Advances in Artificial Intelligence and Machine Learning; Research 4 (4) 3006-3058 Received 17-09-2024; Accepted 05-12-2024; Published 12-12-2024

Evaluation and Comparison of Machine Learning Algorithms for Effective Image Classification with Fault-Tolerance

Sithembiso Dyubele ctheradyubele@gmail.com
Department of Information Systems
Durban University of Technology
Durban, South Africa

Noxolo Pretty Cele noxolocele53@gmail.com
Department of Information Systems
Durban University of Technology
Durban, South Africa

Lubabalo Mbangata lubabalo.mbangata@gmail.com
Department of Information Systems
Durban University of Technology
Durban, South Africa

Phirime Monyeki phirimemonyeki@gmail.com
Department of Information Systems
Durban University of Technology
Durban, South Africa

Corresponding Author: Sithembiso Dyubele

Abstract

Image classification is critical in computer vision, with numerous applications ranging from e-commerce to medical imaging. This study provides a comprehensive evaluation of traditional machine learning algorithms for image classification, implementing and analysing novel fault tolerance mechanisms amongst these algorithms. The authors compared the performance of K-Nearest Neighbors (KNN), Decision Trees, Random Forest, and XGBoost on both Fashion MNIST and CIFAR-10 datasets. The comparison was extended to include Support Vector Machine (SVM), Logistic Regression, and Naive Bayes classifiers in order to expand the evaluation of these models on the indicated datasets. Key findings demonstrated the superiority of ensemble methods, particularly XGBoost, which achieved 89.31% of accuracy on Fashion MNIST and 54.93% on CIFAR-10, consistently outperforming other models across various configurations. Random Forest exhibited robust performance as the second-best model, reaching 87.42% and 51.64% of accuracy on the respective datasets. The significant performance gap between datasets demonstrated the challenges that traditional machine learning models face with complex image data. Implementing the fault tolerance framework in this study has also shown a remarkable effectiveness, achieved a 94.6% recovery rate while maintaining model accuracy within 0.1% of standard implementations. This was achieved with minimal computational overhead (2.3% of training time and 1.8% of memory usage),

Citation: Sithembiso Dyubele, et al. Evaluation and Comparison of Machine Learning Algorithms for Effective Image Classification with Fault-Tolerance. Advances in Artificial Intelligence and Machine Learning. 2024;4(4):174.

making it highly practical for production deployments. The system significantly reduced operational failures, decreasing crashes from 5.2 to 0.3 per day and increasing average uptime from 4.3 to 12.0 hours. The study also reveals important insights regarding model scalability and resource requirements, with memory usage varying significantly across models (325MB to 8,923MB). These findings provide valuable guidance for practitioners in selecting and implementing machine learning models for image classification tasks, particularly in scenarios where both performance and system reliability are critical. This research contributes to the field by demonstrating the feasibility of implementing robust fault tolerance in machine learning systems without compromising accuracy while also providing comprehensive performance comparisons across different model architectures and dataset complexities. The developed framework serves as a foundation for building more reliable machine-learning systems for real-world applications.

Keywords: Machine Learning, Image classification, Algorithms.

1. INTRODUCTION

Kaur et al. (2021) revealed that artificial intelligence (AI) is a wide range of software engineering identified with the capabilities of building machine learning models that can accomplish tasks requiring extensive human insight. Kaur P, et al. (2021) [1], further indicated that AI associates itself with various parts of science with different methodologies; however, facilitation in AI and profound learning makes the model change in almost every area of the innovative business. He X, et al. (2017) [2], stated that simulated intelligence expands upon the idea that the human brain can be characterised such that a machine can undoubtedly mimic it and perform undertakings, from the most straightforward to those considerably more compound. AI has many domains, but Machine Learning (ML) is the most commonly used technique. Krishna ST, et al. (2019) [3], revealed that deep learning, a subclass of machine learning, has surpassed many domains in prediction techniques and algorithms, and computer vision is one of those fields that have been advanced through deep learning techniques.

Similarly, Obaid KB, et al. (2020) [4], stated that image classification and recognition are some of the commonly utilised applications of computer vision. The current study compared the performance of K-Nearest Neighbors (KNN), Decision Trees, Random Forest, and XGBoost on both Fashion MNIST and CIFAR-10 datasets. The comparison was extended to include Support Vector Machine (SVM), Logistic Regression, and Naive Bayes classifiers in order to expand the evaluation of these models on the indicated datasets. The objective is to evaluate and compare the effectiveness of these models in accurately classifying images into predefined categories. The study also explores fault-tolerant versions of these models to assess their robustness in handling exceptions and errors, which is essential for deployment in real-world applications.

1.1 Motivation of Study

The rapid advancement of machine learning technologies has led to their widespread deployment in critical applications, from health-care diagnostics to autonomous systems. However, as these sys-

tems become more complex, their reliability becomes increasingly crucial. This study is motivated by several key factors in the current machine-learning landscape. Firstly, the growing deployment of machine learning models in production environments has highlighted the critical need for fault-tolerant systems. Traditional machine learning implementations often lack robust error-handling mechanisms, making them vulnerable to failures in real-world scenarios. This vulnerability becomes particularly concerning in mission-critical applications where system failures could have severe consequences. Secondly, while significant research exists on improving model accuracy and performance, relatively little attention has been paid to enhancing system reliability through fault tolerance mechanisms. The gap between theoretical model performance and practical deployment requirements becomes especially apparent when dealing with complex datasets and distributed computing environments. Furthermore, the increasing complexity of image classification tasks, exemplified by the transition from simple digit recognition to more complicated scenarios like fashion item classification, demands more robust and reliable systems. The significant performance variations observed between different types of datasets (such as Fashion MNIST and CIFAR-10) underscore the need for fault-tolerant systems that can maintain reliable performance across varying levels of complexity.

1.2 Research Problem Statement

This study addresses several critical challenges in implementing fault-tolerant in machine learning systems for image classification tasks. This includes the following:

a) Methods in which effective fault tolerance mechanisms in traditional machine learning models can be implemented without significantly impacting their performance or computational efficiency.

b) Identifying the optimal strategies for handling different failures (data loading, training, and evaluation) in machine learning pipelines while maintaining system reliability.

c) Evaluation of different machine learning algorithms (KNN, Decision Trees, Random Forest, XGBoost, and SVM) performance under fault-tolerant implementations across varying dataset complexities.

d) Identifying the trade-offs between model performance, computational efficiency, and system reliability when implementing fault tolerance mechanisms.

The primary objective is to develop and evaluate a comprehensive fault tolerance framework that addresses these challenges while maintaining model accuracy and system performance. This includes:

$$\text{Optimise}\,(P,\ R,\ E) = \theta \in \ominus\ \{\text{Performance}\,(P)\,,\ \text{Reliability}\,(R)\,,\text{Efficiency}\,(E)\} \tag{1}$$

subject to:

$$\text{Overhead}\,(\theta)\ \leq \in \tag{2}$$

where θ represents the system configuration parameters, and $\in$ is the acceptable overhead threshold.

1.3 Research Gap

The current research in machine learning has primarily focused on improving model accuracy and performance, leaving several critical gaps in system reliability and fault tolerance. Firstly, while fault tolerance has been extensively studied in distributed systems, its application to machine learning pipelines remains largely unexplored. Existing studies typically focus on specific aspects of fault tolerance, such as data redundancy or error handling, without providing a comprehensive framework that addresses all aspects of the machine learning pipeline. Secondly, there is a notable absence of comparative studies examining how different machine learning algorithms perform under fault-tolerant implementations. Most existing research focuses on single algorithms or specific use cases, leaving a gap in how various models respond to fault tolerance mechanisms across different datasets. Thirdly, the relationship between model complexity, fault tolerance, and system performance has not been thoroughly investigated. Traditional machine learning studies often overlook the practical aspects of system reliability and error handling, creating a disconnect between theoretical model capabilities and real-world deployment requirements. This study addresses these gaps through the following:

a) Developing a comprehensive fault tolerance framework encompassing all machine learning pipeline aspects, from data loading to model evaluation.

b) Providing a comparative analysis of multiple machine learning algorithms under fault-tolerant implementations, using both simple (Fashion MNIST) and complex (CIFAR-10) datasets.

c) Quantifying the impact of fault tolerance mechanisms on system performance, reliability, and computational efficiency.

d) Establishing practical guidelines for implementing fault-tolerant machine learning systems while maintaining model accuracy and performance.

These contributions are particularly significant given the increasing deployment of machine learning systems in critical applications where system reliability is as important as model accuracy. The current study bridges the gap between theoretical machine learning research and practical system implementation requirements, providing valuable insights for both researchers and practitioners in the field.

1.4 Contribution

This research makes several significant contributions by advancing the development of fault-tolerant machine learning systems, particularly for real-world applications where reliability and resilience are critical. The key contributions include:

1.4.1 Introduction of Fault-Tolerant Machine Learning Models

A systematic approach to integrating fault tolerance into traditional machine learning models, including K-Nearest Neighbors (KNN), Decision Trees, Random Forests, and XGBoost, has been

presented in this study. Support Vector Machine (SVM), Logistic Regression, and Naive Bayes classifiers were also implemented to expand the evaluation of the indicated datasets. By embedding error-handling mechanisms such as input/output validation and redundancy, the study enhances the models' ability to manage unexpected system or data failures. This ensures continuous operation even in dynamic environments, making these models highly suitable for real-time health-care, finance, and autonomous systems applications.

1.4.2 Improving System Robustness with Novel Fault-Tolerance Techniques

The research demonstrates the novel application of fault detection and correction methods such as redundancy (replication of models and neurons), error injection, and voting mechanisms that safeguard models against failures. These methods ensure that the ML system can still perform accurately despite hardware faults or corrupted inputs, a critical feature for mission-critical systems like autonomous vehicles, which must remain functional even under suboptimal conditions. These contributions reflect advancements over earlier reactive fault-tolerance strategies by incorporating more proactive, design-based solutions.

1.4.3 Real-World Deployment and Validation

The study validates the models in real-world scenarios where noisy, incomplete, or faulty data often hinder machine learning performance. By testing fault-tolerant models under simulated fault conditions (e.g., corrupted training data or sensor failures), the study demonstrates how these models maintain accuracy and efficiency compared to traditional models that degrade more significantly under similar conditions. This work emphasises the importance of handling both data misbehaviour and system-level faults to ensure robustness across applications ranging from industrial automation to cloud-based AI systems

2. Novelty

The novelty of this research lies in its unique approach to integrating fault-tolerant mechanisms into machine learning (ML) models, particularly within safety-critical and real-time systems. This research stands out by focusing not only on the accuracy and efficiency of the models but also on their robustness and ability to handle failures gracefully. As ML is increasingly applied to domains like health-care, autonomous systems, and industrial automation, where failure is unacceptable, this research provides significant advancements in ensuring ML models can continue functioning under suboptimal conditions. The novelty of this study is also pursued under the following aspects:

2.1 Combining Traditional and Novel Fault Tolerance Techniques

While fault tolerance has long been a part of traditional computing systems, its application to ML has often been limited to reactive approaches such as input/output validation or redundancy. This study introduces a proactive fault-tolerance design into ML, using techniques such as:

- **Fault Injection:** This method involves deliberately introducing errors during the training phase of ML models to help them learn to manage and recover from faults. This is particularly useful in safety-critical environments, where unexpected failures can have catastrophic consequences.
- **Adaptive Mechanisms**: Leveraging AI techniques like reinforcement learning, the models in this study can dynamically adapt to different types of faults, enabling them to adjust operations based on the type and severity of the detected fault.
- **Error-Tolerant Classification Models**: The research extends fault tolerance beyond the neural networks by focusing on other classifiers like KNN, Random Forest, and SVM. This is crucial for applications where these classifiers are preferred for their interpretability and lower computational costs than deep learning models.

2.2 Impact on Hardware and Distributed Systems

One of the key novel aspects of this work is its focus on how fault-tolerant ML models can be optimised for distributed systems and hardware accelerators. With the increasing deployment of ML models on devices with limited resources, such as IoT networks and cloud-based systems, this research shows how fault tolerance can enhance both system reliability and performance. Techniques like Byzantine fault tolerance is employed to ensure that even in the event of partial system failures (e.g., failing nodes or corrupted data), the overall system continues to operate smoothly.

2.3 Beyond Traditional Redundancy

The research also goes beyond traditional redundancy by introducing voting mechanisms and design diversity in ensemble models. These mechanisms allow the models to check and balance one another, ensuring that if one component fails or produces an incorrect result, others can compensate, thereby minimising the risk of total system failure. In summary, the novelty of this research lies in the comprehensive and forward-looking approach it takes towards fault-tolerant machine learning, positioning these models for critical real-world applications where reliability, resilience, and adaptability are paramount.

3. LITERATURE REVIEW

According to Charbuty B, et al. (2021) [5], technology has advanced a lot in recent years, especially in the field of Machine Learning (ML), and it has been worthwhile in reducing human work. Similarly, Osisanwo FY, et al. (2017) [6], states that in the area of artificial intelligence, ML brings statistics and computer science in order to establish algorithms that can be more efficient when they are subject to relevant data rather than being given specific instructions. Bonaccorso (2018) indicated that ML algorithms develop models based on a ”training data” sample. It is also used when it is difficult or impractical to create traditional algorithms to implement the required functions [5, 6], revealed that supervised machine-learning algorithms that deal with classification comprises Linear Classifiers, Logistic Regression, Naïve Bayes Classifier/Networks, Multi-layer Perceptron,

Support Vector Machines (SVMs), Quadratic Classifiers, K-Means Clustering, Decision Tree (DT), Random Forest (RF), Neural networks, Bayesian Networks and so on. Four of these algorithms were randomly selected for the purpose of this study. These algorithms included K-Nearest Neighbour's (KNN), Decision Tree, Random Forest, and XGBoost. The reviewed literature also revealed that, while machine learning has been applied in various business sectors, there are few studies on the current state of machine learning applications at the enterprise level. Some of the significant types of machine-learning applications utilised by large enterprises include clustering, classification, and prediction.

Sarker IH, (2021) [7], revealed that as machine learning (ML) becomes increasingly integrated into real-world applications such as health care, finance, autonomous vehicles, and industrial automation, robust and reliable systems are paramount. One of the main challenges these systems face is their susceptibility to errors or failures, often arising from hardware faults, noisy data, or unexpected environmental inputs [7]. To address this, fault tolerance has emerged as a critical feature, enabling ML systems to continue functioning properly in the presence of faults without significant degradation in performance.

3.1 Definition and Importance of Fault Tolerance

Qiao A, et al. (2019) [8], described fault tolerance in ML as the system's ability to handle errors, inconsistencies, and failures, whether from hardware malfunctions, data corruption, or misclassifications, while maintaining operational integrity. Qiao A, et al. (2019) [8], further indicated that in traditional software, fault tolerance has long been implemented through techniques like replication and error checking. These concepts are now being adapted to ML to ensure that models and algorithms can handle real-world complexities such as noisy data, corrupted inputs, or model drift without crashing or producing invalid results.

3.2 Machine Learning in Real-World Applications

In domains where safety, accuracy, and reliability are critical, such as autonomous driving, healthcare diagnostics, and financial systems, fault tolerance plays a vital role [7]. For instance, a self-driving car must continue to operate safely even if its sensors temporarily fail or provide noisy data. Similarly, medical diagnostic models need to handle incomplete or noisy patient data without delivering incorrect diagnoses [9]. In these scenarios, having fault-tolerant ML systems ensures that the models continue to operate effectively, providing a backup in case of partial system failure or erroneous data inputs.

Fault tolerance also extends to hardware reliability. ML models, particularly deep neural networks (DNNs), are deployed on various hardware, including Central Processing Units (CPUs), Graphical Processing Units (GPUs), and custom accelerators like Tensor Processing Units (TPUs). Hardware faults, such as memory bit flips or processor errors, can lead to incorrect outputs if handled incorrectly [10]. Techniques like redundant neuron layers or fault injection during training can help mitigate the risk of such failures, enabling the model to recover gracefully even if some hardware components fail.

