

Deep Learning-Based Fault Detection System for Structural Defects in Nano-Quadcopters

Project proposed for the KAUST Academy AI summer training 2024.

Shahad Jeza Alamri
Shahad.jeza.alamri@gmail.com

Rema Yasir Albarakati
Rema.y.albarakati@gmail.com

Shatha Turki Alsaidy
shathaalsaidy@gmail.com

Rawan Yasser Albarakati
Rawan.y.albarakati@gmail.com

Salha Nasser Alyami
salha.nasser.ds@gmail.com

August 31, 2024

Dr. Zeashan Khan
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Contents

1 Introduction **3**
1.1 Background . . . 3
1.1.1 Overview of Quadcopters and Their Importance . . . 3
1.1.2 The Significance of Identifying Fault Signatures . . . 3
1.2 Problem Statement . . . 3
1.3 Objectives . . . 3

2 Methods **4**
2.1 Dataset Overview . . . 4
2.2 Data Preprocessing . . . 4
2.3 Feature Engineering . . . 5
2.4 Baseline Models and Results . . . 9
2.5 Proposed Model and Results . . . 10
2.6 Experimental Environment . . . 12
2.7 Discussion . . . 12

3 Conclusion **13**
3.1 Summary of Findings . . . 13
3.2 Future Work . . . 13

1 Introduction

1.1 Background

The rise of unmanned aerial vehicles (UAVs), especially quadcopters, has significantly impacted a wide range of industries, including agriculture and search and rescue. Nano-quadcopters, a specialized category of UAVs, are particularly distinguished by their compact size and exceptional agility, making them ideally suited for operations in confined or hard-to-reach environments. However, with their growing deployment comes the need for reliable systems that can ensure their operational integrity and safety. Structural defects, whether due to wear and tear or unforeseen damage, can lead to critical failures if not detected early. Therefore, identifying fault signatures in real-time is essential to prevent potential accidents and extend the lifespan of these UAVs [6].

1.1.1 Overview of Quadcopters and Their Importance

Quadcopters, a type of UAV with four rotors, are known for their simplicity in design and ease of control, which allows for vertical takeoff and landing, hovering, and agile maneuvering. These characteristics make them valuable tools in a variety of fields, including aerial photography, environmental monitoring, and infrastructure inspection [7]. Nano-quadcopters, in particular, have gained attention due to their small size, which enables them to operate in environments that larger UAVs cannot access. Their ability to carry out missions in tight or hazardous spaces has led to their adoption in indoor surveillance, search and rescue missions, and precision agriculture [4].

1.1.2 The Significance of Identifying Fault Signatures

As nano-quadcopters are increasingly used in important tasks, ensuring their reliability through effective fault detection becomes crucial. Fault signatures are specific patterns or anomalies in sensor data that indicate potential defects or issues. Identifying these signatures early is essential to prevent unexpected failures and maintain the quadcopter's safety and performance.

Traditional methods, such as regular inspections, might not catch small or developing faults in nano-quadcopters due to their complex and compact design [2]. These conventional approaches can miss early signs of wear or damage. Advanced detection systems that use deep learning can help by continuously analyzing data for any unusual patterns. This real-time monitoring helps detect problems before they become serious, ensuring that nano-quadcopters remain reliable and effective for their intended tasks.

1.2 Problem Statement

This research aims to develop a neural network (NN) model to detect structural defects in nano-quadcopters. Using a specialized dataset, we will evaluate how these defects impact the drones' performance. Additionally, we will compare various machine learning (ML) techniques to determine which ones are both efficient and suitable for deployment directly on the UAVs. To improve the model's performance, we will also explore dimensionality reduction techniques, focusing on optimizing fault detection in real-time.

1.3 Objectives

The objectives of this research are to:

1. **Develop a Neural Network Model**: Create and implement a neural network model specifically designed to detect structural defects in nano-quadcopters, improving their operational safety.

2. **Assess the Impact of Structural Defects**: Analyze the effects of various structural defects on the performance of nano-quadcopters using a specialized dataset.

3. **Compare Machine Learning Methods**: Evaluate different machine learning models to determine which methods are the most efficient in terms of speed and computational resource requirements, making them suitable for real-time implementation on UAVs.

4. **Implement Dimensionality Reduction Techniques**: Apply dimensionality reduction techniques to the dataset to enhance the efficiency and speed of fault detection models without compromising accuracy.

