Infant Carrier Seat and Baby Presence Detection on Car Passenger Seat using FIUS Sensor

Course Information Technology
Modules Autonomous Intelligent Systems and Machine Learning
By Dr. Peter Nauth and Dr. Andreas Pech

Anushruthpal Keshavathi Jayapal
Matr. No.: 1502741
anushruthpal.keshavathi-jayapal@stud.fra-uas.de

Aswini Thirumaran
Matr. No.: 1510315
aswini.thirumaran@stud.fra-uas.de

Lavanya Suresh
Matr. No.: 1516065
lavanya.suresh@stud.fra-uas.de

Mandar Gokul Kale
Matr. No.: 1501517
mandar.kale@stud.fra-uas.de

Abstract – This project develops a sensor-based system to detect baby carrier seats and infants in car seats, even under challenging conditions like sunscreens or blankets. It employs machine learning techniques, including Multi-Layer Perceptron (MLPs), Support Vector Machine (SVMs), Random Forest, and XG Boost algorithm to analyze ultrasonic sensor data processed via Fast Fourier Transform (FFT). Model performance is evaluated using classification report analysis and confusion matrices. The system enhances passenger safety by accurately identifying infant presence and minimizing false detections. It also examines environmental impacts on sensor reliability and optimizes data preprocessing for real-time use. Through comprehensive testing, this study provides insights into machine learning applications in automotive safety, contributing to more reliable and intelligent infant detection systems in modern vehicles.

Index Terms – Red Pitaya, Ultrasonic Sensor, FIUS, Machine Learning, Fast Fourier Transform

I. INTRODUCTION

The integration of sensor technology has become a fundamental aspect of modern living spaces, revolutionizing how individuals interact with their environment. In the automotive sector, sensor deployment plays a critical role in ensuring safety and security, particularly in detecting unattended passengers or pets inside vehicles. Traditional sensing solutions, such as cameras, have been widely adopted for this purpose; however, they come with notable limitations, including reduced efficiency in low-light conditions and concerns regarding privacy.

In contrast, innovative approaches leveraging ultrasonic signals, such as the Red Pitaya sensor system, offer a promising alternative. Unlike conventional camera-based solutions, which rely on complex imaging and data-processing mechanisms, this system utilizes a simplified setup consisting of an integrated platform and an ultrasonic sensor. By transmitting and analyzing ultrasonic waves, it can effectively identify the presence of occupants without the drawbacks associated with visual-based methods. This approach not only enhances reliability but also provides a cost-effective and user-friendly solution for occupancy detection.

A distinguishing feature of the Red Pitaya sensor system is its ability to interpret ultrasonic signals reflected from various objects within a vehicle's interior. The frequency spectra of these reflected signals exhibit distinct variations when bouncing off solid objects, such as boxes, compared to human occupants. These differences, influenced by factors such as material composition and surface texture, present a significant opportunity for advancing human presence detection technologies.

Through the application of sophisticated signal processing techniques, the Red Pitaya sensor system demonstrates the capability to distinguish between animate and inanimate objects with high accuracy. This advanced detection mechanism enhances the reliability of occupancy sensing, making it particularly valuable in automotive environments where safety is a priority. By integrating this technology, vehicles can improve their ability to identify unattended occupants, thereby reducing the risks associated with overheating, suffocation, or accidental entrapment.

Machine Learning (ML) is a transformative field of artificial intelligence (AI) that enables computers to learn from data and make predictions or decisions without explicit programming. By recognizing patterns in data, ML systems improve their performance over time, making them invaluable in various industries, including healthcare, finance, automotive, and technology. The rapid advancement of computational power, coupled with the availability of large datasets, has significantly contributed to the widespread adoption of ML.

The first step of Machine Learning is to collect data as depicted in Fig.1. It is a very important task because it will determine how good predictive model can be. But data we gathered are, in most times, unstructured, contain a lot of noise or must take other forms to be useful for our machine learning. So, data need to be cleaned and pre-processed. [1]

At its core, machine learning revolves around algorithms that process data, identify patterns, and make informed decisions. These algorithms can be broadly categorized into three main types: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning is the most widely used category of ML, where the model is trained using labeled data. This means that for every input, there is a corresponding correct output. The model learns by analyzing these input-output pairs and minimizing errors. Supervised learning is used in applications such as email spam detection, medical diagnosis, and stock price prediction. Algorithms like Decision Trees, Support Vector Machines (SVMs), and Neural Networks are commonly used in this type of learning. Classification is one of the most frequently encountered decision-making tasks of human activity. A classification problem occurs when an object needs to be assigned into a predefined group or class based on several observed attributes related to that object.

In contrast, unsupervised learning deals with unlabeled data, where the goal is to find hidden patterns and relationships within the dataset. This method is useful when there is no predefined output, and the algorithm must discover the structure on its own. Clustering and anomaly detection are common tasks in unsupervised learning. For example, businesses use clustering to segment customers based on purchasing behavior, while financial institutions employ anomaly detection to identify fraudulent transactions. Popular unsupervised learning algorithms include K-Means Clustering, Principal Component Analysis (PCA), and Autoencoders.

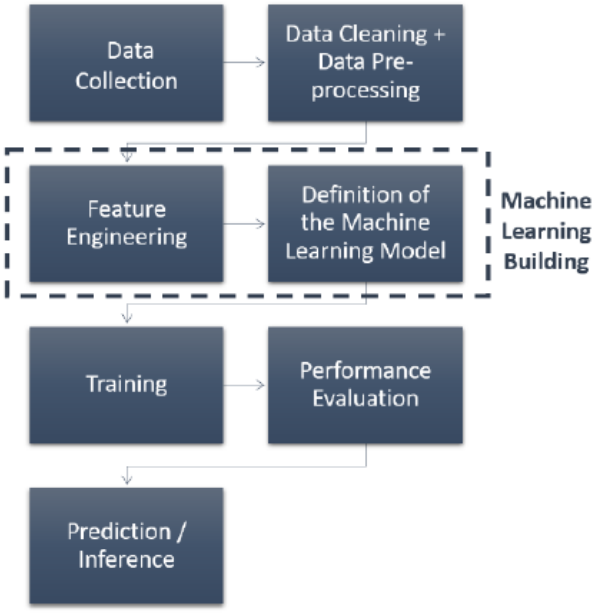

Fig 1: Components of machine learning

The impact of machine learning extends across various industries. In healthcare, ML assists in diagnosing diseases, discovering new drugs, and providing personalized treatment recommendations. The finance sector relies on ML for fraud detection, risk assessment, and algorithmic trading. Autonomous vehicles use ML for object detection, path planning, and decision-making, while natural language processing (NLP) enables AI-driven chatbots, speech recognition, and language translation. The ability of ML to process vast amounts of data and extract meaningful insights has made it indispensable in solving complex real-world problems.

II. METHODOLOGY

The fundamental concepts necessary for the successful implementation of the project is represented here. It integrates advanced technologies, including the Ultrasonic Red Pitaya sensor and Fast Fourier Transform (FFT) algorithms, to capture and analyze critical data within the automotive setting. Additionally, the project incorporates machine learning techniques, with a confusion matrix employed to evaluate classification accuracy.

This combination of technologies not only facilitates sensor data collection and analysis but also contributes to the creation of intelligent systems capable of detecting and responding to changing conditions inside the vehicle, particularly the presence of infants in car seats. By leveraging these approaches, the study aims to enhance safety measures and provide valuable insights into optimizing child passenger protection during travel.

A. Ultrasonic Sensor and Red Pitaya Measurement Board

Red Pitaya is an advanced, open-source measurement and control platform designed for various applications in electronics, signal processing, and automation. It serves as an affordable alternative to high-end lab instruments such as oscilloscopes, signal generators, and spectrum analyzers. With its versatile design, Red Pitaya is widely used by researchers, engineers, and hobbyists for prototyping, data acquisition, and real-time signal processing tasks.

At the core of Red Pitaya is a System-on-Chip (SoC) architecture that integrates an FPGA (Field-Programmable Gate Array) and an ARM processor as in Fig 2. This combination enables real-time signal analysis, making it suitable for applications in telecommunications, industrial automation, and medical instrumentation. The platform supports multiple programming languages, including Python, C/C++, and MATLAB, providing flexibility for users with different technical backgrounds.

One of Red Pitaya's most notable features is its web-based interface, which allows users to operate the device remotely through a browser. This eliminates the need for additional software installations and simplifies interaction with the hardware. Additionally, as an open-source platform, Red Pitaya fosters innovation, allowing users to modify and extend its functionalities to meet specific project requirements.

Red Pitaya is commonly used in signal acquisition and analysis, real-time control systems, and education. It can integrate with a variety of sensors, including ultrasonic sensors, making it ideal for projects that involve object detection, distance measurement, and environmental monitoring.

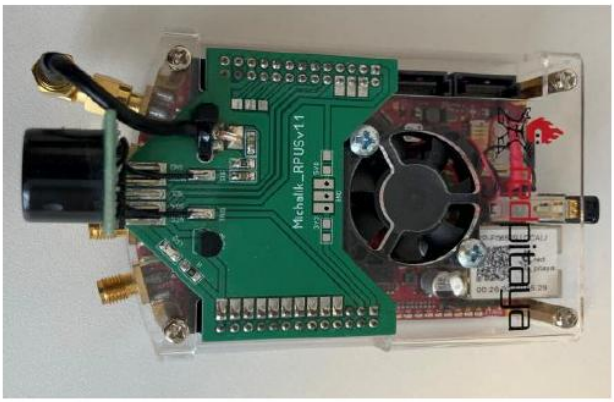

Fig 2 : Red pitaya (ultrasonic sensor)

Ultrasonic sensors are widely used in automation, robotics, and safety applications for distance measurement and object detection. The SRF02 Ultrasonic Sensor is a compact, efficient device that measures distances using ultrasonic sound waves. It operates by emitting an ultrasonic pulse and measuring the time it takes for the echo to return after bouncing off an object. This time delay is then converted into a distance measurement.

The SRF02 Ultrasonic Sensor is designed to provide accurate and efficient distance measurements using ultrasonic sound waves. It operates at a voltage of 5V with a low current consumption of approximately 4mA, making it energy-efficient for continuous operation. The sensor has a measurement range of 15 cm to 6 meters, with a resolution of 1 cm, ensuring precise detection of objects at varying distances. Operating at an ultrasonic frequency of 40 kHz, the SRF02 uses a single transducer for both transmission and reception of sound waves, which simplifies its design while maintaining high accuracy. Additionally, it supports both I2C (Inter-Integrated Circuit) and Serial UART communication protocols, allowing seamless integration with microcontrollers, Raspberry Pi, and platforms like Red Pitaya. These technical specifications make the SRF02 an ideal choice for applications in robotics, industrial automation, automotive safety, and obstacle detection systems.

What sets the SRF02 apart is that it is a single transducer sensor, meaning it uses the same component for both transmitting and receiving ultrasonic waves. This design reduces complexity while maintaining high accuracy. The sensor supports multiple communication protocols, including I2C (Inter-Integrated Circuit) and Serial UART,

making it highly compatible with microcontrollers, Raspberry Pi, and platforms like Red Pitaya.

Integrating the SRF02 ultrasonic sensor with Red Pitaya enables real-time distance measurement and advanced signal processing for applications in automotive safety, robotics, and industrial automation. In vehicles, this combination helps detect obstacles, unattended passengers, or pets, improving occupancy detection using machine learning. In robotics, it supports autonomous navigation and collision avoidance, allowing robots to adjust movements based on detected obstacles. In industrial settings, it facilitates automated monitoring, detecting structural anomalies, leaks, or fluid levels through ultrasonic analysis. This integration enhances efficiency, safety, and automation across various industries.

B. Analog to Digital Converter (ADC)

An Analog-to-Digital Converter (ADC) is a crucial component in modern electronics, enabling the transformation of continuous analog signals into discrete digital representations. This conversion is essential for processing, storing, and analyzing real-world signals such as sound, temperature, voltage, and light intensity using digital systems like microcontrollers, computers, and embedded platforms. ADCs are widely used in various industries, including telecommunications, medical instrumentation, industrial automation, and consumer electronics, allowing seamless interaction between analog input sources and digital processing systems.

The process of analog-to-digital conversion consists of three fundamental steps: sampling, quantization, and encoding. Sampling involves capturing values from the continuous analog signal at specific time intervals. The rate at which these samples are taken is called the sampling rate, which must be at least twice the highest frequency in the signal, as stated by the Nyquist-Shannon theorem, to avoid loss of information. Once the signal is sampled, the next step is quantization, where the sampled values are mapped to the nearest available discrete level within a predefined range. This process introduces a small error known as quantization noise, which affects the accuracy of the digital representation. The final step, encoding, involves converting the quantized values into a binary format so they can be processed by digital systems such as microprocessors and signal processors.

The accuracy and efficiency of an ADC depend on key parameters such as resolution, sampling rate, input voltage range, and signal-to-noise ratio. Resolution, also known as bit depth, defines how many discrete levels an ADC can use to represent an analog signal. A higher resolution results in a more precise digital representation. For example, an 8-bit ADC provides 256 discrete levels, while a 16-bit ADC provides 65,536 levels, allowing for much finer detail. The sampling rate determines how frequently an ADC captures data points from the analog signal, measured in samples per second. A higher sampling rate ensures better representation of fast-changing signals, such as those in high-speed data acquisition systems. The ADC's input voltage range determines the maximum and minimum voltages it can process, requiring signal conditioning when the input exceeds the predefined limits. Signal-to-noise ratio is another critical factor, as it determines how much of the ADC's output corresponds to the actual signal versus unwanted noise.

Different ADC architectures are designed to optimize speed, accuracy, and power efficiency. The Successive Approximation Register (SAR) ADC is widely used in microcontrollers and industrial applications due to its good balance of speed and resolution. Flash ADCs provide extremely fast conversions but require a large number of comparators, making them suitable for high-speed applications like radar and oscilloscopes. Sigma-Delta ADCs offers high resolution with slower conversion times and are commonly used in audio and precision measurement devices. Dual-Slope ADCs, though slower, provide excellent accuracy and noise immunity, making them ideal for digital multimeters and scientific instruments.

The applications of ADCs span across multiple industries. In telecommunications, they play a vital role in converting analog voice signals into digital data for efficient transmission. In audio processing, ADCs ensure high-fidelity sound recording and playback. Medical devices such as electrocardiograms and MRI scanners rely on ADCs for accurate digitization of physiological signals. Industrial automation uses ADCs to monitor environmental conditions such as temperature, pressure, and vibration. In automotive systems, ADCs are integrated into engine control units, airbag sensors, and advanced driver-assistance systems.

As technology continues to advance, ADCs are becoming faster, more power-efficient, and more accurate. They are essential for enabling seamless integration between the analog and digital worlds, improving the performance of modern electronic systems. With ongoing innovations in semiconductor design, ADCs will continue to play a crucial role in fields such as artificial intelligence, Internet of Things (IoT), and autonomous systems, driving new possibilities in digital signal processing.

C. Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is an essential algorithm used in signal processing and various scientific and engineering applications. It efficiently computes the Discrete Fourier Transform (DFT), converting a time-domain signal into its frequency-domain representation. This transformation is crucial for analyzing the frequency content of signals, providing insights into their underlying properties. By decomposing a signal into its constituent sinusoidal components, FFT allows researchers and engineers to better understand, modify, and optimize signals for a wide range of applications.

FFT data plays a fundamental role in identifying the frequency components of a given signal. Each frequency component corresponds to a distinct frequency bin, indicating the magnitude and phase of that particular frequency within the signal. This breakdown enables signal analysis and feature extraction in various domains, from audio processing to structural health monitoring. Compared to computing the DFT directly, FFT significantly reduces computational complexity from $O(N^2)$ to O(N log N), making it feasible for real-time signal processing and large-scale data analysis.

$$X(k) = \sum_{n=0}^{N-1} x(n).\, e^{-\frac{j2\pi}{N}.kn}$$

X(k) is the frequency-domain representation of the signal. x(n) is the input signal in time domain. N is the total number of samples

In audio signal processing, FFT data is extensively used for tasks such as equalization, filtering, and spectrum analysis. By analyzing the frequency components of an audio signal, engineers can remove noise, enhance sound quality, and detect specific frequency bands of interest. Music applications utilize FFT for pitch detection, audio synthesis, and automatic transcription, while speech processing systems leverage FFT data for voice recognition and speech enhancement techniques.