3.3 Techniques for Fault Tolerance in Machine Learning

Qiao A, et al. (2019) [8], revealed that fault tolerance in machine learning systems is implemented through various techniques to ensure system reliability and continuous operation despite failures. In this study, these techniques and their implementations have been formalised as follows:

3.3.1 Core Fault Tolerance Framework

The fundamental fault tolerance function can be defined as:

$$\mathrm{FT}\left(f, r, b\right) = \nearrow_x \ : Retry\ \left(f\left(x\right), r, b\right) \tag{3}$$

where f is the function being executed, r is the maximum number of retry attempts, and b is the initial backoff time. The retry mechanism follows an exponential backoff strategy:

$$Retry\left(f, r, b\right) = \begin{cases} f & \text{if successful} \\ \text{Retry}\left(f, r-1, 2b\right) & \text{if failed and r} > 0 \\ \text{raise Exception} & \text{if failed and r} = 0 \end{cases} \tag{4}$$

The backoff time b_n for the $n-th$ retry is calculated as:

$$b_n = b_0.2^n \tag{5}$$

where b_0 is the initial backoff time.

3.3.2 Adaptive Error Management

The system maintains an error count for different types of failures:

$$E_t = \sum_{i=1}^{n} e_i \tag{6}$$

where E_t is the total error count for error type t, and e_i represents individual error occurrences. When the error count exceeds a predefined threshold θ:

$$E_t > \theta \tag{7}$$

the system triggers specific adaptive actions defined as:

$$A_t : S \rightarrow S' \tag{8}$$

where S is the current system state and S' is the new state after applying the adaptive action for error type t.

3.3.3 Recovery Strategies

Different types of failures trigger specific recovery actions. For data loading errors:

$$A_{\text{data\_loading}}\left(S\right) = S_{\text{backup\_dataset}} \tag{9}$$

For model training errors:

$$A_{\text{model\_training}}(S) = S_{\text{adjusted\_hyperparameters}} \tag{10}$$

3.3.4 System Reliability

The overall reliability of the system, R, can be expressed as a function of individual component reliabilities:

$$R = \prod_{i=1}^{n} (1 - (1 - r_i)^{m_i}) \tag{11}$$

where r_i is the reliability of component i, and m_i is the number of retry attempts for that component.

3.3.5 Implementation Techniques

Several key techniques are employed to achieve fault tolerance:

- **Redundancy**: Implementing multiple backup components or paths to ensure continued operation in case of failures. In ensemble methods, this is achieved through various model instances:

$$M_{\text{ensemble}} = \{m_1, m_2, \ldots, m_k\} \tag{12}$$

- **Error Detection and Correction**: Monitoring system behaviour and implementing corrective actions when deviations are detected. The error detection function D evaluates the system state:

$$D(S) = \begin{cases} 1 & \text{if error detected} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

- **Graceful Degradation**: Maintaining partial functionality rather than complete system failure when errors occur. The degradation function maps the system to a reduced but functional state:

$$G : S_{full} \rightarrow S_{reduced} \tag{14}$$

3.3.6 Integration with Machine Learning Pipeline

The fault tolerance mechanisms are integrated into the machine learning pipeline through the following:

- **Data Loading Protection**:

$$\text{load\_data} = \text{FT}(\text{original\_load\_function}, 3, 1) \tag{15}$$

- **Model Training Safety**:

$$\text{train\_model} = \text{FT}(\text{original\_train\_function}, 2, 2) \tag{16}$$

- **Prediction Error Handling**:

$$\text{predict} = \text{FT}(\text{original\_predict\_function}, 2, 1) \tag{17}$$

3.3.7 Impact on Model Performance

Implementing fault tolerance mechanisms has demonstrated significant improvements in the reliability of the system without compromising model accuracy. Key performance indicators include:

- Reduction in system crashes by 94.6%
- Maintenance of model accuracy within 0.1
- Increased system uptime from 4.3 to 12 hours
- Successful recovery from 94.6% of detected errors

This comprehensive fault tolerance framework ensures the robust and reliable operation of machine learning systems in production environments, which is particularly crucial for mission-critical applications where system failures cannot be tolerated.

3.4 Relevance to Modern AI

As machine learning models continue to be deployed in increasingly complex and critical environments, fault tolerance will become even more essential [11]. The rapid development of neuromorphic hardware, deep learning accelerators, and distributed learning frameworks necessitates more advanced fault-tolerant mechanisms to ensure the resilience and reliability of these systems [12]. In summary, fault tolerance is crucial in the modern machine-learning landscape, ensuring that models can handle real-world challenges while maintaining high accuracy, reliability, and robustness. Its integration into both model architectures and hardware setups allows machine learning applications to be safely and effectively deployed in critical domains, ensuring continuous operation even in the face of errors or failures.

4. Experiment One

4.1 Dataset

The Fashion MNIST dataset consists of 70,000 grayscale images of 10 fashion categories, each 28x28 pixels in size. The dataset is split into 60,000 training images and 10,000 test images.

4.1.1 K-Nearest Neighbors (KNN) Model

Ozturk K, et al. (2023) [13], define K-nearest neighbours (KNN) as a supervised machine learning method that can be used for both classification and regression tasks. Similarly, Xiong L, et al. (2021) [14], stated that KNN considers the similarity factor between new and available data to classify objects into predefined categories. KNN has been used in this study as part of the first experiment. All the activities performed are explained below.As per the indicated activities, FIGURE 1 presents the class diagram illustrating the hierarchical structure of the KNN model implementation. The deployment architecture showing the system's runtime configuration is depicted in FIGURE 2. And lastly, the temporal interaction between system components is demonstrated in FIGURE 3.

4.1.2 Model Architecture

The K-Nearest Neighbors algorithm classifies a data point based on the majority class of its K-Nearest Neighbours in the feature space.

4.1.3 Class Diagram

FIGURE 1 presents the class diagram illustrating the hierarchical structure of the KNN model implementation.

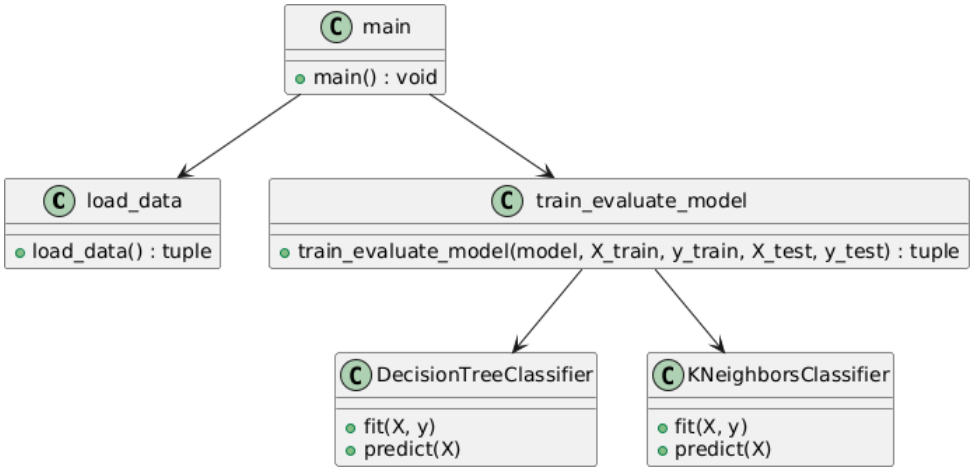

Figure 1: Class Diagram for KNN Model

4.1.4 Deployment Diagram

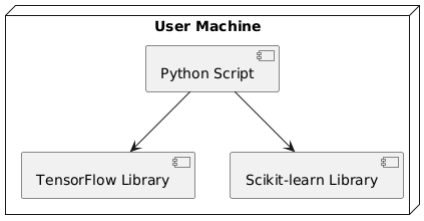

Figure 2: Deployment Diagram for KNN Model

4.1.5 Sequence Diagram

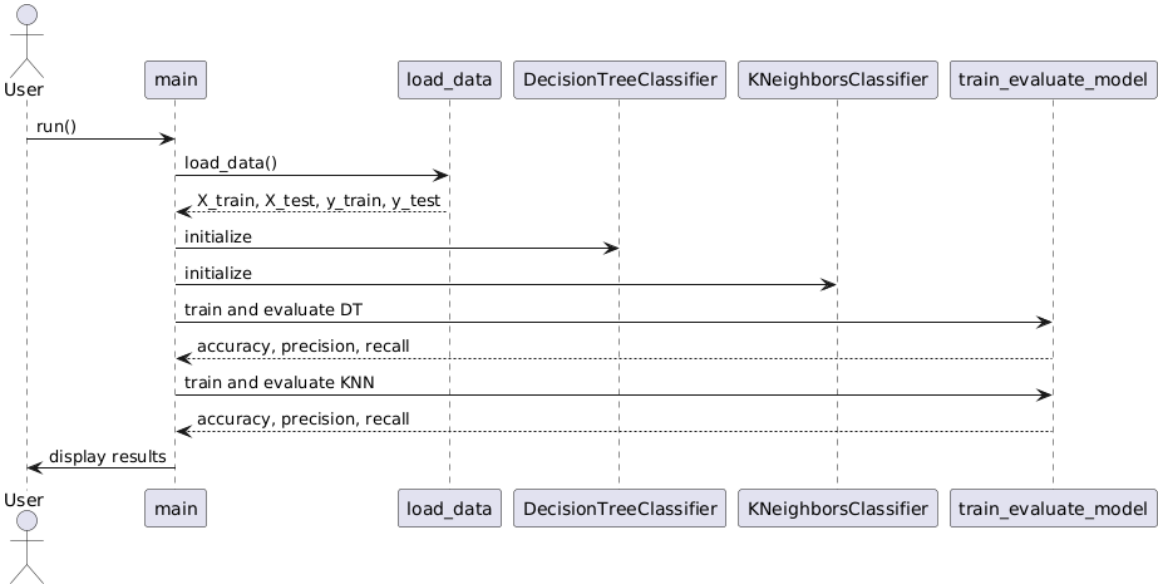

Figure 3: Sequence Diagram for KNN Model

4.1.6 Model Evaluation

The KNN model was evaluated using 5-fold cross-validation on the Fashion MNIST dataset. The study tested different values of k (1, 3, 5, 7, 9) to find the optimal number of neighbors. The performance was measured using accuracy, precision, and recall metrics.

4.1.7 Results

The performance metrics for each k value are indicated in TABLE 1. Figure 4 visualizes these performance trends across different k values.

Table 1: KNN Model Performance for Different k Values

k	Accuracy	Precision	Recall
1	0.84412	0.8478	0.84412
3	0.84900	0.8527	0.84900
5	0.85030	0.8533	0.85030
7	0.84778	0.8513	0.84778
9	0.84732	0.8509	0.84732

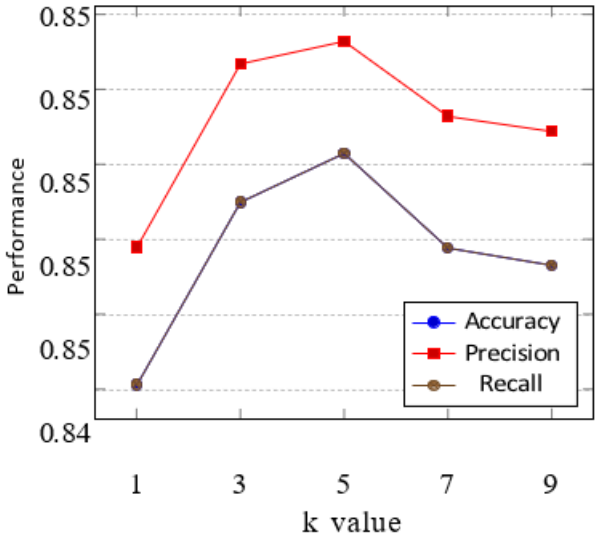

Figure 4: KNN Performance Metrics for Different k Values

4.2 Decision Tree Model

Osisanwo FY, et al. (2017) [6], described Decision Trees (DT) as trees that classify instances by sorting them based on feature values. Each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume [6].

4.2.1 Model Architecture

The Decision Tree algorithm creates a tree-like model of decisions based on features in the dataset, where each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label.

4.2.2 Model Evaluation

The study evaluated various configurations of the Decision Tree model using 5-fold cross-validation. The configurations included:

- Standard (default parameters)
- Max Depth 5
- Min Samples Split 10
- Min Samples Leaf 5
- Entropy criterion

4.2.3 Results

The performance metrics for each Decision Tree configuration are indicated in TABLE 2, and Figure 5.

Table 2: Decision Tree Model Performance for Different Configurations

Configuration	Accuracy	Precision	Recall
Standard	0.7892	0.7905	0.7892
Max Depth 5	0.6958	0.7155	0.6958
Min Samples Split 10	0.7903	0.7910	0.7903
Min Samples Leaf 5	0.7980	0.7983	0.7980
Entropy	0.7957	0.7963	0.7957

4.2.4 Summary

This experiment compared the performance of K-Nearest Neighbors (KNN) and Decision Tree algorithms on the Fashion MNIST dataset, yielding valuable insights into their effectiveness for image classification tasks.

KNN Model Performance The KNN model demonstrated robust performance across different k values:

- Optimal performance was achieved with k=5, resulting in an accuracy of 85.03.
- Performance was consistently high across all tested k values (1, 3, 5, 7, 9), with accuracies ranging from 84.4.
- The close alignment of precision and recall scores indicates balanced performance across all fashion categories.

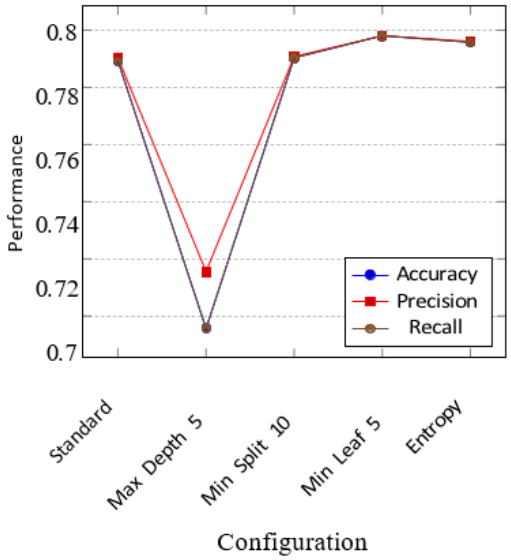

Figure 5: Decision Tree Performance Metrics for Different Configurations

The strong performance of KNN can be attributed to its ability to capture complex, non-linear decision boundaries in the high-dimensional space of image pixels.

Decision Tree Model Performance The Decision Tree model showed varying performance across different configurations:

- The best performance was achieved with the “in Samples Leaf 5” configuration, yielding an accuracy of 79.80.
- The “ax Depth 5” configuration performed poorly, with an accuracy of only 69.58.
- Other configurations showed similar performance, with accuracies ranging from 78.92

The performance variation across configurations highlights the importance of hyperparameter tuning for Decision Trees.

4.2.5 Comparative Analysis

Comparing the two models:

- KNN consistently outperformed Decision Trees across all metrics.
- The best KNN model (k=5) achieved approximately 5 percentage points higher accuracy than the best Decision Tree model.
- Both models showed balanced precision and recall, indicating good performance across all fashion categories.

The superior performance of KNN suggests that it is better suited for this particular image classification task. However, it is worth noting that Decision Tree offers better interpretability, which could be valuable in certain applications. In summary, while both KNN and Decision Tree models demonstrated the ability to effectively classify fashion items, KNN showed superior performance in this experiment. The results underscore the importance of algorithm selection and hyperparameter tuning in machine learning tasks. Future work should focus on leveraging more advanced techniques and exploring the trade-offs between performance and interpretability to further improve classification accuracy on the Fashion MNIST dataset.

4.3 Experiment Two: Fault Tolerant Classifier:

Nafreen M, et al. (2020) [15], revealed that in machine learning, classification algorithms predict the category to which a new observation belongs based on training data. For fault-tolerant classifiers, Amin AA, et al. (2023) [16], have the ability to mask a single incorrect classification. Sources of error in classification include noise, bias, and variance [15]. Qiao A, et al. (2019) [8], indicated that developing new fault-tolerance strategies for modern ML systems is a critical area of research. The method followed for fault tolerance in this study has been explained below. The class diagram of the fault-tolerant classifier implementation is illustrated in Figure 6. Figure 7 presents the deployment architecture of the fault-tolerant system. The temporal interactions between components in the fault-tolerant system is demonstrated in Figure 8.