5. **Optimize for Real-Time Application**: Ensure that the proposed models, particularly the neural network, are optimized for real-time fault detection, enabling them to be installed directly on the UAVs for continuous monitoring and fault detection.

2 Methods

This section details the methodologies employed in our research, encompassing data collection, preprocessing, feature engineering, algorithm selection, model training, and evaluation. Each step is critical to ensuring the development of an effective and efficient model for detecting structural defects in nano-quadcopters.

2.1 Dataset Overview

The dataset utilized in this research is known as *CrazyPAD* [3] (Crazyflie Propeller Anomaly Data), which is specifically designed to address the impact of structural defects on the performance of nano-quadcopters.

The CrazyPAD dataset was collected using the Crazyflie 2.1 nano-quadrotor shown in Figure 1. It contains a wide range of data derived from rigorous simulations and multiple scenarios of structural defects and their consequences on UAV performance.

The dataset is categorized into two primary data types: Controller data and Motor data. The Controller Data encompasses critical parameters related to the UAV's control inputs and its estimated state in the global reference system. This includes estimated position coordinates, control commands, gyroscope measurements, acceleration data, actuator thrust, and setpoints.

Motor Data focuses on the performance metrics of the individual motors and additional flight dynamics. It includes motor power readings, PWM output values, battery charge status, and repeated estimated position data from the Controller Data. Additionally, it captures velocity and acceleration metrics, and spatial position data.

Data collection was conducted during each flight, with data transmitted in binary form via a radio channel. This method ensured a comprehensive record of the UAV's performance throughout its flight path, capturing a sequence of parameters linked to the flight time.

2.2 Data Preprocessing

The initial dataset comprised 576,718 entries with 21 features and 60 classes. During the preprocessing phase, several steps were undertaken to ensure the quality and suitability of the data for subsequent analysis. The preprocessing of the dataset involved the following steps:

- The class `cut_propeller_M3_2mm`, had a high number of missing values, was dropped from the dataset.

- Missing values in the feature `cut_propeller_M3_3mm` were imputed using the median value of the feature.



Figure 1: Crazyflie 2.1 Nano Drone used for dataset collection.

- Missing values in other features were imputed using their respective median values.
- Categorical labels were encoded to facilitate their use in machine learning models.
- For normalization, we conducted evaluations using QuantileTransformer alone, QuantileTransformer in combination with StandardScaler, and StandardScaler alone. Our results indicate that QuantileTransformer provides a slight improvement in performance compared to the other normalization methods.

2.3 Feature Engineering

Feature engineering was a crucial step in this research to improve the model's ability to capture significant aspects of the UAV's performance and enhance prediction accuracy. This process involved generating new features that provide a more comprehensive representation of the UAV's behavior in flight.

The need for feature engineering arises from the complex nature of UAV dynamics, where raw data may not sufficiently capture the intricate relationships between various parameters. By deriving additional features, the model can better understand and predict faults, leading to more accurate and reliable outcomes.

The added features were inspired by the physical properties and flight dynamics of UAVs [1] [5]. For example, spherical coordinates provide insights into how the UAV's position changes relative to a central point, offering a more natural representation of its flight path and making it easier to analyze directional trends and potential deviations in flight stability. These enhancements were expected to improve the model's ability to detect subtle patterns and anomalies, leading to more accurate fault detection. The added features includes:

- **Magnitude of the Position Vector:** A new feature was calculated by computing the magnitude of the position vector using the x, y, and z coordinates. Here, x, y, and z refer to the estimated positions in the global reference system:

$$\text{pos_mag} = \sqrt{x^2 + y^2 + z^2}$$

This feature provides a single scalar value representing the overall distance from the origin, simplifying the analysis of the UAV's spatial position.

- **Conversion to Spherical Coordinates:** Cartesian coordinates were converted to spherical coordinates using the following equations, where r is the radial distance, θ is the polar angle, and ϕ is the azimuthal angle:

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arccos\left(\frac{z}{r}\right)$$
$$\phi = \arctan 2(y, x)$$

This transformation allows for a more intuitive understanding of the UAV's position and movement in three-dimensional space, especially in scenarios involving radial distances and angular directions.

- **Magnitude of Velocity and Acceleration:** The magnitudes of both velocity and acceleration vectors were computed as follows, where v_x, v_y, v_z represent the components of velocity, and a_x, a_y, a_z represent the components of acceleration:

$$\text{vel_mag} = \sqrt{v_x^2 + v_y^2 + v_z^2}$$
$$\text{acc_mag} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

These features encapsulate the overall speed and rate of change of speed, respectively, providing a more direct measure of the UAV's dynamic behavior.

