseline accuracy and faster training times due to its optimized neural engine and Metal acceleration. However, its form factor, operating system, and lack of dedicated edge-AI toolkit support limited its suitability for deployment in constrained or rugged environments.

In contrast, the ASUS NUC Pro, equipped with an Intel Core Ultra 7 165H processor and integrated Intel Arc GPU, provided a compact and efficient alternative for edge inference. With support for up to 48 GB of DDR5 memory and a higher clock speed of 4.8 GHz, the NUC was optimized for low-latency, on-device signal classification. It supports AI acceleration through Intel's OpenVINO toolkit and runs on flexible operating systems including Windows, Ubuntu, RHEL, and ChromeOS Flex. Its lightweight form factor (1.65 lbs) and small dimensions (117 x 112 x 37 mm) make it ideal for field use in mission-critical environments.

While the MacBook served as an effective prototyping and experimentation tool, the Intel NUC aligned more closely with our deployment objectives. Its compliance with Size, Weight, Power, and Cost (SWaP-C) constraints makes it a viable platform for military, aerospace, and remote sensing use cases. In summary, we found the MacBook Pro well-suited for initial model training, and the Intel NUC essential for real-world deployment.

Spec / Feature	MacBook Pro (M1)	ASUS NUC PRO
Processor	Apple M1	Intel Core Ultra 7 165H
GPU	Integrated 8-core GPU	Integrated Intel Arc GPU
RAM	8 GB unified memory	Up to 48 GB DDR5-5600
Acceleration	Apple Neural Engine + Metal	OpenVINO Toolkit
Operating System	macOS Sequoia 15.0.1	Windows / Ubuntu / RHEL / ChromeOS Flex
Weight	3.4 lbs (1.55 kg)	1.65 lbs (0.75 kg)
Form Factor	Laptop (13-inch)	Mini PC (117 x 112 x 37 mm)
Clock Speed	3.2 GHz	4.8 GHz
Ports	USB-C, USB-A, HDMI via hub	2x Thunderbolt 4, 2x HDMI, 4x USB, LAN

Figure 6: Comparison of Macbook vs Intel NUC

Agile Workflow

To manage development efficiently and ensure iterative progress, we adopted an Agile-based sprint model throughout the Smart Signal Detection project. Each sprint focused on a specific objective that incrementally advanced the functionality and performance of the final system.

In Sprint 1, we focused on hardware benchmarking by comparing training and inference speed between the MacBook Pro and the Intel NUC. A test image classifier was used to establish a performance baseline for both platforms. This informed our deployment strategy by highlighting the NUC's edge-optimized capabilities.

Sprint 2 concentrated on building the signal synthesizer. We developed a Python-based script to generate synthetic RF signals using randomized sine, square, and sawtooth waveform combinations. This component was foundational to creating large, diverse datasets for model training and evaluation.

In Sprint 3, we implemented and trained several machine learning models, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Random Forest classifiers. These models were tested across time-domain, frequency-domain, and hybrid signal representations to assess architectural effectiveness.

Sprint 4 was dedicated to model benchmarking and analysis. We evaluated each model's training accuracy, inference time, and ability to handle noisy and nonconforming signals. Results from this phase guided model selection for deployment on the Intel NUC.

This structured, incremental workflow enabled us to validate each component of the system in stages and supported seamless integration and real-time testing throughout development.

Sprint Phase	Key Deliverables
Sprint 1: Hardware Benchmarking	Benchmarked training and inference speed between MacBook Pro and Intel NUC using a test image classifier.
Sprint 2: Signal Synthesizer	Developed a Python-based script to generate synthetic RF signals using sine, square, and sawtooth waveforms.
Sprint 3: Model Development	Built, trained, and evaluated CNN, RNN, and Random Forest models across time, frequency, and hybrid domains.
Sprint 4: Model Benchmarking & Analysis	Conducted detailed benchmarking of all models based on training accuracy, inference time, and robustness to noise and nonconforming signals.