Another significant application of FFT data is in structural engineering, where vibration analysis helps in detecting mechanical defects in buildings, bridges, and machinery. By converting vibration signals into the frequency domain, FFT allows engineers to identify resonant frequencies, structural weaknesses, and irregularities that may indicate wear, fatigue, or potential failures. This approach improves predictive maintenance strategies, reducing downtime and preventing catastrophic failures in industrial machinery.

In communication systems, FFT plays a vital role in modern digital signal processing techniques. It is widely used in modulation, demodulation, and channel estimation, ensuring efficient data transmission over noisy or bandwidth-limited communication channels. Orthogonal Frequency Division Multiplexing (OFDM), a key technology in 4G and 5G wireless networks, relies on FFT for encoding and decoding signals efficiently, minimizing interference and maximizing data throughput. Similarly, radar and sonar systems utilize FFT to analyze reflected signals, enabling precise target detection and tracking.

FFT data also serves as the foundation for advanced digital signal processing (DSP) techniques, such as digital filtering, convolution, and correlation. By manipulating FFT data, researchers can design adaptive filters, remove unwanted noise from signals, and perform spectral analysis to detect hidden patterns. In biomedical engineering, FFT is used in ECG and EEG analysis to detect abnormalities in heartbeats or brain activity, aiding in early disease diagnosis and medical research.

Overall, FFT is an indispensable tool in signal analysis, offering a powerful method to extract valuable insights from complex signals. Whether in audio engineering, industrial diagnostics, telecommunications, or biomedical research, FFT data provides a comprehensive frequency-domain perspective, enabling researchers and engineers to make informed decisions, optimize system performance, and enhance technological advancements. As computational power continues to grow, FFT will remain a critical component in real-time and large-scale data

analysis, shaping future innovations in digital signal processing and beyond

D. Confusion Matrix:

A confusion matrix is a crucial evaluation tool in machine learning used to analyze the performance of classification models. It provides a structured way to compare the model's predictions with the actual values, offering deeper insights into how well the classifier distinguishes between different classes. Unlike a simple accuracy score, which may sometimes be misleading, a confusion matrix breaks down predictions into various categories, allowing for a more detailed assessment of model performance.

In a binary classification problem, a confusion matrix is a 2x2 table that consists of four essential components as depicted in *Fig 3*:

True Positive (TP): The model correctly predicted a positive outcome (the actual outcome was positive).

True Negative (TN): The model correctly predicted a negative outcome (the actual outcome was negative).

False Positive (FP): The model incorrectly predicted a positive outcome (the actual outcome was negative). Also known as a Type I error.

False Negative (FN): The model incorrectly predicted a negative outcome (the actual outcome was positive). Also known as a Type II error.

The confusion matrix is not limited to binary classification; it can be extended to multi-class classification problems as well. In this case, the matrix grows in size, with each row representing the actual class and each column representing the predicted class. The diagonal entries indicate correct predictions, while off-diagonal values represent misclassifications. By analyzing this matrix, one can determine which classes the model struggles with and take corrective measures such as collecting more data, tuning hyperparameters, or adjusting the decision threshold.

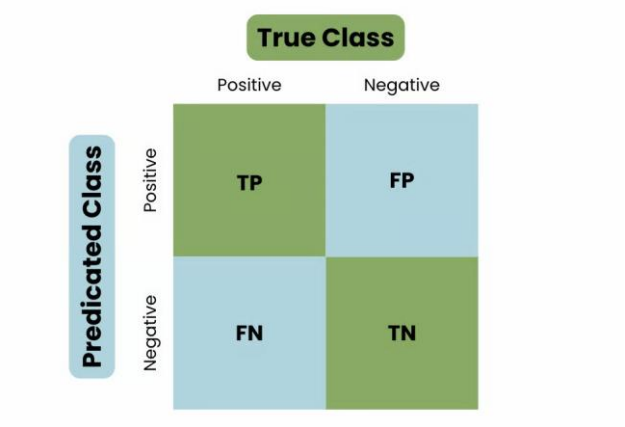

Fig 3 : Confusion matrix

One of the primary advantages of using a confusion matrix is that it enables the computation of various performance metrics. Accuracy, the most common metric, is calculated as the sum of correctly predicted instances (TP + TN) divided by the total number of instances. However, accuracy alone can be misleading, especially when dealing with imbalanced datasets. For example, if 95% of the data belongs to one class, a model that always predicts the majority class may achieve high accuracy but still fail to perform well in identifying the minority class.

E. Classification Report:

In machine learning, classification models are evaluated using various performance metrics. The classification report is one of the most comprehensive evaluation tools, providing detailed insights into how well a model classifies different categories. It summarizes key metrics: precision, recall, F1-score, and support for each class in a classification task. This report is particularly useful in scenarios where accuracy alone is insufficient, especially when dealing with imbalanced datasets where one class significantly outnumbers the others.

The classification report consists of multiple performance metrics that assess the model's effectiveness

Precision (Positive Predictive value): Precision measures how many of the instances predicted as a certain class actually belong to that class.

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)}$$

A high precision score indicates that when the model predicts a class, it is usually correct. It is important in situations where false positives are costly. In spam detection, precision is critical because falsely classifying a legitimate email as spam could cause problems.

Recall (Sensitivity or True Positive Rate): The Recall measures how many actual instances of a class were correctly identified.

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$

A high recall means the model captures most of the relevant cases. It is crucial in scenarios where false negatives are costly. In medical diagnosis, missing a cancer diagnosis (false negative) is more dangerous than falsely predicting cancer (false positive).

F1-Score (Harmonic Mean of Precision and Recall): The F1-score is the harmonic mean of precision and recall, balancing both metrics.

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The F1-score is particularly useful when the dataset is imbalanced. A high F1-score means the model has a good balance between precision and recall. In fraud detection, both false positives and false negatives are undesirable, so a balanced F1-score is important.

Support: Support refers to the number of actual occurrences of a class in the dataset. It is not a measure of performance but rather a count of instances in each class. Useful for analyzing whether a model is biased towards a majority class. If a dataset has 1000 instances of class A and only 10 instances of class B, support helps indicate this imbalance.

The classification report is crucial in evaluating machine learning models as it aids in model selection, handling imbalanced datasets, and guiding model improvement. By comparing precision, recall, and F1-score across different models, it helps in choosing the most suitable one for a given task. This is particularly important when dealing with imbalanced datasets, where accuracy alone can be misleading—for example, in fraud detection or rare disease diagnosis, where one class is significantly underrepresented. Additionally, the classification report guides model improvement by identifying weaknesses; if recall is low, the model may be missing too many positive cases, requiring adjustments to increase positive predictions. Conversely, if precision is low, the model may be generating too many false positives, indicating the need for better feature selection or threshold tuning. Thus, the classification report provides valuable insights to refine and optimize machine learning models effectively.

F. Machine Learning Models for Classification and Regression

Classification is one of the most frequently encountered decision making tasks of human activity. A classification problem occurs when an object needs to be assigned into a predefined group or class based on a number of observed attributes related to that object. [2]

Pattern recognition is the scientific discipline whose goal is the classification of objects into a number of categories or classes. Depending on the application, these objects can be images or signal waveforms or any type of measurements that need to be classified. There is the general term "pattern" to these objects. There are two approaches in Pattern recognition to classification procedure:

1. Decision-Theoretic Approach (Statistical Approach):

- represent the pattern as a vector in a vector space (Feature space).

- use a decision algorithm (mainly statistical) to decide which class the pattern is in.

2. Structural Approach (Syntactic Approach):

- represent the pattern by its structure, e.g. a string, a symbols, a graph connecting the primary elements, etc.;

- use parsing (grammatical) or graph matching to perform classification.

Machine learning algorithms play a crucial role in predictive analytics by enabling automated decision-making and pattern recognition. Various models have been developed to address classification and regression tasks, each with its strengths and limitations. This section provides an overview of four widely used machine learning techniques—Random Forest (RF), Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), and

Extreme Gradient Boosting (XGBoost)—highlighting their mechanisms, advantages, and disadvantages.

1. *XGBoost (Extreme Gradient Boosting)*

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that relies on gradient-boosted decision trees. Known for its speed and high predictive accuracy, it is often used for structured data tasks, including classification and regression [3]. XGBoost employs a boosting framework that gradually improves weak learners (decision trees) by correcting the errors made by their predecessors. This iterative process creates a strong and efficient model that reduces bias and variance.

XGBoost operates within a gradient boosting framework, where new trees are added incrementally to correct the errors of previous trees. XGBoost incorporates several optimizations to enhance its performance. It uses gradient-based optimization, where each new tree addresses the residual errors of the previous trees through the gradient descent method, allowing for more accurate predictions.

Additionally, XGBoost employs regularization techniques such as L1 (Lasso) and L2 (Ridge) to control model complexity and mitigate overfitting. The model also utilizes column and row sampling, which randomly selects subsets of features and data points, reducing correlations between trees and improving generalization. Unlike traditional gradient boosting, which builds trees sequentially, XGBoost takes advantage of parallel processing, significantly boosting its speed. Furthermore, XGBoost efficiently handles missing data by determining the optimal split directions based on the available information, making it robust for real-world datasets with missing values.

The key steps in training XGBoost model are:
Initialize the model: Start with a weak base learner (a single decision tree).

Compute Residuals: Determine the difference between predicted values and actual target values.

Train a New Tree: A new tree is created to minimize residual errors by learning the gradient of the loss function.

Update Model Weights: The contributions of each tree are adjusted with the learning rate to ensure gradual improvement of predictions.

Repeat Until Convergence: Steps 2-4 are repeated until a stopping criterion is met (e.g., reaching a defined threshold).

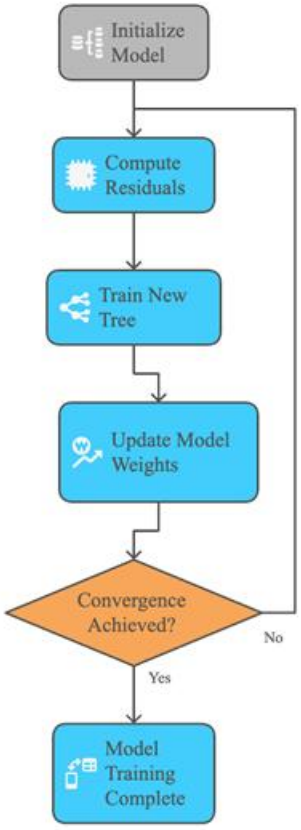

Fig 4 : XGBoost modelling steps

XGBoost offers superior performance compared to conventional boosting methods due to its efficient parallel processing capabilities, making it significantly faster. It also incorporates enhanced regularization techniques, including L1 and L2 regularization, which help mitigate overfitting as in Fig 4. The model is highly versatile, accommodating various objective functions such as regression, classification, and ranking, allowing it to be adapted for different challenges. Additionally, XGBoost effectively manages missing data by automatically determining optimal handling strategies, making it robust for real-world datasets.

However, XGBoost comes with certain drawbacks. While it is designed for speed, fine-tuning hyperparameters for large datasets can be resource-intensive, requiring substantial computational power. Furthermore, its performance is highly sensitive to hyperparameter selection, such as learning rate, tree depth, and the number of trees, necessitating careful tuning to achieve optimal results.

2. *Random Forest (RF)*

Random Forest is an ensemble learning technique that constructs several decision trees during the training process, yielding the most frequent class from the individual trees (for classification) or the average prediction (for regression) [4]. This method enhances predictive performance and minimizes overfitting by averaging multiple decision trees trained on different subsets of the training data.

The random forest (RF) is a tree-structured ensemble learning method as shown in Fig 5. The basic idea of building a random forest is: First, randomly extract new sample sets from the original data set with bootstrap resampling method, and use the new samples to construct classification decision trees; then, suppose there are feature, randomly select features as candidate split attributes, and select one of feature attributes for splitting according to the Gini index; the above process would be iterative executed, and finally, without cutting the decision tree, directly integrate the generated decision trees form a random forest to classify the new data, and the classification results are obtained by the majority voting strategy, the class with the most votes is the final classification result. [5]

The algorithm generates multiple decision trees using various subsets of the training data. Each tree is built from a bootstrap sample of the original dataset—a random selection made with replacement. During the construction of these trees, a random selection of features is chosen for each split, preventing the model from relying too heavily on any single feature. This element of randomness produces a diverse collection of trees, which, when aggregated, yield more reliable and generalized predictions. The final output is obtained by averaging the predictions from all individual trees for regression tasks or by selecting the majority vote for classification tasks. [3]

Fig 5 : Working Principle of Random Forest model

Random Forest offers several advantages, making it a robust machine learning model. It effectively reduces the risk of overfitting by averaging multiple decision trees, particularly when handling noisy datasets. Additionally, it provides insights into feature importance, aiding in feature selection and improving the understanding of the data's structure. The model is also highly flexible, capable of handling both classification and regression tasks while efficiently processing large, high-dimensional datasets.

However, Random Forest has certain limitations. Training multiple trees can be computationally expensive, requiring significant memory and processing power. Furthermore, while individual decision trees are interpretable, the overall structure of a Random Forest model becomes complex, reducing its clarity and making it challenging to explain its decision-making process.

3. *Support Vector Machine (SVM)*

The Support Vector Machine (SVM) is a supervised learning algorithm primarily used for classification tasks, but it can also be applied to regression. SVM aims to find the optimal hyperplane that separates data points from different classes with the largest margin. Maximizing this margin improves the model's robustness and generalization capability. [6]

SVM operates by transforming input features into a high-dimensional space through a kernel function, which allows it to handle non-linear decision boundaries. The algorithm then identifies the hyperplane that best separates the classes by maximizing the margin—the distance between the hyperplane and the closest data points from each class, known as support vectors as shown in Fig 6. Common kernel functions include linear, polynomial, and radial basis function (RBF), each enabling the model to recognize different patterns in the data. [7]

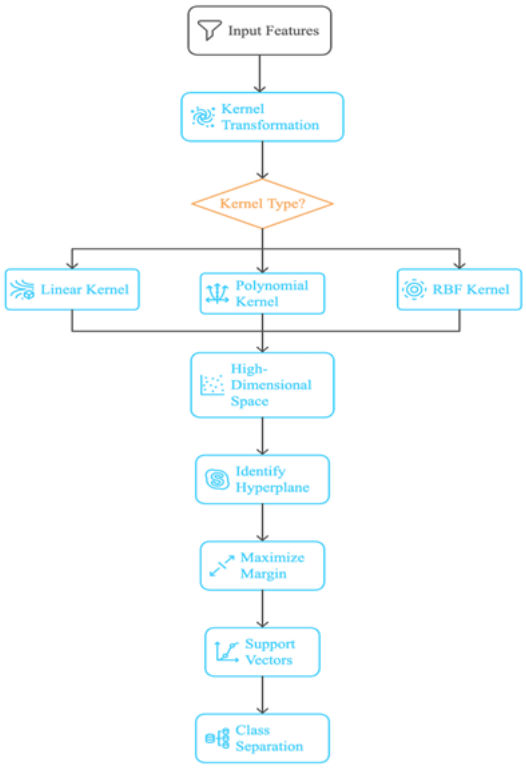

Fig 6 : SVM model workflow

Support Vector Machine (SVM) is particularly effective in high-dimensional spaces, making it advantageous when the number of features exceeds the number of samples. It is also memory efficient, as the model relies only on a subset of training points, known as support vectors, for its decision function. Additionally, SVM is highly versatile, capable of modeling complex non-linear decision boundaries through various kernel functions, allowing it to adapt to different types of data distributions.

However, SVM comes with certain drawbacks. Training can be computationally intensive, especially for large datasets and when using non-linear kernels. Moreover, the performance of SVM is highly sensitive to hyperparameter selection, particularly the choice of kernel and regularization parameters, requiring careful tuning to achieve optimal results.

4. *Multi-Layer Perceptron (MLP)*

A Multi-Layer Perceptron (MLP) is a type of feedforward artificial neural network comprising at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. MLPs can model complex, non-linear relationships and are widely utilized for classification and regression tasks. [8]

MLPs operate by transmitting input data through interconnected layers of neurons. Each neuron carries out a linear transformation of its inputs, followed by a non-linear activation function, such as ReLU (Rectified Linear Unit) or sigmoid. The network learns by adjusting the weights of these transformations to minimize the difference between predicted outputs and actual targets, a process supported by backpropagation and optimization techniques like gradient descent. [4]

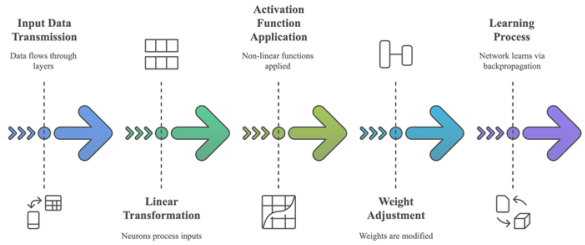

Fig 7 : MLP model steps

Multi-Layer Perceptrons (MLPs) are highly capable of modeling complex patterns, making them effective for capturing intricate non-linear relationships that simpler models might fail to recognize as in Fig 7. They offer

significant flexibility, as the number of layers and neurons can be adjusted to suit specific problems and datasets. Additionally, MLPs are widely applicable across various domains, including image and speech recognition, due to their ability to process diverse data types efficiently.