- **Direction Cosines for Velocity and Acceleration:** Direction cosines for velocity and acceleration vectors in the x, y, and z directions were calculated using the following equations:

$$\text{v_cos_x} = \frac{v_x}{\text{vel_mag}}, \quad \text{v_cos_y} = \frac{v_y}{\text{vel_mag}}, \quad \text{v_cos_z} = \frac{v_z}{\text{vel_mag}}$$
$$\text{a_cos_x} = \frac{a_x}{\text{acc_mag}}, \quad \text{a_cos_y} = \frac{a_y}{\text{acc_mag}}, \quad \text{a_cos_z} = \frac{a_z}{\text{acc_mag}}$$

These features are useful for analyzing the directional tendencies of the UAV's motion, offering insights into how the UAV moves relative to each axis.

- **Angles Between Velocity and Acceleration Vectors:** The angles between the velocity and acceleration vectors were computed using the dot product as follows:

$$\text{vel_acc_angle} = \arccos\left(\frac{v_x \cdot a_x + v_y \cdot a_y + v_z \cdot a_z}{\text{vel_mag} \times \text{acc_mag}}\right)$$

This feature helps in understanding the relationship between the UAV's current direction of motion and the applied forces, which is critical for assessing maneuverability and control.

- **Trigonometric Transformations:** Trigonometric transformations (cosine and sine) for the roll and yaw angles were applied using the following equations, where roll and yaw represent the UAV's roll and yaw angles:

$$\text{roll_sin} = \sin(\text{radians}(\text{roll})), \quad \text{roll_cos} = \cos(\text{radians}(\text{roll}))$$

$$\text{yaw_sin} = \sin(\text{radians}(\text{yaw})), \quad \text{yaw_cos} = \cos(\text{radians}(\text{yaw}))$$

These transformations capture periodic trends and relationships in the UAV's orientation, aiding in the detection of anomalies in its rotational dynamics.

These feature engineering steps expanded the dataset to a total of 38 features, enhancing the model's ability to learn complex patterns related to the UAV's performance and potential structural defects.

To enhance our feature selection process, we employed ranking-based methods, specifically the Leave-One-Feature-Out (LOFO) importance technique and SHAP (SHapley Additive exPlanations) analysis, applied to the XGBoost model. As illustrated in Figure 2 , the SHAP analysis provides insight into the most influential features, with battery voltage (pm.vbatMV), altitude (stateEstimate.z), and motor outputs (motor.m3, motor.m2) being the most critical. This suggests that these features have a significant impact on the model's predictions.

Similarly, Figure 3 depicts the LOFO feature importance results for the same XGBoost model, which also highlights the battery charge, yaw measurements, and motor outputs as highly influential. This consistency between SHAP and LOFO analyses reinforces the importance of these features in the context of UAV fault detection. Additionally, LOFO analysis identifies spatial and orientation-related features, such as position magnitude and theta (theta), as relevant, indicating that both spatial dynamics and motor outputs are crucial for accurate fault detection.

