COS70008 – Technology Innovation Project and Research

Final Project Report

CyberShield AI – Design 1

Student ID: 104837257

Student Name: Arun Ragavendhar Arunachalam Palaniyappan

Date: 30 May 2025

Contents

Word Count (excluding References, Table of contents and Captions): 2213

Abbreviations

AI: Artificial Intelligence
ML: Machine Learning
CPS: Cyber-Physical Systems
CSV: Comma-Separated Values
REST API: Representational State Transfer Application Programming Interface
UI: User Interface
ORM: Object Relational Mapping
SQL: Structured Query Language
PDF: Portable Document Format
AWS: Amazon Web Services
MISP: Malware Information Sharing Platform
RBAC: Role-Based Access Control
GCP: Google Cloud Platform
CI/CD: Continuous Integration / Continuous Deployment
.exe: Executable file (Windows application format)
.pdf: Portable Document Format
.docx: Microsoft Word Document Format
SVM: Support Vector Machine
PCA: Principal Component Analysis
MLP: Multi-Layer Perceptron

List of Figures and Tables

1. Introduction

This report is about the individual contributions and achievements in a group project, done over a period of 12 weeks. The main purpose is to present and summarise the outcomes of the final phase of the project.

It covers the below things, done individually:

- Learning about the exact project and client requirements.
- Finding the individual learning issue.
- Doing individual research to cover the technical gap.
- Individual contribution to:
    - ♦ System design and architecture of Design 1
    - ♦ Figma Prototype and User Interface of Design 1
    - ♦ Coding, Implementation and Testing of Design 1
    - ♦ Final Project Demonstration and Delivery to the client

2. Project Background and Requirements

2.1. Main project requirement

The main project requirement was to build a three-tier web application system which detects, predicts, classifies and analyses malware and malicious attacks, using a Hybrid Machine learning model.

It must be able to detect and classify multiple malware types and should have unique and innovative features, different from already existing implementations in the industry.

It should be able to detect known, unknown malware and obfuscated malware (malware that hides itself very well).

2.2. Project Background and Motivation

Cyber attackers have been improving their attack methods, using stealth techniques like obfuscation, memory-only execution, and behavioural masking to get past traditional detection tools. These threats are dangerous in cyber-physical systems, where software directly affects physical operations [1].

Despite advancements in security tools, detecting unknown or hidden malware remains difficult. Existing solutions rely on static rules or known signatures, which fail against new and modified threats [2].

The motivation for this project was to bridge that gap by creating a web application system that uses hybrid machine learning models to detect, predict and analyse these advanced malware threats. The detailed problem statement is covered in the next section.

3. Problem Statement

3.1 Problem statement from client:

- Search for and find out about different malicious attacks and cyberthreats
- Look for, find, and analyse useful malware datasets that are publicly available and are from reliable sources.
- Apply different machine learning models,
    - To identify different kinds of cyber-attacks
    - To recognise abnormal behaviour for early threat detection.
- Train and implement different hybrid AI model pipelines using the collected datasets.
- Integrate the Model with a 3-tier web application.
- The web application that should accept a file upload, scan it and return the scan result.

The client asked for five Unique designs having different Hybrid models and different web applications, and three of the designs were approved for implementation.

3.2 Tasks done to solve the problem statement

Individual learning gaps were identified and listed as below:

- How to build and combine hybrid ML models?
- How to choose the right malware dataset?
- How to connect ML models to a full-stack web application?

Then, carried out independent research to cover the learning gaps and learn about the below things:

- CIC-MalMem2022 malware dataset (Canadian Institute for Cybersecurity) [3].
- Autoencoders for anomaly detection [6].
- Random Forests for classification [5].
- REST APIs and full-stack web development (Vue.js, Flask, MySQL).

After that, started working on one complete end-to-end system design (Design 1).

- Created the end-to-end full system design architecture (by end of week 6).
- Created a working Figma prototype (by end of week 6).
- Implemented, tested and demonstrated the complete system (by end of week 11).

The upcoming section explains in detail about all the individual tasks done.

4. Design concept

4.1 Design 1 – CyberShield AI

From the five submitted designs, Design 1 was chosen, fully implemented and the project was successfully demonstrated and delivered to the client.

Main Individual contribution was the complete end to end design and development of Design 1 – CyberShield AI.

GitHub link of the Project: https://github.com/Arun-2208/TIP-Project-1

Figma prototype link:
https://www.figma.com/design/LeODBt9l38BV7VTPap9Sub/CyberShield-AI?node-id=125-3496&p=f&t=gZKmVH2bDgBXLgeS-0

4.2 Technology stack and Project Methodology

The project used a 3-sprint Agile approach to build the system quickly and adjust as needed. Each sprint focused on one part—model, backend, or frontend—to keep progress clear and feedback fast.

Technology stack:

Component	Technology
Frontend	Vue.js + Tailwind CSS for front inter user interface
Backend	Flask (Python) for ML model serving and API orchestration
Machine Learning	Scikit-learn (Random Forest), TensorFlow/Keras (Autoencoder)
Data Handling	Pandas, NumPy, StandardScaler (for feature engineering and transformation)
Database	Oracle SQL accessed via SQLAlchemy ORM
Model Management	Local .pkl file serialisation and parameter versioning
Deployment	Currently running on Localhost
Version Control	GitHub repository to track commits, issues, documentation, and

Table 1: Technology stack for Design 1

4.3 Dataset

The **CIC-MalMem2022** dataset was used.

It is a publicly available and well-documented malware memory dataset from the Canadian Institute for Cybersecurity [3].

It contains labelled samples of both **benign** and **malicious** memory behaviours. It was collected from real-world malware families like **trojans, ransomware, and spyware** [3].

Main highlights:

- **55 numeric features per sample**, covering memory, system, and behavioural patterns
- Over **58,000 rows** of data [3].
- Malware samples are obfuscated.
- Dataset is already balanced, cleaned up and scaled. So, it can be directly used for model training.
- Supports both anomaly detection and classification tasks.

4.4 Machine learning models

- **Autoencoder**: A neural network trained to learn normal behaviour. It was trained purely on benign data. It flags anything different from benign as “anomalous”, if the output is above a fixed threshold [6]. It was used for anomaly detection.