Figure 7: Agile Workflow Timeline

Approach and Tradeoffs

To evaluate model performance across both architectural design and signal representation, we implemented and compared three distinct machine learning approaches: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Random Forest Classifiers (RFs). Each model offered unique strengths and tradeoffs that were tested under varying noise conditions and signal domains.

CNNs were used to extract spatial features from 1D RF signals using a series of convolutional layers followed by fully connected layers. This architecture proved effective for clean datasets,

offering fast inference and scalable performance. As shown in Figure 8, the CNN architecture incorporated three convolutional layers and two dense layers to classify signals into six predefined classes.

RNNs, specifically those based on Gated Recurrent Units (GRUs), were chosen for their ability to capture temporal dependencies in sequential data. These networks demonstrated strong performance under noisy conditions, learning structure across time steps even when the signal was distorted. The RNN architecture, depicted in Figure 9, relied on stacked GRU layers followed by dropout and dense layers for classification. While RNNs provided excellent noise robustness, they required more computational resources and exhibited slower inference times compared to CNNs.

In contrast, Random Forest classifiers served as a lightweight, interpretable alternative. Built on top of decision trees, these models used handcrafted statistical features such as skewness, kurtosis, and energy to classify signals. Figure 10 illustrates the ensemble-based design of the Random Forest, which produced extremely fast inference, making it well-suited for constrained edge environments. However, its accuracy was slightly lower and highly dependent on the quality of feature engineering.

We also evaluated each model across three signal representations: time-domain, frequency-domain, and hybrid-domain. Time-domain inputs preserved the raw waveform shape, making them ideal for capturing direct amplitude patterns. Frequency-domain inputs, generated via log-scaled FFT, emphasized spectral characteristics and were particularly useful for identifying modulation patterns. Hybrid-domain representations combined both time and frequency vectors into a dual-channel (512×2) input. This format provided balanced performance and demonstrated the best generalization across noise levels and nonconforming signals.

By leveraging multiple models and signal domains, our system was able to adapt to various deployment scenarios, optimizing for speed, accuracy, or noise robustness as required by the operational context.

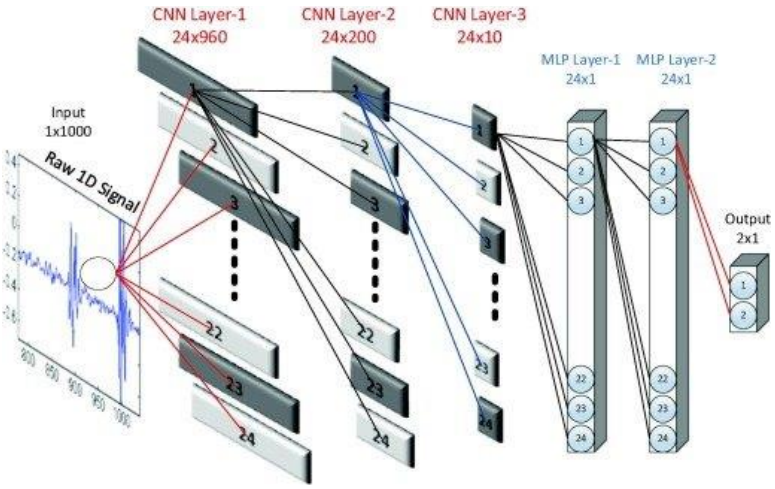

Figure 8: CNN architecture used for signal classification with three convolutional layers and two MLP layers.