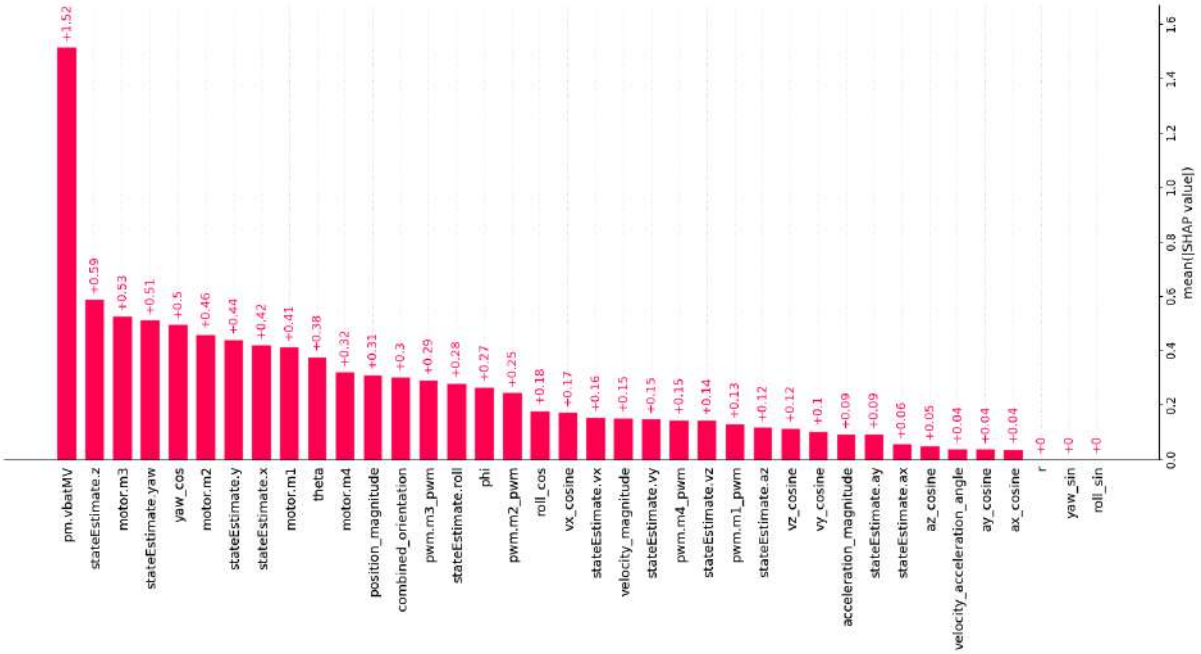


Figure 2: SHAP feature importance for XGBoost model.

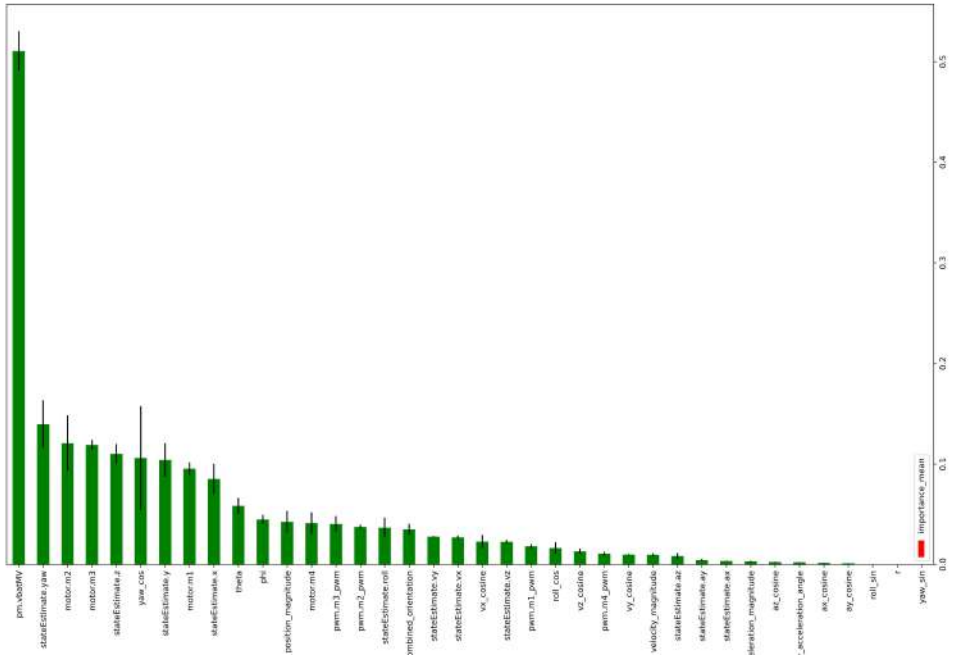


Figure 3: LOFO feature importance for XGBoost model.

We aimed to assess how different thresholds of selected features impact model performance, including accuracy, precision, recall, F1 score, and computational efficiency in terms of training and inference times. The feature selection thresholds ranged from considering all features to selecting the top 15, 20, 25, 30, and 35 features. We also examined the effects of different scaling methods, specifically Standard scaling and Quantile scaling, to understand their influence on model performance. Further settings can be found in Table 1. The results are summarized in Table 2, highlighting that selecting a subset of features using SHAP and LOFO can lead to comparable or even improved performance compared to using all features, with potential benefits in computational efficiency.

Table 1: XGBoost Model Parameters in Features Selection Experiment.

Number of Estimators	Max Depth	Learning Rate	Number of Classes	Device	Number of Folds
100	6	0.3	58	cuda	5

Table 2: Performance Comparison of Various Feature Selection Techniques.