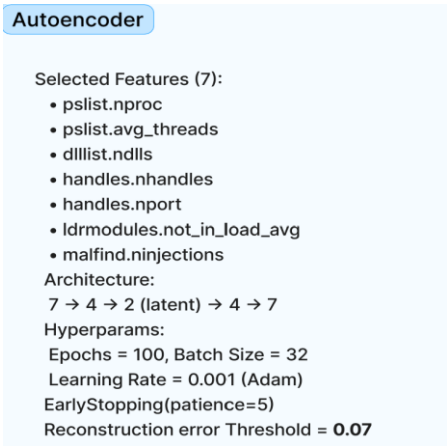

Figure 1: Design Architecture of Autoencoder [6]

- **Random Forest Classifiers**: 2 Random forests classifiers. One model for binary classification (malware vs benign). Another for multi-class classification (e.g., trojan, ransomware, spyware) [5].

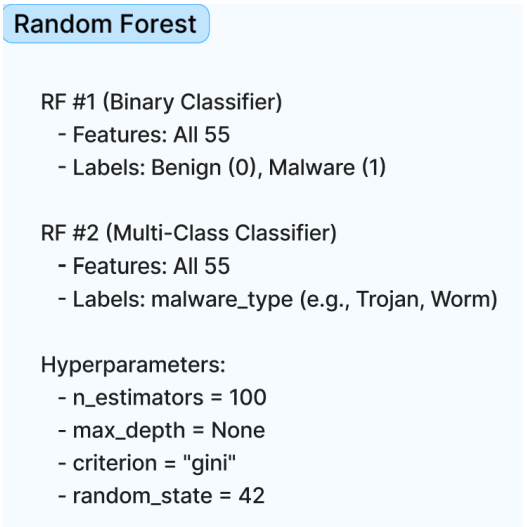

Figure 2: Design Architecture of Random Forest Classifier [5]

4.5 Sequential Hybrid Machine learning pipeline

These models were combined in a sequential pipeline to improve efficiency and accuracy.

The Autoencoder runs first to provide an anomaly score for every file. Then RF 1 checks if it’s malware. RF 2 runs only if malware is detected, which avoids unnecessary classification and reduces errors from false positives [2,5,6].

Steps:

1. Autoencoder gives anomaly score.
2. RF 1: detects if malware or benign.
3. RF 2 (run only if malware detected): Trojan, spyware, or ransomware.
4. Prediction results are stored in an Oracle SQL database, along with a **future risk rating** calculated using the Predictive Risk Rating Module, which combines current and historical anomaly data [2].
5. A complete PDF of the scan report can be generated if the User/admin wants it.

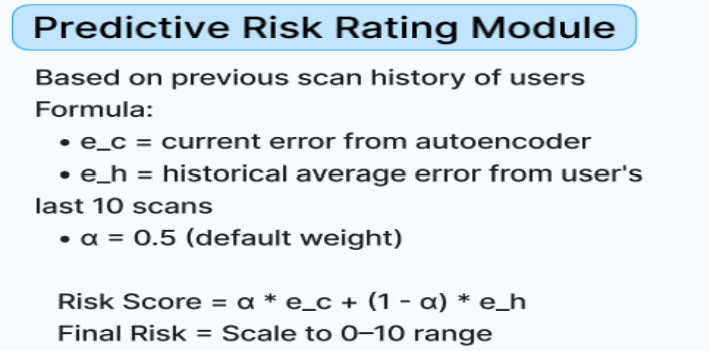

Figure 3: Predictive Risk Rating Module [2]

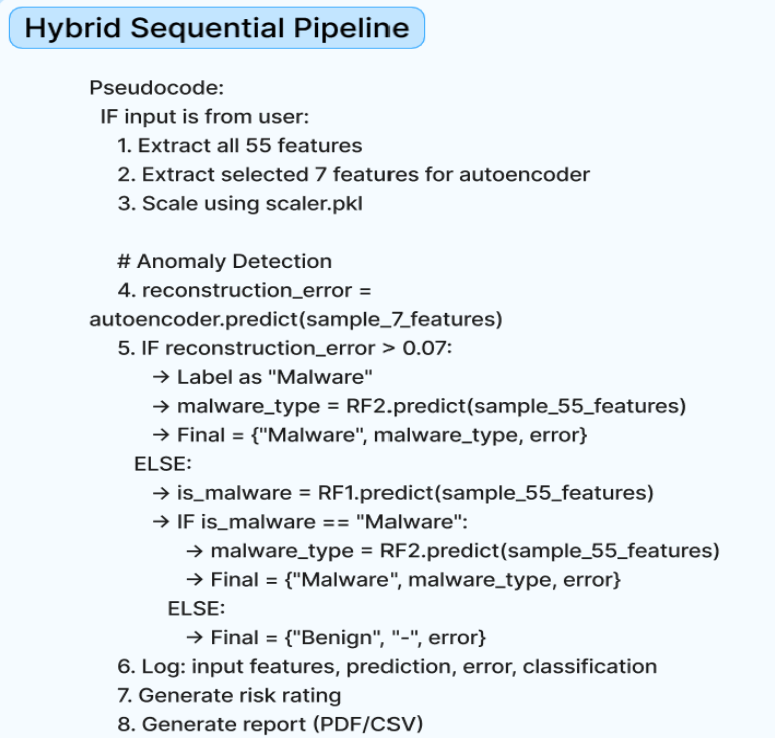

Figure 4: Hybrid Sequential Pipeline [2,5,6]

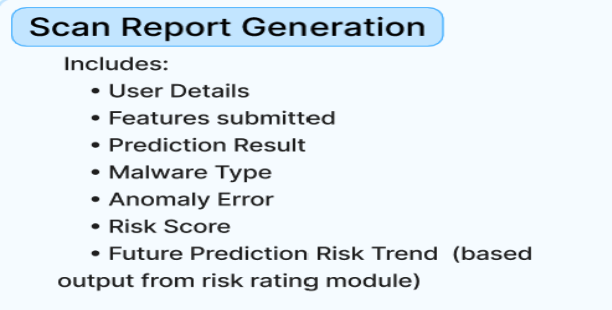

Figure 5: Scan Report Generator [1]

4.6 Database

MySQL database used with two main tables:

- **users** – Stores user login details, credentials, and roles (user or admin).
- **scan_history** – Stores all scan records, including file name, anomaly score, prediction results, malware type, risk score, and timestamp.

4.7 REST API

The backend logic is exposed through a set of **Flask-based REST APIs**. These APIs handle all core functions of the application, including:

- Scanning uploaded files and returning prediction results.
- Saving and retrieving scan history.
- Downloading PDF reports.
- Triggering model retraining with updated datasets and parameters.

4.8 Figma Prototype

The user interface was first designed using **Figma**. This included layouts for login, scanning, history, admin tools, and analytics.