Despite these advantages, MLPs come with certain challenges. Training deep networks can be computationally demanding, requiring substantial resources, especially for large-scale applications. Moreover, MLPs are prone to overfitting if not properly regularized, which can negatively impact their ability to generalize to unseen data.

By considering these factors, these machine learning models were utilized in our project to enhance predictive accuracy and handle complex data patterns effectively. Their selection was based on their ability to balance performance, computational efficiency, and adaptability to the given dataset.

III. IMPLEMENTATION

A. Measurement Setup and Data Collection

1). Measurement Environment

A Ford Fiesta v16 model was used for real-time data collection purposes, as depicted in Fig 8. This vehicle was stationed in a controlled environment, specifically a basement garage at Frankfurt University of Applied Sciences, ensuring minimal external interference. The experimental setup was designed to simulate a typical in-vehicle scenario, focusing on detecting the presence of an infant carrier seat and later whether baby is present in the carrier seat.

A Red Pitaya measurement board with an FIUS ultrasonic sensor was strategically mounted on the dashboard on the passenger side.

The FIUS sensor was strategically positioned to ensure comprehensive data collection for detecting an infant car seat and the presence of a baby. The sensor was mounted on the car dashboard as illustrated in Fig. 10 at a height of 74 cm and the passenger seat height was measured at 29.5 cm, with a fixed positioning of 9.6 cm along the adjustment rail, maintaining consistency in data collection. This sensor was positioned at an angle of 28 degrees relative to the reference line, ensuring optimal coverage of the designated measurement zone as shown in the Fig 9.

Fig 8 : Ford Fiesta v16 model car

The distance from the sensor to the passenger seat backrest intersection was measured at 99 cm, forming a cone-shaped detection area extending downward. This placement allowed the sensor to cover both the seat and part of the backrest, ensuring accurate monitoring of the seat's occupancy status. The setup is illustrated in Fig 9, providing a side view of the experimental configuration.

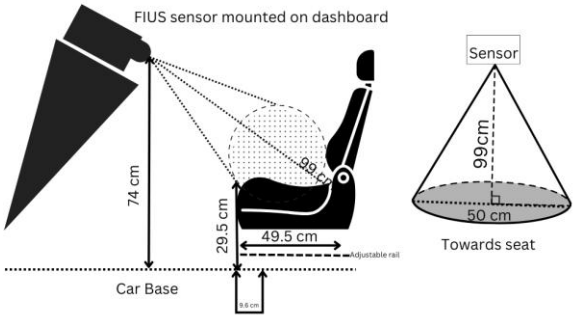

Fig. 9. Side view of the setup and sensor measurement zone

The infant carrier seat was securely mounted on the passenger seat using a seatbelt in a rear-facing orientation as in Fig 11. The distance from the sensor to the carrier seat handle was recorded as 23 cm, while the distance to the carrier seat body was 28.5 cm as shown in Fig 11. This precise positioning minimized obstructions and ensured reliable data acquisition for detecting seat occupancy.

Fig. 10. Sensor placement on Dashboard measurements

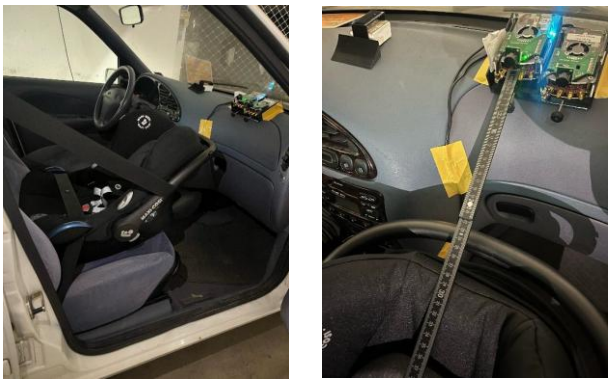

Fig.11. Carrier seat placement on passenger seat

For baby presence detection, two different baby dolls were used to simulate real infant seating conditions. Doll 1 measured 40 cm in height and was made of plastic, while Doll 2 measured 57 cm and was composed of plastic stuffed with cotton as shown in Fig 13. Both dolls were tested in multiple seating postures, including upright, lying down, and strapped positions, to replicate various real-world scenarios as illustrated in Fig 12. This setup allowed the sensor to capture critical environmental parameters, ensuring accurate and unbiased classification of seat occupancy and baby presence.

2) Hardware and Software Configuration

To facilitate real-time data acquisition, two Red Pitaya sensors were deployed within the vehicle on dashboard as shown in figure on passenger seat side. One of the sensor (on the left) was wirelessly connected via a GUI interface, configured through a UDP Client. The connected sensor was assigned a static IP address (192.168.129.1) for seamless communication with the data collection system.

Fig. 12. Carrier seat occupied with Baby

Fig.13. Used baby as subject and its height measurement (Baby1: 40 cm and Baby2: 57cm)

In our experiment, data acquisition can be performed directly in either ADC (Analog-to-Digital Converter) format or FFT (Fast Fourier Transform) format. However, we specifically opted to collect data in ADC format and subsequently convert it into FFT manually. This approach was chosen to enhance the accuracy and control over the signal processing pipeline.

By acquiring raw ADC readings, we retain the highest level of signal fidelity, allowing for customized preprocessing steps before applying FFT. This ensures that noise reduction techniques, such as low pass filtering and signal normalization, can be effectively implemented prior to frequency domain transformation. Additionally, manually converting ADC data into FFT provides flexibility in choosing optimal parameters for the transformation, such as windowing functions and resolution settings, which may not be adjustable when using direct FFT output from the measurement software.

Furthermore, performing FFT conversion manually allows us to analyze signal characteristics at different processing stages, aiding in a more refined feature extraction process.

This approach ultimately improves the reliability and interpretability of the frequency domain representation, leading to more robust feature engineering and better classification performance in our machine learning model.

GUI Software Setup for Measurement Acquisition:

The GUI software plays a critical role in configuring the measurement parameters and facilitating data collection. The following steps outline the setup and usage of the software:

Sensor Connection and Initialization: Upon launching the UDP Client software, the user must input the wireless network credentials corresponding to the specific sensor (designated as "Sensor 1" in our setup).A successful connection is indicated by a green signal at the bottom left of the GUI interface. In case of connectivity issues or interruptions, the sensor can be restarted using the designated "Restart Sensor" button. Additionally, the connection status can be verified through a separate "Check Connection" button, ensuring seamless communication with the hardware.

Data Storage and Organization: The recorded measurements are systematically saved in a designated directory: *\GUI_V0.23_2024-03-15\GUI_SW\save*

This ensures that all measurement files are stored in a structured manner, allowing for efficient data retrieval and processing.

Measurement Data Structure: The measurement scans are performed based on the input measurement counts from user. This approach ensures comprehensive data acquisition across different experimental conditions, enhancing the reliability of subsequent analysis.

The GUI provides an editable column where users can manually specify file names for each recorded dataset. This feature facilitates better dataset organization, allowing users to tag specific readings based on experimental conditions.

To initiate ADC data collection, the user must select the ‘ADC checkbox within the GUI interface. Once selected, clicking the ‘Start ADC button triggers the recording of time-domain data. The GUI automatically stops the measurement process upon completion and saves the resulting ADC data under the designated file name.

B. Data Collection

The ADC dataset was acquired using Red Pitaya technology to monitor and analyze conditions within a vehicle. The data collection process focused on capturing signals from ADC sensors placed inside the vehicle to study different scenarios involving a baby seat as shown in Fig 14. Each acquisition recorded real-time variations within the environment.

These ADC sensors form an integral part of the vehicle’s monitoring system, capturing a wide range of analog signals. With high precision and reliability, the sensors detect even the slightest variations, ensuring accurate monitoring of the baby seat’s surroundings.

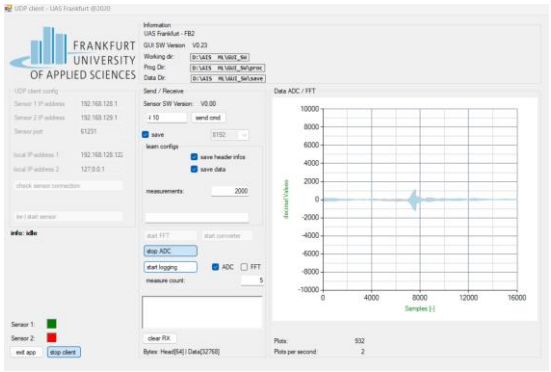

Fig .14. ADC Data Collection from measurement software

For this experiment, measurements were conducted for three specific tasks. First, data was collected to differentiate between an empty seat and a carrier seat to establish baseline conditions. Second, the presence of a baby in the carrier seat was analyzed to determine distinct signal variations. Third, the impact of obstructions, such as a blanket, sunscreen was examined to assess the system’s ability to detect covered infants.

To enhance data variability, systematic changes were introduced during the data collection process. The baby’s head position was tilted and hand positions were adjusted. Moreover, conditions were varied by covering the baby with a blanket, or sunscreen and altering the position of the carriage within the vehicle.

In total, 83,000 unique data samples were collected for the study. The samples were categorized as follows: 25,000 samples correspond to empty passenger seats, 25,000 samples correspond to passenger seat mounted with baby carrierseat , 21,000 samples represent carrier seats with a baby, and 12,000 samples involve scenarios where the baby was covered by obstacles such as a blanket or sunscreen. For the tasks, Task 1 (Seat Classifcation) utilized 50,000 samples, Task 2 (detecting baby presence in the carrier seat) used 46,000 samples, and Task 3 (identifying a baby in the carrier seat with obstructions like blankets or sunscreen) involved 33,000 samples.So, in total we used 129,000 samples for training and testing inclusive of all models.

This dataset provides a robust foundation for further signal processing and classification using FFT-based analysis, allowing for the detection and differentiation of these conditions based on frequency-domain characteristics.

C. Pre-processing of Data

Once the sensor data is collected, the data will undergo a preprocessing to make it fit for the modelling process.

ADC Values Description: The dataset contains raw ADC values that represent the time-domain data captured by the sensor. These values are sampled at a frequency of 1.953 MHz, with each sample recorded as a 12-bit amplitude value, providing high precision for analyzing the signal's variations over time.

By analyzing these raw ADC values, the signal strength and round-trip time (in microseconds) can be extracted, which are key to determining the distance between the sensor and the first detected object. These time-domain values are crucial for object detection and classification. However, the ADC values alone do not provide frequency-based insights; for such analysis, the data must be processed through the FFT (Fast Fourier Transform).

Header Description: The dataset header (of length 16) contains important metadata that provides context for the data collected. Below is the structure of the header and its corresponding descriptions:

Table 1: Dataset Header of raw ADC data

Parameter	**Description**
Data Length	16384 samples
Class Detected	1: Object, 2: Person
Measurement Type	0: FFT (frequency-domain), 1: ADC (time-domain)
Dependent on Table Element D	Frequency resolution (Δf) or sampling time (Δt) — currently irrelevant
Sampling Frequency (f_s)	1.953 MHz (Hz)
ADC Resolution	12 bits
Distance to First Object (Round-Trip Time)	Measured in microseconds (μs) — distance between sensor and first detected object
FFT Window Length	1024, 2048, 4096, or 8192 depending on configuration
Software Version (RP)	Version used in data acquisition
Aux 1	Additional data parameter (context-specific)
Aux 2	Additional data parameter (context-specific)
Data (Amplitudes and Frequencies)	Recorded amplitude values across frequencies ranging from 34.9 kHz to 44.9 kHz
Frequency Range	34.9 kHz to 44.9 kHz, with increments of 1.08-1.09 kHz
Frequency Resolution (Δf)	(Hz) — frequency resolution for FFT
Sampling Time (Δt)	(ns) — time between samples for ADC (currently irrelevant)

The Table 1 provides an overview of the key parameters in the dataset header, offering essential information for interpreting the data collected during the experiment.

1) Data Cleaning and Formatting

The collected raw ADC data was stored in text format and contained multiple headers, metadata, and noise artifacts as shown in Fig 15.

The preprocessing phase involved the following steps:

CSV Conversion: The data was converted from .txt to .csv format for structured storage and ease of analysis.

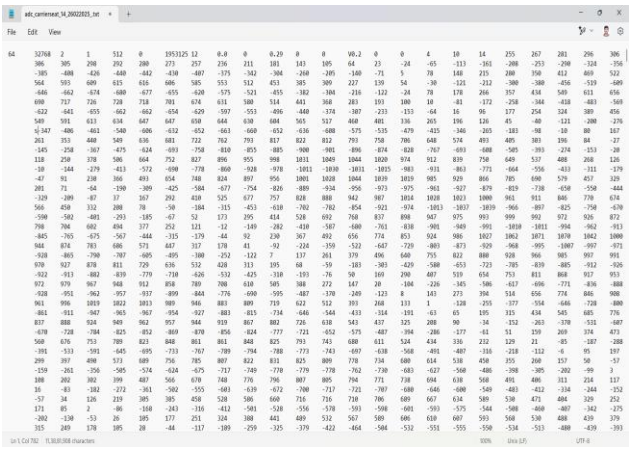

Fig.15. Raw AC Data

Merging Data Files: Since multiple recordings were taken for each scenario, all .csv files were combined into a single dataset for analysis as shown in *Fig.*.

```
# Combine all CSV data
combined_df = pd.concat(dfs, ignore_index=True)
combined_df
```

	0	1	2	3	4	5	6	7	8	9	…	16390	16391	16392	16393	16394	16395	16396	16397	16398	16399
0	64	32768	1	1	512	0	1953125	12	0.0	0	…	4	0	4	2	-3	-6	-7	-5	-2	-5
1	64	32768	1	1	512	0	1953125	12	0.0	0	…	-9	-8	-8	-11	-14	-16	-10	-12	-11	-12
2	64	32768	1	1	512	0	1953125	12	0.0	0	…	25	20	17	14	12	8	6	4	2	3
3	64	32768	1	1	512	0	1953125	12	0.0	0	…	-93	-96	-97	-100	-101	-101	-103	-108	-108	-109
4	64	32768	1	1	512	0	1953125	12	0.0	0	…	7	7	13	18	19	21	20	16	13	11
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
20995	64	32768	1	1	512	0	1953125	12	0.0	0	…	4	6	7	7	15	15	11	8	6	8
20996	64	32768	1	1	512	0	1953125	12	0.0	0	…	0	2	4	4	7	15	18	20	21	21
20997	64	32768	1	1	512	0	1953125	12	0.0	0	…	-2	2	3	-1	0	3	4	5	5	12
20998	64	32768	1	1	512	0	1953125	12	0.0	0	…	21	27	25	24	20	21	24	21	16	17
20999	64	32768	1	1	512	0	1953125	12	0.0	0	…	14	16	18	18	16	19	21	21	20	23

21000 rows × 16400 columns

Fig.16.Combined dataframe of ADC data

Header Removal: The first 16 columns of each measurement, containing metadata such as sampling frequency and measurement timestamps, were removed to retain only the numeric ADC values as depicted in Fig. 17.

```
#remove headers
raw_data = combined_df.iloc[:, 16:]
raw_data
```

	16	17	18	19	20	21	22	23	24	25	…	16390	16391	16392	16393	16394	16395	16396	16397	16398	16399
0	-6	-8	-6	-6	38	51	53	55	52	54	…	4	0	4	2	-3	-6	-7	-5	-2	-5
1	-12	-10	-11	-7	-3	34	21	3	-16	-35	…	-9	-8	-8	-11	-14	-16	-10	-12	-11	-12
2	4	3	3	-1	-11	1	18	33	49	65	…	25	20	17	14	12	8	6	4	2	3
3	-109	-107	-107	-107	-106	-10	-9	-13	-16	-22	…	-93	-96	-97	-100	-101	-101	-103	-108	-108	-109
4	10	10	6	-33	-47	-62	-62	-58	-57	-48	…	7	7	13	18	19	21	20	16	13	11
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
20995	11	11	12	16	116	95	74	48	26	10	…	4	6	7	7	15	15	11	8	6	8
20996	20	17	18	18	18	152	146	135	116	93	…	0	2	4	4	7	15	18	20	21	21
20997	9	10	15	19	177	171	167	158	152	141	…	-2	2	3	-1	0	3	4	5	5	12
20998	17	17	13	11	14	-20	-37	-54	-69	-76	…	21	27	25	24	20	21	24	21	16	17
20999	22	26	30	35	40	43	136	149	150	143	…	14	16	18	18	16	19	21	21	20	23

21000 rows × 16384 columns

Fig.17. Header removed from ADC data

2) Low-Pass Filtering and Signal-to-Noise Ratio Calculation

Given that real-world data contains noise, a low-pass filter was applied to smoothen the signal and remove high-frequency disturbances as in Fig 18. Additionally, the signal-to-noise ratio (SNR) was computed to evaluate data quality before proceeding with further transformations as shown in Fig 19.

```
from scipy.signal import butter, filtfilt

def butter_lowpass_filter(signal, cutoff=30e3, fs=1e6, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, signal)

filtered_data = np.array([butter_lowpass_filter(row) for row in adc_data])
filtered_data
```

Fig. 18. Applying Low pass filter to the ADC data to reduce Noise in the signal

```
def calculate_snr(signal, noise):
    power_signal = np.mean(signal ** 2)
    power_noise = np.mean(noise ** 2)
    return 10 * np.log10(power_signal / power_noise)

snr_after = calculate_snr(filtered_data, adc_data - filtered_data)
print(f"SNR After Fixing: {snr_after:.2f} dB")
```

```
SNR After Fixing: 20.94 dB
```

Fig. 19. Signal to Noise Ratio (SNR) Calculation

3)FFT Processing of ADC Data

The ADC data is first converted into a NumPy array for efficient numerical computation. Since raw ADC values are in the time domain, a Hanning window is applied to the signal before performing the Fast Fourier Transform (FFT). The purpose of this windowing function is to reduce spectral leakage, which occurs when non-periodic signals are transformed into the frequency domain.