Scaling	Selection	Threshold	Accuracy	Precision	Recall	F1	Training Time	Infer Time
Standard	Xgboost	All	96.06%	96.06%	96.06%	96.06%	137.13	0.18
Quantile	Xgboost	Top 15	90.43%	90.43%	90.43%	90.43%	109.86	**0.15**
Standard	Xgboost	Top 15	90.41%	90.41%	90.41%	90.41%	**109.19**	**0.15**
Quantile	Xgboost	Top 20	93.94%	93.94%	93.94%	93.94%	113.24	0.16
Quantile	Xgboost	Top 25	94.99%	94.99%	94.99%	94.99%	140.00	0.16

Quantile	Xgboost	Top 30	94.72%	94.72%	94.72%	94.72%	128.82	0.18
Quantile	Xgboost	Top 35	96.10%	96.10%	96.10%	96.10%	136.94	0.18
Quantile	LOFO	Top 15	92.46%	92.46%	92.46%	92.46%	109.66	**0.15**
Quantile	LOFO	Top 20	95.79%	95.79%	95.79%	95.79%	116.05	0.16
Quantile	LOFO	Top 25	96.12%	96.12%	96.12%	96.12%	**143.68**	0.16
Quantile	LOFO	Top 30	96.09%	96.09%	96.09%	96.09%	133.24	0.18
Quantile	LOFO	Top 35	96.05%	96.05%	96.05%	96.05%	138.04	0.19
Quantile	SHAP	Top 15	93.72%	93.72%	93.72%	93.72%	111.66	**0.15**
Quantile	SHAP	Top 20	93.72%	93.72%	93.72%	93.72%	111.66	**0.15**
Quantile	SHAP	Top 25	**96.28%**	**96.28%**	**96.28%**	**96.28%**	144.31	0.19
Quantile	SHAP	Top 30	96.17%	96.17%	96.17%	96.17%	127.41	0.17
Quantile	SHAP	Top 35	95.99%	95.99%	95.99%	95.99%	130.98	0.17

To better understand the significance of battery voltage in detecting faults, we analyzed the distribution of the feature 'pm.vbatMV', which represents the battery voltage in millivolts (mV). The plot in Figure 4 illustrates the density distribution of 'pm.vbatMV' across different labels. By overlaying Kernel Density Estimates (KDE) for each label, we can observe how the battery voltage varies between normal and fault conditions. This visualization highlights the critical role of monitoring battery voltage to ensure that the nano-quadcopters maintain adequate power for operation and to prevent potential issues related to battery depletion.

2.4 Baseline Models and Results

This research employed various baseline machine learning (ML) models to detect faults in UAVs. The models were chosen based on their popularity and effectiveness in classification tasks, focusing on balancing accuracy and inference time, which is a critical factor for real-time fault detection in UAV systems. The baseline models evaluated include XGBoost, Random Forest, k-Nearest Neighbors (KNN), and LightGBM. Table 4 summarizes the performance metrics, including accuracy, precision, recall, F1 score, training time, and inference time. Below is a detailed discussion of the performance and trade-offs observed with each baseline model.

- **XGBoost (Extreme Gradient Boosting):** XGBoost is an efficient, scalable, and accurate implementation of gradient boosting machines. It's particularly well-suited for tasks where both training time and prediction accuracy are critical. XGBoost achieved an accuracy of 96.17%, with balanced precision, recall, and F1 scores. Notably, it exhibited an inference time of 3.64 seconds, making it a strong candidate for real-time fault detection, though it had a longer training time of 288.91 seconds.
- **Random Forest:** As an ensemble learning method, Random Forest builds multiple decision trees and aggregates their predictions to enhance robustness and reduce overfitting. It performed slightly better than XGBoost in terms of accuracy (96.74%) but had a similar inference time of 3.46 seconds. This delay might make Random Forest less suitable for scenarios where rapid decision-making is required, and its training time was 284.07 seconds.
- **k-Nearest Neighbors (KNN):** KNN is a straightforward, instance-based learning algorithm. However, it performed relatively lower in this task, with an accuracy of 95.89%. One notable aspect is its extremely fast training time of 0.08 seconds compared to its much slower inference time of