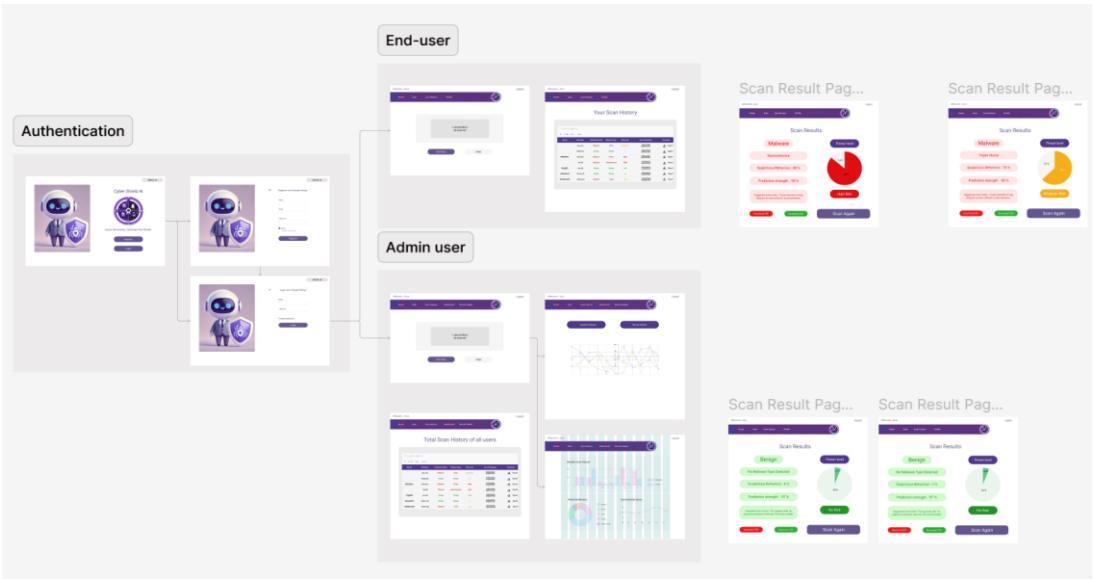

Figure 6: Figma Prototype for Design 1

4.7 Web Application

The web application has two login options: **regular user** and **admin**. Both use the same interface but the admin has extra features. The user interface layout which had been drafted using Figma, was implemented with Vue.js. It connects to the backend via the Flask REST APIs, explained in the previous section.

4.8 Complete System Design Architecture

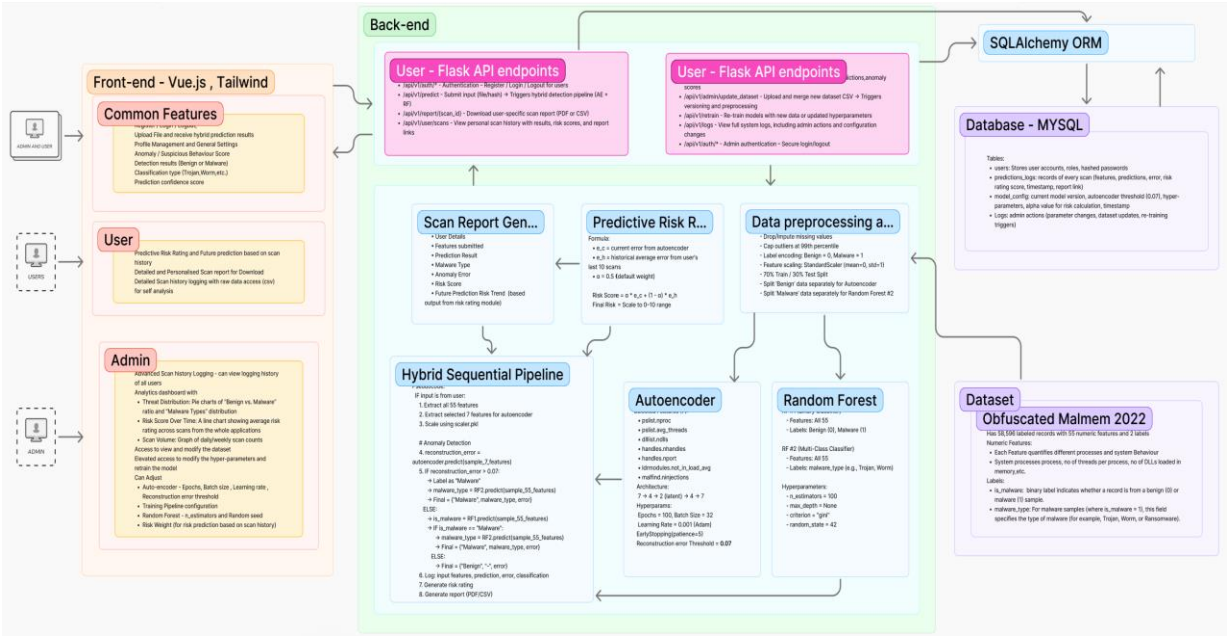

Figure 7: Complete System Design Architecture

4.9 Key Features of the CyberShield AI system

Hybrid AI model pipeline features:

- Can do anomaly detection
- Can detect and classify 3 different malware types:
    1. Trojan
    2. Ransomware
    3. Spyware
- Can do future prediction. Offers a calculated risk rating based on the user's previous scan histories.

Regular User Features:

- Upload a file and run a malware scan.
- View results instantly (safe or malware).
- See anomaly score, malware type (if any), and accuracy.
- Download a detailed PDF scan report.
- View scan history.
- Secure Registration and login with authorised access.

Admin Features (Includes all user features +):

- Access **Analytics Dashboard** showing:
    - Number of scans done **today**, **last week**, and **last month.**
    - **Average detection rate per malware type.**
    - **Malware type composition** (percentage split of trojan, spyware, ransomware).
- **Edit dataset** and update training data.
- **Retrain the model** with new parameters (epochs, learning rate and batch size) [1].
- View updated model results and logs.