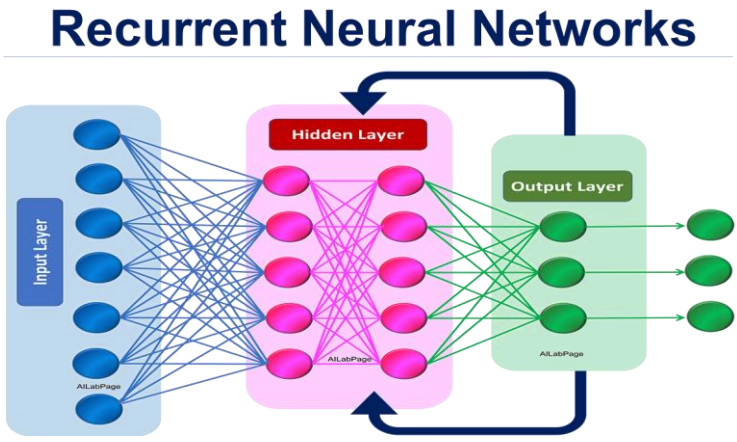

Figure 9: RNN model used for classifying sequential RF signals with GRU layers capturing temporal patterns.

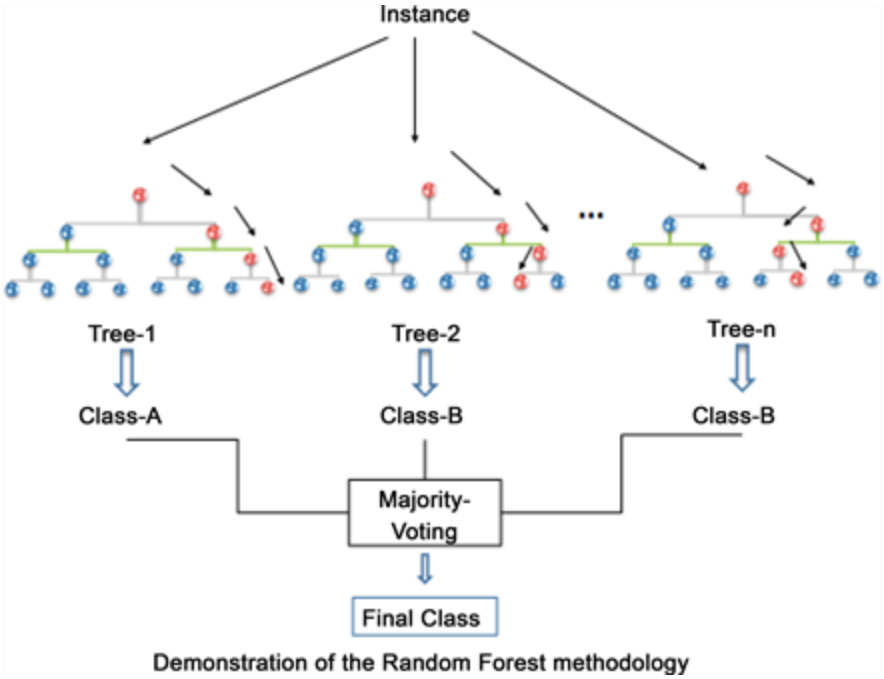

Figure 10: Random Forest ensemble classifier built on top of decision trees and majority voting.

Absolutely! Here's the finalized version of the **Project Implementation Process** section you can copy and paste **right before your Results and Analysis** section in your final report:

Project Implementation Process

To implement our Smart Signal Detection system, we followed a structured process built on iterative development through Agile sprints.

We began by building a signal synthesizer that generates synthetic RF signals composed of sine, square, and sawtooth waveforms. These signals were augmented with varying levels of Gaussian noise and transformed into time, frequency, and hybrid domain representations.

We then trained three model types — CNN, RNN (GRU-based), and Random Forest classifiers — using domain-specific preprocessing pipelines. Each model was tested on clean and noisy datasets, as well as unseen (nonconforming) samples.

To evaluate the performance of each model, we benchmarked training time, inference time, and accuracy across devices: the MacBook Pro and the Intel NUC. We recorded how increasing training dataset size affected generalization and analyzed performance tradeoffs across the three domains (time, frequency, and hybrid).

Figures 11–16 illustrate key implementation results, such as latency comparisons, noise resilience, and architecture performance on different domains. These outcomes guided our final deployment strategy on the Intel NUC using OpenVINO for optimized edge inference.

Results and Analysis:

Laptop vs. NUC Performance Comparison

To assess the deployment potential of our Smart Signal Detection system, we benchmarked all model types—CNN, RNN, and Random Forest—on three critical metrics: training time, inference latency, and classification accuracy across both conforming and nonconforming datasets.