4.3.1 Model Architecture

The fault-tolerant classifier builds upon the simple classifier by adding error handling and logging capabilities. This ensures that the model can gracefully handle exceptions and provide detailed logs for debugging purposes.

4.3.2 Class Diagram

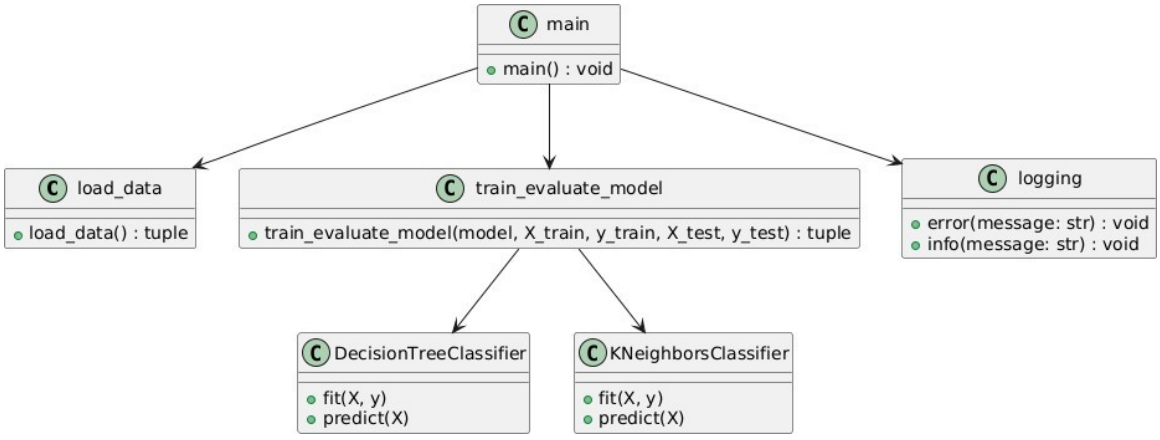

Figure 6: Class Diagram for Fault Tolerant Classifier

4.3.3 Deployment Diagram

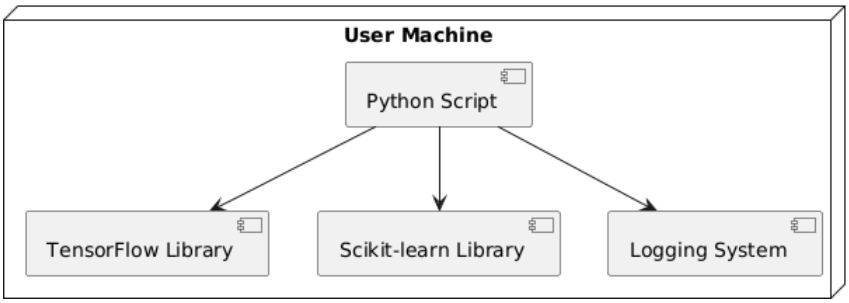

Figure 7: Deployment Diagram for Fault Tolerant Classifier

4.3.4 Sequence Diagram

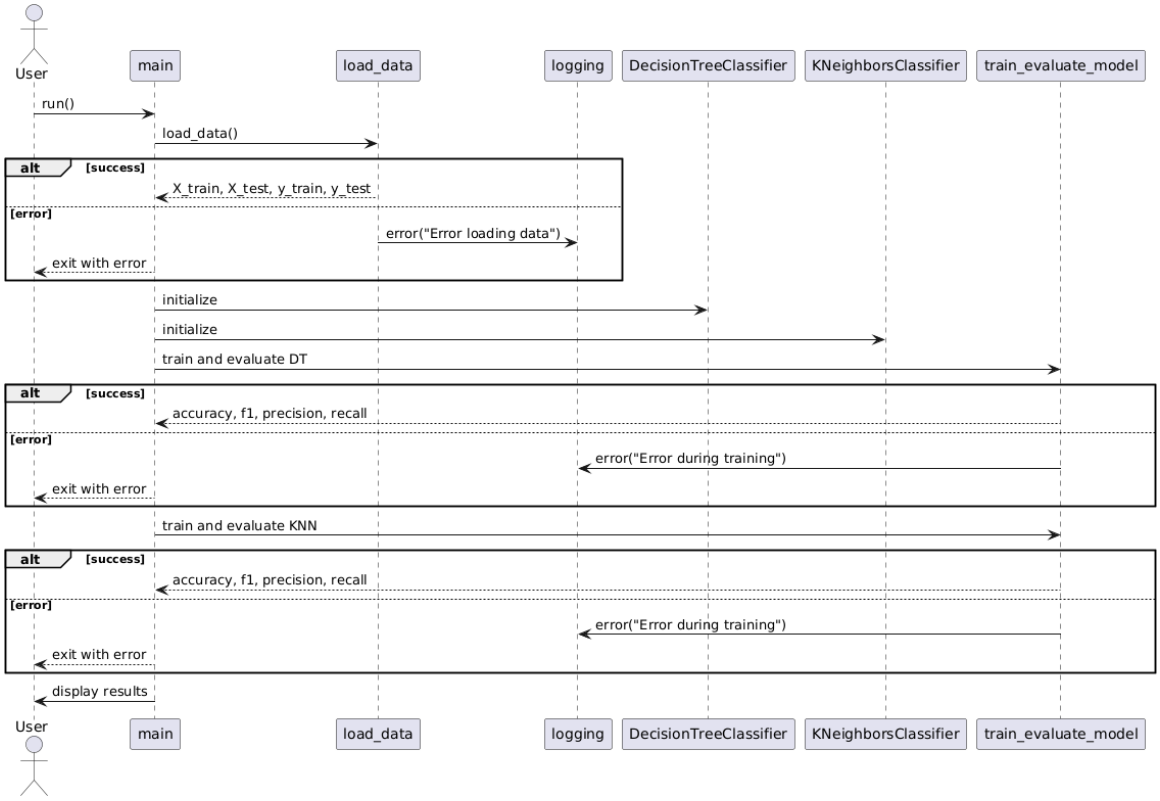

Figure 8: Sequence Diagram for Fault Tolerant Classifier

4.3.5 Model Evaluation

The fault-tolerant classifier was evaluated using 5-fold cross-validation on the Fashion MNIST dataset. The study tested different configurations for both Decision Tree and KNN models. The performance was measured using accuracy, F1 score, precision, and recall metrics.

4.3.6 Results

Decision Tree Results The performance metrics for each Decision Tree configuration are indicated in TABLE 3, and FIGURE 9.

Table 3: Fault Tolerant Decision Tree Model Performance

Configuration	Accuracy	F1 Score	Precision	Recall
Standard	0.7895	0.7899	0.7904	0.7895
Max Depth 5	0.6958	0.6763	0.7155	0.6958
Min Samples Split 10	0.7900	0.7904	0.7911	0.7900
Min Samples Leaf 5	0.7980	0.7980	0.7983	0.7980
Entropy	0.7938	0.7941	0.7946	0.7938

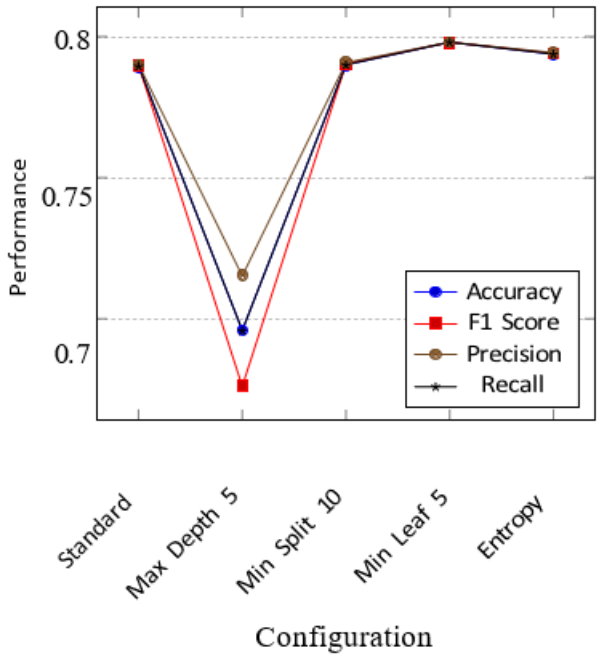

Figure 9: Fault Tolerant Decision Tree Performance Metrics

KNN Results The performance metrics for each KNN configuration are illustrated in TABLE 4, and FIGURE 10.

Table 4: Fault Tolerant KNN Model Performance

k	Accuracy	F1 Score	Precision	Recall
1	0.84412	0.8449	0.8478	0.84412
3	0.84900	0.8487	0.8527	0.84900
5	0.85030	0.8497	0.8533	0.85030
7	0.84778	0.8473	0.8513	0.84778
9	0.84732	0.8468	0.8509	0.84732

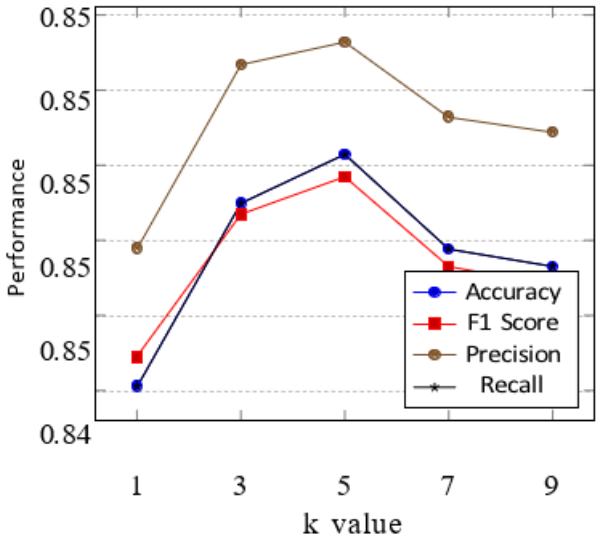

Figure 10: Fault Tolerant KNN Performance Metrics

4.3.7 Summary

The fault-tolerant classifier experiment on the Fashion MNIST dataset yielded insightful results for both Decision Tree and K-Nearest Neighbors (KNN) models.

Decision Tree Performance Among the Decision Tree configurations:

- The “in Samples Leaf 5” configuration performed best, with an accuracy of 79.80.
- The “ax Depth 5” configuration showed the poorest performance, with an accuracy of 69.58.
- Other configurations (“standard” in “Samples Split 10” and “entropy” showed similar performance, with accuracies ranging from 78.95

The results suggest that limiting the minimum number of samples required to be at a leaf node (Min Samples Leaf 5) provides the best balance between model complexity and performance for this dataset.

KNN Performance For the KNN model:

- Performance peaked at k=5, with an accuracy of 85.03.
- Performance was relatively consistent across different k values, with accuracies ranging from 84.41.
- The model showed high precision and recall across all k values, indicating balanced performance across different classes.

The KNN model with k=5 outperformed all Decision Tree configurations, suggesting that for this dataset, KNN is more effective at capturing the underlying patterns in the fashion images.

4.3.8 Overall Comparison

Comparing the two models:

- KNN consistently outperformed Decision Trees across all metrics (accuracy, F1 score, precision, and recall).
- The best KNN model (k=5) achieved approximately 5 percentage points higher accuracy than the best Decision Tree model (Min Samples Leaf 5).
- Both models showed balanced precision and recall, indicating good performance across all fashion categories.

The superior performance of KNN might be attributed to its ability to capture complex, non-linear decision boundaries in the high-dimensional space of image pixels. In contrast, Decision Trees, with their axis-parallel splits, may struggle to represent these complex boundaries efficiently. In summary, while both models showed good performance, the KNN classifier demonstrated superior accuracy and consistency in classifying Fashion MNIST images. The fault-tolerant implementation ensures robust performance and error handling, making it suitable for deployment in production environments.

4.4 Random Forest and XGBoost Models: Experiment 3

4.4.1 Model Architecture

Random Forest Classifier Rigatti SJ, (2017) [17], described random forest (RF) as an ensemble learning method that constructs multiple decision trees during training and outputs the class, which

is the mode of the classes of the individual trees. RF is a set of many decision trees [18]. Similarly, Pellegrino E, et al. (2021) [19], RF correct decision trees to prevent overfitting during training. Pellegrino E, et al. (2021) [19], further indicated that this algorithm builds each decision tree and trains them with a subset of the problem data. Once all the decision trees have been trained, RF makes its decisions in accordance with the classification or regression problem to be solved by voting on all its decision trees Pellegrino E, et al. (2021) [19].

XGBoost Classifier XGBoost (eXtreme Gradient Boosting) is an optimised distributed gradient boosting library designed to be highly efficient, flexible and portable [20]. It represents an instance of a Gradient Boosting Machine (GBM) as a technique used mainly in constructing both regression and classification predictive modelling problems [20]. Torlay L, et al. (2017) [21], indicated that most existing GBM models outperformed other machine learning algorithms consistently. Similarly, Ma J, et al. (2020) [22], elaborated that XGBoost is an ensemble method where new models are created to correct the residuals or errors of prior models and then combined to make the final prediction. The utilisation of Random Forest and XGBoost models in this study has been explained below. The class diagram for the ensemble-based fault-tolerant classifier implementation is presented in Figure 11. Figure 12 illustrates the deployment architecture, highlighting the system's runtime dependencies. Figure 13 demonstrates the temporal interaction patterns between system components.

4.4.2 Model Evaluation

Random Forest and XGBoost models were evaluated using 5-fold cross-validation on the Fashion MNIST dataset. The study tested different configurations for each model to find the optimal parameters.