4.10 Design Implementation and Processes

Step 1: Model Training

- Autoencoder trained on benign data to learn normal patterns.
- Random Forest models trained on labelled samples for binary and multi-class classification.
- Models tested, validated and saved as .pkl files [1].

```
autoencoder.compile(optimizer=Adam(learning_rate=learning_rate), loss=MeanSquaredError())
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = autoencoder.fit(
    X_train_ae, X_train_ae,
    validation_data=(X_val_ae, X_val_ae),
    epochs=epochs,
    batch_size=batch_size,
    callbacks=[early_stop],
    verbose=1
)

ae_model_path = os.path.join(MODEL_DIR, 'autoencoder_model.h5')
if os.path.exists(ae_model_path):
    os.remove(ae_model_path)
autoencoder.save(ae_model_path)
output_log.append(f" Autoencoder model trained for {len(history.history['loss'])} epochs (
```

Figure 8: Autoencoder [6]

```
# ---- RF1 ----
df_rf1 = pd.read_csv(RF1_DATA_PATH)
X_rf1 = df_rf1.drop(columns=['Class']).copy()
y_rf1 = df_rf1['Class'].apply(lambda x: 0 if x == 'Benign' else 1)
for col in X_rf1.columns:
    upper = X_rf1[col].quantile(0.99)
    X_rf1.loc[:, col] = np.where(X_rf1[col] > upper, upper, X_rf1[col])
scaler_rf1 = StandardScaler()
X_rf1_scaled = scaler_rf1.fit_transform(X_rf1)
with open(os.path.join(MODEL_DIR, 'scaler_rf1.pkl'), 'wb') as f:
    pickle.dump(scaler_rf1, f)
X_train_rf1, _, y_train_rf1, _ = train_test_split(X_rf1_scaled, y_rf1, test_size=0.3, random_state=42, stratify=y_rf1)
rf_binary = RandomForestClassifier(n_estimators=100, random_state=42)
rf_binary.fit(X_train_rf1, y_train_rf1)
with open(os.path.join(MODEL_DIR, 'rf_binary.pkl'), 'wb') as f:
    pickle.dump(rf_binary, f)
output_log.append(f" Random Forest (binary) trained with 100 trees (Accuracy: {rf_binary.score(X_train_rf1, y_train_rf1):.2%})

# ---- RF2 ----
df_rf2 = pd.read_csv(RF2_DATA_PATH)
X_rf2 = df_rf2.drop(columns=['Category']).copy()
y_rf2 = df_rf2['Category']
for col in X_rf2.columns:
    upper = X_rf2[col].quantile(0.99)
    X_rf2.loc[:, col] = np.where(X_rf2[col] > upper, upper, X_rf2[col])
X_rf2_scaled = scaler_rf1.transform(X_rf2)  # reuse scaler
X_train_rf2, _, y_train_rf2, _ = train_test_split(X_rf2_scaled, y_rf2, test_size=0.3, random_state=42, stratify=y_rf2)
rf_multi = RandomForestClassifier(n_estimators=100, random_state=42)
rf_multi.fit(X_train_rf2, y_train_rf2)
with open(os.path.join(MODEL_DIR, 'rf_multiclass.pkl'), 'wb') as f:
    pickle.dump(rf_multi, f)
output_log.append(f" Random Forest (multi-class) trained with 100 trees (Accuracy: {rf_multi.score(X_train_rf2, y_train_rf2):

output_log.append(" All models trained and saved successfully.")
```

Figure 9: Random Forest Classifier [5]

Figure 10: Model getting Trained [6]

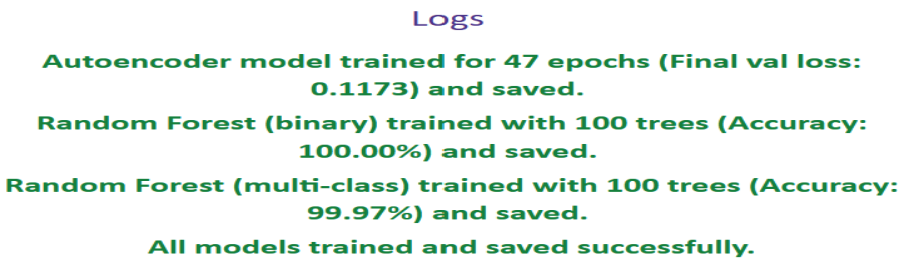

Figure 11: Model finished training

Step 2: Backend Development

- Flask used to create REST API routes for scanning, scan history, retraining, and report generation and admin analytics.
- Prediction results stored in Oracle SQL using SQLAlchemy ORM.