The results revealed a clear tradeoff between development efficiency and operational deployment. The MacBook Pro, equipped with a discrete GPU and a higher base clock speed, achieved faster training and inference times. This made it highly effective during the development phase, where rapid iteration was necessary. However, the Intel NUC consistently demonstrated higher classification accuracy, particularly as dataset size increased.

More importantly, the NUC's compact design, low power consumption, and compatibility with Intel's OpenVINO toolkit made it the superior platform for edge deployment. Its performance, while slightly slower than the MacBook in raw speed, remained well within the bounds required for real-time signal classification.

This distinction highlights a core insight: the MacBook Pro is an excellent tool for prototyping and iterative model development in lab settings. In contrast, the Intel NUC is purpose-built for field-ready inference, satisfying key constraints related to Size, Weight, Power, and Cost (SWaP-C)—crucial for military, aerospace, and embedded use cases.

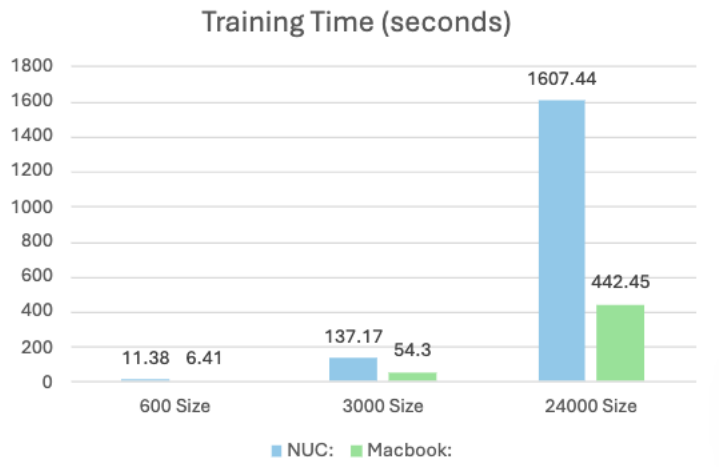

Figure 11: Training Time on the Mac vs NUC

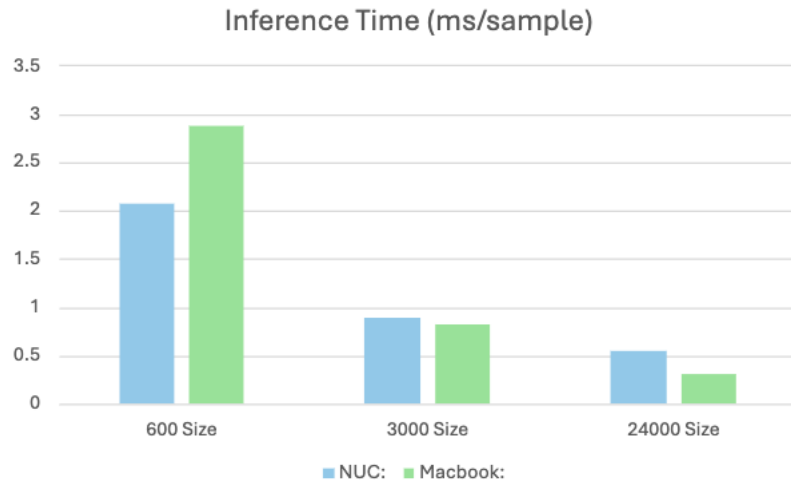

Figure 12: Inference Time on the Mac vs NUC

Figure 13: Training Accuracy on the Mac vs NUC

Impact of Training Size

To better understand the influence of training data volume on model performance, we evaluated Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Random Forest classifiers using datasets containing 600, 3,000, and 24,000 samples. This experiment provided insight into each model's learning capacity and scalability.