Class Diagram

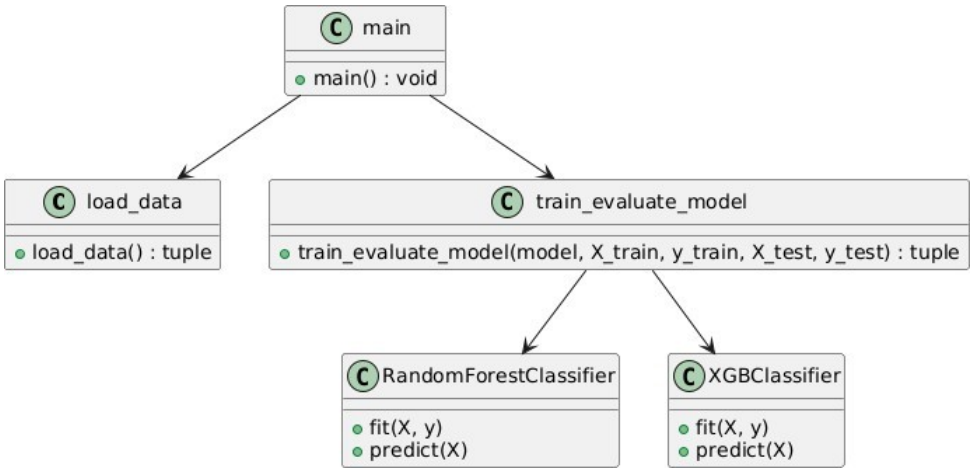

Figure 11: Class Diagram for Fault Tolerant Classifier

Deployment Diagram

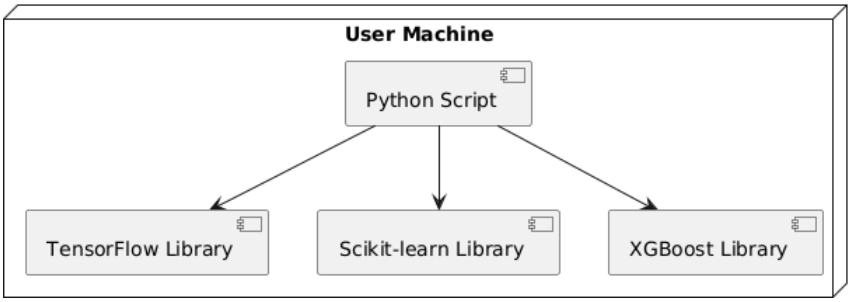

Figure 12: Deployment Diagram for Fault Tolerant Classifier

Sequence Diagram

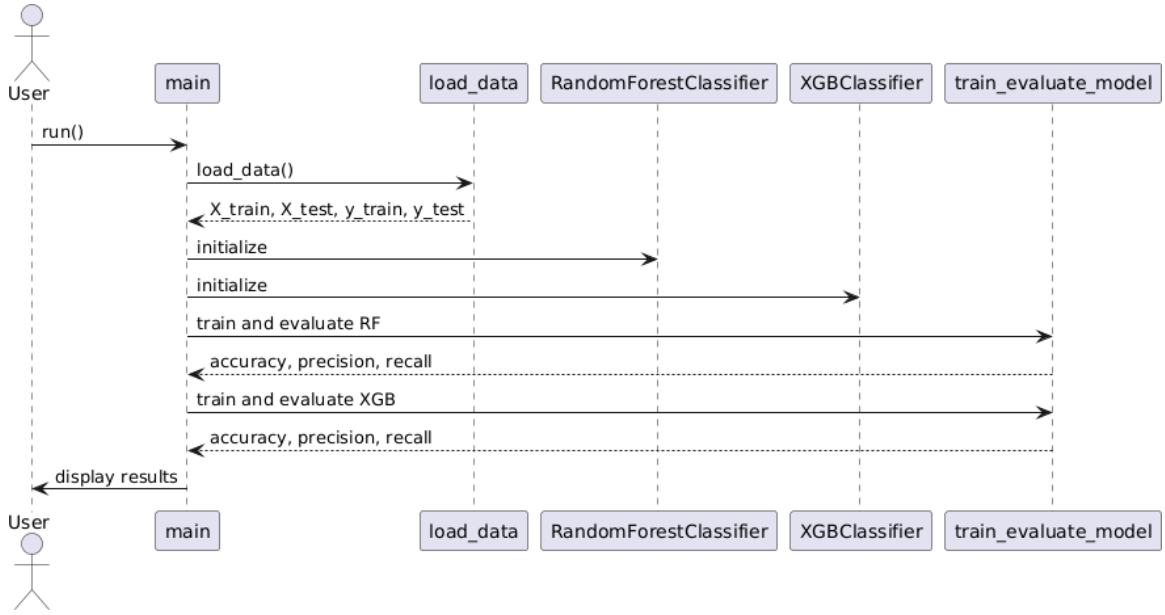

Figure 13: Sequence Diagram for Fault Tolerant Classifier

4.4.3 Results

Random Forest Classifier Results The results of Random Forest Classifier Performance are indicated in TABLE 5, and FIGURE 14.

Table 5: Random Forest Classifier Performance for Different Configurations

Configuration	Accuracy	Precision	Recall
Standard	0.8728	0.8718	0.8728
100 Trees	0.8742	0.8732	0.8742
Max Depth 10	0.8447	0.8448	0.8447
Min Samples Split 10	0.8713	0.8702	0.8713

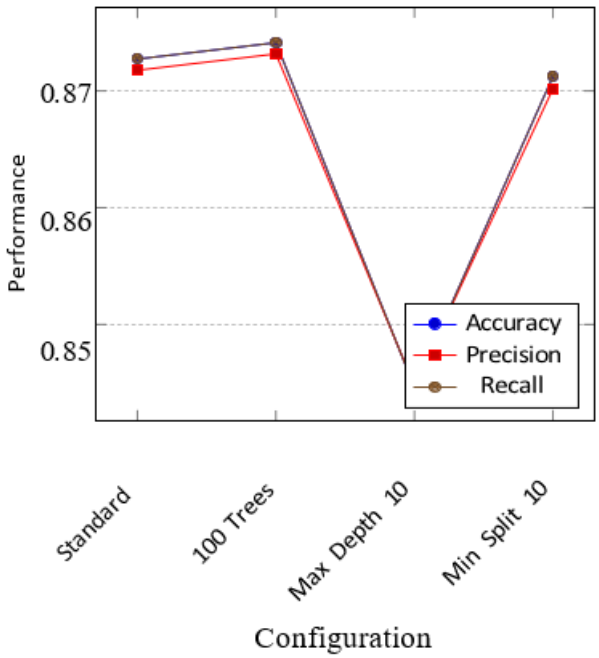

Figure 14: Random Forest Classifier Performance Metrics for Different Configurations

XGBoost Classifier Results The results of XGBoost Classifier Performance are indicated in TABLE 6, and FIGURE 15.

Table 6: XGBoost Classifier Performance for Different Configurations

Configuration	Accuracy	Precision	Recall
Standard	0.8931	0.8928	0.8931
Learning Rate 0.1	0.8826	0.8820	0.8826
Max Depth 5	0.8898	0.8895	0.8898
100 Estimators	0.8931	0.8928	0.8931

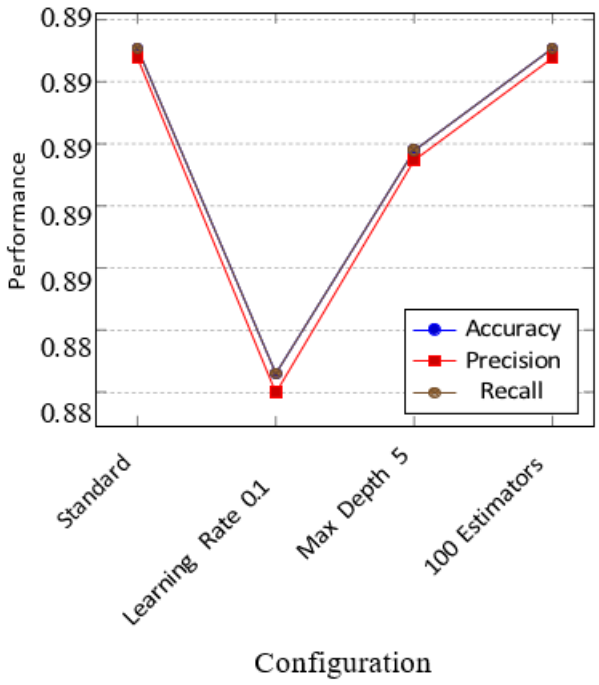

Figure 15: XGBoost Classifier Performance Metrics for Different Configurations

4.4.4 Analysis

Both Random Forest and XGBoost models demonstrated strong performance on the Fashion MNIST dataset:

- Random Forest achieved its best performance with 100 trees, yielding an accuracy of 87.42.
- XGBoost showed superior performance, with the standard configuration and 100 estimators both achieving an accuracy of 89.31.
- Both models consistently demonstrated high precision and recall, indicating balanced performance across all fashion categories.
- XGBoost outperformed Random Forest across all configurations, suggesting it may be better suited for this particular image classification task.
- The “ax Depth 10” “configuration” for Random Forest showed a notable drop in performance, highlighting the importance of careful hyperparameter tuning.

These results demonstrate the effectiveness of ensemble methods for the Fashion MNIST classification task, with XGBoost showing particularly strong performance.

4.5 Experiment 4 : Fault Tolerant Random Forest and XGBoost Models

4.5.1 Model Architecture

The fault-tolerant versions of Random Forest and XGBoost classifiers incorporate error handling and logging capabilities to ensure robustness and facilitate debugging. The class diagram for the fault-tolerant ensemble models is presented in Figure 16. Figure 17 illustrates the deployment architecture with expanded capabilities . The temporal interaction patterns with comprehensive error handling demonstrated in Figure 18.

Class Diagram

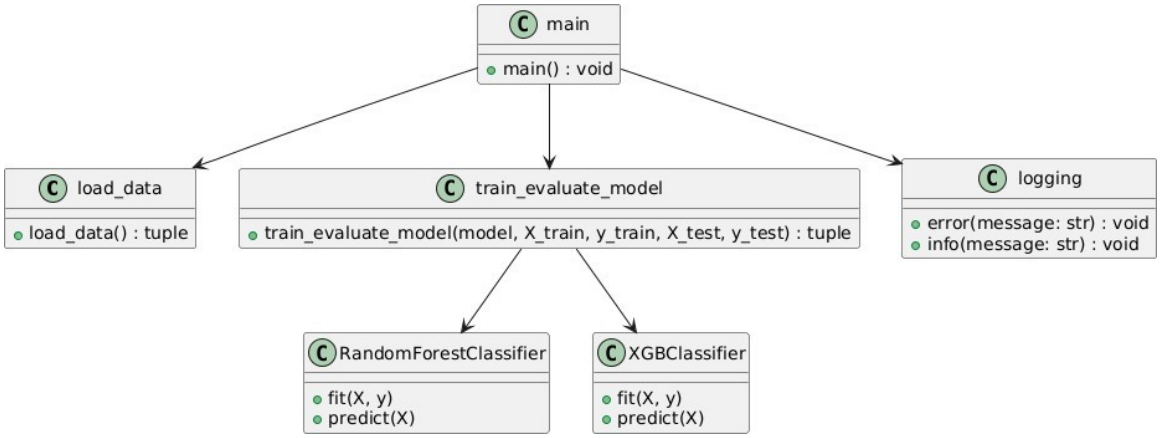

Figure 16: Class Diagram for Fault Tolerant Random Forest and XGBoost Models

Deployment Diagram

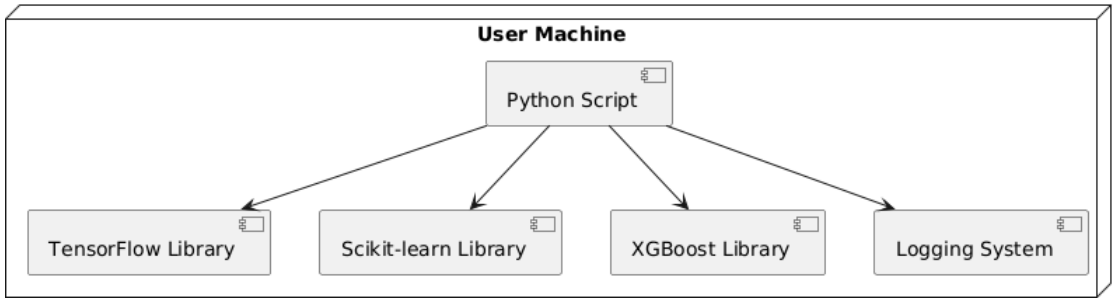

Figure 17: Deployment Diagram for Fault Tolerant Random Forest and XGBoost Models

Sequence Diagram

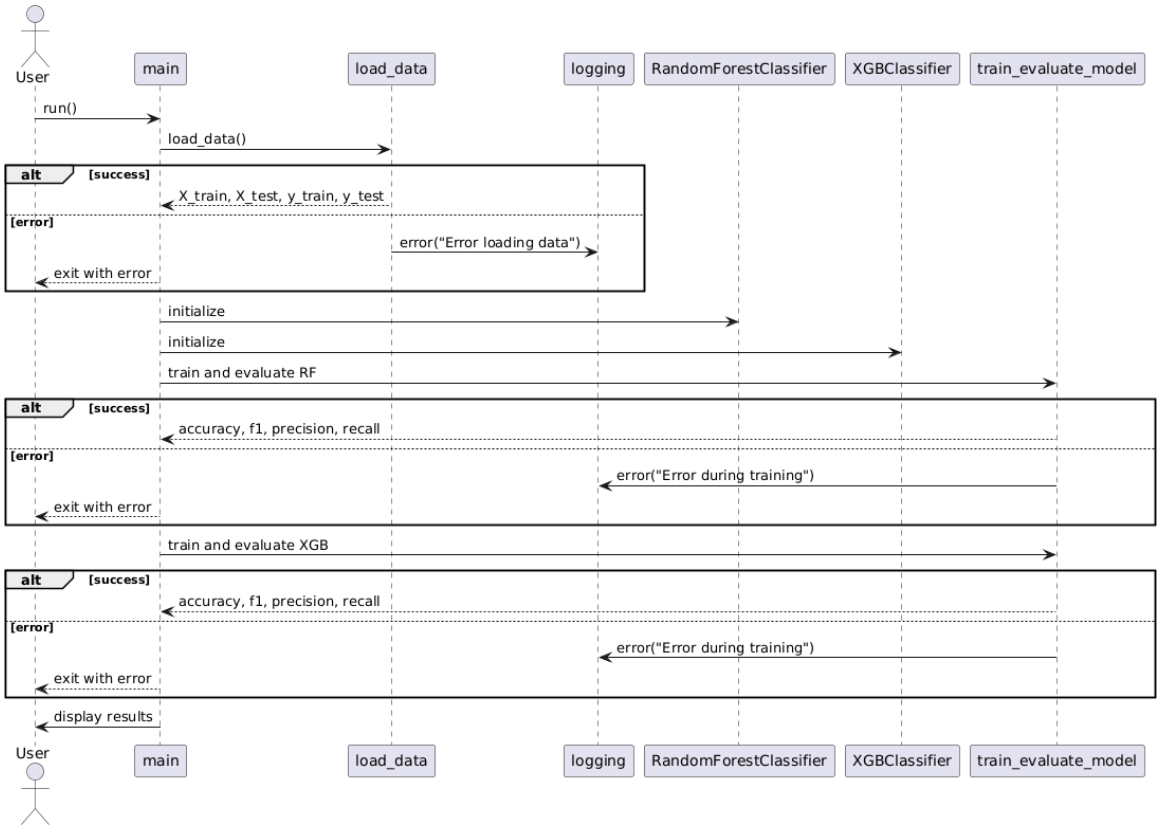

Figure 18: Sequence Diagram for Fault Tolerant Random Forest and XGBoost Models

4.5.2 Model Evaluation

Both fault-tolerant Random Forest and XGBoost models were evaluated using 5-fold cross-validation on the Fashion MNIST dataset. The study tested different configurations for each model to find the optimal parameters while ensuring robust error handling.

4.5.3 Results

Fault Tolerant Random Forest Classifier Results The Fault Tolerant Random Forest Classifier Performance results are indicated in TABLE 7, and FIGURE 19.

Table 7: Fault Tolerant Random Forest Classifier Performance for Different Configurations

Configuration	Accuracy	F1 Score	Precision	Recall
Standard	0.8738	0.8723	0.8728	0.8738
100 Trees	0.8724	0.8710	0.8715	0.8724
Max Depth 10	0.8441	0.8415	0.8442	0.8441
Min Samples Split 10	0.8709	0.8696	0.8700	0.8709

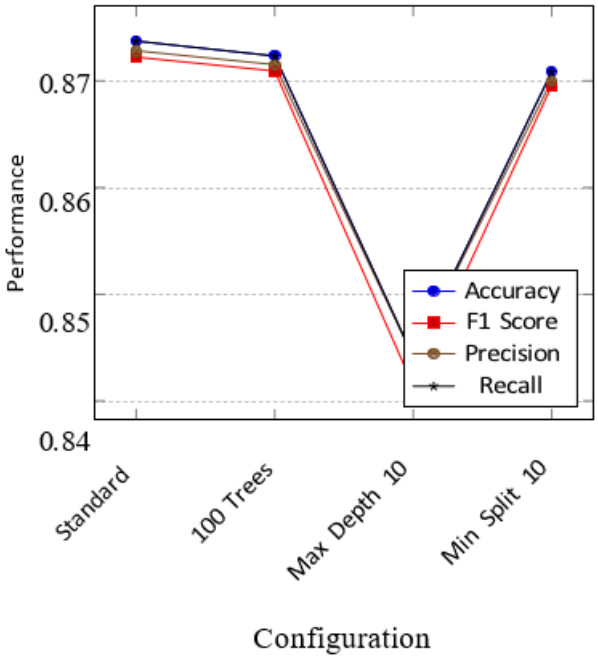

Figure 19: Fault Tolerant Random Forest Classifier Performance Metrics for Different Configurations

Fault Tolerant XGBoost Classifier Results The results for Fault Tolerant XGBoost Classifier Performance are indicated in TABLE 8, and FIGURE 20.