Once the windowed ADC data is ready, an FFT operation is performed. This transforms the time-domain signal into its corresponding frequency components, allowing for spectral analysis as shown in Fig 20. The frequency bins are then computed using the known sampling rate of 1.953 MHz, which helps in mapping FFT results to actual frequency values.

```python
# Convert the pandas Series to a numpy array
adc_array = adc_data_selected_columns

# Choose a window function - Hanning window in this case
window = np.hanning(len(adc_array))

# Apply the window function to your data
windowed_adc_data = adc_array * window

# Perform FFT on the windowed data
fft_result = np.fft.fft(windowed_adc_data)

# Frequency bins (assuming you know the sampling rate)
sampling_rate = 1953125  # Example: 1000 Hz, replace with your actual sampling rate
min_freq = 35000  # 35 kHz
max_freq = 45000  # 45 kHz

n = len(adc_array)
freq = np.fft.fftfreq(n, d=1/sampling_rate)
# Calculate the magnitude and phase of the FFT result
magnitude = np.abs(fft_result)
phase = np.angle(fft_result)

# Create a DataFrame
fft_df = pd.DataFrame({
    'Frequency': freq,
    'FFT Magnitude': magnitude,
    'Phase': phase
})


fft_df.head()  # Display the first few rows of the DataFrame
```

	Frequency	FFT Magnitude	Phase
0	0.000000	88895.479976	0.000000
1	92.390019	42749.875976	3.114080
2	184.780038	3517.831335	-2.625055
3	277.170057	2235.324771	0.837800
4	369.560076	1569.457500	-1.524020

Fig.20. ADC to FFT Conversion

The FFT result consists of complex numbers, which are further processed to extract two key features:

Magnitude: Represents the strength of each frequency component.
Phase: Indicates the phase shift of each frequency component.

```
fft_df
```

	Frequency	FFT Magnitude	Phase	Object_Presence
0	0.000000	88895.479976	0.000000	0
1	92.390019	42749.875976	3.114080	0
2	184.780038	3517.831335	-2.625055	0
3	277.170057	2235.324771	0.837800	0
4	369.560076	1569.457500	-1.524020	0
...	...	...	...	...
21135	-461.950095	513.908183	-1.315854	0
21136	-369.560076	1569.457500	1.524020	0
21137	-277.170057	2235.324771	-0.837800	0
21138	-184.780038	3517.831335	2.625055	0
21139	-92.390019	42749.875976	-3.114080	0

21140 rows × 4 columns

Fig. 21. FFT data with Label of Classification

Finally, the computed frequency values, magnitudes, and phases are stored in a structured Data frame. A label of classification is added for the data as shown in Fig 21. This allows for easy visualization and analysis of the frequency spectrum, which is crucial for object detection and classification in signal processing applications.

D. Feature Extraction

To enhance the classification performance, statistical and spectral features were extracted from the FFT-transformed data as depicted in *Fig.*. The following features were computed for each dataset:

1) Spectral Features

Mean FFT Magnitude: Measures the average intensity of frequency components.

Standard Deviation of FFT Magnitude: Captures the variability in frequency magnitudes.

Maximum and Minimum FFT Magnitude: Identifies peak and trough values in the frequency spectrum.

Median FFT Magnitude: Represents the central tendency of the FFT spectrum.

Total Energy (Sum of FFT Magnitudes): Provides insight into the overall signal strength.

Spectral Entropy: Measures the randomness of energy distribution across the frequency spectrum.

Spectral Centroid: Weighted mean of the frequencies present in the signal

Spectral Bandwidth: Spread of the frequencies around the centroid.

Spectral Flatness: Ratio of geometric mean to arithmetic mean of the spectrum.

2) Phase Features

Phase Variance: Measures the dispersion of phase values.

Phase Mean: Computes the average phase angle.

Phase Difference: Calculates the mean difference between consecutive phase values.

The extracted features were stored as structured datasets for further processing.

```python
import numpy as np
from scipy.stats import entropy
from scipy.signal import find_peaks

def extract_features(fft_freqs, fft_mags, fft_phase):
    # Normalize magnitudes to prevent division errors
    norm_mags = fft_mags / np.sum(fft_mags) if np.sum(fft_mags) > 0 else fft_mags

    # Spectral Centroid (Weighted Mean of Frequencies)
    spectral_centroid = np.sum(fft_freqs * norm_mags) / np.sum(norm_mags)

    # Spectral Bandwidth (Spread around centroid)
    spectral_bandwidth = np.sqrt(np.sum(norm_mags * (fft_freqs - spectral_centroid) ** 2))

    # Spectral Flatness (Geometric Mean / Arithmetic Mean)
    spectral_flatness = np.exp(np.mean(np.log(fft_mags + 1e-10))) / np.mean(fft_mags + 1e-10)

    # Phase Features
    phase_variance = np.var(fft_phase)  # Variance of phase angles
    phase_mean = np.mean(fft_phase)  # Mean of phase angles
    phase_diff = np.mean(np.diff(fft_phase))  # Average phase difference between frequencies

    return {
        "mean_fft": np.mean(fft_mags),
        "std_fft": np.std(fft_mags),
        "max_fft": np.max(fft_mags),
        "min_fft": np.min(fft_mags),
        "median_fft": np.median(fft_mags),
        "sum_fft": np.sum(fft_mags),
        "spectral_entropy": entropy(norm_mags),  # Energy spread
        "spectral_centroid": spectral_centroid,
        "spectral_bandwidth": spectral_bandwidth,
        "phase_variance": phase_variance,
        "phase_mean": phase_mean,
        "phase_diff": phase_diff,
    }
```

Fig.22.Feature extraction from FFT data

E. Augmentation via Perturbation

To improve model robustness, synthetic data augmentation was performed by adding minor perturbations to the extracted feature set. A perturbation factor of ±3% was applied to introduce variability without distorting key signal characteristics. This was achieved using:

Both original and perturbed datasets were horizontally concatenated, followed by vertical stacking of empty-seat and carrier-seat cases or carrier seat with and without baby cases. This resulted in a final dataset used for model training.

```python
# Function to apply perturbation based on reference values
def add_perturbation(reference_features, num_rows, perturb_range=0.03):
    perturbed_data = []

    for _ in range(num_rows):
        perturbed_features = {
            key: value * (1 + np.random.uniform(-perturb_range, perturb_range))
            for key, value in reference_features.items()
        }
        perturbed_data.append(perturbed_features)

    return pd.DataFrame(perturbed_data)
```

Fig.23.Generating Perturbed data based on extracted features

F. Model Training and Evaluation

To classify passenger seat occupancy and detect the presence of a baby in a baby carrier seat, and also with obstructions different machine learning models were employed. The classification was divided into three tasks:

1. **Passenger Seat Occupancy Detection**: To determine whether a seat was empty or mounted with a baby carrier seat, Extreme Gradient Boosting (XGBoost) and a Multilayer Perceptron (MLP) neural network were utilized. These models were trained on labeled seat occupancy data to distinguish between an empty seat and one containing a baby carrier effectively.

2. **Baby Presence Detection**: To classify whether a baby was present in the baby carrier, a Random Forest (RF) classifier and a Support Vector Machine (SVM) were implemented. These models were trained to differentiate between an occupied carrier with and without a baby inside.

3. **Baby Presence Detection with Obstructions:** To determine whether a baby was present in a baby carrier even when covered with a blanket or sunscreen, an Extreme Gradient Boosting (XGBoost) model was utilized These models were trained on labeled data to distinguish between an occupied baby carrier without baby and one where the baby was obscured by coverings.

The dataset was split into 80% for training and 20% for testing to ensure robust model evaluation. A confusion matrix and classification report were generated for each classification task to assess model performance, analyze misclassification patterns, and refine model parameters accordingly.

IV. RESULTS

Task 1: Seat Occupancy Detection

A. Implementation of XGBoost Model for Seat Occupancy Classification

The XGBoost (Extreme Gradient Boosting) model was implemented to classify seat occupancy conditions using FFT-transformed ADC data. The process involved training the model on raw FFT data, extracting and augmenting features, and refining the model through feature selection to improve accuracy.

1) Initial Model Training on Raw FFT Data

The raw ADC signals were first transformed into the frequency domain using Fast Fourier Transform (FFT) as explained in our previous sections.

```python
# Load your dataset (df_combined)
X = dataset_1_rawdata.drop(columns=['Object_Presence'])  # Features
y = dataset_1_rawdata['Object_Presence']  # Labels

# Split into training & testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features (XGBoost handles unscaled data well, but scaling can help)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert data into DMatrix format (optimized for XGBoost)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Set up XGBoost parameters
params = {
    'objective': 'binary:logistic',  # For binary classification
    'eval_metric': 'logloss',       # Loss function
    'max_depth': 6,                  # Depth of trees
    'learning_rate': 0.1,            # Step size shrinkage
    'n_estimators': 1000,            # Number of trees
    'random_state': 42
}

# Train XGBoost model
clf = xgb.XGBClassifier(**params)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report for rawdata:\n", classification_report(y_test, y_pred))
```

Fig.24.XG Boost model

The XGBoost model was trained using raw data with an 80-20 split for training and testing. The feature set was standardized using StandardScaler before training. The model was configured for binary classification with the "binary:logistic" objective function and evaluated using the logloss metric. Key hyperparameters included a maximum tree depth of 6, a learning rate of 0.1, and 1000 estimators as in Fig 24. The training and testing data were converted into XGBoost's optimized DMatrix format for efficient computation. After training, predictions were made on the test set, and model performance was assessed using accuracy and a classification report to evaluate its effectiveness in detecting object presence in seats.

The XGBoost model trained using raw data for detecting empty versus carrier seat conditions achieved an overall accuracy of 54.01 percent, indicating that the model's performance is only slightly better than random guessing as in Fig 25. The classification report shows a precision of 55 percent for empty seats and 53 percent for carrier seats, while the recall values are 56 percent and 52 percent, respectively. The f1-score for both classes is approximately 54 percent, suggesting a balance between precision and recall. The macro and weighted average scores also remain around 54 percent, indicating that the model performs similarly across both classes without significant bias.

```
Accuracy: 0.5401044138585667

Classification Report for rawdata:
              precision    recall  f1-score   support

         0.0       0.55      0.56      0.55      4271
         1.0       0.53      0.52      0.53      4157

    accuracy                           0.54      8428
   macro avg       0.54      0.54      0.54      8428
weighted avg       0.54      0.54      0.54      8428
```

Fig. 25.Classification report of XG Boost model trained with raw data

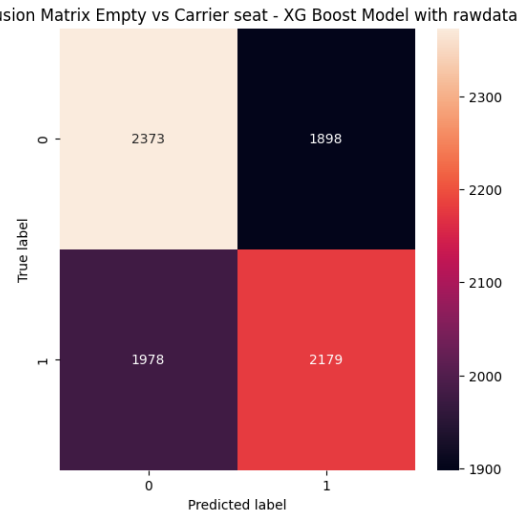

Fig. 26. Confusion Matrix for XG Boost model Raw data

The confusion matrix as in Fig 26 revealed misclassifications, particularly between empty and occupied seat classes, leading to the need for feature extraction

2) *Re-training after Feature Extraction and Augmentation*

To improve model performance, additional spectral features were extracted, including mean, variance, skewness, and kurtosis of FFT magnitudes. Additionally, dominant frequencies, spectral entropy, and frequency band energy distributions were incorporated as shown in Fig 27. To further enhance the robustness of the model, data augmentation techniques were applied to simulate realistic variations in the data. Despite these modifications upon training the model with same parameters as previous, the model exhibited overfitting, achieving high training accuracy but poor generalization to unseen test data.

```
# Extract features for fft_empty
reference_features_empty = extract_features(fft_frequencies_empty, fft_magnitudes_empty, fft_phase_empty)
reference_features_empty
```

```
{'mean_fft': np.float64(1388.8087999687539),
 'std_fft': np.float64(1270.5024649486377),
 'max_fft': np.float64(88895.4799756217),
 'min_fft': np.float64(8.222425953483421),
 'median_fft': np.float64(1103.5419233163489),
 'sum_fft': np.float64(29359418.031339455),
 'spectral_entropy': np.float64(9.696724055325122),
 'spectral_centroid': np.float64(-10.742604308958107),
 'spectral_bandwidth': np.float64(611614.4854242009),
 'phase_variance': np.float64(3.286245977227801),
 'phase_mean': np.float64(0.0001486089240108682 2),
 'phase_diff': np.float64(-0.0001473144568057513)}
```

Fig. 27. Features Extracted from empty seat FFT data

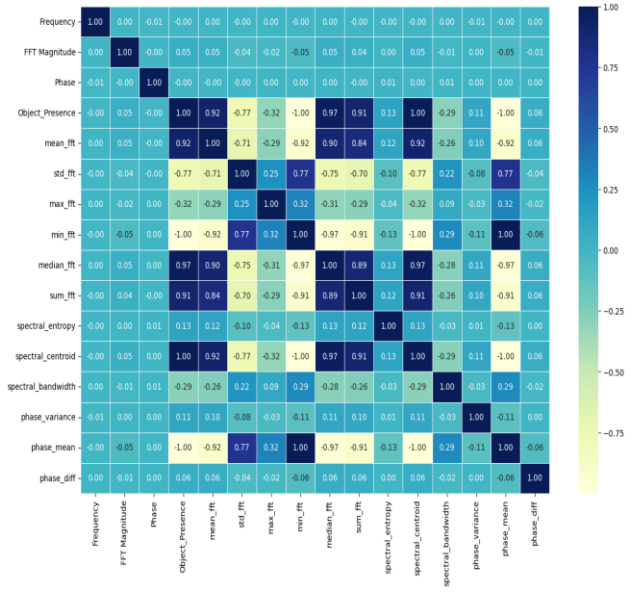

Fig. 28. Correlation matrix of extracted features

The classification report for the model trained on extracted features data as in Fig 29, shows a perfect classification performance with an accuracy of 100%. Both classes, empty (0) and carrier (1), achieved precision, recall, and F1-scores of 1.00, indicating that the model correctly identified all instances without any misclassification. However, this result suggests that the model may be overfitting to the training data, as it performs exceptionally well on the test set but might not generalize well to unseen data as plotted in Fig 30. This overfitting could be due to the inclusion of too many features or the model becoming too tailored to the specific dataset. The same XGBoost model as used previously was applied, but with the addition of extracted features, which likely contributed to this high performance.

```
Accuracy: 1.0

Classification Report for extracted features data:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      4271
         1.0       1.00      1.00      1.00      4157

    accuracy                           1.00      8428
   macro avg       1.00      1.00      1.00      8428
weighted avg       1.00      1.00      1.00      8428
```