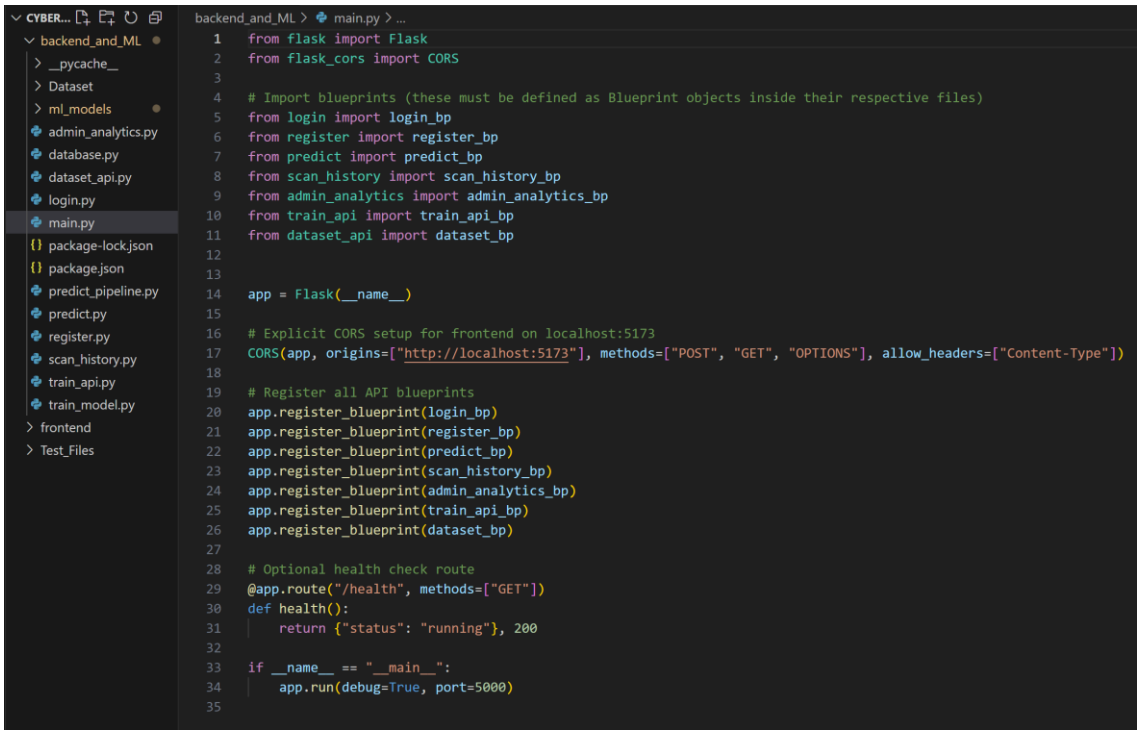

Figure 12: Flask server successfully running

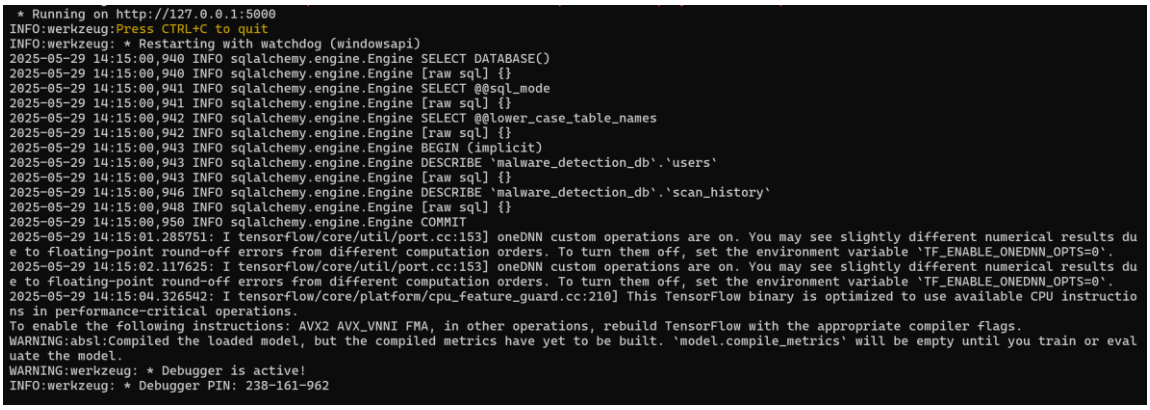

Figure 13: Flask server successfully running

Step 3: Frontend Development

- Vue.js used to build UI based on Figma layout.
- Pages include register, login, dashboard, file upload, scan history, and analytics.
- Tailwind CSS applied for responsive design.