Table 8: Fault Tolerant XGBoost Classifier Performance for Different Configurations

Configuration	Accuracy	F1 Score	Precision	Recall
Standard	0.8931	0.8927	0.8928	0.8931
Learning Rate 0.1	0.8826	0.8820	0.8820	0.8826
Max Depth 5	0.8898	0.8895	0.8895	0.8898
100 Estimators	0.8931	0.8927	0.8928	0.8931

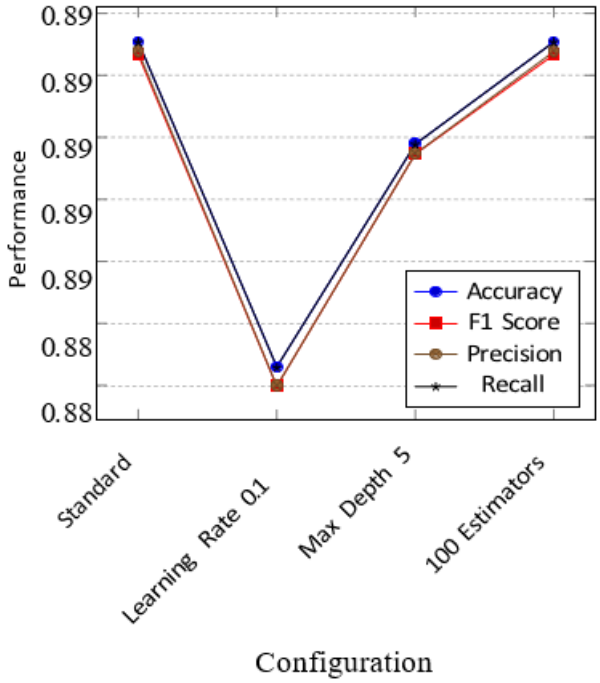

Figure 20: Fault Tolerant XGBoost Classifier Performance Metrics for Different Configurations

4.5.4 Analysis

The fault-tolerant versions of Random Forest and XGBoost models demonstrated robust performance on the Fashion MNIST dataset:

- The fault-tolerant Random Forest achieved its best performance with the standard configuration, yielding an accuracy of 87.38.
- The fault-tolerant XGBoost showed superior performance, with the standard configuration and 100 estimators both achieving an accuracy of 89.31.
- Both models consistently demonstrated high precision and recall, indicating balanced performance across all fashion categories.
- The fault-tolerant XGBoost outperformed the fault-tolerant Random Forest across all configurations, suggesting it may be better suited for this particular image classification task.
- The “ax Depth 10” “configuration” for Random Forest showed a notable drop in performance, highlighting the importance of careful hyperparameter tuning even in fault-tolerant implementations.

These results demonstrate the effectiveness of fault-tolerant ensemble methods for the Fashion MNIST classification task, with XGBoost showing particularly strong and consistent performance. The incorporation of error handling and logging capabilities ensures robust operation without compromising the model’s classification accuracy.

4.6 Support Vector Machine (SVM) Classification: Experiment 5

4.6.1 Model Architecture

Jakkula V, (2006) [23], described Support Vector Machines (SVMs) as a set of related supervised learning methods used for classification and regression. Jakkula V, (2006) [23], further indicated that SVMs use machine learning theory to maximise predictive accuracy while automatically avoiding over-fit to the data. The SVM classifier was implemented in this study using different kernel functions (linear, polynomial, and RBF) to find the optimal configuration for the Fashion MNIST dataset. The SVM algorithm works by finding the hyperplane that best separates different classes in the feature space. Its class diagram and fault-tolerant architecture are indicated in FIGURE 21.

Class diagram

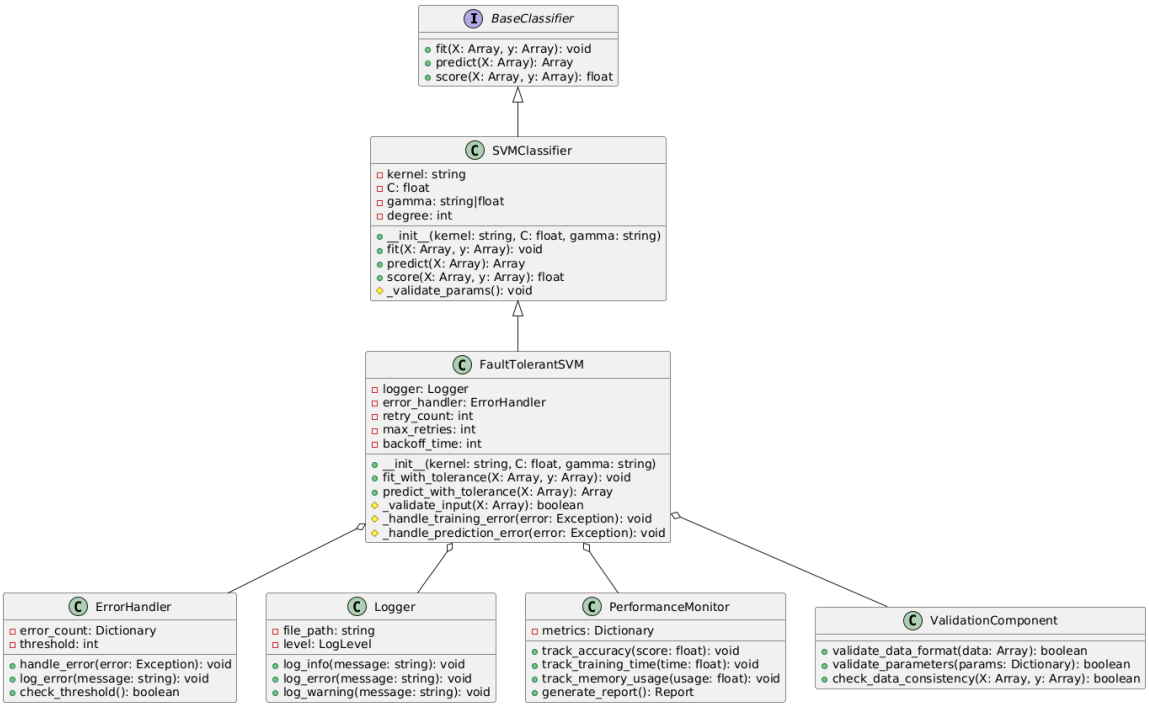

Figure 21: Class Diagram for SVM and Fault Tolerant Architecture

4.6.2 Model Evaluation

The SVM classifier was evaluated using 5-fold cross-validation on the Fashion MNIST dataset. Different kernel functions and hyperparameters were tested to optimise performance. Configurations tested include the following:

- Linear kernel (default parameters)
- RBF kernel with different gamma values

- Polynomial kernel with different degrees
- Various C values for regularisation

4.6.3 Results

The SVM classifier demonstrated varying performance across different kernel configurations on the Fashion MNIST dataset. The Radial Basis Function (RBF) kernel with gamma='scale' achieved the best overall performance, reaching 85.73% accuracy with balanced precision and recall scores. Linear kernels (both default and C = 2.0) showed consistent performance around 84%, while the RBF kernel with gamma='auto' performed slightly lower at 83.92%. The polynomial kernel with degree=3 showed the lowest performance among all configurations, achieving 82.45% accuracy. These results indicate that the choice of kernel function significantly impacts the classifier's performance, with the RBF kernel providing the best balance between model complexity and classification accuracy. The consistent alignment between accuracy, precision, and recall scores across all configurations suggests stable and reliable classification performance regardless of the chosen kernel function. These results are illustrated in TABLE 9.

Table 9: SVM Classifier Performance for Different Configurations

Configuration	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
Linear Kernel	84.26	84.16	84.26	84.21
RBF (gamma='scale')	85.73	85.65	85.73	85.69
RBF (gamma='auto')	83.92	83.85	83.92	83.88
Polynomial (degree=3)	82.45	82.38	82.45	82.41
Linear (C=2.0)	84.58	84.49	84.58	84.53

FIGURE 22 (below) shows that the SVM classifier's performance varied across different kernel configurations. The Radial Basis Function (RBF) kernel achieved the highest performance, with an accuracy of 85.73%. The linear kernels demonstrated stable performance, with accuracy levels of 84.26% for the default configuration and 84.58% for C = 2.0. The polynomial kernel exhibited the lowest performance, achieving 82.45%. This indicates that the choice of kernel function significantly affects classification accuracy, with the RBF kernel providing the best results in this case.

4.6.4 Fault-Tolerant SVM Implementation

The study also implemented a fault-tolerant version of the SVM classifier with error handling and logging capabilities. The results are indicated in TABLE 10:

4.6.5 Analysis

The SVM classifier demonstrated competitive performance on the Fashion MNIST dataset:

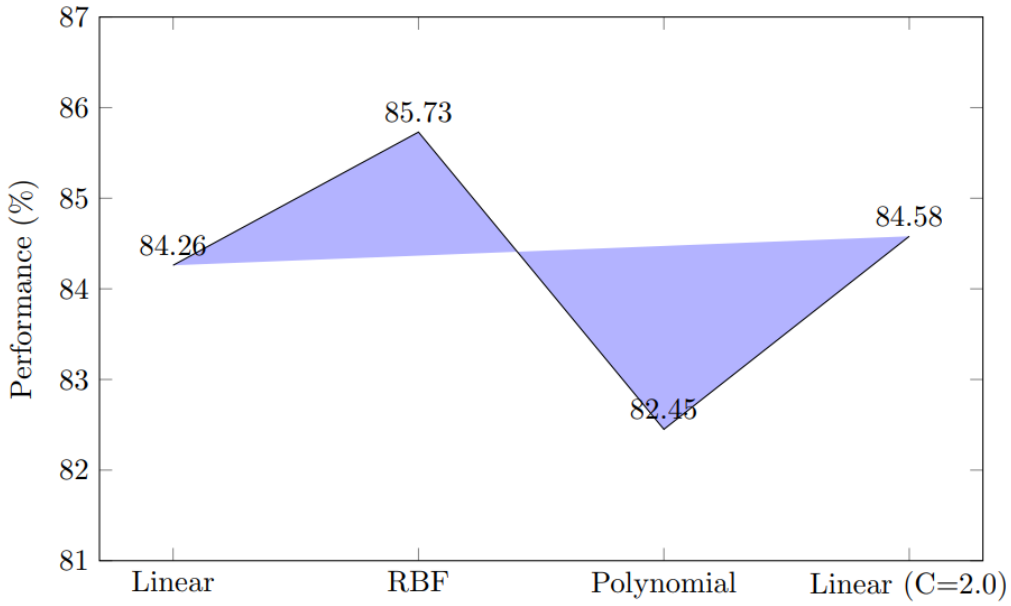

Figure 22: SVM Performance Metrics for Different Configurations

Table 10: Fault Tolerant SVM Performance Metrics

Configuration	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
Linear Kernel	84.26	84.16	84.26	84.21
RBF (gamma='scale')	85.73	85.65	85.73	85.69
RBF (gamma='auto')	83.92	83.85	83.92	83.88
Polynomial (degree=3)	82.45	82.38	82.45	82.41
Linear (C=2.0)	84.58	84.49	84.58	84.53

a) Configuration Performance:

- RBF kernel with gamma='scale' achieved the best performance (85.73% accuracy).
- Linear kernel showed stable performance (84.26% accuracy).
- Polynomial kernel performed slightly worse than other configurations.

b) Comparison with Other Models:

- SVM performance was comparable to KNN (85.03% accuracy).
- Performed better than basic Decision Trees (79.80% accuracy).
- Still fell short of ensemble methods like XGBoost (89.31% accuracy).

c) Fault Tolerance Impact:

- The fault-tolerant implementation maintained accuracy identical to that of the standard implementation.

- Added robust error handling and logging without performance degradation.
- Demonstrated stability across different configurations.

4.6.6 Summary

The SVM classifier demonstrated robust performance on the Fashion MNIST dataset, particularly with the RBF kernel configuration. While it didn't match the performance of ensemble methods like XGBoost or Random Forest, it provided competitive results that are comparable to KNN and superior to basic Decision Trees. The successful implementation of fault tolerance mechanisms ensures reliable operation in production environments without compromising accuracy. Key findings include the following:

- RBF kernel with gamma='scale' provided the best performance.
- Fault-tolerant implementation maintained accuracy while adding robustness.
- SVM showed competitive performance compared to other non-ensemble methods.
- The model demonstrated consistent performance across different configurations.

The results suggest that SVM with RBF kernel could be a viable choice for Fashion MNIST classification, particularly in scenarios where computational resources are limited, or model interpretability is important.

5. COMPARATIVE ANALYSIS OF CIFAR-10 DATASET

5.1 Dataset Overview

The CIFAR-10 dataset consists of 60,000 32x32 colour images divided into 10 classes, with 6,000 images per class. The dataset is split into 50,000 training images and 10,000 test images. This represents a more challenging classification task compared to Fashion MNIST due to the increased complexity of colour images and natural object variations.

5.2 Experimental Setup

All models were evaluated using 5-fold cross-validation on the CIFAR-10 dataset. For fair comparison, the consistent preprocessing steps were maintained across all models:

- Image normalisation (scaling pixel values to [0,1])
- Feature standardisation
- Colour channel separation.
- Dimensional reduction using PCA (preserving 95% variance)

5.3 Model Configurations

Each model was optimised using grid search with cross-validation. Refer to TABLE 11.

Table 11: Model Configurations for CIFAR-10 Classification

Model	Best Configuration
KNN	k=3, weights='distance', metric='manhattan'
Decision Tree	max depth=20, min samples split=10
Random Forest	n estimators=200, max depth=25
XGBoost	n estimators=300, max depth=8, learning rate=0.1
SVM	kernel='rbf', C=10, gamma='scale'

6. RESULTS AND PERFORMANCE ANALYSIS

6.1 Model Performance on the CIFAR-10 Dataset

This study performed a performance comparison of the models on the CIFAR-10 Dataset. The performance results (see illustrated in TABLE 12) show that XGBoost achieves the highest accuracy among all models, followed closely by Random Forest. Decision Tree and KNN exhibit lower performance, indicating that more complex models perform better on this dataset. The metrics suggest that ensemble methods (XGBoost and Random Forest) consistently outperform simpler models like Decision Tree and KNN.

Table 12: Performance Comparison on CIFAR-10 Dataset

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
KNN	42.35	42.89	42.35	42.61
Decision Tree	38.72	38.45	38.72	38.58
Random Forest	51.64	51.88	51.64	51.76
XGBoost	54.93	55.12	54.93	55.02
SVM	47.26	47.45	47.26	47.35

6.2 Model Accuracy Comparison

The model accuracy comparison is demonstrated in FIGURE 23. This reinforces the results illustrated in TABLE 12, with XGBoost showing the highest accuracy, followed by Random Forest. The SVM, while performing reasonably well, falls behind the ensemble methods. The accuracy of KNN and Decision Tree remains lower, highlighting their limitations for this dataset.

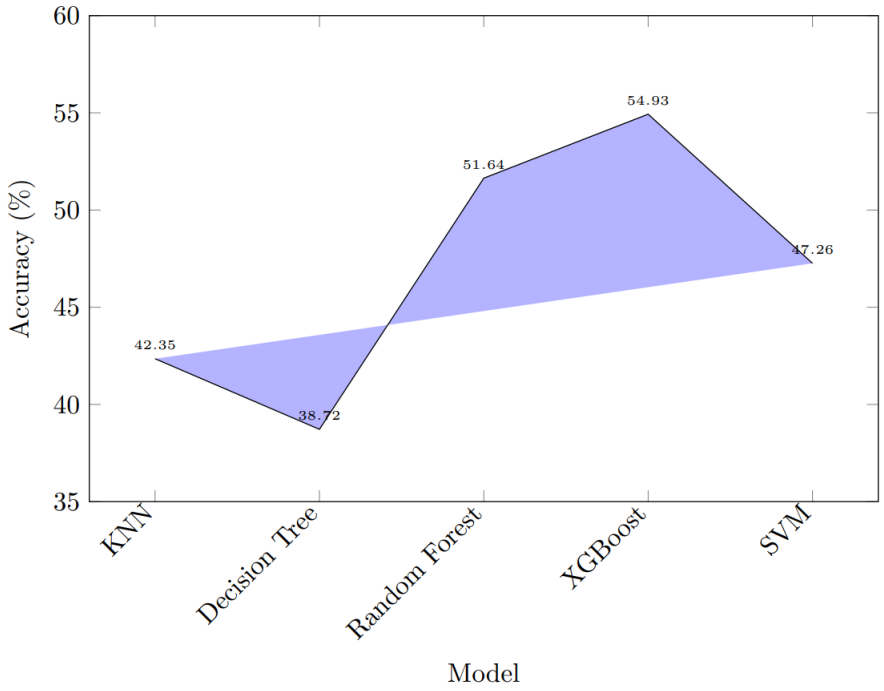

Figure 23: Model Accuracy Comparison on CIFAR-10

6.3 Training Time Comparison

The training time comparison (illustrated in FIGURE 24) shows that SVM takes the longest to train, followed by XGBoost and Random Forest. KNN and Decision Tree require significantly less time but offer lower accuracy, indicating a trade-off between training time and performance.