Fig. 29. Classification report of XG Boost model trained with feature extracted data depicting Overfitting

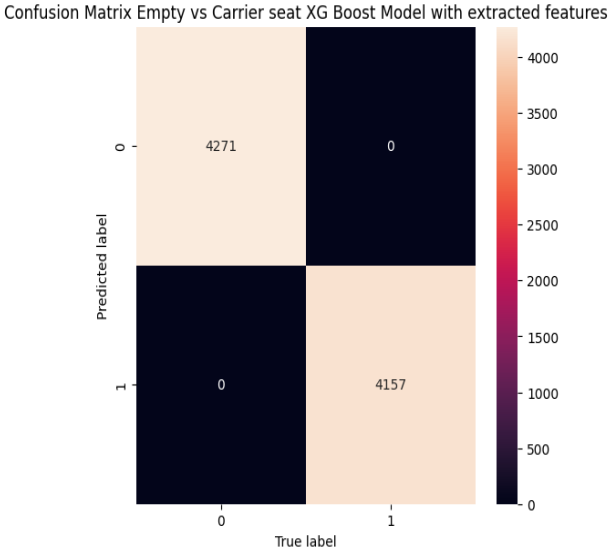

Fig. 30. Confusion Matrix for XG Boost model trained with features extracted data showing Overfitting

3) *Addressing Overfitting Using Feature Selection*

To mitigate overfitting, a correlation matrix as in Fig 28, analysis was performed to identify and remove redundant features. Features with correlation coefficients greater than 0.8 were considered redundant and excluded from the dataset as shown in Fig 31 and Fig 32. This approach reduced the complexity of the model and improved

generalization. After filtering the features, the XGBoost model was retrained with optimized parameters.

```python
# Compute the correlation matrix
corr_matrix = dataset_1.corr()

# Set a threshold for correlation (e.g., 0.8)
threshold = 0.8

# Find pairs of highly correlated features
highly_correlated = np.where(np.abs(corr_matrix) > threshold)

# Create a set to store the columns to drop
to_drop = set()

# Loop through the indices of the highly correlated pairs
for i, j in zip(*highly_correlated):
    if i != j:  # Avoid diagonal (self-correlation)
        feature_i = corr_matrix.columns[i]
        feature_j = corr_matrix.columns[j]
        # Ensure we do NOT drop 'Object_Presence'
        if feature_j != "Object_Presence":
            to_drop.add(feature_j)  # Drop one of the correlated features

# Drop the highly correlated features from the dataframe
dataset_1_reduced = dataset_1.drop(columns=to_drop)

corr_matrix = dataset_1_reduced.corr()
fig= plt.subplots(figsize=(15, 10))
ax= sns.heatmap(corr_matrix,
                annot=True,
                linewidths=0.5,
                fmt='.2f',
                cmap='YlGnBu')

# Display the dropped features and new dataframe shape
print("Dropped features: ", to_drop)
print("New dataframe shape: ", dataset_1_reduced.shape)
```

Fig. 31. Addressing Overfitting Using Feature Selection

Dropped features: {'mean_fft', 'spectral_centroid', 'sum_fft', 'min_fft', 'median_fft', 'phase_mean'}

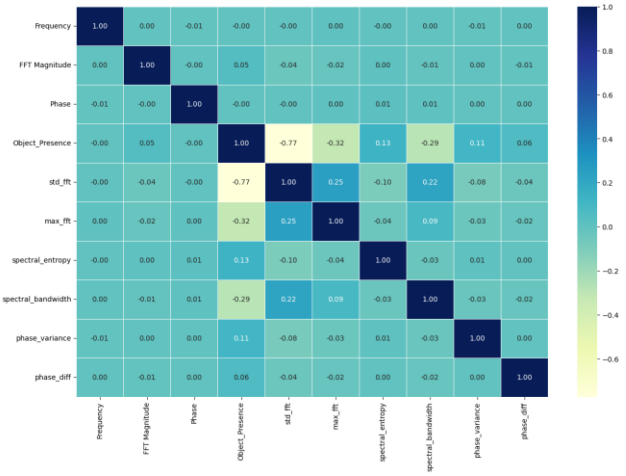

Fig. 32.Correlation Matrix for selected features

4) *Final Model Evaluation and Performance Metrics*

After applying feature selection and optimizing model parameters, the model achieved a final accuracy of 91.58%. The confusion matrix showed significant improvements with fewer misclassifications.

```
Accuracy: 0.9157570004746084

Classification Report for selected features data:
              precision    recall  f1-score   support

         0.0       0.92      0.92      0.92      4271
         1.0       0.91      0.91      0.91      4157

    accuracy                           0.92      8428
   macro avg       0.92      0.92      0.92      8428
weighted avg       0.92      0.92      0.92      8428
```

Fig. 33. Classification report of XG Boost model trained with feature selected data

The classification report as in Fig 33 for the model trained with the reduced features dataset shows a high classification performance, achieving an accuracy of 91.58%. The model demonstrated precision, recall, and F1-scores of 0.92 and 0.91 for both classes, empty (0) and carrier (1) respectively indicating that it performed well in correctly classifying both classes. This suggests that the feature selection process, based on the correlation matrix, helped improve the model's efficiency by eliminating less relevant features, leading to better overall performance. The accuracy and other metrics are consistently high, indicating that the reduced feature set still retains the essential information needed for accurate classification which is observed in the confusion matrix as in Fig 34.

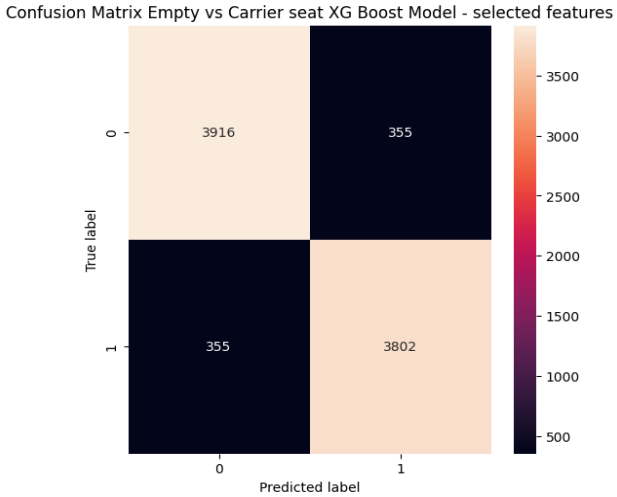

Fig. 34. Confusion Matrix for XG Boost model trained with features extracted data

For the XGBoost model, the ROC curve as in Fig 35, was analyzed to evaluate its classification performance, along with feature importance to identify the most influential predictors and it was observed that most of the selected features influenced for prediction, however the

contribution of the standard deviation of FFT was very high as shown in Fig 36.

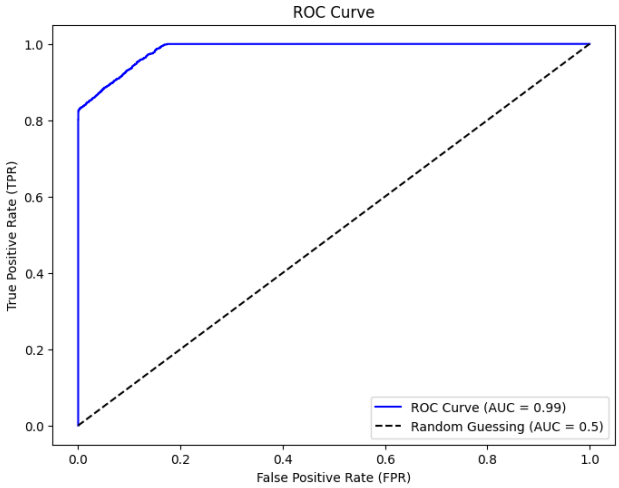

Fig. 35. ROC Curve

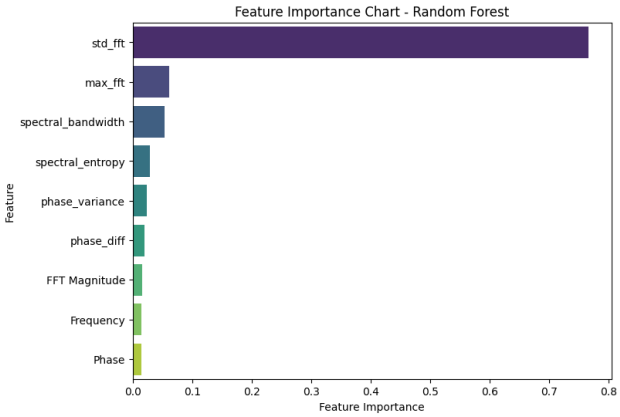

Fig. 36. Feature Importance chart - XG Boost for selected features

5) XG Boost Result analysis

The XGBoost model initially performed poorly on raw FFT data, achieving an accuracy of approximately 54%. This indicated challenges in capturing meaningful patterns from the raw data. After performing feature extraction, the model achieved perfect accuracy (100%), but this was an indication of overfitting, as the model demonstrated high performance on the training set but struggled to generalize on unseen data. In the final step, by applying feature selection techniques based on correlation analysis, the model's complexity was reduced, leading to a significant improvement in performance. This resulted in an accuracy of approximately 91.58%, with high precision and recall scores. This experiment demonstrated that feature selection and engineering are crucial steps in improving the model's generalization ability, making XGBoost a powerful tool for the seat occupancy classification task.

The comparison of different training scenarios is summarized in the Table 2 below.

Table 2 : XG Boost Model Performance Comparison for Seat Occupancy Classification

Training Scenario	Accuracy	Precision	Recall	F1-Score
Raw Data	54.01%	Moderate	Moderate	Moderate
Feature Extracted Data	100%	High	High	Perfect (Overfitting)
Feature Selected Data	91.58%	High	High	Improved

B. Implementation of MLP Model for Seat Occupancy Classification

A Multi-Layer Perceptron (MLP) model was implemented as in Fig 37 to classify seat occupancy conditions using FFT-transformed ADC data. The process involved training the model on raw FFT data, extracting and augmenting features, and refining the model through feature selection to improve accuracy.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Load dataset
X = dataset_1.drop(columns=['Object_Presence']).values
y = dataset_1['Object_Presence'].values

# Split data into training & testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build MLP model with an explicit Input() layer
mlp_model = Sequential([
    Input(shape=(X_train.shape[1],)),  # Explicit Input layer
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])

# Compile model
mlp_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train model
mlp_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate model
mlp_loss, mlp_acc = mlp_model.evaluate(X_test, y_test)
print(f"MLP Accuracy: {mlp_acc:.4f}")
```

Fig. 37.MLP Model

1) *Initial Model Training on Raw FFT Data*

The raw ADC signals were first transformed into the frequency domain using Fast Fourier Transform. The Multi-Layer Perceptron, a neural network-based algorithm, was selected for training with parameters including an input layer matching the number of FFT bins, two hidden layers with 64 and 32 neurons, respectively, and a ReLU activation function. The dataset was split into 80% training and 20% testing.

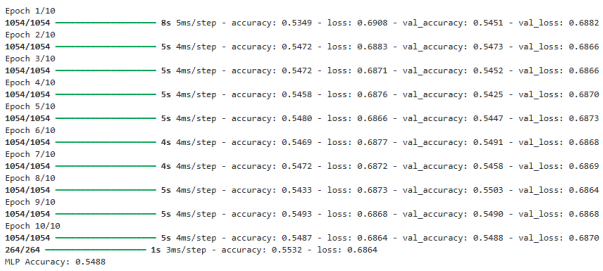

Fig. 38.Training Output (epochs) for raw data

```
264/264 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
MLP Accuracy: 0.5488

Classification Report for rawdata:
              precision    recall  f1-score   support

         0.0       0.55      0.59      0.57      4271
         1.0       0.55      0.51      0.53      4157

    accuracy                           0.55      8428
   macro avg       0.55      0.55      0.55      8428
weighted avg       0.55      0.55      0.55      8428
```

Fig. 39. Classification report of MLP model trained with raw data

The Multi-Layer Perceptron model was trained on the raw data, which consisted of Fast Fourier Transform derived features from the raw ADC signals as in Fig 38. After training, the model achieved an accuracy of approximately 54.88%. The classification report in Fig 39 showed that the MLP had similar performance for both classes: it achieved a precision of 0.55 and recall of 0.59 for the "empty seat" class (0.0), and a precision of 0.55 and recall of 0.51 for the "occupied seat" class (1.0). These results indicate that the model struggled with distinguishing between the two classes effectively. The F1-score, which balances precision and recall, was also relatively low at 0.55 for both classes, further highlighting the need for model refinement and feature enhancement. The macro and weighted averages for precision, recall, and F1-score were all around 0.55, suggesting that the MLP model's performance was consistent across the classes but still suboptimal.

However, the initial results showed low accuracy, indicating that the raw FFT features alone were not sufficient for effective classification as seen in Fig 40. The confusion matrix revealed misclassifications, particularly between empty and occupied seat classes. This highlighted the need for further feature extraction and refinement to improve model performance.

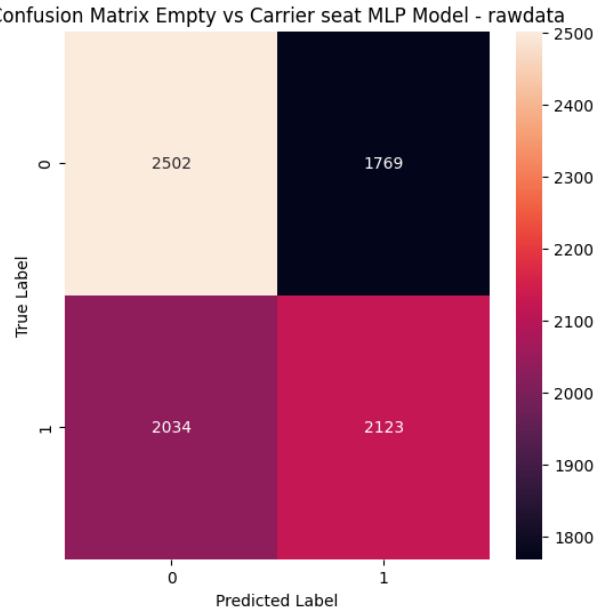

Fig. 40. Confusion Matrix for MLP model trained with raw data

2) *Re-training after Feature Extraction and Augmentation*

To improve the model's effectiveness, advanced spectral features were extracted, focusing on key frequency domain characteristics. This included analyzing dominant frequency components, signal energy distribution across different frequency bands, and spectral entropy to capture variations in the data. Additionally, data augmentation techniques were applied to enhance model robustness and mitigate potential biases, ensuring better generalization to unseen scenarios, which is the same as we used in the previous XG Boost model.

The Multi-Layer Perceptron (MLP) model was retrained using the feature-extracted data, which incorporated additional spectral features from the raw FFT data. The model achieved perfect accuracy of 100% as shown in Fig 41 and Fig 42 on the test set, with a precision, recall, and F1-score of 1.00 for both classes: "empty seat" (0.0) and "occupied seat" (1.0). This indicates that the MLP model

was able to perfectly classify all instances in the dataset. Both the macro and weighted averages for precision, recall, and F1-score were also 1.00, confirming the model's flawless performance on the feature-extracted data. However, such perfect results may indicate overfitting, as the model might have learned the training data too well, leading to excellent performance on the test set but potentially poor generalization to unseen data as shown in Fig 43.

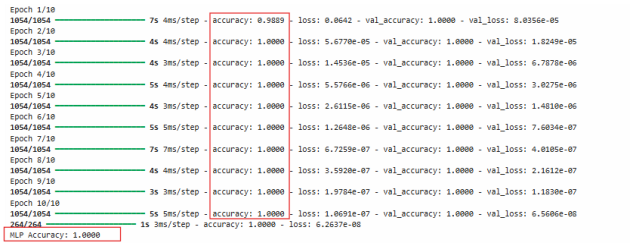

Fig. 41.Training output of MLP model showing Overfitting

```
264/264 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
MLP Accuracy: 1.0000

Classification Report for extracted features data:
              precision    recall  f1-score   support