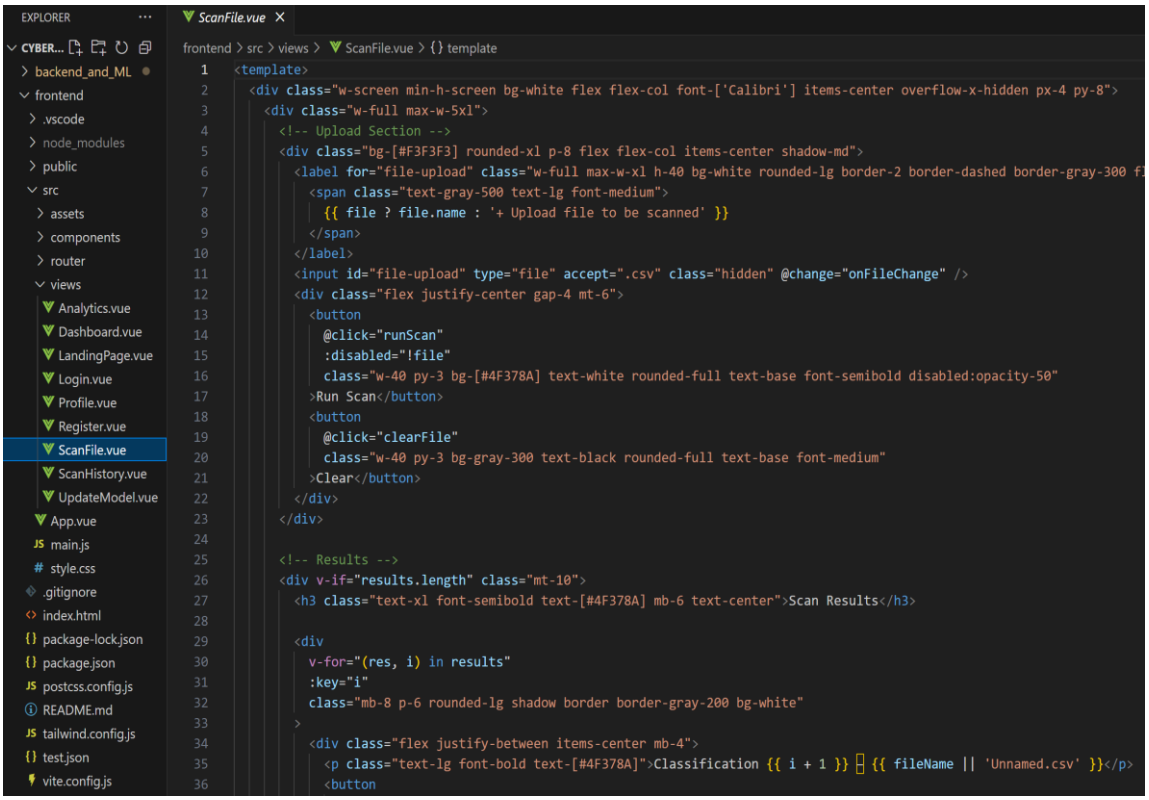

Figure 14: Front end build using Vue.js views and components

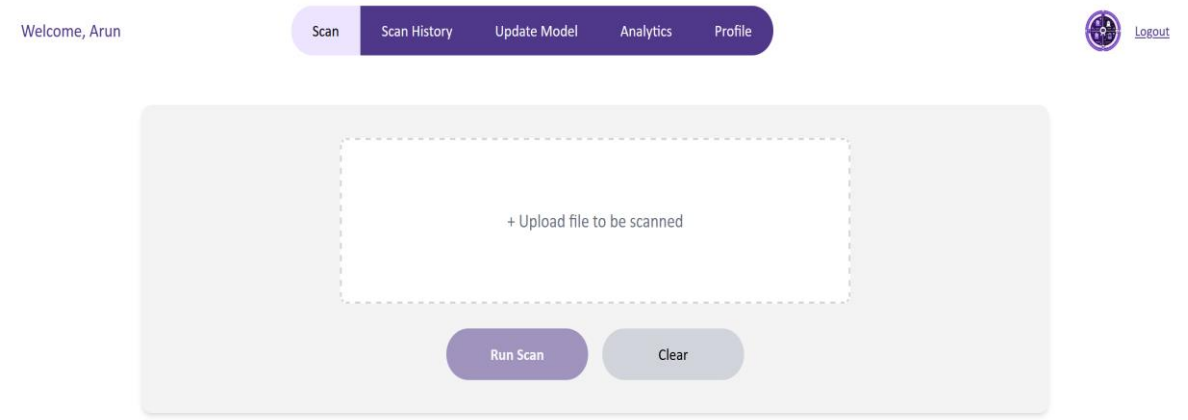

Figure 15: Front end page where user/admin can scan a file

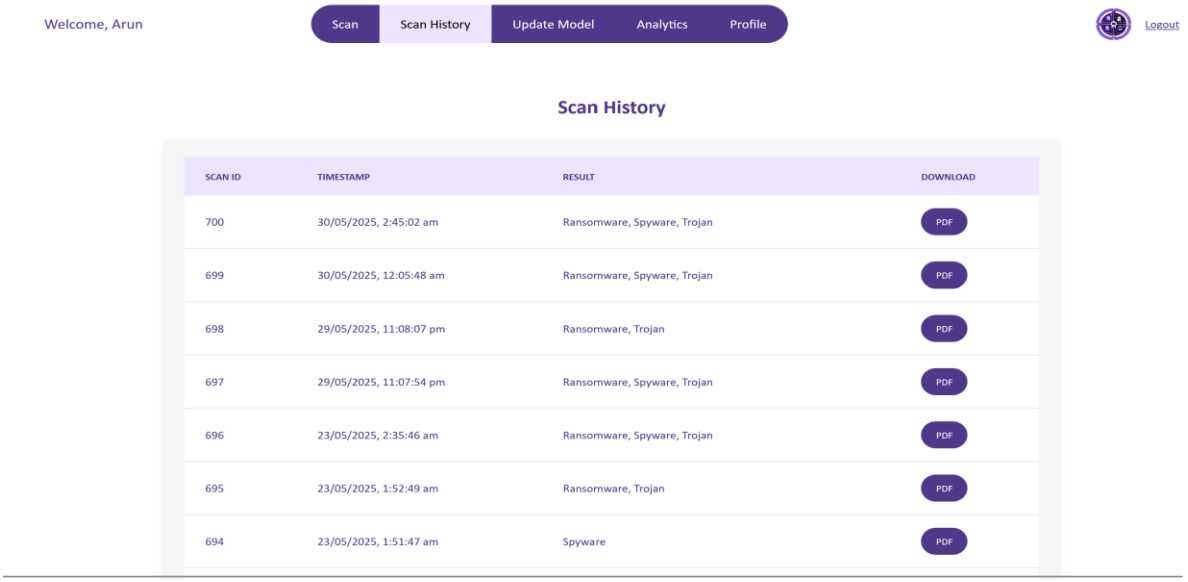

Figure 16: Front end page where user/admin can view scan history

Step 4: System Integration

- Frontend connected to backend via Axios.
- The data for the scanning, reporting, and history updates, were stored in the backend.
- Anomaly score, malware type, and future risk rating were displayed on the front end.

Step 5: Testing and Review

- Testing was done by scanning .csv files and scan results shown in below images.
- Final build reviewed by tutor and demonstrated to client with all key features working.