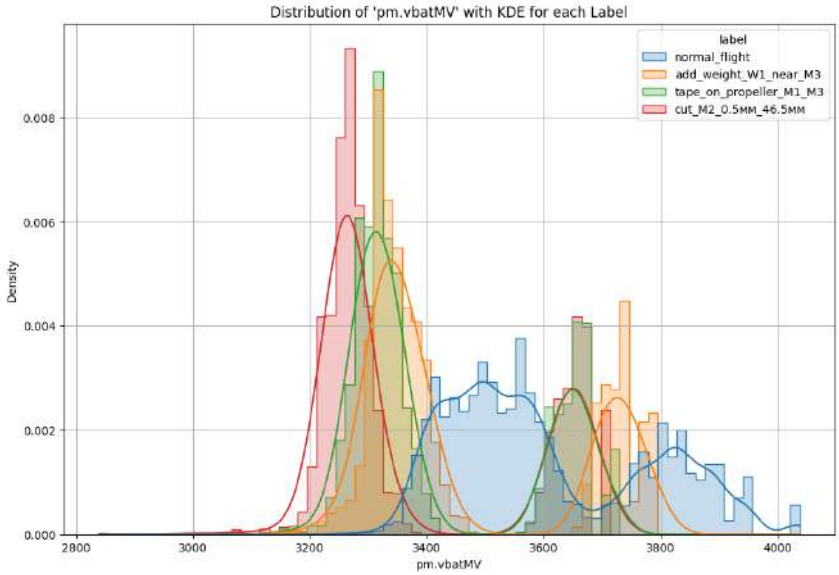


Figure 4: Distribution of 'pm.vbatMV' with Kernel Density Estimates (KDE) for various label, showing the variation in battery voltage across different fault conditions.

58.74 seconds. This phenomenon is due to the nature of KNN, where the model simply memorizes the training data and performs computations during inference, making it computationally expensive when predicting new instances, especially in a high-dimensional space.

- **LightGBM (Light Gradient Boosting Machine):** LightGBM is designed for efficiency and speed, with optimizations that make it ideal for large-scale data. It achieved an impressive accuracy of 98.67%, slightly higher than the other models. However, it had the longest training time of 2367.61 seconds and a substantial inference time of 145.76 seconds. This suggests a trade-off where LightGBM may excel in accuracy but at the cost of longer processing times, which could be a limiting factor in real-time applications.

2.5 Proposed Model and Results

The proposed model for fault detection in UAVs is a Deep Neural Network (DNN), which demonstrated superior performance across all evaluated metrics. The architecture of the model is illustrated in Figure 7, consisting of three fully connected layers with 1024, 512, and 256 hidden units, respectively, each followed by ReLU activations and 8% dropout. The model was developed through a series of iterative improvements, starting with a simpler architecture and gradually enhancing its complexity and performance through careful adjustments to its hyperparameters.

Initially, the model was constructed with two fully connected layers and a learning rate of 0.001. The hidden units in these layers were 32 and 16, and the batch size was set to 32. This preliminary model achieved an accuracy of approximately 82%, indicating a promising start. However, in an attempt to improve performance, an additional layer was added, the learning rate was increased to 0.01, and the hidden units were adjusted to 64, 32, and 16 across the layers. Unfortunately, this configuration led to a significant drop in accuracy, with results falling into the 40% range.

To address these issues, batch normalization and dropout were introduced. Dropout was initially set at 20%, but it led to overfitting, so it was gradually reduced to 15%, 10%, and finally 8%, where the model began to show better generalization. The hidden units were then increased in a stepwise fashion, first to 128, 64, and 32, which improved accuracy to the 60% range, and eventually to the final configuration of 1024, 512, and 256 hidden units. These adjustments led to a significant improvement in the model's performance.

Moreover, the batch size was also incrementally increased, starting from 32 and going up to 512. However, it was observed that increasing the batch size did not have as substantial an impact on the model's performance as the adjustments to the hidden units. The learning rate of 0.001 consistently provided the best results, as any increase in the learning rate resulted in a decline in model performance.

The final configuration and hyperparameters of the model are outlined in Table 3.

Table 3: Hyperparameters of the Proposed DNN Model.

Layers	Batch Size	Hidden Units L1	Hidden Units L2	Hidden Units L3	Learning Rate
4	128	1024	512	256	0.001

The DNN model in this study achieved the highest accuracy, precision, recall, and F1 scores, all at 98.66%. Notably, it also had the lowest inference time of 0.92 seconds, although it required a longer training time of 533.73 seconds. The training and validation losses shown in Figure 5 were closely monitored throughout the training process, showing a steady decrease and indicating effective learning and generalization. Additionally, the confusion matrix in Figure 6 revealed that the DNN performed exceptionally well across all classes, with minimal misclassifications, underscoring its robustness in detecting faults. The combination of high accuracy, efficient inference time, and strong performance metrics makes the DNN the most effective model for fault detection in UAVs, particularly when both accuracy and real-time performance are critical.