As shown in Figure 14, all models demonstrated improved accuracy as the dataset size increased, validating the expected relationship between data volume and model generalization. Random Forest classifiers performed unexpectedly well on smaller datasets, achieving over 45% accuracy with just 600 samples, likely due to their reliance on statistical feature extraction rather than deep pattern learning.

CNN and RNN architectures displayed more significant gains as the sample size scaled. Both models surpassed 70% accuracy when trained on 24,000 samples. RNNs slightly outperformed CNNs at higher volumes, showcasing their superior ability to capture sequential and noisy patterns, which are characteristic of real-world RF environments.

These results highlight the importance of sufficient data volume when training deep learning models for robust edge-based signal classification.

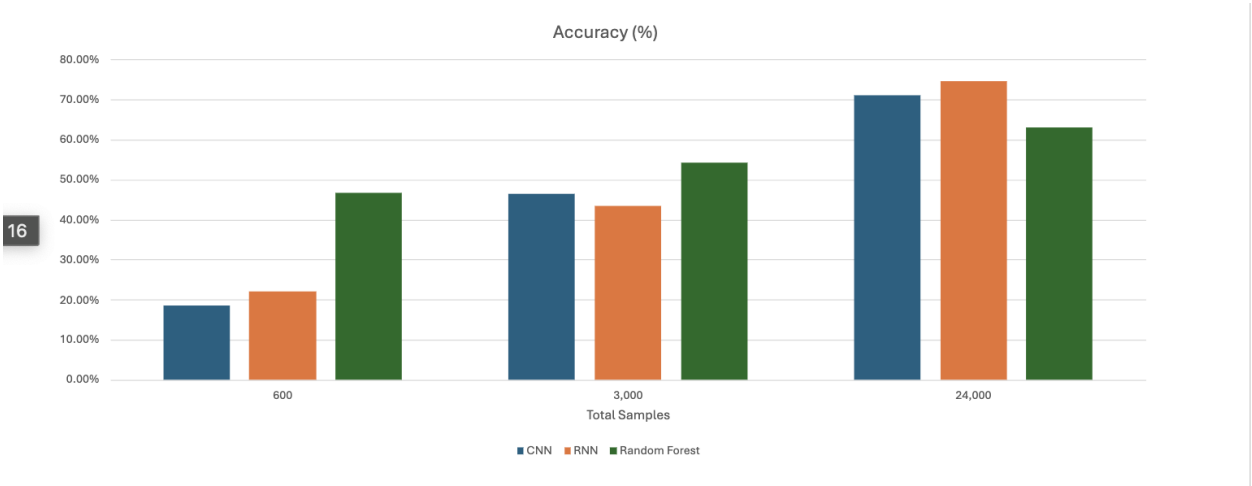

Figure 14: Model accuracy

Accuracy in Noisy Environments:

To evaluate model robustness under signal degradation, we tested each classifier—CNN, RNN, and Random Forest—on datasets embedded with Gaussian noise ranging from 0 to 1.0 standard

deviation. This evaluation was conducted across all three data representations: Time-Domain, Frequency-Domain (FFT), and Hybrid.

As illustrated in Figure 15, each model displayed varying levels of noise tolerance. CNNs performed reliably up to moderate noise levels ($\sigma \leq 0.1$), maintaining stable accuracy across domains. However, their performance deteriorated significantly at high noise ($\sigma = 1.0$), particularly in the frequency domain where distortion heavily affected spatial patterns.

RNNs exhibited the highest resilience to noise. Their sequential processing architecture, especially in GRU-based implementations, enabled the models to retain critical temporal features, resulting in stronger performance under heavy distortion. This was especially evident in the time and hybrid domains.

Random Forests, while offering fast inference and interpretability, were most affected by increased noise. Their performance declined sharply, with the steepest drop observed in the frequency domain. This is likely due to their reliance on statistical features, which become less informative in highly corrupted signals.

These results underscore the fundamental tradeoff between inference efficiency and resilience. While CNNs and Random Forests are lightweight and fast, RNNs present a stronger option for environments where signal clarity cannot be guaranteed.