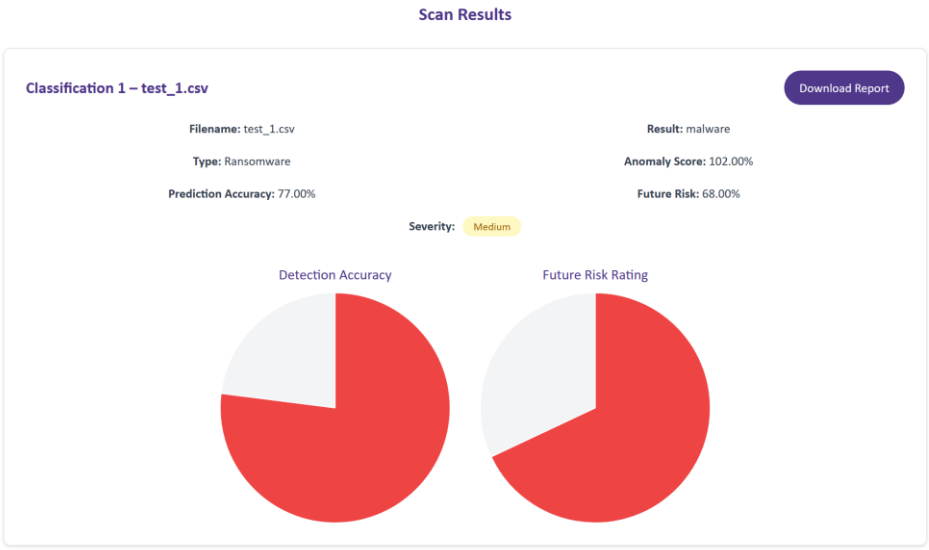

Figure 17: Scan results-malware type-1

Figure 18: Scan results-malware type -2

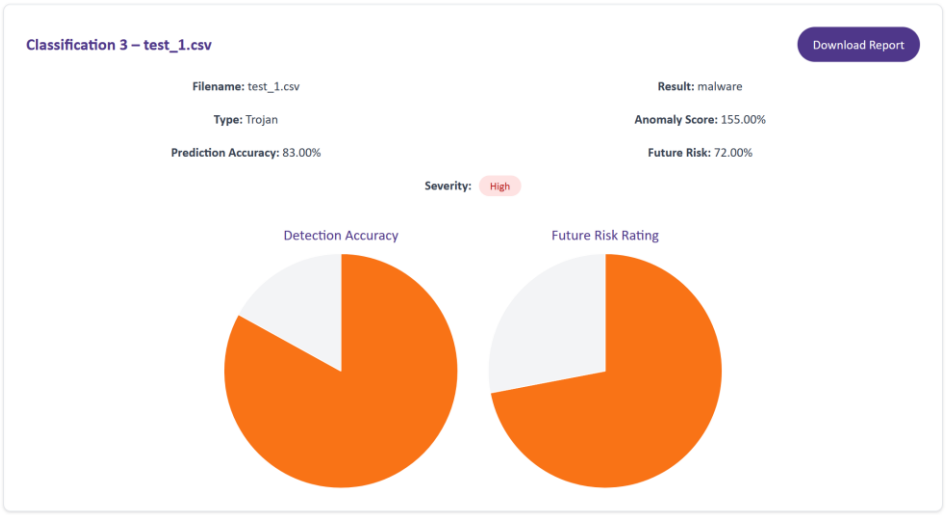

Figure 19: Scan results—malware type -3

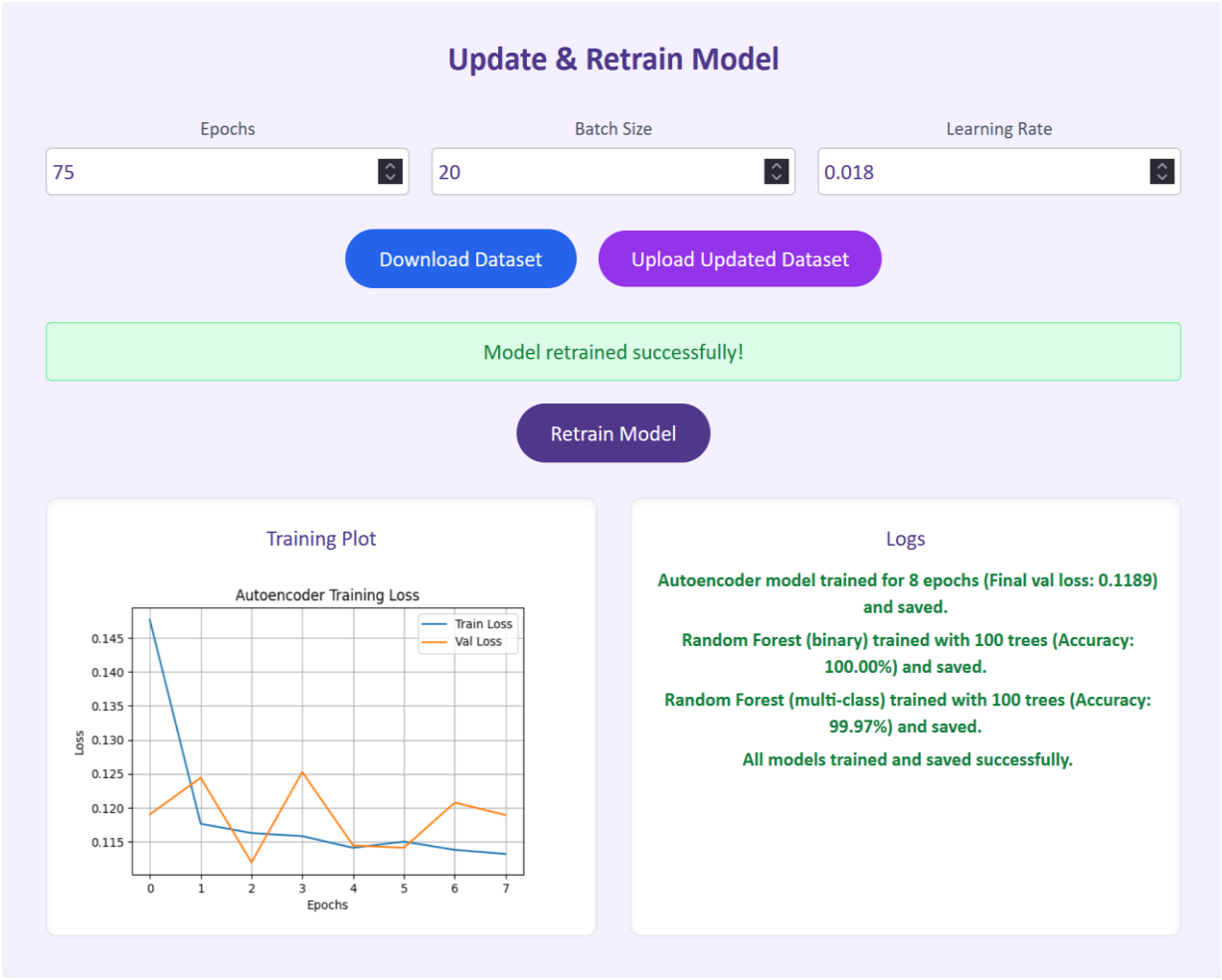

Figure 20: Successful Model retrained by the admin

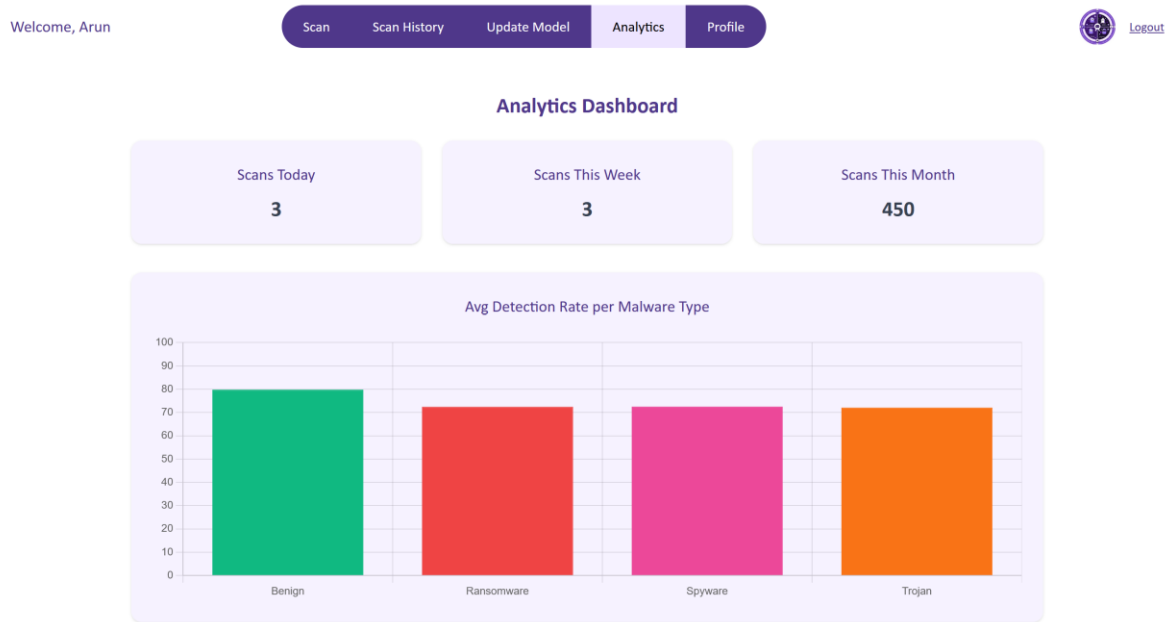

Figure 21: Admin Analytics Dashboard features

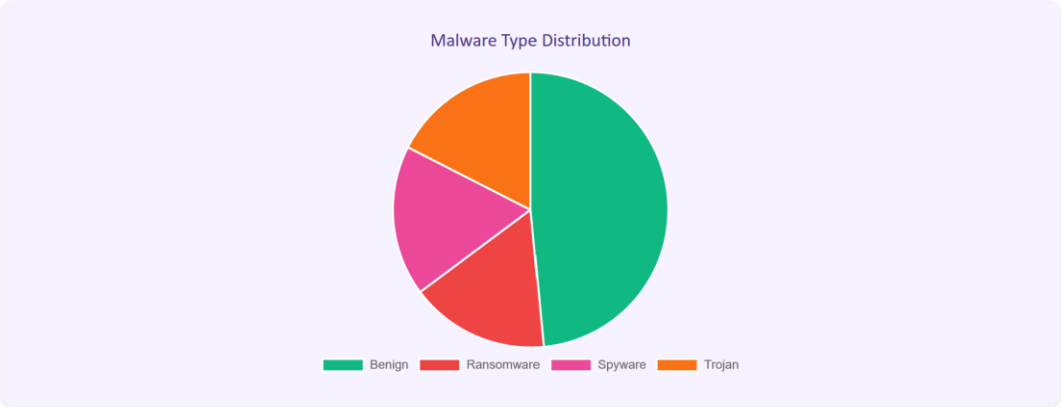

Figure 22: Admin Analytics Dashboard features

4.11 Constraints and Limitations

- **CSV-only input**: The system currently supports only CSV file uploads for scanning.
- **Threshold sensitivity**: The Autoencoder requires precise threshold tuning. A value too low (below 0.07) causes false alerts; too high (above 0.15) may miss real threats [6].
- **Retraining compatibility**: After retraining, the new model must follow the same structure as the existing one. Otherwise, the system may not load it correctly, causing prediction errors or failures [1].
- **Limited malware types**: As of now, detection is limited to trojan, ransomware, and spyware, as the dataset used includes only these three categories [3].
- **Local-only deployment**: The system runs on local host. Cloud hosting and live real-time detection are not yet implemented.
- **Model simplicity**: To keep the system lightweight and suitable for a university project, simple and fast models were chosen instead of complicated Machine learning techniques [1].

4.12 Compatibility and Benefits of Implemented Design

The final design fully meets the client's core requirements and works as intended:

- Can detect, classify, predict both known malware and new unknown threats using a sequential hybrid AI approach.
- Secure user login and registration is implemented for controlled access.
- Delivers complete scan results with:
    - Anomaly score (from Autoencoder)
    - Malware detection (from RF)
    - Malware type classification (3 types – Trojan, Ransomware, Spyware)
    - Risk score and prediction accuracy
- Gives full admin control through the dashboard:

        - View and manage all scan records
        - Extract and update the dataset directly
        - Edit hyperparameters and retrain the model anytime
- Includes a detailed admin analytics dashboard that shows:
        - Total scans by day, week, and month
        - Detection rates for each malware type
        - Percentage of trojan, spyware, and ransomware cases
- Current model can readily be scaled to detect and classify more malware types, by just updating the dataset from the live web application.
- Easy to update into live usage for an interested buyer.
        - Can be easily integrated inside an existing application
        - Project is packaged and ready for direct cloud deployment.

5. Conclusion and Recommendations

5.1 Conclusion

The objective of this project was to design a full-stack web application that uses a hybrid machine learning model to detect and classify malware, with anomaly detection and behavioural analysis for improved accuracy. *Design 1 – CyberShield AI* has successfully achieved this goal.

The system links and meets all the project and client requirements:

- It detects, predicts, classifies multiple malware types using a two-step sequential hybrid AI Model pipeline. It also does anomaly detection and future risk prediction.
- It has been trained on a public dataset and integrated with a 3-tier web application.
- Includes both User and Admin access (extra features). Users can scan and view results. Admins can also retrain models, update datasets, and access analytics dashboard.