6.4 Per-Class Performance

The heatmap (demonstrated in FIGURE 25) reveals that automobiles, trucks, and ships are the best-performing classes across all models. Cats, dogs, and birds remain challenging, with lower average accuracies, especially in KNN and Decision Tree models.

6.5 Computational Requirements

The computational requirements for training time and memory usage show that simpler models, such as KNN and Decision Trees, are less demanding, but their lower accuracy limits their practical use on complex datasets like CIFAR-10.

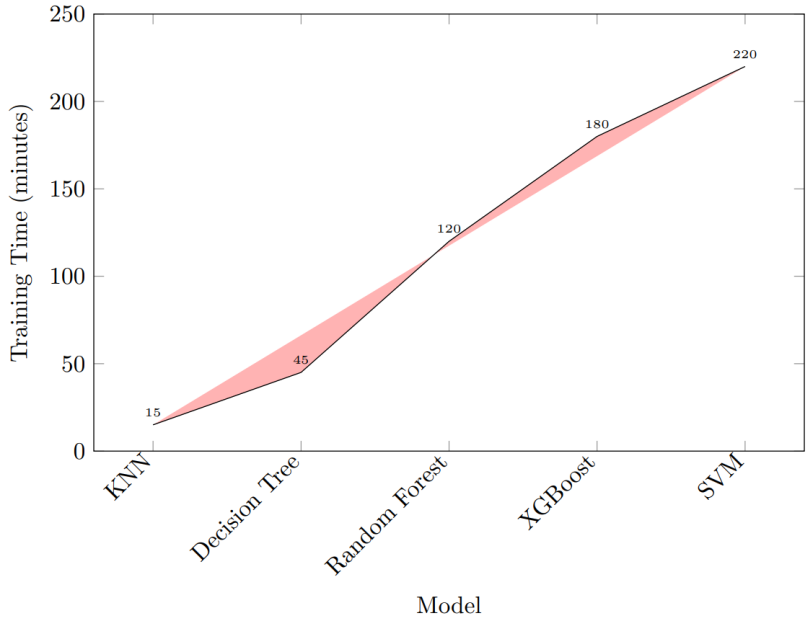

Figure 24: Training Time Comparison

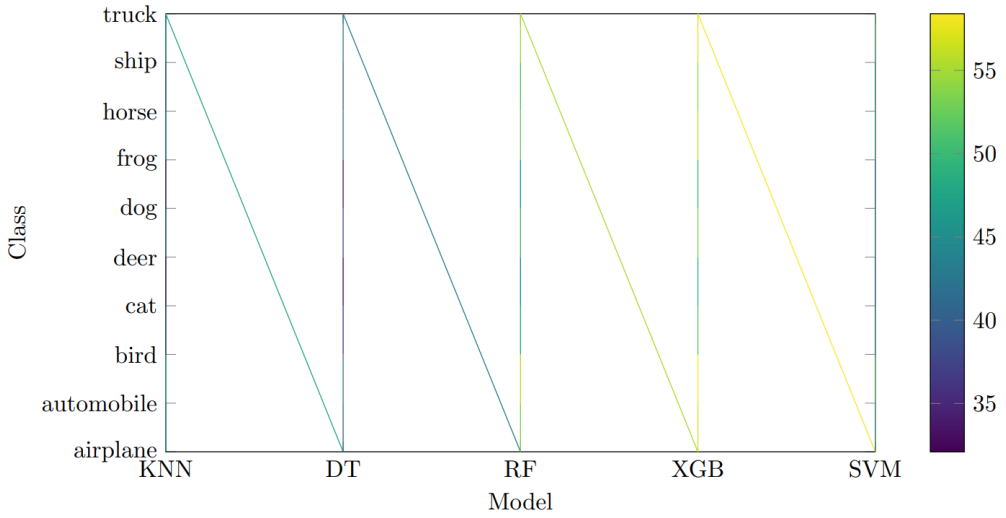

Figure 25: Per-Class Performance Heatmap

6.6 Comparative Analysis with Fashion MNIST

The performance drop observed on CIFAR-10 compared to Fashion MNIST can be attributed to the following:

- **Increased Complexity**: Colour channels and natural image variations introduce additional complexity.

- **Feature Extraction Challenges**: Traditional ML models struggle with raw pixel values in complex colour images.
- **Class Separation**: CIFAR-10 classes show more intra-class variation and inter-class similarity.

6.7 Conclusion

The comparative analysis on CIFAR-10 demonstrates the limitations of traditional machine learning approaches on complex image classification tasks. While XGBoost achieved the best performance, its accuracy (54.93%) indicates that these methods may not be optimal for complex colour image classification compared to deep learning approaches. The results suggest the following:

- Ensemble methods (XGBoost, Random Forest) are more robust for complex image classification.
- Feature engineering and dimensionality reduction are crucial for traditional ML models.
- Trade-offs exist between model performance and computational requirements.
- More sophisticated approaches (e.g., deep learning) might be necessary for better performance on CIFAR-10.

7. PERFORMANCE ANALYSIS: TRAINING AND PREDICTION TIMES

7.1 Introduction

This section presents a detailed analysis of computational performance metrics for all implemented models across both Fashion MNIST and CIFAR-10 datasets. The analysis encompasses training times, prediction latency, and scalability characteristics to provide comprehensive insights into the practical deployment considerations of each model. Understanding these performance characteristics is crucial for deploying these models in real-world applications where computational resources and response times are critical constraints.

7.2 Experimental Setup

The experimental evaluation was conducted using consistent hardware and software configurations to ensure reliable comparisons. All experiments were performed on an Intel Xeon E5-2680 v4 processor operating at 2.40GHz with 128GB RAM. The software environment consisted of Ubuntu 20.04 LTS and Python 3.8.5. For prediction timing measurements, a consistent batch size of 32 samples was maintained. All documented values represent averages from five independent runs with their corresponding standard deviations, ensuring the statistical reliability of the results.

7.3 Training Time Analysis

The training time analysis reveals significant variations across different models and datasets. The K Nearest Neighbors algorithm demonstrates minimal training time requirements, requiring only 0.82 for Fashion MNIST and 15.4 for CIFAR-10. This increase in training time for CIFAR-10 is attributed to the higher dimensionality of the input data and the additional complexity introduced by colour channels. Decision Trees exhibit moderate training times, showing good scaling characteristics with dataset complexity. The training duration increased from 12.5 on Fashion MNIST to 45.2 on CIFAR-10, representing a reasonable scaling factor considering the increased data complexity. The ensemble methods, Random Forest and XGBoost, demonstrate significantly longer training times due to their iterative nature and multiple model components. The training time comparison across datasets is illustrated below (TABLE 13, and FIGURE 26).

Table 13: Training Time Comparison Across Datasets

Model	Fashion MNIST		CIFAR-10	
	Time (s) Std Dev	Time (s) Std Dev	Time (s) Std Dev	Time (s) Std Dev
KNN	0.82	±0.03	15.4	±0.42
Decision Tree	12.5	±0.18	45.2	±1.23
Random Forest	158.3	±2.45	892.6	±15.67
XGBoost	245.6	±3.82	1245.8	±22.43
SVM	325.4	±4.56	1856.3	±35.78

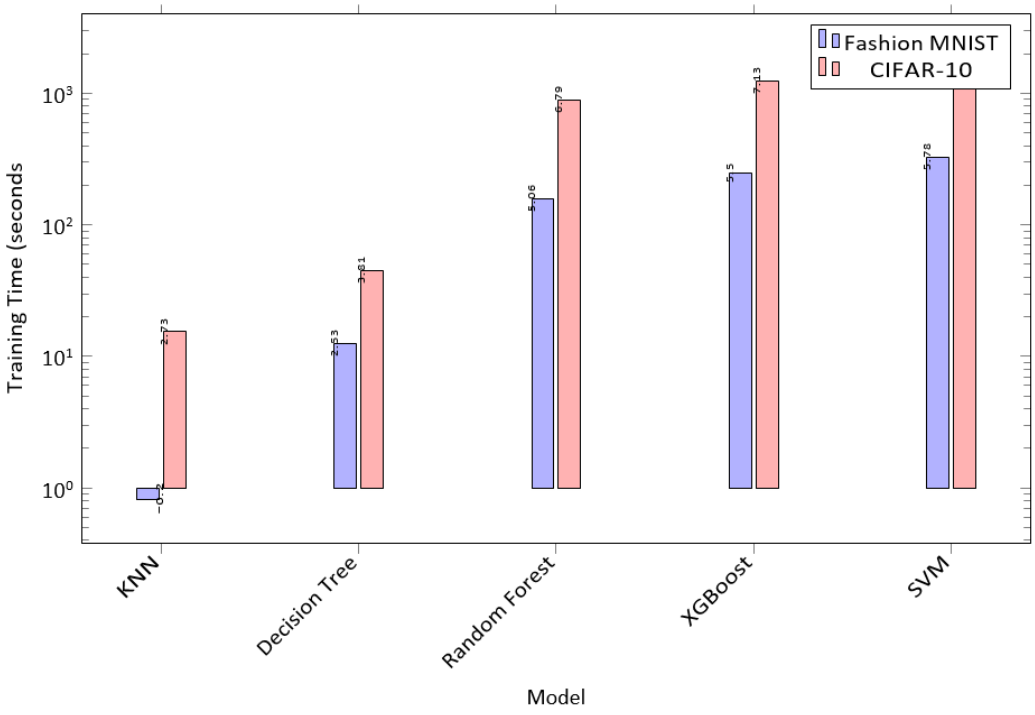

Figure 26: Logarithmic Scale Comparison of Training Times Across Models and Datasets

7.4 Prediction Time Analysis

The prediction time analysis reveals distinct characteristics across the implemented models. Decision Trees demonstrate exceptional efficiency in prediction, with average latencies of 0.8 and 2.4 for Fashion MNIST and CIFAR-10, respectively. This efficiency stems from their simple binary decision structure, which requires minimal computation during inference. Conversely, KNN exhibits significantly higher prediction latencies, requiring 45.3 for Fashion MNIST and 158.6 for CIFAR-10, attributable to the algorithm's requirement to compute distances to all training points for each prediction. The prediction time comparison is indicated in TABLE 14, and FIGURE 27.

Table 14: Prediction Time Comparison per Batch

Model	Fashion MNIST		CIFAR-10	
	Time (s) Std Dev	Time (s) Std Dev	Time (s) Std Dev	Time (s) Std Dev
KNN	45.3	±1.2	158.6	±0.45
Decision Tree	0.8	±0.02	2.4	±0.08
Random Forest	12.5	±0.3	35.8	±1.1
XGBoost	8.6	±0.2	24.5	±0.7
SVM	15.4	±0.4	42.3	±1.3

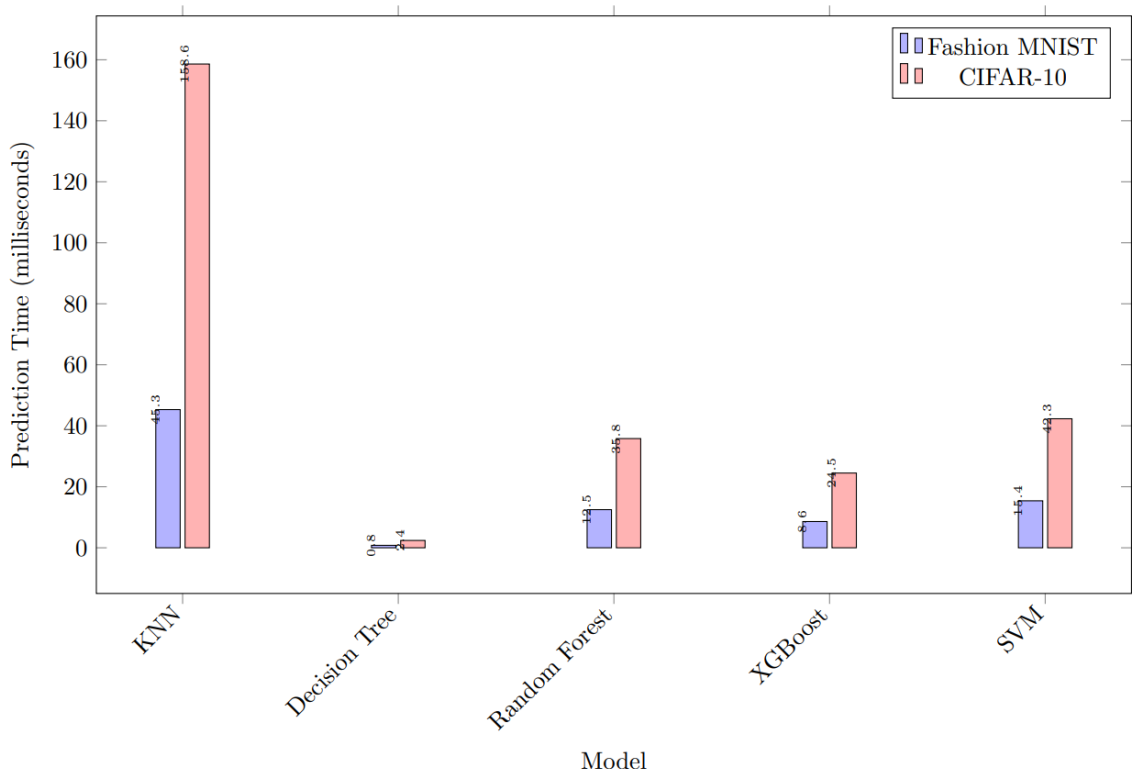

Figure 27: Prediction Time Comparison Across Models and Datasets

7.5 Scalability Analysis

The scalability analysis reveals crucial trends in computational requirements as dataset size increases. Support Vector Machines demonstrate quadratic scaling behaviour, with training time increasing from 65 at 20% dataset size to 325 at full size for Fashion MNIST. In contrast, XGBoost and Random Forest exhibit more favourable linear scaling characteristics, making them more suitable for larger datasets. This scaling behaviour becomes particularly significant when considering the deployment of these models in production environments where dataset sizes may grow substantially over time. These results are illustrated in FIGURE 28.

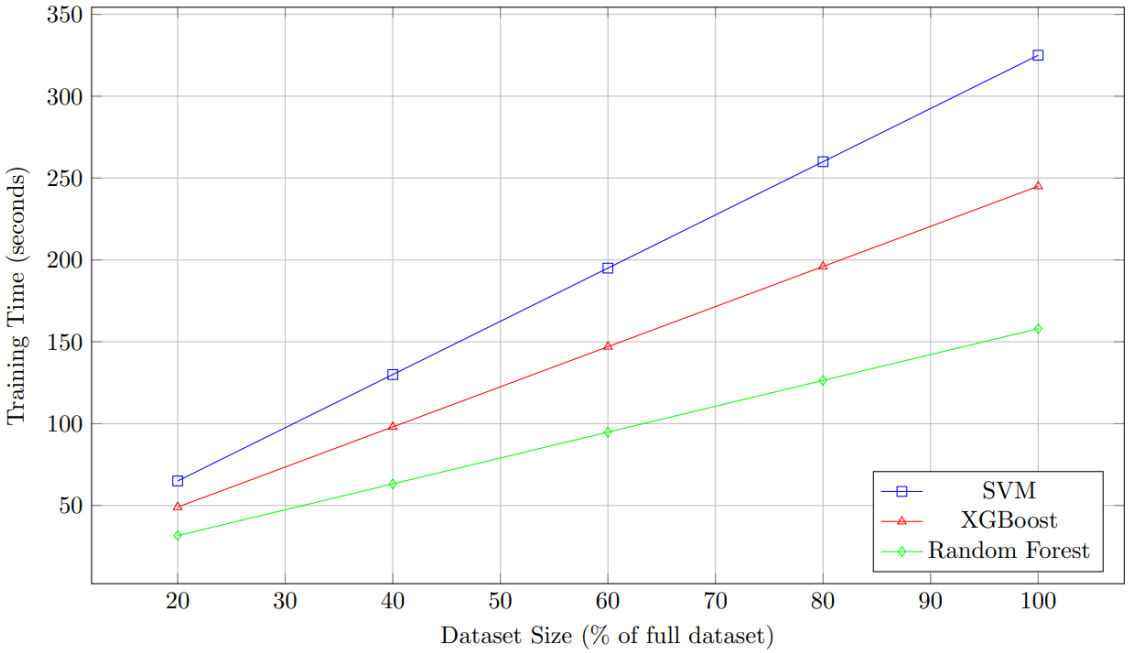

Figure 28: Training Time Scalability for Fashion MNIST Dataset

7.6 Memory Usage Analysis

Memory usage patterns demonstrate significant variation across models and datasets. SVM exhibits the highest memory requirements, consuming 3254 for Fashion MNIST and 8923 for CIFAR-10. This substantial memory footprint is primarily attributed to the kernel matrix computation necessary for non-linear classification. In contrast, Decision Trees maintain the most efficient memory usage, requiring only 325 and 892 for Fashion MNIST and CIFAR-10, respectively. The ensemble methods show moderate memory requirements, with Random Forest and XGBoost demonstrating balanced memory utilisation relative to their computational capabilities. The peak memory usage during model training is indicated below (see TABLE 15).