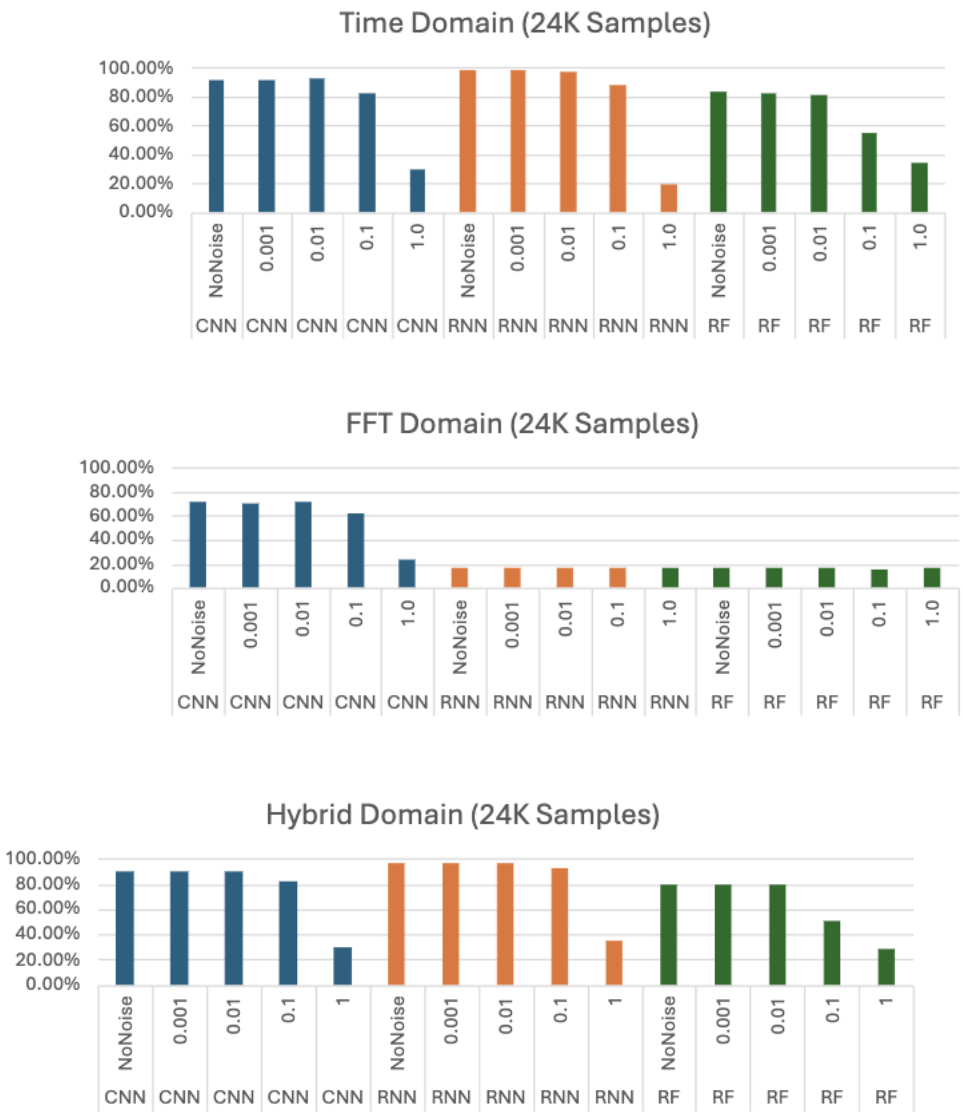

Figure 15. Accuracy across Gaussian noise levels (0 to 1.0) for CNN, RNN, and Random Forest models across Time, Frequency, and Hybrid domains.

Domain Comparison

To evaluate the influence of signal representation on model performance, we tested CNNs, RNNs, and Random Forest classifiers on a dataset of 24,000 clean samples distributed across three domains: Time, Frequency (FFT), and Hybrid. All experiments were conducted on the Intel NUC using deployment-ready configurations.