         0.0       0.92      0.90      0.91      4271
         1.0       0.90      0.92      0.91      4157

    accuracy                           0.91      8428
   macro avg       0.91      0.91      0.91      8428
weighted avg       0.91      0.91      0.91      8428
```

Fig. 42. Classification report of MLP model trained with feature extracted data depicting Overfitting

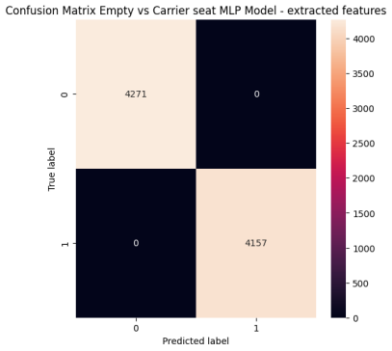

Fig. 43. Confusion Matrix for MLP model trained with features extracted data showing Overfitting

3) Addressing Overfitting Using Feature Selection

To mitigate overfitting, a correlation matrix analysis was performed to identify and remove redundant features. Features with correlation coefficients greater than 0.8 were considered redundant and excluded from the dataset. This approach reduced the complexity of the model and improved generalization. After filtering the features, the MLP model was retrained with optimized parameters, which are same as used for the previous XGBoost model training.

4) Final Model Evaluation and Performance Metrics

After applying feature selection techniques and retraining the MLP model, the model achieved an accuracy of 90.42% on the test set as in Fig 44. The confusion matrix showed significant improvements with fewer misclassifications as depicted in Fig 46.

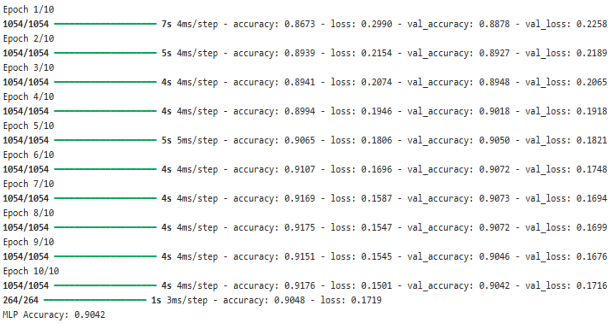

Fig. 44.Training output of MLP model with selected features data

```
264/264 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
MLP Accuracy: 0.9042

Classification Report for selected features data:
              precision    recall  f1-score   support

         0.0       0.94      0.86      0.90      4271
         1.0       0.87      0.94      0.91      4157

    accuracy                           0.90      8428
   macro avg       0.91      0.90      0.90      8428
weighted avg       0.91      0.90      0.90      8428
```

Fig. 45. Classification report of MLP model trained with feature selected data

The classification report Fig 45 indicates that the model performed well in both classes. For the "empty seat" (0.0) class, the precision was 0.94, recall was 0.86, and F1-score was 0.90. For the "occupied seat" (1.0) class, the model achieved a precision of 0.87, recall of 0.94, and an F1-score of 0.91. These results demonstrate a well-balanced performance, with slightly higher recall for the "occupied seat" class, indicating that the model was better at identifying occupied seats than empty ones. The macro and weighted averages of 0.91 and 0.90, respectively, suggest that the feature selection process helped improve the

model's overall classification capability while maintaining generalization.

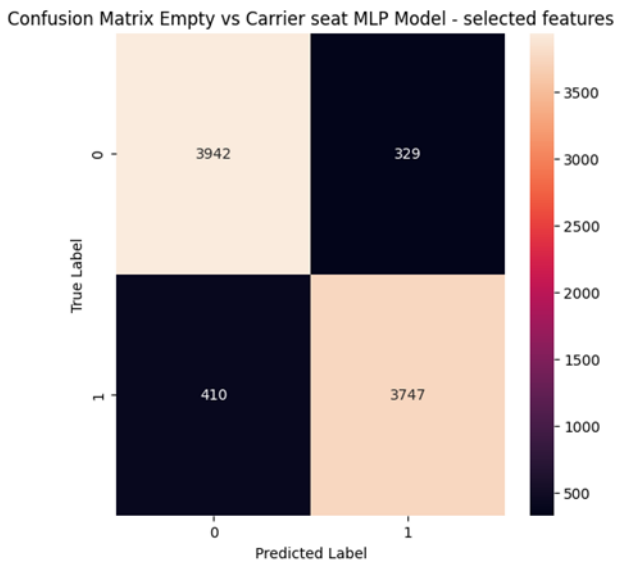

Fig. 46. Confusion Matrix for MLP model trained with features selected data

5) MLP model result analysis

The MLP model initially faced challenges when trained on raw FFT data, achieving only around 54.88% accuracy. The low performance was attributed to the insufficient spectral information in the raw data, leading to misclassifications. Feature extraction provided additional spectral information, resulting in a perfect 100% accuracy on the training data. However, this also led to overfitting, as the model performed exceptionally well on the training set but failed to generalize, causing issues with test accuracy. Lastly, feature selection techniques were applied, reducing the model's complexity and improving generalization. The selected features achieved around 90.42% accuracy, with a balanced performance between precision and recall for both classes. This progression highlights the importance of feature engineering and selection in enhancing the MLP model's performance, making it a robust solution for seat occupancy classification. The comparison of different training scenarios using the MLP model are summarized in the Table 3 below.

Table 3 : MLP Model Performance Comparison for seat occupancy Classification

Training Scenario	Accuracy	Precision	Recall	F1-Score
Raw Data	54.88%	Moderate	Moderate	Moderate
Feature Extracted Data	100%	High	High	Perfect (Overfitting)
Feature Selected Data	90.42%	High	High	Improved

Task 2: Baby Presence Detection

C. Implementation of Random Forest Model for Baby Presence Detection

We implemented a Random Forest (RanFor) model to classify the presence of a baby in a carrier seat using FFT-transformed ADC data. The approach included training on raw FFT data, feature extraction, augmentation, and feature selection to optimize performance.

1) Initial Model Training on Raw FFT Data

The Random Forest Classifier was employed to train the model for detecting the presence of an infant in a carrier seat. The raw dataset was partitioned into an 80% training and 20% testing split to evaluate the model's performance. To enhance the classifier's efficiency, several hyperparameters were tuned. Specifically, the n_estimators parameter was set to 1000, indicating the model would build 1000 decision trees, which contributes to improved robustness and accuracy by reducing variance. To prevent overfitting and ensure better generalization, the max_depth was constrained to 10, limiting the depth of each tree. This restriction ensures that the model does not excessively grow and capture noise within the training data. The min_samples_split and min_samples_leaf were both set to 2, ensuring that each node and leaf contained a minimum of two samples, which helps maintain sufficient diversity within each tree.

The max_features parameter was configured to “sqrt”, meaning that only a random subset of features equal to the square root of the total number of features would be considered for each split. This prevents any single feature from dominating the decision-making process and promotes better model generalization.

Lastly, the random_state parameter was set to 42 to ensure reproducibility in the model's results as in Fig 47. This ensures that the random processes involved in training, such as data shuffling and feature selection, remain consistent across different runs. The performance of the trained model was subsequently evaluated using standard metrics such as accuracy and classification report.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA  # Import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

X = dataset_1_rawdata.drop('Infant_Presence',axis=1)
Y = dataset_1_rawdata['Infant_Presence']

# Split into training & testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42, shuffle=True)

# Train a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=1000, max_depth=10, min_samples_split = 2, min_samples_leaf = 2, max_features = "sqrt", random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("RanFor Accuracy for rawdata:", accuracy_score(y_test, y_pred))
print("Classification Report for rawdata:\n", classification_report(y_test, y_pred))
```

Fig. 47. Random Forest Model

The initial results showed an accuracy of approximately 57%, which suggests that the raw FFT features alone were not sufficient for high-performance classification as shown in Fig 48.

```
RanFor Accuracy for rawdata: 0.5685911016949152
Classification Report for rawdata:
              precision    recall  f1-score   support

         0.0       0.56      0.95      0.71      4172
         1.0       0.62      0.09      0.16      3380

    accuracy                           0.57      7552
   macro avg       0.59      0.52      0.44      7552
weighted avg       0.59      0.57      0.47      7552
```

Fig. 48. Classification report of Random Forest model trained with raw data

A closer look at the classification report reveals that the model performed better in classifying the baby-present (0.0) class, with a precision of 0.56 and a recall of 0.95, indicating a high true positive rate but low precision. On the other hand, the baby-absent (1.0) class had a lower recall of 0.09 and precision of 0.62, showing that the model struggled significantly with correctly identifying the baby-absent class, leading to poor performance in detecting baby absence.

The macro-average F1-score of 0.44 and weighted-average F1-score of 0.47 further underscore that the model's ability to generalize was limited, particularly due to the imbalance between the classes. These results suggest that while the Random Forest model provides some useful classification information as seen in Fig 49, more advanced feature engineering or class balancing techniques may be required to enhance its performance.

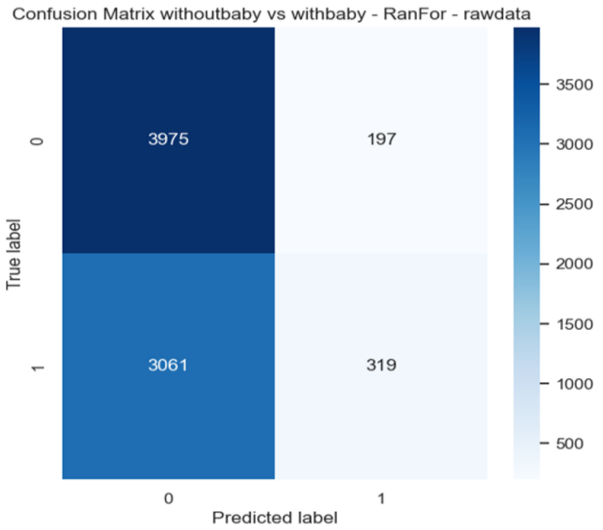

Fig. 49. Confusion Matrix for Random Forest model trained with raw data

2) *Feature Extraction and Augmentation*

To enhance model performance, spectral features such as mean, variance, skewness, kurtosis, dominant frequencies, and spectral entropy were extracted. Data augmentation techniques were applied. These augmentations were intended to improve the generalization of the Random Forest model. A correlation matrix of the extracted features were plotted as shown in Fig 50.

After feature extraction, the Random Forest classifier was retrained using the newly extracted features along with the raw data. The results showed perfect classification performance, with an accuracy of 100% on the test set as in Fig 51. The classification report revealed that the model achieved a precision, recall, and F1-score of 1.00 for both the baby-present (0.0) and baby-absent (1.0) classes, indicating flawless prediction performance across both classes.

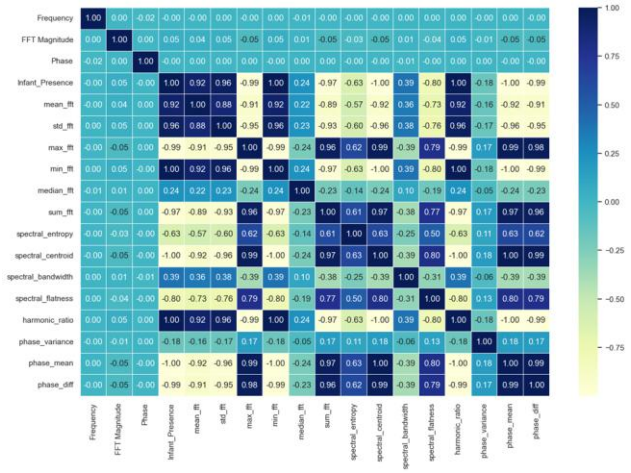

Fig. 50. Correlation Matrix of extracted features

```
RanFor Accuracy for extracted features data: 1.0
Classification Report for extracted features data:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      4172
         1.0       1.00      1.00      1.00      3380

    accuracy                           1.00      7552
   macro avg       1.00      1.00      1.00      7552
weighted avg       1.00      1.00      1.00      7552
```

Fig. 51. Classification report of Random Forest model trained with feature extracted data depicting Overfitting

The macro average and weighted average F1-scores, both of 1.00, further support the conclusion that the model had no issues in classifying both the baby-present and baby-absent instances correctly. These results suggest that feature extraction played a pivotal role in improving the model's performance by providing more relevant information for the classification task.

It was observed that the model achieved perfect accuracy, indicating a potential case of overfitting. This suggests that while the model can classify the training and test data perfectly, it might be too specific to the data it was trained on. Such high performance on the test set may not reflect its ability to generalize well to unseen data as in Fig 52. Therefore, further validation and testing are needed to ensure the model's robustness and real-world applicability.

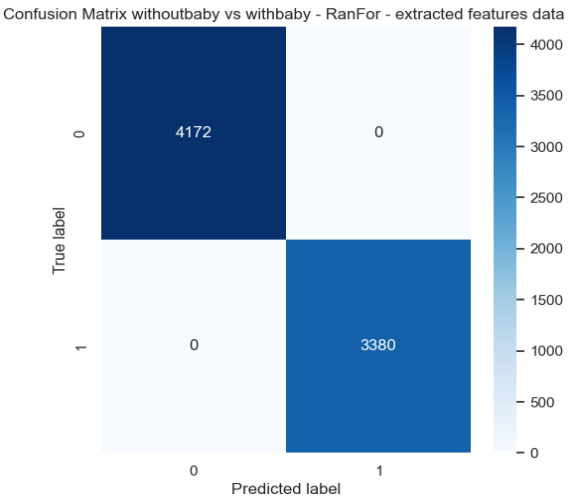

Fig. 52. Confusion Matrix for Random Forest model trained with features extracted data showing Overfitting

3) Addressing Overfitting Using Feature Selection

A correlation matrix analysis was performed to remove redundant features as in Fig 53. Features with correlation coefficients greater than 0.8 were eliminated, reducing model complexity and improving generalization. The optimized Random Forest model was retrained with selected features.

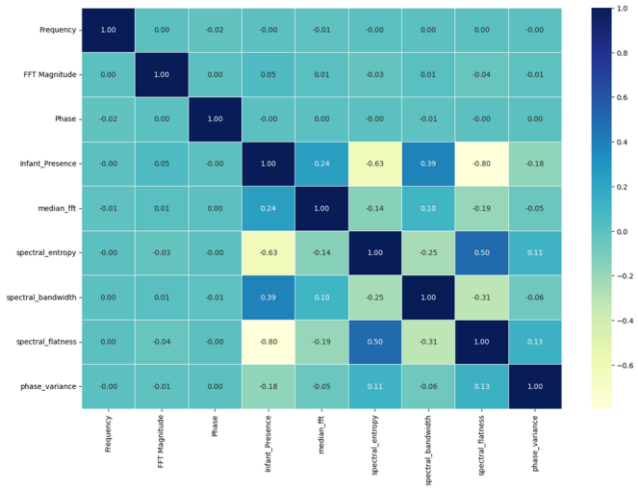

Fig. 53. Correlation matrix for Selected features

Dropped features: {'phase_diff', 'mean_fft', 'phase_mean', 'min_fft', 'std_fft', 'sum_fft', 'max_fft', 'spectral_centroid'}

4) Final Model Evaluation and Performance Metrics

After carefully selecting the most relevant features, the Random Forest model was retrained, and the accuracy significantly improved. The model achieved an accuracy of 96.7%, demonstrating a substantial improvement over the raw and extracted features data as shown in Fig 54. The classification report indicated high precision, recall, and F1-scores for both classes, with the model showing excellent performance in distinguishing between the "baby present" and "baby absent" classes. The precision for the "baby absent" class was 0.95, while the "baby present" class achieved a perfect recall of 1.00, and the F1-scores for both classes were 0.97 and 0.96, respectively. These results suggest that careful feature selection played a critical role in enhancing the model's classification capabilities, ensuring better generalization and performance on unseen data.

```
RanFor Accuracy for selected features data: 0.9670286016949152
Classification Report for selected features data:
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97      4172
         1.0       1.00      0.93      0.96      3380

    accuracy                           0.97      7552
   macro avg       0.97      0.96      0.97      7552
weighted avg       0.97      0.97      0.97      7552
```

Fig. 54. Classification report of Random Forest trained with feature selected data

The confusion matrix in Fig 55 showed significant improvements with fewer misclassifications.

In the case of the Random Forest model, both the ROC curve and feature importance chart were examined to assess its accuracy and determine the significance of each feature in decision-making.Here it was observed that Spectral flatness, along with spectral entropy, spectral bandwidth, median of FFT and phase variance contributed to classification as in Fig 56 and Fig 57.