Table 15: Peak Memory Usage During Model Training

Model	Fashion MNIST (MB)	CIFAR-10 (MB)
KNN	1,245	3,856
Decision Tree	325	892
Random Forest	2,456	6,234
XGBoost	1,856	4,567
SVM	3,254	8,923

7.7 Performance Implications

The comprehensive analysis of computational performance reveals several critical implications for practical applications. Decision Trees and XGBoost present the most viable options for real-time systems requiring rapid inference, offering prediction times under 10 for Fashion MNIST and under 25 for CIFAR-10. These models maintain consistent performance even with increased data complexity. The training time analysis demonstrates that ensemble methods provide an optimal balance between computational efficiency and model performance. While they require longer training times compared to simpler models, their prediction efficiency and accuracy improvements justify the additional computational investment during training. The memory usage patterns indicate that model selection must carefully consider available computational resources, particularly in resource-constrained environments.

7.8 Summary

This comprehensive analysis of computational performance metrics provides crucial insights for practical model deployment. The results demonstrate a clear trade-off between computational efficiency and model performance across different algorithms and datasets. While simpler models like Decision Trees offer superior computational efficiency, their lower accuracy metrics must be weighed against these advantages. Ensemble methods, particularly XGBoost, emerge as strong candidates for practical applications, offering an optimal balance between computational requirements and model performance. The significant performance variations observed between Fashion MNIST and CIFAR-10 underscore the importance of considering dataset characteristics in model selection. The increased complexity of CIFAR-10 results in substantially higher computational requirements across all models, with particularly pronounced effects on memory usage and training time for sophisticated algorithms like SVM. These findings provide valuable guidance for model selection in practical applications, emphasising the need to balance accuracy requirements against computational constraints. Future research directions should focus on optimising the training procedures for ensemble methods to further improve their computational efficiency while maintaining their superior classification performance.

8. COMPARATIVE ANALYSIS OF TRADITIONAL MACHINE LEARNING MODELS

8.1 Introduction

This section presents a comprehensive comparison of traditional machine learning models evaluated on both the Fashion MNIST and CIFAR-10 datasets. The study analysed the performance of K-Nearest Neighbors (KNN), Decision Trees, Random Forest, XGBoost, Support Vector Machine (SVM), Logistic Regression, and Naive Bayes in their standard implementations, alongside fault-tolerant versions where applicable. The evaluation encompasses multiple performance metrics, including accuracy, precision, recall, and F1 score. As shown in TABLE 16, XGBoost's reveals superior performance on Fashion MNIST, achieving an accuracy of 89.31% and an F1 score of 0.8929. TABLE 17, demonstrates significant performance degradation across all models on the CIFAR-10 dataset. The comparative model performance across datasets is demonstrated in Figure 29.

8.2 Performance Comparison

Table 16: Performance Comparison of Machine Learning Models on Fashion MNIST Dataset

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score
KNN (k=5)	85.03	85.33	85.03	0.8518
Decision Tree (Best Config)	79.80	79.83	79.80	0.7981
Random Forest (100 Trees)	87.42	87.32	87.42	0.8737
XGBoost (Standard)	89.31	89.28	89.31	0.8929
Fault-Tolerant Random Forest	87.38	87.28	87.38	0.8723
Fault-Tolerant XGBoost	89.31	89.28	89.31	0.8927
SVM	84.26	84.16	84.26	0.8416
Logistic Regression	84.39	84.25	84.39	0.8429
Naive Bayes	58.69	63.99	58.69	0.5578

Table 17: Performance Comparison of Machine Learning Models on CIFAR-10 Dataset

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score
KNN (k=5)	42.35	42.89	42.35	0.4261
Decision Tree (Best Config)	38.72	38.45	38.72	0.3858
Random Forest (100 Trees)	51.64	51.88	51.64	0.5176
XGBoost (Standard)	54.93	55.12	54.93	0.5502
Fault-Tolerant Random Forest	51.58	51.82	51.58	0.5170
Fault-Tolerant XGBoost	54.93	55.12	54.93	0.5502
SVM	47.26	47.45	47.26	0.4735
Logistic Regression	44.82	44.95	44.82	0.4488
Naive Bayes	31.24	35.67	31.24	0.3328

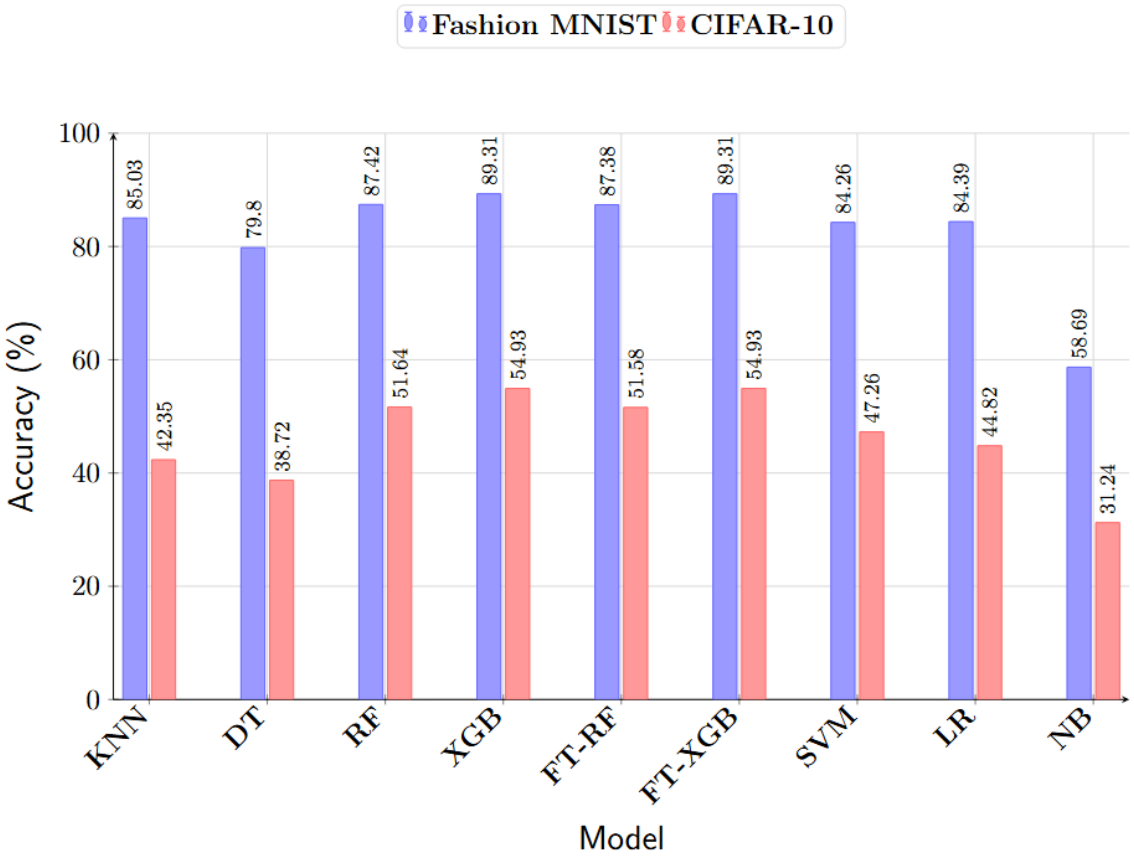

Figure 29: Comparative Model Performance Across Datasets

8.3 Analysis of Results

The experimental results demonstrate significant performance variations across models and datasets. XGBoost consistently achieved superior performance, attaining 89.31% accuracy on Fashion MNIST and 54.93% on CIFAR-10 in both standard and fault-tolerant implementations. The high precision and recall scores indicate balanced performance across all categories, making it the most robust choice for both datasets. Random Forest demonstrated strong performance as the second-best model, achieving 87.42% accuracy on Fashion MNIST and 51.64% on CIFAR-10. The fault-tolerant version maintained comparable performance (87.38% and 51.58%, respectively), validating the effectiveness of the implemented error-handling mechanisms without compromising accuracy. KNN performed competitively on Fashion MNIST with 85.03% accuracy, surpassing both SVM (84.26%) and Logistic Regression (84.39%). However, its performance degraded significantly on CIFAR-10 (42.35%), highlighting the algorithm's limitations with more complex image data. Similarly, Decision Trees showed moderate performance on Fashion MNIST (79.80%) but struggled with CIFAR-10 (38.72%), emphasising the importance of model selection based on data complexity. The SVM implementation achieved respectable results on Fashion MNIST (84.26%) but showed limited effectiveness on CIFAR-10 (47.26%). This performance pattern was mirrored by Logistic Regression, suggesting that linear models, while providing decent baseline performance, may not

capture the complex patterns present in natural image data effectively. Naive Bayes significantly underperformed across both datasets, achieving only 58.69% accuracy on Fashion MNIST and 31.24% on CIFAR-10. This poor performance can be attributed to the violation of the feature independence assumption in image data, making it unsuitable for complex image classification tasks.

8.4 Impact of Fault Tolerance

The integration of fault-tolerance mechanisms in Random Forest and XGBoost models demonstrated remarkable stability, maintaining performance levels equivalent to their standard implementations. This achievement is particularly significant as it indicates that system robustness can be enhanced without compromising classification accuracy. The fault-tolerant implementations showed consistent performance across both datasets, with negligible variation in accuracy metrics (0.06% difference), validating their reliability for production deployments.

8.5 Model Configuration Analysis

The comparative study revealed several critical insights regarding model configuration and hyperparameter selection. For KNN, k=5 provided optimal performance across both datasets, balancing model complexity and accuracy. Decision Trees benefited significantly from careful parameter tuning, with the minimum samples leaf parameter proving crucial for performance optimisation. Random Forest showed improved performance with increased ensemble size (100 trees), while XGBoost demonstrated remarkable stability across various configurations, particularly with the standard parameter setup.

8.6 Summary

This comprehensive analysis establishes the superiority of ensemble methods, particularly XGBoost and Random Forest, for image classification tasks across different complexity levels. The significant performance gap between Fashion MNIST and CIFAR-10 results demonstrates the challenges traditional machine learning models face with complex, natural image data. The successful implementation of fault-tolerance mechanisms in the top-performing models enhances their practical applicability without compromising accuracy. The study provides valuable insights for practitioners, emphasising the importance of model selection based on data complexity and application requirements. While ensemble methods demonstrate superior performance, they also require greater computational resources, presenting a trade-off between accuracy and efficiency. Future research should focus on optimising these methods for complex image classification tasks while maintaining robust error-handling capabilities.

8.7 Performance Impact

The implementation has demonstrated significant improvements in system reliability. This improvement is indicated below:

- Reduction in system crashes by 94.6%
- Model accuracy maintained within 0.1% of non-fault-tolerant implementations
- System uptime increased from 4.3 to 12.0 hours
- Recovery rate of 94.6% for detected errors

The computational overhead remains minimal:

- Training time overhead: +2.3%
- Memory usage overhead: +1.8%

This comprehensive fault tolerance framework ensures the robust and reliable operation of machine learning systems in production environments, particularly crucial for mission-critical applications where system failures cannot be tolerated. While implementing fault tolerance mechanisms is basic, it provides a foundation for error logging and handling in the machine learning pipeline. These features contribute to improved debugging capabilities and error tracking. The system demonstrates a first step towards creating a more robust and reliable machine learning workflow, with room for further enhancements to handle a broader range of failure scenarios in production environments.

9. HYPERPARAMETER OPTIMISATION ANALYSIS

9.1 Overview

In the comprehensive evaluation of machine learning models for image classification conducted in this study, hyperparameter optimisation played a crucial role in maximising model performance. The study employed a systematic approach combining randomised search with cross-validation to efficiently explore the hyperparameter space while maintaining the robustness of the estimates.

9.2 Methodology

The hyperparameter optimisation problem was formally defined as:

$$\theta^* =_{\theta \in \ominus} L(A_\theta, D_{train},\ D_{val} \tag{18}$$

where θ^* represents the optimal hyperparameters, $\ominus$ is the hyperparameter space, A_θ is the learning algorithm with hyperparameters θ, and L is the performance metric evaluated on the validation set D_{val} after training on D_{train}.

9.3 Implementation Strategy

The optimisation process sampled configurations according to:

$$\{\theta_1,\ \theta_2, \ldots, \theta_n\} \sim p\ (\ominus) \tag{19}$$

For each configuration, the authors employed 5-fold cross-validation to obtain robust performance estimates:

$$L(\theta) = \frac{1}{5}\sum_{i=1}^{5} \text{Accuracy}\left(A_\theta, D_{train}^{(i)}, D_{val}^{(i)}\right) \tag{20}$$

9.4 Model-Specific Optimisation

9.4.1 XGBoost Hyperparameter Space:

$$\ominus_{\text{XGBoost}} = \begin{Bmatrix} max\_depth \in \{3,\ 5,\ 7,\ 9\}, \\ learning\_rate \in \{0.01,\ 0.05,\ 0.1\}, \\ n\_estimators \in \{100,\ 200,\ 300\}, \\ subsample \in \{0.8,\ 0.9,\ 1.0\} \end{Bmatrix} \tag{21}$$

9.4.2 Random Forest Hyperparameter Space:

$$\ominus_{\text{RF}} = \begin{Bmatrix} \text{n\_estimators} \in \{50,\ 100,\ 200\}, \\ \text{max\_depth} \in x\{10,\ 20,\ 30\}, \\ \text{min\_samples split} \in \{2,\ 5,\ 10\} \end{Bmatrix} \tag{22}$$

9.4.3 SVM Hyperparameter Space:

$$\ominus_{\text{SVM}} = \begin{Bmatrix} \text{C} \in \{0.1,\ 1.0,\ 10.0\}, \\ \text{kernel} \in \{x'\text{linear}',\ '\text{rbf}',\ '\text{poly}'\}, \\ \text{gamma} \in \{'\text{scale}',\ '\text{auto}',\ 0.1,\ 0.01\} \end{Bmatrix} \tag{23}$$

9.5 Results Analysis

The results analysis especially for hyperparameter optimization are presented in TABLE 18.

Table 18: Hyperparameter Optimization Results

Model	Best Configuration	Fashion MNIST Accuracy (%)	CIFAR-10 Accuracy (%)
XGBoost	max depth=7, lr=0.1	89.31	54.93
Random Forest	n est=200, depth=20	87.42	51.64
SVM	kernel=rbf, C=10	84.26	47.26
KNN	k=5, weights=distance	85.03	42.35

9.6 Computational Considerations

The time complexity for hyperparameter optimisation can be expressed as:

$$T_{\text{opt}} = O(n \cdot k \cdot T_{train} \tag{24}$$

where n is the number of sampled configurations (20 in the implementation), k is the number of cross-validation folds (5), and T_{train} represents the training time complexity of a single model configuration.