As illustrated in Figure 16, we analyzed each model's training duration, inference speed, and classification accuracy under consistent conditions:

CNNs delivered strong performance across all domains and achieved the highest overall accuracy when trained on frequency-domain (FFT) representations. Their ability to learn spatial patterns made them well-suited for both structured time and spectral data.

RNNs excelled in the time and hybrid domains, leveraging their GRU-based architecture to capture sequential dependencies. However, this came at the cost of higher computational overhead, as RNNs required longer training cycles and exhibited increased inference latency due to their temporal processing.

Random Forests stood out for their efficiency. They achieved the lowest inference time and were easy to interpret, making them ideal for edge applications where resources are constrained. Nevertheless, they lagged in classification accuracy, particularly in the frequency domain, where their reliance on handcrafted features proved limiting.

This comparison highlights the tradeoffs between accuracy, speed, and complexity across domains and architectures. While CNNs offered a strong balance of speed and accuracy, RNNs were more robust to sequence-based variations, and Random Forests prioritized deployment efficiency.

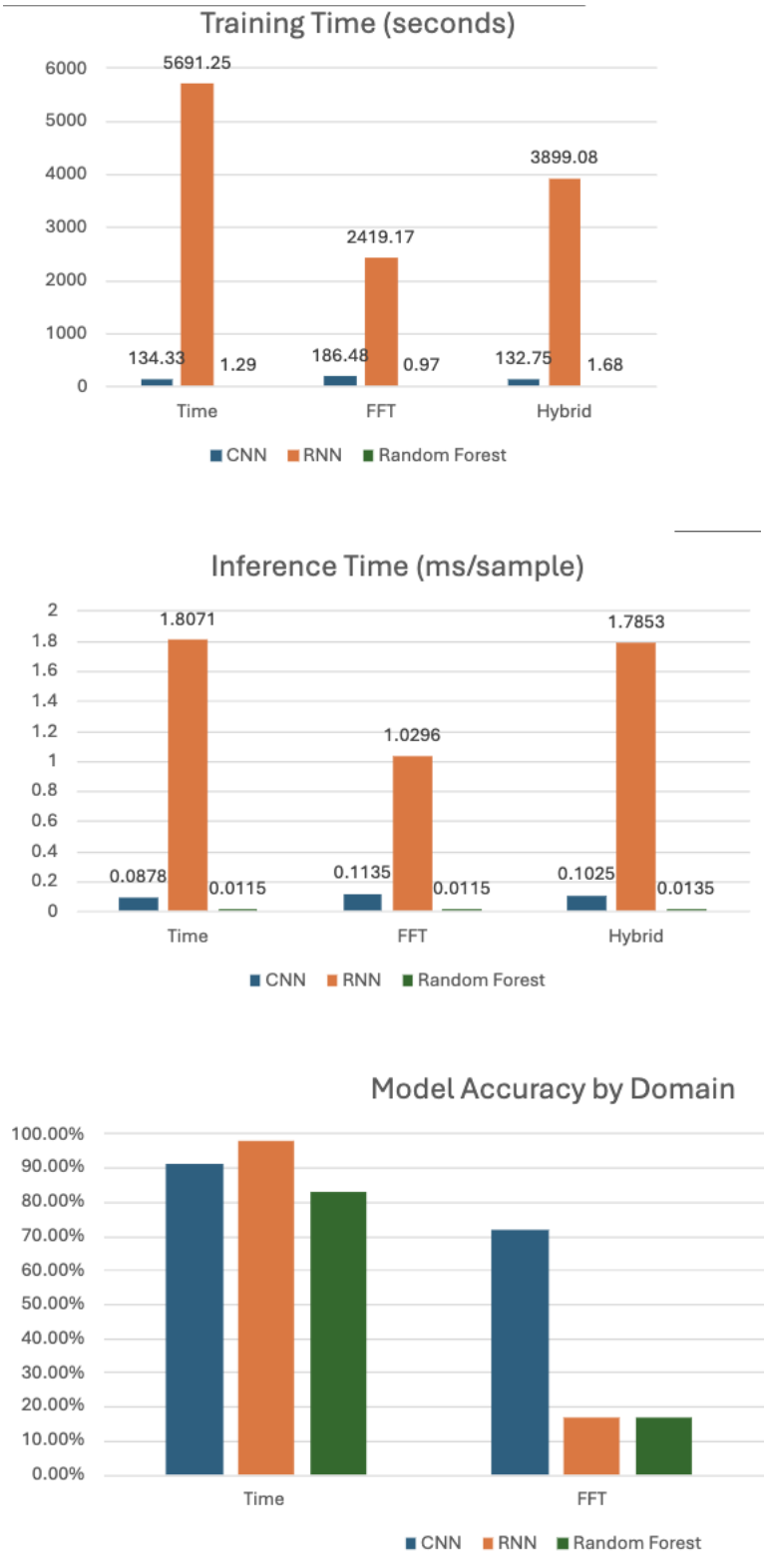

Figure 16. Comparison of model performance across Time, Frequency, and Hybrid domains using 24,000 clean samples evaluated on the Intel NUC.

Conclusion

We successfully developed and demonstrated a scalable, real-time RF signal classification system that runs autonomously on edge hardware. By training lightweight neural networks across multiple signal domains and noise levels, we built models that are not only accurate but resilient—especially under nonconforming and noisy signal conditions.

Our deployment on the Intel NUC showcased its viability as an edge-AI platform, balancing performance with power efficiency and field-readiness. With support for OpenVINO acceleration, the NUC proved capable of running real-time inference without reliance on cloud infrastructure.

The Smart Signal Detection system offers broad applicability across defense, aerospace, satellite communication, and emergency response scenarios—anywhere rapid signal intelligence is required at the edge.

This project confirms that edge-AI for RF classification is not just technically feasible—it's operationally deployable.

Future Work

To further enhance the scalability, robustness, and real-world applicability of the Smart Signal Detection system, several key areas have been identified for future development.