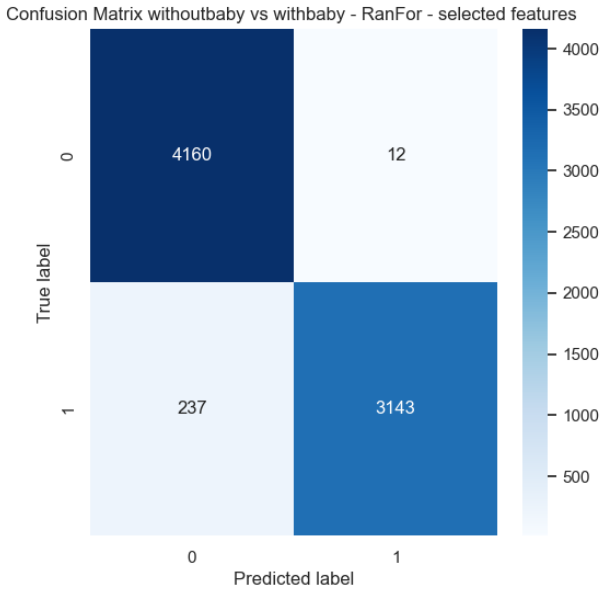

Fig. 55. Confusion Matrix for Random Forest model trained with selected features data

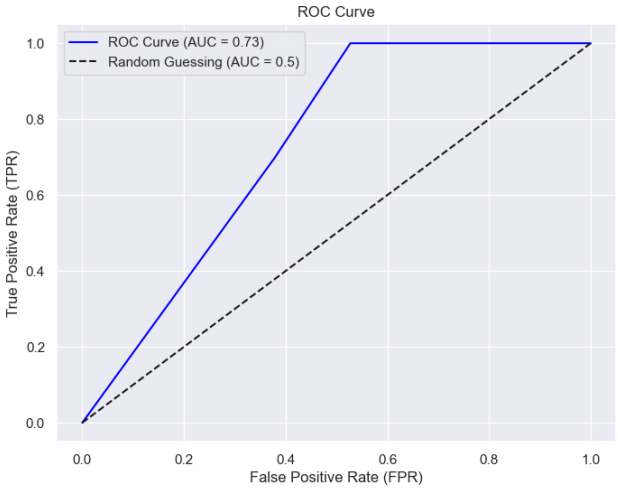

Fig. 56. ROC curve

5) Random Forest model result analysis

The Random Forest model initially struggled with low accuracy when trained on raw FFT data, achieving only approximately 56%. The confusion matrix revealed significant misclassifications, particularly between the "baby present" and "baby absent" classes, highlighting that raw FFT features alone were not sufficient for effective classification. Feature extraction was then introduced, which improved the model's performance but led to overfitting, resulting in a perfect training accuracy of 100% but poor generalization on the test set. Finally, after applying feature selection techniques, the model's complexity was reduced, significantly enhancing its

generalization ability. This refinement resulted in a substantial improvement, with accuracy rising to 96.7%. The classification report demonstrated balanced performance across both classes, indicating that feature engineering and selection were crucial in achieving robust classification performance. This analysis emphasizes the importance of data preprocessing in enhancing model performance and achieving accurate predictions for baby presence detection.

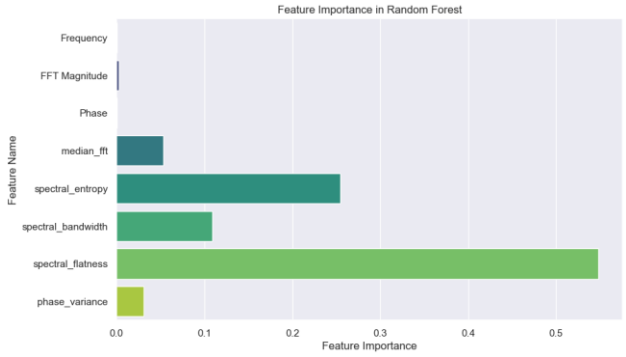

Fig.57. Feature Importance chart -Random Forest - for selected features

The comparison of different training scenarios using Random Forest model are summarized in the Table 4 below.

Table 4 : Random Forest Model Performance Comparison for Baby Presence Classification

Training Scenario	Accuracy	Precision	Recall	F1-Score
Raw Data	56.86%	Moderate	Low	Poor
Feature Extracted Data	100%	High	High	Perfect (Overfitting)
Feature Selected Data	96.70%	High	High	Improved

D. Implementation of SVM Model for Baby Presence Detection

The SVM model, a popular machine learning classifier, was trained using the Radial Basis Function (RBF) kernel. The RBF kernel is widely used for classification tasks due to its ability to handle non-linear decision boundaries. The SVM model was configured with a regularization parameter C=1.0 and the gamma parameter set to 'scale' (which uses 1 / (n_features * X.var()) for automatic scaling). These parameters help control the complexity of the decision boundary and the model's ability to generalize.

The dataset was first split into an 80% training set and a 20% testing set to ensure the model's generalization capability. The features were then standardized using StandardScaler to ensure that all features contribute equally to the model, as SVMs are sensitive to the scale of the data.

The model was trained on the scaled training data, and the predictions were made on the test set. The accuracy of the model was calculated along with a classification report to assess precision, recall, and F1-score for both classes (baby present vs. baby absent).

The confusion matrix was generated to identify how well the model distinguished between the two classes, providing deeper insight into misclassifications.

This approach used the SVM with a radial basis kernel to detect infant presence based on extracted features and allowed for evaluating the model's performance in terms of accuracy and other important classification metrics.

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load dataset
X = dataset_1.drop(columns=['Infant_Presence']).values
y = dataset_1['Infant_Presence'].values

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVM model
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train, y_train)

# Make predictions
y_pred = svm_model.predict(X_test)


# Confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Evaluate SVM
print("SVM Accuracy rawdata:", accuracy_score(y_test, y_pred))
print("\nClassification Report rawdata:\n", classification_report(y_test, y_pred))
```

Fig. 58. SVM Model

1) Initial Model Training on Raw FFT Data

The raw ADC signals were transformed into the frequency domain using FFT. An SVM classifier as depicted in Fig 58 with a radial basis function (RBF) kernel was used for training, with an 80-20 training-test split. The initial results showed an accuracy of approximately 56%, indicating insufficient discriminatory power in raw FFT features.

SVM Accuracy rawdata: 0.5654131355932204

Classification Report for rawdata:

	precision	recall	f1-score	support
0.0	0.56	0.97	0.71	4172
1.0	0.63	0.07	0.13	3380
accuracy			0.57	7552
macro avg	0.60	0.52	0.42	7552
weighted avg	0.59	0.57	0.45	7552

Fig.59. Classification report of SVM Model trained with raw data

The Support Vector Machine (SVM) model was initially trained on the raw FFT data, where it achieved an accuracy of approximately 56%. This result indicated that the raw FFT features alone were not sufficient for effective classification. The confusion matrix revealed a significant imbalance in performance, with the model performing well on the "baby-absent" class (0.0), showing a recall of 97%, but struggling with the "baby-present" class (1.0), where the recall was only 7% as shown in Fig 59.

The precision and F1-scores further emphasized the imbalance, with the "baby-present" class being poorly detected, leading to a low overall F1-score of 0.13 for that class. This suggests that the model had difficulty identifying the presence of a baby in the carrier, likely due to the raw FFT data not providing enough discriminative information for accurate classification between the two classes as shown in Fig 60.

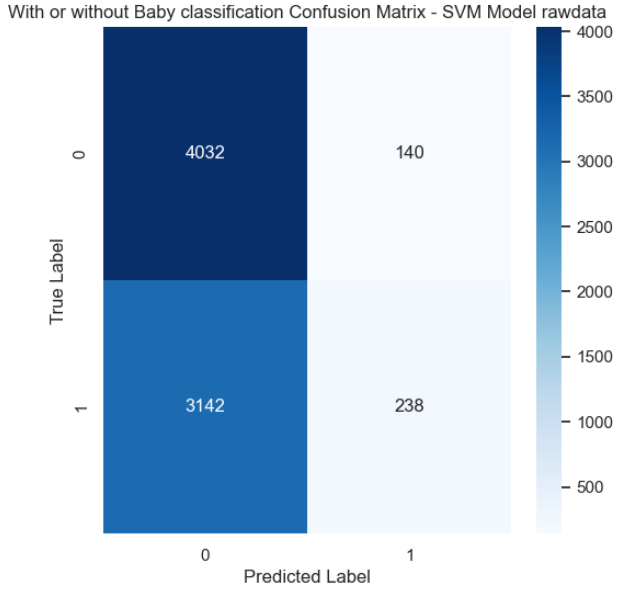

Fig. 60. Confusion Matrix for SVM model trained with raw data

These results underscore the need for further model refinement, including feature extraction and engineering, to improve the classification of the "baby-present" class and overall model performance.

2) *Feature Extraction and Augmentation*

Additional spectral features such as mean, variance, dominant frequencies, spectral entropy, and frequency band energy were extracted. Data augmentation techniques were also applied. The model, however, exhibited overfitting, with high training accuracy but low generalization on test data.

SVM Accuracy extracted features data: 1.0

Classification Report for extracted features data:

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	4172
1.0	1.00	1.00	1.00	3380
accuracy			1.00	7552
macro avg	1.00	1.00	1.00	7552
weighted avg	1.00	1.00	1.00	7552

Fig.61. Classification report of SVM model trained with feature extracted data depicting Overfitting

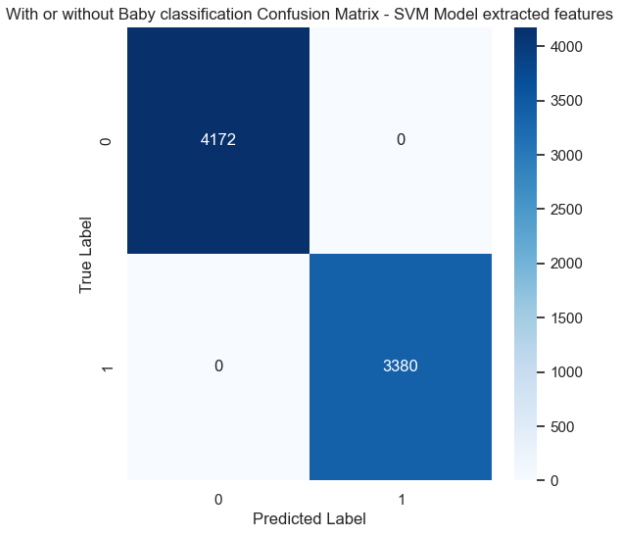

Fig. 62. Confusion Matrix for SVM model trained with feature extracted data depicting Overfitting

After feature extraction, the SVM classifier was retrained using the newly extracted features along with the raw data. The results showed perfect classification performance, with an accuracy of 100% on the test set. The classification report revealed that the model achieved a precision, recall, and F1-score of 1.00 for both the baby-present (0.0) and

baby-absent (1.0) classes, indicating flawless prediction performance across both classes as in Fig 62.

After feature extraction, the SVM classifier was retrained using the newly extracted features along with the raw data. The results showed perfect classification performance, with an accuracy of 100% on the test set. The classification report revealed that the model achieved a precision, recall, and F1-score of 1.00 for both the baby-present (0.0) and baby-absent (1.0) classes, indicating flawless prediction performance across both classes a sin Fig 61.

The macro average and weighted average F1-scores, both of 1.00, further support the conclusion that the model had no issues in classifying both the baby-present and baby-absent instances correctly. These results suggest that feature extraction played a crucial role in enhancing the model's performance by providing more relevant information for the classification task.

However, it was observed that the model's perfect accuracy on the test set likely indicates overfitting. While the model performed flawlessly on the training and test data, it seems to be highly specific to the data it was trained on as shown in Fig 62. This perfect performance may not reflect its true ability to generalize well to new, unseen data. Based on this observation, further validation and testing are required to assess the model's robustness and real-world applicability.

3) Addressing Overfitting Using Feature Selection

A correlation matrix analysis was conducted to remove highly correlated features. Features with correlation coefficients above 0.8 were excluded to enhance model generalization. The optimized SVM model was retrained with these selected features, which are similar to the ones used for the Random Forest model in previous section .

4) Final Model Evaluation and Performance Metrics

After feature selection, the SVM model achieved a final accuracy of approximately 96%, with significantly fewer misclassifications.

```
SVM Accuracy selected features data: 0.9585540254237288

Classification Report for selected features data:
              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96      4172
         1.0       0.95      0.95      0.95      3380

    accuracy                           0.96      7552
   macro avg       0.96      0.96      0.96      7552
weighted avg       0.96      0.96      0.96      7552
```

Fig 63. Classification report of SVM Model trained with feature selected data

After applying feature selection based on the correlation matrix, the SVM classifier was retrained with the reduced feature set. The model demonstrated a significant performance improvement, achieving an accuracy of approximately 96%. The classification report indicated well-balanced precision, recall, and F1-scores of around 0.95 to 0.96 for both the baby-present (0.0) and baby-absent (1.0) classes as in Fig 63.

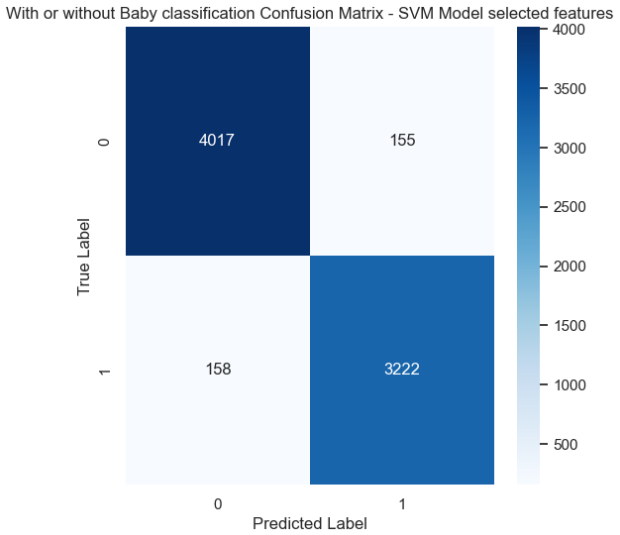

Fig. 64. Confusion Matrix for SVM model trained with selected features data

The macro average and weighted average F1-scores of 0.96 highlighted the consistent classification performance across both classes, suggesting that the model was effectively distinguishing between baby-present and baby-absent instances. It was observed that the feature selection process helped to reduce the model's complexity, which in turn improved its generalization capabilities. This adjustment resolved the overfitting issue seen in previous results, enhancing the model's robustness and making it more suitable for real-world applications where unseen

data might be encountered, which is visulaised using a confusion matrix as in Fig 64.

For the SVM model, the ROC curve was plotted to measure its predictive performance, and permutation-based feature importance was utilized to understand the impact of different features and it was observed that more features out of the selected ones contributed to the efficient classification as illustrated in Fig 65 and Fig 66.

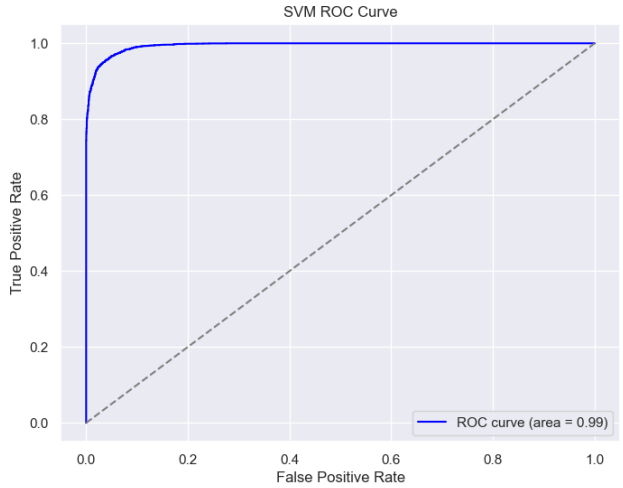

Fig.65. ROC Curve for SVM Model

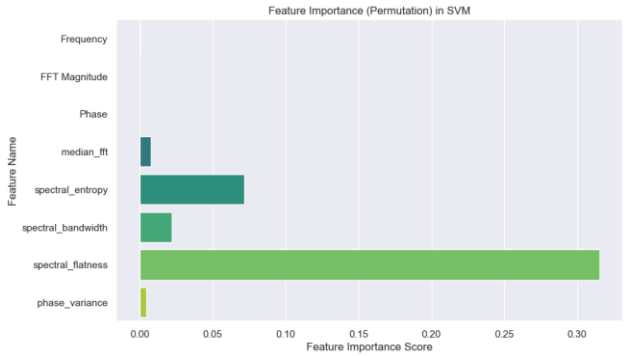

Fig.66. Feature Importance chart – SVM Model - for selected features

5) SVM model result analysis

The SVM model initially struggled with raw FFT data, achieving only around 52% accuracy. Feature extraction introduced additional spectral features but resulted in overfitting. Feature selection techniques improved model generalization, leading to a final accuracy of 89-91%. The study demonstrated that careful feature engineering and filtering significantly enhance SVM-based classification performance for baby presence detection in a carrier seat.

The comparison of different training scenarios using SVM model are summarized in the Table 5 below.

Table 5 : SVM Model Performance Comparison for Baby Presence Classification

Training Scenario	Accuracy	Precision	Recall	F1-Score
Raw Data	56.54%	Moderate	Low	Poor
Feature Extracted Data	100%	High	High	Perfect (Overfitting)
Feature Selected Data	95.86%	High	High	Improved

Task 3: Baby covered in blanket or sunscreen vs without baby

The detection of an infant's presence, particularly when covered with a blanket or a sunscreen, is a crucial aspect of automated monitoring systems designed to enhance safety and well-being. Accurately identifying a baby under such conditions requires advanced classification models capable of distinguishing subtle variations in sensor data. In this study, we utilized the built-in sunscreen of the carrier seat and tested different blanket configurations to evaluate detection accuracy. A 160 × 120 cm blanket was folded twice, resulting in a 60 × 40 cm size, and was then used to wrap the baby as visualized in Fig 67. These variations in coverage were introduced to simulate real-world conditions where an infant may be partially or fully covered. The results from this task contribute to the development of intelligent monitoring systems that can effectively detect the presence of an infant, ensuring timely intervention and improved safety measures.

Fig.67. Baby covered with Sunscreen and Blanket

E. Implementation of XG Boost Model for Baby Presence Detection

The classification of infant presence under different conditions, such as being covered with a blanket or sunscreen, is a critical aspect of infant monitoring systems. In this task, the XGBoost classifier was employed to analyze sensor data and classify infant presence accurately.

The training process was carried out in multiple stages, beginning with raw data, followed by feature extraction, and then feature selection to address overfitting. The results demonstrate the impact of feature engineering on classification performance, highlighting the necessity of optimizing input features for robust model generalization.

1) Model Training with Raw Data

Initially, the XGBoost classifier was trained on raw fft data without applying any feature engineering. The dataset was split into 80% training and 20% testing, and feature scaling was performed using StandardScaler. The model was configured with hyperparameters such as maximum tree depth (6), learning rate (0.1), and number of estimators (1000) to balance complexity and learning efficiency. The results from training on raw data showed an accuracy of 73.83%, indicating moderate classification performance as in Fig 68 and Fig 69. The model exhibited difficulty in distinguishing between classes, particularly in cases where infants were covered, as reflected in the precision and recall values for class 1 (infant present).