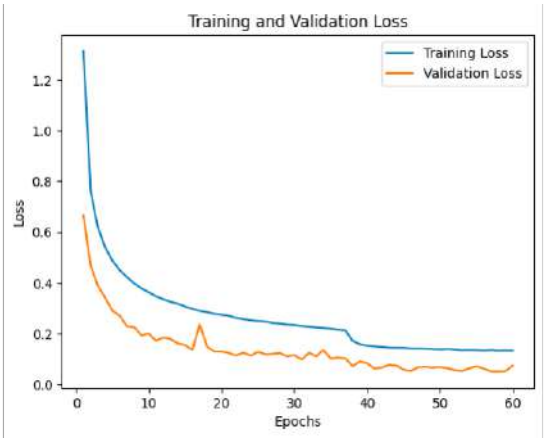


Figure 5: Training Loss and Validation losses.

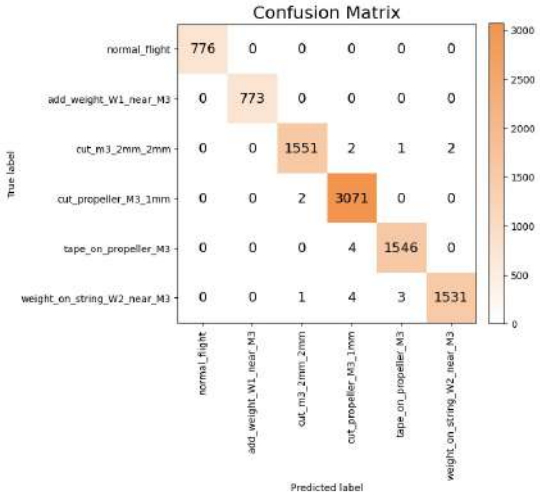


Figure 6: Confusion Matrix of Key Classes.

Table 4: Comparison of Model Performance Metrics Across Different Models.

Model's Name	Accuracy	Precision	Recall	F1	AUC	Training Time (s)	Inference Time (s)
XGBoost	96.17%	96.17%	96.17%	96.17%	0.9994	288.91	3.64
Random Forest	96.74%	96.74%	96.74%	96.73%	0.9978	284.07	3.46
KNN	95.89%	95.89%	95.89%	95.88%	0.9997	**0.08**	58.74
LightGBM	**98.67%**	**98.67%**	**98.67%**	**98.67%**	**0.9999**	2367.61	145.76
DNN	98.56%	98.66%	98.65%	98.65%	**0.9999**	533.73	**0.92**

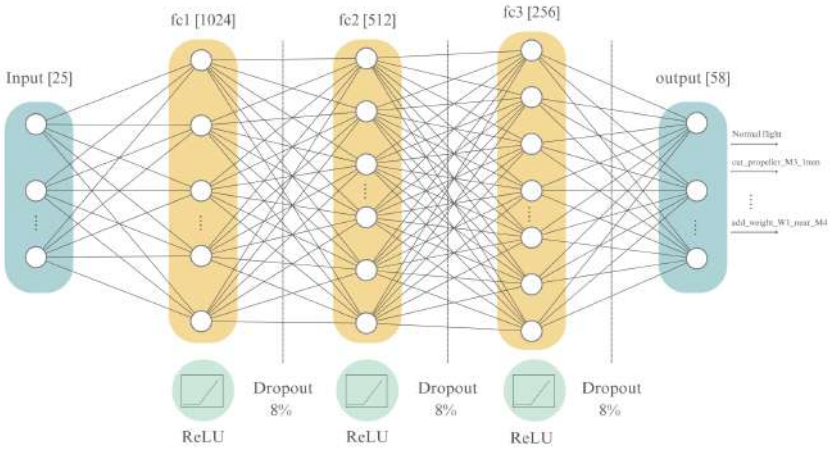


Figure 7: Architecture of the Proposed Deep Neural Network (DNN) for UAV Fault Detection.

2.6 Experimental Environment

The experiments for this research were conducted on the Ibex computational cluster. Ibex is equipped with a diverse set of CPU and GPU architectures, providing robust computational power for handling large-scale ML tasks. Specifically, the configuration used for this study included:

- **CPU:** x86_64 with 64 cores
- **GPU:** NVIDIA A100-SXM4-80GB
- **RAM:** 503.68 GB

These specifications provided the necessary computational power for running the various ML models efficiently.