First, expanding the scope of hardware benchmarking is crucial. While the Intel NUC has proven viable for edge inference, future work should include comparative evaluation of alternative edge platforms such as the NVIDIA Jetson Nano, Google Coral TPU, and ARM-based System-on-Chips (SoCs). These platforms may offer more favorable tradeoffs in size, power consumption, and specialized AI acceleration, depending on the deployment environment.

Second, incorporating real-world RF data is essential to improve model generalization and operational readiness. By integrating software-defined radios (SDRs), the system can be exposed to authentic electromagnetic environments with unpredictable interference patterns, allowing the models to be trained and validated on signals beyond the synthetic dataset used in this project.

Next, continued optimization of model architectures can lead to significant gains in efficiency and performance. Future iterations may benefit from exploring attention-based networks, transformer architectures, or lightweight CNN variants such as MobileNet. These improvements can reduce latency and resource usage while preserving—or even enhancing—classification accuracy under varying signal conditions.

In addition, automating the end-to-end signal analysis workflow will increase system usability and operational value. A future iteration of the system should include real-time signal detection, autonomous classification, anomaly monitoring, and automated alerting. This would allow for seamless deployment in high-risk, time-sensitive applications.

Finally, expanding operational use cases beyond initial demonstrations will validate the versatility of the Smart Signal Detection system. Potential domains include tactical military scenarios involving jamming detection, autonomous drone navigation in contested airspace, RF communication monitoring during disaster response, and spectrum analysis on low-Earth orbit CubeSats.

References

[1] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. [Online]. Available: https://www.tensorflow.org/

[2] Keras Team, “Keras API Reference,” [Online]. Available: https://keras.io/

[3] NumPy Developers, “Fast Fourier Transform (FFT),” [Online]. Available: https://numpy.org/doc/stable/reference/routines.fft.html

[4] Intel Corporation, “OpenVINO Toolkit Overview,” [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html

[5] Intel Corporation, “Intel NUC Product Brief,” [Online]. Available: https://www.intel.com/content/www/us/en/products/details/nuc.html