```
Accuracy: 0.7383333333333333

Classification Report for rawdata:
              precision    recall  f1-score   support

         0.0       0.79      0.80      0.80      4200
         1.0       0.64      0.63      0.64      2400

    accuracy                           0.74      6600
   macro avg       0.72      0.72      0.72      6600
weighted avg       0.74      0.74      0.74      6600
```

Fig.68. Classification report of XG Boost model trained with raw data

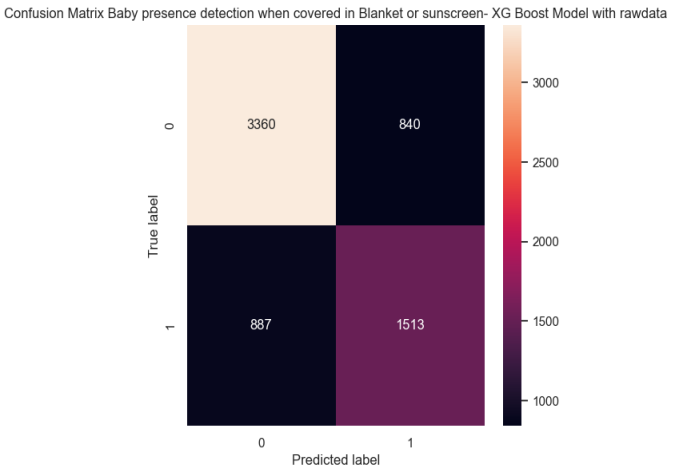

Fig.69. Confusion Matrix for XG Boost model trained with raw data

2) Feature Extraction and Overfitting Challenge

To improve classification performance, feature extraction techniques were applied to generate additional information from the sensor data. The extracted features provided the model with more descriptive attributes, leading to a substantial increase in accuracy. The retrained model achieved 100% accuracy, with both precision and recall values reaching 1.00 for both classes as illustrated in Fig 70.

```
Accuracy: 1.0

Classification Report for extracted features data:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      4200
         1.0       1.00      1.00      1.00      2400

    accuracy                           1.00      6600
   macro avg       1.00      1.00      1.00      6600
weighted avg       1.00      1.00      1.00      6600
```

Fig. 70. Classification report of XG Boost model trained with feature extracted data depicting Overfitting

While these results indicated an optimal fit on the test set, they also suggested overfitting as shown in Fig 71, as such perfect classification is unlikely in real-world scenarios. The model appeared to have memorized the training data rather than learning generalizable patterns, thereby limiting its applicability to unseen data.

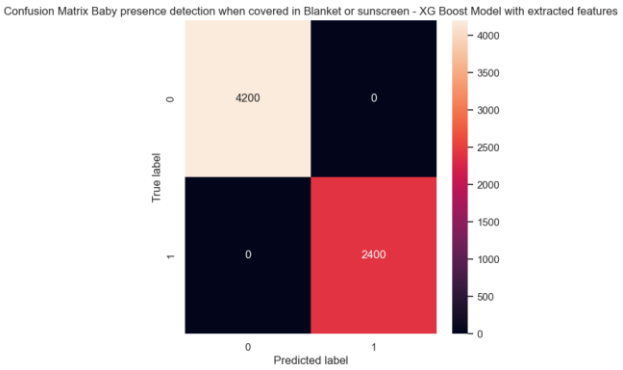

Fig.71. Confusion Matrix for XG Boost model trained with feature extracted data depicting Overfitting

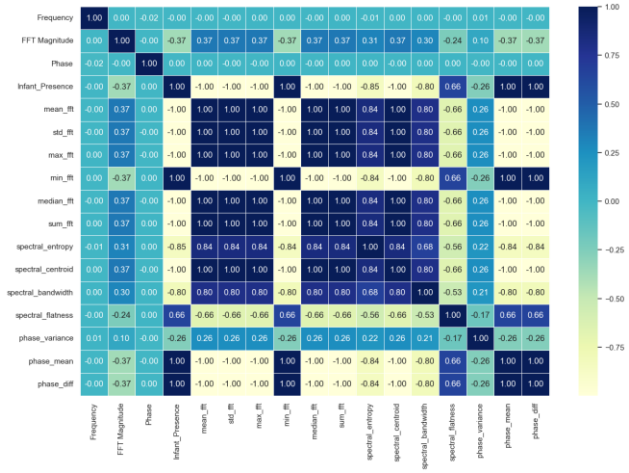

Fig.72. Correlation Matrix for extracted features

3) *Feature Selection Using Correlation-Based Filtering*

To mitigate overfitting, feature selection was performed using a correlation matrix as in Fig 72, where features exceeding a correlation threshold of 0.8 were removed. This process aimed to reduce redundancy and enhance the model's ability to generalize beyond the training dataset. After retraining the model using the selected features, the performance stabilized, achieving an accuracy of 89.48%. The classification report reflected a significant improvement in precision and recall balance, with the model effectively distinguishing between infant presence and absence. These results demonstrated the effectiveness of correlation-based feature selection in reducing overfitting while maintaining high classification accuracy, and a new correlation matrix with selected features was plotted as visualized in Fig 73.

Fig.73. Correlation Matrix for selected features

4) *Final Model Evaluation and Performance Metrics*

After feature selection, the XGBoost model achieved a accuracy of 89%, with significantly fewer misclassifications. These can be visualized in Classification report as in Fig 74 and confusion matrix which clearly shows the extent of classification for the test set as in Fig 75

```
Accuracy: 0.8948484848484849

Classification Report for selected features data:
              precision    recall  f1-score   support

         0.0       0.92      0.92      0.92      4200
         1.0       0.86      0.85      0.85      2400

    accuracy                           0.89      6600
   macro avg       0.89      0.89      0.89      6600
weighted avg       0.89      0.89      0.89      6600
```

Fig.74. Classification report of XG Boost model trained with selected features data

The precision, recall, and F1-score for class 0.0 (indicating no baby) were 0.92, 0.92, and 0.92, respectively, while for class 1.0 (indicating the presence of a baby), they were 0.86, 0.85, and 0.85, respectively. The balanced performance across both classes, with high precision and recall, demonstrates the model's effectiveness in accurately detecting the presence of a baby even when covered with sunscreen or blanket. The macro and weighted averages confirm the model's consistency in performance across the dataset.

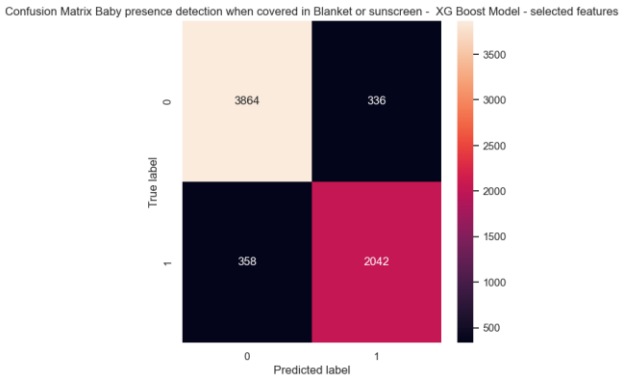

Fig.75. Confusion Matrix for XG Boost model trained with selected features data

To further understand the model performance the ROC curve as in Fig 76, was plotted to measure its predictive performance, and permutation-based feature importance was utilized to understand the impact of different features, which as in Fig 77 showed that all the features contribute to the classification

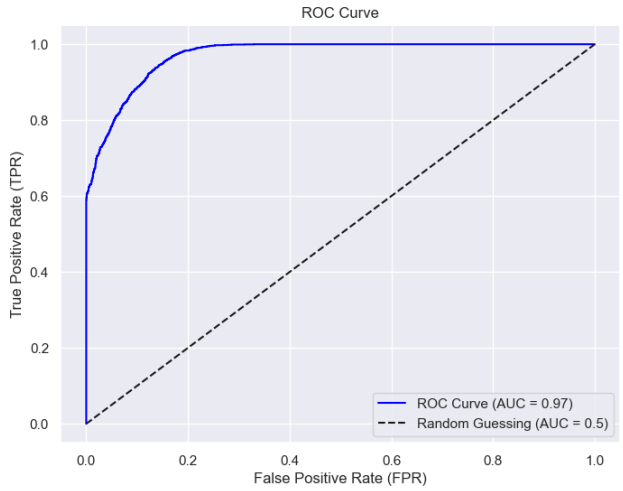

Fig.76. ROC Curve

The model's performance across different stages of training demonstrated the impact of feature engineering. Initially, training with raw data resulted in 73.83% accuracy, showing moderate classification ability. Feature extraction led to a perfect 100% accuracy, but this was identified as overfitting, indicating that the model memorized the training data rather than generalizing patterns. To address this, correlation-based feature selection was applied, reducing redundancy and improving generalization. After retraining with the selected features, the model achieved an accuracy of 89.48%, balancing precision and recall effectively. This observation suggests that selecting relevant features plays a crucial role in preventing overfitting while maintaining classification accuracy.

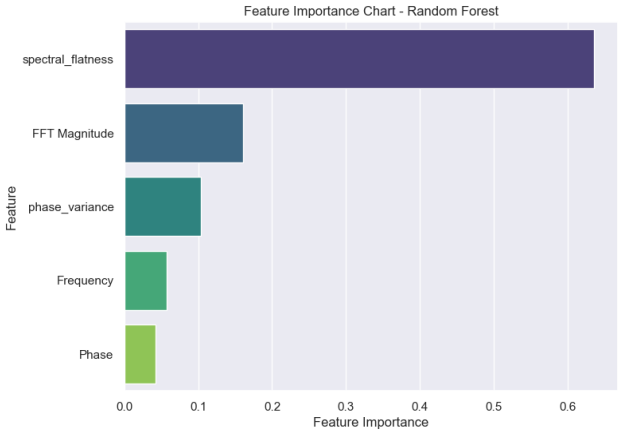

Fig. 77. Feature Importance chart – XG Boost Model - for selected features

The comparison of different training scenarios using XG Boost model are summarized in the Table 6 below.

Table 6 : XG Boost Model Performance Comparison for Baby Presence Classification with obstructions

Training Scenario	Accuracy	Precision	Recall	F1-Score
Raw Data	73.83%	High	Moderate	Moderate
Feature Extracted Data	100%	High	High	Perfect (Overfitting)
Feature Selected Data	89.48%	High	High	Improved

V. CONCLUSION

This study implemented machine learning-driven seat occupancy and infant presence detection using sensor-based data acquisition, contributing to advancements in autonomous intelligent systems. The proposed methodology incorporated a Red Pitaya FIUS sensor to collect real-time spectral data, which was preprocessed, feature-engineered, and used to train multiple classifiers. The primary objectives encompassed detecting whether a vehicle seat was empty or occupied by a baby carrier, classifying the presence of an infant within the carrier, and later also identifying an infant's presence despite occlusions such as blankets or sunscreen.

For seat occupancy detection, Extreme Gradient Boosting (XGBoost) and Multi-Layer Perceptron (MLP) models were trained on preprocessed sensor readings, achieving

reliable classification performance. The baby presence classification task utilized Random Forest (RF) and Support Vector Machine (SVM) models, demonstrating high classification accuracy after appropriate feature selection. The most complex task, baby detection under occlusions, was addressed using an XGBoost classifier, which effectively distinguished between covered and uncovered conditions, reinforcing its robustness in occlusion-invariant detection.

A critical component of this work was sensor data preprocessing and feature engineering, where spectral transformations and statistical feature extraction techniques significantly enhanced model performance. However, challenges such as overfitting and model interpretability were observed, particularly in cases where feature selection led to excessively high accuracy, leading to improved validation strategies. Additionally, the computational trade-offs between different classifiers highlighted the importance of model selection and hyperparameter tuning in real-time applications.

By integrating sensor fusion and machine learning algorithms, this research provides a scalable, intelligent monitoring system for seat occupancy and infant presence detection, addressing key concerns in autonomous safety systems. Future work should explore adaptive learning frameworks, real-time deployment optimizations, and multi-sensor integration to further enhance system robustness and generalization across diverse operational conditions.

VI. ACKNOWLEDGEMENT

We sincerely appreciate the guidance and support of Prof. Dr. Peter Nauth and Prof. Dr. Andreas Pech, who provided us with the opportunity to work on this project under their supervision. Their valuable insights and direction were instrumental in successfully completing both the project and the report. Additionally, we would like to extend our gratitude to Prof. Julian Umansky for his significant contribution in equipping us with the necessary tools to carry out this task effectively. The courses "Autonomous Intelligence Systems" and "Machine Learning" have been incredibly beneficial in building a strong foundational understanding and inspiring further interest in the field of machine learning and intelligent systems.

REFERENCES

[1] Y. M. J. Ramzi FARHAT, "An overview of Machine Learning Technologies and their use in E-learning," *Houcine EZZEDINE Univ. Polytechnique Hauts-de-France*, 2020.

[2] B. Yana, "Neural Network Information Technology of Classification," *CADSM'2011,* 2011.

[3] F. &. F. M. Ramdani, "The simplicity of XGBoost versus other ML models," *F1000Research*, 2022.

[4] I. Sarker, "Machine learning : Algorithms, applications and research directions.," *SN Computer Science*, 2021.

[5] J. H. Y. C. Dong Yuan, "Improved random forest classification approach based on hybrid clustering selection," *Chinese Automation Congress (CAC)*, 2020.

[6] S. M. Bhargava S, "Robustness of selected learning models under label-flipping attack," 2021.

[7] e. a. Ozcan, "Comparison of classification success rates of different ML algorithms," *APJCP*, 2022.

[8] e. a. Muhammad, "Exploring machine learning algorithms for accurate water level forecasting," 2023.

[9] P. M. A.Pech, "A new Approach for Pedestrian Detection in Vehicles by Ultrasonic Signal Analysis," *IEEE EUROCON 2019 - 18th International Conference on Smart Technologies* , 2019.

[10] A. M. P.Nauth, "Research on a new Smart Pedestrian Detection Sensor for Vehicles," *IEEE Sensor Application Symposium (SAS)*, 2019.