2.7 Discussion

The results reveal an important trade-off between accuracy and inference time across the models. The Deep Neural Network (DNN) stands out with the highest accuracy, precision, recall, and F1 scores, all at 98.56%. Notably, it also features the fastest inference time of 0.92 seconds among all models, which is a significant advantage for scenarios requiring rapid responses. However, this performance comes with a longer training time of 533.73 seconds.

XGBoost provides a balanced approach with high accuracy (96.17%) and a relatively fast inference time of 3.64 seconds. While its inference time is longer than the DNN's, it still performs well in real-time fault detection applications where quick decision-making is necessary.

The k-Nearest Neighbors (KNN) model, despite its extremely fast training time of 0.08 seconds, suffers from a long inference time of 58.74 seconds. This is due to KNN's need to compute distances between the test instance and all training instances, which becomes increasingly time-consuming with larger datasets.

LightGBM achieves the highest accuracy among the baseline models (98.67%) but has the longest training time of 2367.61 seconds and a substantial inference time of 145.76 seconds. This suggests that LightGBM is better suited for applications where accuracy is prioritized and model updates are infrequent, but its long inference time may limit its use in time-sensitive applications without further optimization.

In summary, the choice of model for fault detection in UAVs should carefully consider the trade-offs between accuracy, training time, and inference time. For applications demanding both high accuracy and fast inference, the DNN proves to be the most effective option, balancing these factors exceptionally well.

3 Conclusion

3.1 Summary of Findings

This study evaluates various machine learning models for UAV fault detection, highlighting their trade-offs between accuracy and inference time.

The Deep Neural Network (DNN) achieved the highest accuracy (98.56%) and the fastest inference time (0.92 seconds), though it requires longer training (533.73 seconds). XGBoost provides a good balance with high accuracy (96.17%) and a relatively fast inference time (3.64 seconds), making it suitable for real-time applications. The k-Nearest Neighbors (KNN) model has a very fast training time (0.08 seconds) but suffers from a long inference time (58.74 seconds). LightGBM, while having the highest accuracy (98.67%), faces challenges with the longest training (2367.61 seconds) and inference times (145.76 seconds).

In summary, the DNN offers the best combination of accuracy and inference speed, making it ideal for applications requiring rapid responses and high precision.

3.2 Future Work

Building on these findings, future research should focus on:

- **Hybrid Models:** Integrate motor and controller fault data to enhance UAV fault detection.
- **Efficient Algorithms:** Explore more efficient algorithms and hybrid approaches for balancing complexity and performance.
- **Feature and Data Impact:** Investigate the effects of different feature sets and preprocessing methods on model performance.
- **Diverse Datasets:** Expand the study to include varied datasets and scenarios for a comprehensive understanding of model capabilities.
- **Edge Computing:** Explore the deployment of the DNN model on low-power edge devices, such as onboard UAV processors. This will require investigating model compression techniques like model quantization, pruning, and knowledge distillation to ensure that the model remains efficient and effective within the limited computational resources available on edge devices.

These steps can help advance fault detection systems for UAVs, improving their effectiveness and efficiency.

References

[1] Samir Bouabdallah and Roland Siegwart. Design and control of a miniature quadrotor. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 2279–2284. IEEE, 2005.

[2] Ming Gou, Hao Tang, Lei Song, Zhong Chen, Xiaoming Yan, Xiangwen Zeng, and Wenlong Fu. Research on defect diagnosis of transmission lines based on multi-strategy image processing and improved deep network. *Processes*, 12(9), 2024.

[3] Kamil Masalimov, Tagir Muslimov, Evgeny Kozlov, and Rustem Munasypov. Crazypad: A dataset for assessing the impact of structural defects on nano-quadcopter performance. *Data*, 9(6), 2024.

[4] Nabil A. Mohamed, Ehab Gomaa, and Ahmed Ahmed. A comprehensive review on uavs applications for civil engineering: From the structural inspection to modeling. *Journal of Building Engineering*, 26:100879, 2019.

[5] Qing Ou, XiaoQi Chen, David Park, Aaron Marburg, and James Pinchin. Integrated flight dynamics modelling for unmanned aerial vehicles. In *2008 IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, pages 570–575, 2008.

[6] Radosław Puchalski and Wojciech Giernacki. Uav fault detection methods, state-of-the-art. *Drones*, 6(11), 2022.

[7] Hazim Shakhatreh, Alaa Sawalmeh, Ala Al-Fuqaha, Zhengguo Dou, Emad Almaita, Issa Khalil, Nasir S Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019.