All tasks, from initial research and dataset selection to model development, system integration, and testing, were completed within the project timeline of 12 weeks. The final solution is accurate, stable, and ready for future expansion. It can be made more suitable for real-world use with further improvements as per the below recommendations.

5.2 Future Recommendations

Implementing the below improvements will help CyberShield AI improve further and become an industry ready, advanced malware detection system:

- **Cloud deployment**: Hosting the system on a cloud platform (e.g., AWS, Azure) for public access and scalability [2].
- **Real-time detection**: Adding live memory or file monitoring to detect threats in real time, not just from uploaded samples.

- **Ensemble Hybrid model execution**: Running the Autoencoder and Random Forest models simultaneously to reduce scan time and improve performance [5,6].
- **Extended malware coverage**: Using larger datasets with more malware categories to go beyond the current three classifications [2,3].
- **Support for More File Types**: Extending support for more file types other than CSV to formats like .exe, .pdf, or .docx, using safe extraction and pre-processing methods.
- **Advanced User Access Management (RBAC)**: Introducing role-based access control with activity logs to support secure multi-user environments.
- **Automated Dataset Updates**: Creating a system for admins to periodically import and validate updated datasets, improving detection for emerging malware types [3].
- **Encrypted Report Storage**: Encrypting stored scan results and PDF reports to protect user data and maintaining data privacy standards [2].
- **Model Versioning and Rollback**: Implementing version control for ML models, allowing admins to roll back to stable versions if a new model causes issues after retraining.
- **Integration with Threat Intelligence Feeds**: Pulling live threat data from platforms like VirusTotal or MISP to improve scan context and improve results [4].
- **Sandbox Simulation**: Adding a safe environment to execute suspicious files and analyse their behaviour before classifying, especially useful for detecting stealthy malware [1].

6. References

[1] A. I. Abubakar, B. Pranggono, A. Khan, and M. Iqbal, "A survey on feature selection and feature extraction for malware detection," *J. Netw. Comput. Appl.*, vol. 203, p. 103356, 2022. [Online]. Available: https://doi.org/10.1016/j.jnca.2022.103356

[2] M. Alazab, A. Awajan, A. Abdallah, H. Alqahtani, and A. Alzahrani, "Intelligent forensics: AI-based threat hunting and digital forensics in cyberspace," *Future Gener. Comput. Syst.*, vol. 128, pp. 84–104, 2022. [Online]. Available: https://doi.org/10.1016/j.future.2021.11.008

[3] Canadian Institute for Cybersecurity, "CIC-MalMem2022 dataset," *University of New Brunswick*, 2022. [Online]. Available: https://www.unb.ca/cic/datasets/malmem2022.html

[4] M. Dastbaz, B. Akhgar, and L. Lecky-Thompson, "Cyber forensics in the cloud: State-of-the-art and current challenges," *Comput. Secur.*, vol. 105, p. 102345, 2021. [Online]. Available: https://doi.org/10.1016/j.cose.2021.102345

[5] R. Patel, B. Malhotra, and S. Jain, "Cybersecurity and AI: A strategic review of machine learning-based malware detection techniques," *ACM Comput. Surv.*, vol. 56, no. 2, pp. 1–37, 2023. [Online]. Available: https://doi.org/10.1145/3576123

[6] Y. Chen, H. Zhang, Y. Shen, and L. Tian, "Autoencoder-based anomaly detection with hybrid feature learning for malware detection," *IEEE Access*, vol. 10, pp. 12101–12114, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3145405