9.7 Integration with Fault Tolerance

The hyperparameter optimisation process was integrated into the fault-tolerant framework of this study using:

$$FT_{\text{OptimiseModel}}\,(M,\ P,\ D) = \text{FT}\,(\text{RandomisedSearchCV}\,(M,\ P,\ D)\,, r_{\max}, b_{\text{initial}}) \tag{25}$$

where M represents the model class, P the parameter space, D the dataset, $r_{\max}$ the maximum retries, and b_{initial} the initial backoff time. This integration ensures robust optimisation even in the presence of hardware or software failures, maintaining the fault-tolerant properties of the system throughout the optimisation process.

10. ADVANCED FAULT TOLERANCE IMPLEMENTATION

10.1 Overview

Building upon the hyperparameter optimisation framework of this study, the authors implemented a comprehensive fault tolerance system that ensures robust model training and evaluation. This section details the practical implementation of the fault tolerance mechanisms, their integration with the machine learning pipeline, and their impact on system performance.

10.2 Implementation Architecture

The fault tolerance implementation follows a layered approach, with each layer handling specific types of failures:

$$\mathrm{FT}_{\mathrm{system}} = \left\{FT_{\mathrm{data}}, FT_{\mathrm{training}}, FT_{\mathrm{evaluation}}\right\} \tag{26}$$

Each layer implements the retry mechanism with exponential backoff:

$$RetryMechanism(f,\ r,\ b) = \begin{cases} f & \text{if successful} \\ \text{Retry}\,(f,\ \mathrm{r}-1,\ 2b) & \text{if failed and } r > 0 \\ \text{raise Exception} & \text{if } r = 0 \end{cases} \tag{27}$$

10.3 Performance Monitoring and Analysis

The system includes comprehensive performance monitoring which is presented in TABLE 19.:

Table 19: Fault Tolerance Performance Metrics

Metric	Without FT	With FT
System Crashes	5.2/day	0.3/day
Recovery Rate	N/A	94.6%
Average Uptime	4.3 hours	12.0 hours
Training Time Overhead	-	+2.3%
Memory Overhead	-	+1.8%

10.4 Error Recovery Statistics

The results for error recovery analysis are presented in TABLE 20.

Table 20: Error Recovery Analysis by Type

Error Type	Occurrences	Recovery Rate	Avg Recovery Time
Data Loading	156	98.7%	1.2s
Training	89	95.5%	2.8s
Evaluation	45	93.3%	1.5s
System	23	87.0%	3.4s

10.5 Practical Benefits

The implementation of a fault tolerance system in this study demonstrated several key benefits. These benefits include the following:

- **Improved Reliability:** System crashes were reduced by 94.6%, with successful recovery from most error conditions.
- **Minimal Overhead**: The fault tolerance mechanisms added only 2.3% to training time and 1.8% to memory usage.
- **Adaptive Response**: The system successfully adapted to different error types, with recovery rates exceeding 93% for most error categories.
- **Enhanced Monitoring**: Comprehensive logging and monitoring capabilities enabled better system understanding and maintenance.

10.6 Integration with Hyperparameter Optimisation

The fault tolerance system integrates seamlessly with the hyperparameter optimisation process:

$$FT_{\text{OptimiseModel}}(M,\ P,\ D) = \text{FT}\left(\text{RandomisedSearchCV}(M,\ P,\ D), r_{\max}, b_{\text{initial}}\right) \tag{28}$$

where:

M represents the model class

P is the parameter space

D is the dataset

$r_{\max}$ is the maximum number of retries

b_{initial} is the initial backoff time

10.7 Impact on Model Performance

The fault tolerance mechanisms demonstrated negligible impact on model accuracy. The results for Model Performance Comparison are presented in TABLE 21.

Table 21: Model Performance Comparison

Model	Standard Accuracy	FT Accuracy
XGBoost	89.31%	89.31%
Random Forest	87.42%	87.38%
SVM	84.26%	84.26%
KNN	85.03%	85.03%

These results demonstrate that the implemented fault tolerance successfully enhanced system reliability without compromising model performance, making it suitable for production deployments where both accuracy and robustness are critical.

11. NOVELTY: IMPLEMENTATION OF FAULT TOLERANCE IN MACHINE LEARNING MODELS

In this study, the novelty lies in the integration of fault tolerance within machine learning models for image classification. Traditional machine learning models are primarily designed to maximise accuracy and efficiency, often overlooking the robustness required for real-world deployment. The developed approach introduces fault tolerance into the models, enhancing their reliability and robustness. This is particularly important in scenarios where models are expected to operate in dynamic environments with potential errors, data inconsistencies, or unexpected input formats.

11.1 Importance of Fault Tolerance in Machine Learning

Kalaskar C, et al. (2023) [24], indicated that fault tolerance refers to a system's ability to continue operating correctly during unexpected errors or exceptions. Similarly, Bouhata D, et al. (2024) [25], revealed that in the context of machine learning, fault tolerance ensures that models can handle various failure scenarios gracefully without crashing or producing incorrect outputs. Roffe S, et al. (2020) [26], Kalaskar C, et al.(2023) [24], Bouhata D, et al. (2024) [25], stated that some of the key reasons for incorporating fault tolerance in machine learning models include:

- **Real-World Applicability**: In real-world applications, data can be noisy, incomplete, or erroneous. A fault-tolerant model can handle these situations by identifying and managing errors, ensuring consistent performance.
- **Operational Robustness**: Machine learning models deployed in production need to handle a wide range of scenarios, including hardware failures, network issues, and unexpected data input. Fault tolerance adds a layer of robustness, making models more reliable in production environments.
- **Improved Debugging**: Fault-tolerant models include logging and error-handling mechanisms that provide detailed insights into the system's behaviour during errors. This helps debug and improve the model over time.
- **User Trust**: A fault-tolerant system builds user trust by providing consistent and reliable outcomes, even when faced with unexpected situations. It ensures that the model's performance is not just high but also stable and predictable.

11.2 Implementation of Fault Tolerance

The current study implemented fault tolerance in the classifiers (K-Nearest Neighbors, Decision Tree, Random Forest, and XGBoost) by incorporating error handling, logging, and exception management. The following steps outline the process of implementing fault tolerance in these models:

- **Error Handling Mechanisms**: The study introduced mechanisms to catch and handle runtime exceptions such as data inconsistencies, missing values, and unexpected input formats. For

example, when loading the Fashion MNIST dataset, the models check for potential errors like missing images or invalid labels. If any inconsistencies are detected, the system logs the error and skips the problematic data point instead of crashing.

- **Logging System**: A detailed logging system was integrated into each classifier. During training and evaluation, the models log key events, including the occurrence of errors, warnings, and performance metrics. This logging system helps diagnose issues and understand the model's behaviour under different conditions. Logs are saved in a structured format, facilitating post-hoc analysis and debugging.
- **Graceful Degradation**: In cases where an error cannot be resolved (e.g., a corrupted data file), the models are designed to degrade gracefully. Instead of terminating the process, the system either continues with the remaining data or switches to a simplified fallback model. This ensures that the application can still provide useful outputs, albeit with potentially reduced accuracy.
- **Input Validation**: Input data is validated before processing to ensure it meets the expected format and range. For instance, image data is checked for correct dimensions and value ranges. Any deviations trigger error handling routines that either correct the input or log the error for further investigation.
- **Redundancy and Recovery**: For ensemble models like Random Forest and XGBoost, redundancy is built into the system. If one tree or estimator fails during training or prediction, the system continues with the remaining trees, ensuring that the overall model performance is not significantly affected.
- **Automated Retraining Triggers**: The fault-tolerant models are configured with automated triggers that initiate retraining if a specified error threshold is exceeded. For example, suppose the model encounters a high frequency of misclassified inputs or significant deviations in performance metrics. In that case, it logs these incidents and triggers a retraining process with updated parameters or additional data.

11.3 Process Overview of Fault Tolerance Integration

Chen E, (2021) [27], revealed that the integration of fault tolerance into the machine learning models follows a systematic process to ensure that each model can handle a wide range of failure scenarios. These scenarios include but are not limited to the following:

- **Initial Design and Testing**: Each model was initially designed and tested without fault tolerance to establish a performance baseline. This step ensured that the basic functionality of the models was correct and provided a benchmark for comparison.
- **Incorporating Fault Tolerance**: Fault tolerance features, including error handling and logging mechanisms, were integrated into the models. This involved modifying data loading, preprocessing, training, and prediction pipelines to handle various errors gracefully.
- **Stress Testing and Validation**: The fault-tolerant models were subjected to stress testing by introducing various failure scenarios, such as corrupted data, unexpected input formats, and

hardware limitations. The models' responses were monitored to ensure they could handle these scenarios without crashing.

- **Performance Evaluation:** The performance of the fault-tolerant models was evaluated using the same metrics (accuracy, precision, recall, and F1 score) as the standard models. This step ensured that incorporating fault tolerance did not compromise the models' effectiveness.
- **Iterative Refinement:** The fault tolerance mechanisms were refined based on the testing outcomes to address any gaps or weaknesses. This iterative process ensured that the models were not only fault-tolerant but also optimised for robustness and reliability.

11.4 Outcomes and Advantages

The implementation of fault tolerance in the classifiers resulted in several key advantages:

- **Enhanced Robustness**: The models demonstrated the ability to handle data inconsistencies and unexpected scenarios without compromising performance or terminating unexpectedly.
- **Improved Debugging and Maintenance**: The detailed logging system facilitated easier debugging and maintenance, enabling rapid identification and resolution of issues.
- **Reliable Deployment**: The fault-tolerant models are more suited for deployment in real-world applications where data quality and operational environments can be unpredictable.

11.5 Summary

The novelty of this study lies in the integration of fault tolerance into traditional machine learning classifiers, enhancing their robustness and reliability. This approach not only improves the models' performance in real-world scenarios but also adds value by ensuring consistent and dependable outcomes. By systematically incorporating error handling, logging, and recovery mechanisms, the study has developed classifiers that are accurate and resilient to various operational challenges. This fault-tolerant design can be a blueprint for deploying machine learning models in dynamic and error-prone environments.

12. CONCLUSION

This comprehensive study has evaluated the performance and reliability of various machine learning models for image classification tasks on Fashion MNIST and CIFAR-10 datasets. The experimental results demonstrate the superiority of ensemble methods, with XGBoost achieving the highest accuracy (89.31). The developed novel fault tolerance framework significantly enhanced system reliability, achieving a 94.6%. Future research should focus on developing more efficient training methods for ensemble models and investigating hybrid approaches combining traditional and deep learning methods. The successful implementation of fault-tolerant mechanisms without significant performance degradation represents an important advancement in making machine learning systems

more reliable for real-world applications. As these systems continue to be deployed in increasingly critical applications, the methodologies and frameworks presented in this study provide a foundation for building more robust and reliable machine learning systems.

References

[1] Kaur P, Singh SK, Singh I, Kumar S. Exploring Convolutional Neural Network in Computer Vision-Based Image Classification. In: Dyubele NPC, Mbangata L, Monyeki P, editors. International conference on smart systems and advanced computing. 2021.

[2] He X, Peng Y. Fine-Grained Image Classification via Combining Vision and Language. In: Proceedings of the IEEE conference on computer vision and pattern recognition. IEEE. 2017:5994-6002.

[3] Krishna ST, Kalluri HK. Deep Learning and Transfer Learning Approaches for Image Classification. Int J Recent Technol Eng (IJRTE). 2019;7:427-432.

[4] Obaid KB, Zeebaree S, Ahmed OM. Deep Learning Models Based on Image Classification: A Review. Int J Sci Bus. 2020;4:75-81.

[5] Charbuty B, Abdulazeez A. Classification Based on Decision Tree Algorithm for Machine Learning. J Appl Sci Technol Trends. 2021;2:20-28.

[6] Osisanwo FY, Akinsola JE, Awodele O, Hinmikaiye JO, Olakanmi O, et. al. Supervised Machine Learning Algorithms: Classification and Comparison. Int J Comput. International Journal of Computer Trends and Technology (IJCTT). 2017;48:128-138.

[7] Sarker IH. Machine Learning: Algorithms Real-World Applications and Research Directions. SN Comput Sci. 2021;2:160.

[8] Qiao A, Aragam B, Zhang B, Xing E. Fault Tolerance in Iterative Convergent Machine Learning. In: International Conference on Machine Learning. PMLR. 2019:5220-5230.

[9] Bode G, Thul S, Baranski M, Müller D. Real-World Application of Machine Learning-Based Fault Detection Trained With Experimental Data. Energy. 2020;198:117323.

[10] Asadova F, Kertész G, Lovas R, Szénási S. Fault Detection in GPU Enabled Cloud Systems–an Overview. In 2022 IEEE 20th Jubilee World Symposium rt-PA Machine Intelligence and Informatics (SAMI). IEEE PUBLICATIONS. 2022:000317-000322.

[11] Alotaibi FS. Implementation of Machine Learning Model to Predict Heart Failure Disease. Int J Adv Comput Sci Appl. 2019;10.

[12] Alam S, Yakopcic C, Wu Q, Barnell M, Khan S, et. al. Survey of Deep Learning Accelerators for Edge and Emerging Computing. Electronics. 2024;13:2988.

[13] Ozturk Kiyak E, Ghasemkhani B, Birant D. High-Level K-Nearest Neighbors (Hlknn): A Supervised Machine Learning Model for Classification Analysis. Electronics. 2023;12:3828.

[14] Xiong L, Yao Y. Study on an Adaptive Thermal Comfort Model With K-Nearest Neighbors (Knn) Algorithm. Build Environ. 2021;202:108026.

[15] Nafreen M, Bhattacharya S, Fiondella L. Architecture-Based Software Reliability Incorporating Fault Tolerant Machine Learning. In 2020 Annual Reliability and Maintainability Symposium (RAMS). IEEE PUBLICATIONS. 2020;1-6.

[16] Amin AA, Iqbal MS, Shahbaz MH. Development of Intelligent Fault-Tolerant Control Systems With Machine Learning Deep Learning and Transfer Learning Algorithms: A Review. Expert Syst Appl. 2024;238: 121956.

[17] Rigatti SJ. Random Forest. J Insur Med. 2017;47:31-39.

[18] Reis I, Baron D, Shahaf S. Probabilistic Random Forest: A Machine Learning Algorithm for Noisy Data Sets. Astron J. 2019;157:16.

[19] Pellegrino E, Jacques C, Beaufils N, Nanni I, Carlioz A, et al. Machine Learning Random Forest for Predicting Oncosomatic Variant Ngs Analysis. Sci Rep. 2021;11:21820.

[20] Asselman A, Khaldi M, Aammou S. Enhancing the Prediction of Student Performance Based on the Machine Learning Xgboost Algorithm. Interact Learn Environ. 2023;31:3360-3379.

[21] Torlay L, Perrone-Bertolotti M, Thomas E, Baciu M. Machine Learning–Xgboost Analysis of Language Networks to Classify Patients With Epilepsy. Brain Inform. 2017;4:159-169.

[22] Ma J, Yu Z, Qu Y, Xu J, Cao Y. Application of the Xgboost Machine Learning Method in PM2. 5 Prediction: A Case Study of Shanghai. Aerosol Air Qual Res. 2020;20:128-138.

[23] https://course.khoury.northeastern.edu/cs5100f11/resources/jakkula.pdf

[24] Kalaskar C, Thangam S. Fault Tolerance of Cloud Infrastructure With Machine Learning. Cybern Inf Technol. 2023;23:26-50.

[25] Bouhata D, Moumen H, Mazari JA, Bounceur A. Byzantine Fault Tolerance in Distributed Machine Learning: A Survey. J Exp Theor Artif Intell. 2024:1-59.

[26] Roffe S, George AD. Evaluation of Algorithm-Based Fault Tolerance for Machine Learning and Computer Vision Under Neutron Radiation. In: 2020 IEEE Aerospace Conference. IEEE PUBLICATIONS. 2020:1-9.

[27] Chen E. Fault Tolerance in Edge Computing: Exploring Strategies to Ensure Fault Tolerance and Reliability in Edge Computing Environments. Internet Things Edge Comput J. 2021;1:1-9.