COS70008 – Technology Innovation Project and Report

Assignment – 3
Innovation Concept Report

Thursday 04:30 PM – 06:30 PM - Group No: 4

Student Name	**ID**
Arun Ragavendhar Arunachalam Palaniyappan	104837257
Vatana Chhorn	104198682
Huu Tien Dang	104677479
Klos Wasanapitanon	104760955
Nichakorn Srisupornwong	104835031

Table of Contents

Executive Summary

This report presents five distinct system designs, each built to detect a specific type of cyberattack: memory-based malware, phishing through SMS and URLs, volumetric DDoS attacks, and hybrid malware that combines static file analysis with behavioural observation. All designs follow a hybrid machine learning approach, combining anomaly detection models (such as Autoencoder or PCA) with classification models (such as Random Forest, Logistic Regression, or SVM).

The report details each system's methodology, constraints, technical specifications, and architectural layout. Development methodologies—Agile or Waterfall—were chosen based on scope and complexity. For each design, the report includes structured information on model selection, feature engineering, decision logic, API workflows, and user interface prototypes built in Figma. Public datasets such as Obfuscated-MalMem2022, CIC-DDoS2019, BIG 2015, and phishing message corpora were used to ensure threat relevance. Each design has been evaluated for its limitations, such as data imbalance, processing constraints, and potential evasion risks. Corresponding mitigation strategies—such as feature selection, model tuning, and access control, have also been recommended.

The final section presents a comparative justification of the five proposed systems. It explains the rationale behind their model combinations, identifies the design with the strongest technical foundation, and outlines the next steps for implementation. Three of the five systems will be selected for development as full stack three-tier web applications by the client. GitHub repositories for each have been prepared to support source control and documentation.

In summary, this report documents the full design and technical groundwork for five Hybrid AI-based malware detection systems, offering a complete foundation for the project implementation.

Name	Student ID	Contribution Summary
Arun Ragavendhar Arunachalam Palaniyappan	104837257	Developed Design 1 (Memory based Malware Detection). Completed Part A, justification and evaluation section, Figma, datasets, and formatted and compiled the whole document into a single, unified tone. Ensured overall clarity and alignment with client requirements.
Vatana Chhorn	104198682	Developed Design 2 (Phishing Detection System). Researched relevant models and datasets, created Figma prototype, and wrote the design description, architecture and all activities for Design 2.
Huu Tien Dang	104677479	Developed Design 4 (Botnet Detection System). Created architecture diagrams, datasets, defined technical constraints, Figma and contributed to the specifications section and all activities for Design 4.
Klos Wasanapitanon	104760955	Developed Design 3 (DDoS Detection System). Worked on architecture diagrams, datasets, system components, extensive and detailed Figma prototype work and designing and all activities for Design 3.
Nichakorn Srisupornwong	104835031	Developed Design 5 (Static and Dynamic Malware Detection). Architecture diagrams, datasets, defined use of Random Forest + SVM and all activities for Design 5.

1. Part A

1.1 Project Overview

As cyber threats continue to evolve in sophistication, traditional malware detection tools are becoming increasingly ineffective. Static, signature-based systems often fail to recognise new, obfuscated, or fileless threats, especially in environments such as Cyber-Physical Systems (CPS), where software directly controls physical devices. This project presents five unique system designs aimed at detecting and analysing different types of cyberattacks. These include memory-based malware, phishing through SMS and URLs, DDoS attacks, and threats that require both static and behavioural analysis. Each system is built using a hybrid AI approach that combines anomaly detection (like Autoencoder or PCA) with classification (like Random Forest, Logistic Regression, or SVM). These model combinations were chosen based on how the data behaves and how well the system can explain and respond to both known and unknown attacks.

The goal is to develop three fully functioning **three-tier web applications** that can not only detect malware but also analyse cyber-physical system (CPS) behaviours. This helps in identifying malicious activity based on how programs act, not just how they look. Each web system includes a user interface, machine learning backend, and secure database, built using public datasets suited to each threat type. The project responds to the challenge of early and accurate detection in environments where attackers often use hidden or fast-changing techniques. Traditional static or rule-based tools often fall short, especially when dealing with obfuscated or fileless malware. This is where hybrid AI methods help—combining structure, behaviour, and pattern recognition to improve threat visibility.

All five designs include detailed architecture diagrams, Figma-based UI flows, backend routing logic, and appropriate security features. Public datasets such as Obfuscated-MalMem2022, CIC-DDoS2019, BIG 2015, and phishing corpora have been used to ensure that detection systems match the nature of the threats they target.

1.2 Client Requirements

The project is shaped by three main client goals. First, to design five AI-based models, each targeting a specific threat using different data and combination of ML methods. Second, to create five User Interface prototypes in Figma and fully develop three of them as complete three tier web applications. Third, to include behaviour-based detection in the developments—watching how programs behave rather than just scanning their files.

The client has outlined a clear set of functional and technical expectations that drive the scope of the project:

Malware Detection Focus: Each system must be capable of detecting and analysing a specific class of malicious attack. These include but are not limited to:

- Static and dynamic malware analysis
- Memory based malware analysis
- Phishing (via email, SMS, and URLs)

- Fileless attacks and cyber-physical behavioural anomalies
- DDoS detection

Dataset Alignment: All detection systems must be trained on publicly available and reputable datasets suited to their target threat. For example, the CIC-MalMem2022 dataset supports memory-based malware classification, while the CIC-DDoS2019 dataset is used for traffic-based DDoS attack detection.
Hybrid AI Models: Each design must use at least two ML techniques in a hybrid configuration. For example, a model combining anomaly detection (unsupervised) with classification (supervised). This dual-layer approach should improve detection accuracy for both known and unknown threats.
System Architecture: The client requires full design documentation including:

- End-to-end **system architecture diagrams.**
- Clearly defined **decision logic**, model pipelines, and user flows.
- Identification of decision points such as threshold values, risk scoring formulas, tuning parameters.

Web Application Features:

- A responsive **frontend interface** for three different user types (guest, registered user, admin).
- A fully functional **backend** using Flask or FastAPI for serving ML models and APIs.
- A structured **SQL or NoSQL database** to store user data, scan history, logs, and configuration settings.

Prototype Demonstration:

- Five Figma prototypes to visualise different solution designs.
- Three selected systems to be developed and demonstrated as functional web applications.
- Final project presentation including fully working implementations, source code and documentation.

Behavioural Analysis Capability:

- Each system should incorporate behavioural analysis or sandboxing to monitor cyber-physical responses over time, particularly for threats that do not leave clear static traces
- The implementation should simulate safe environments where malicious behaviours can be observed and recorded

Current Scope, Constraints and Exclusions:

- Deployment will be limited to **localhost or test servers** due to time and resource limitations
- Integration with **external tools** like threat intelligence feeds or SIEM dashboards is outside the current scope.
- User interfaces will support only English at this stage; multi-language and accessibility features may be explored in future phases.
- In summary, the client expects the project to deliver 3 innovative web application systems that use AI not only to detect threats but also to visualise and explain them in a user-friendly way.

2. Part B

2.1 Design 1 - Memory-Based Malware Detection, Classification and behavioural analysis (AE + RF)

2.1.1 Methodology

Cyber Shield AI is to be developed using a structured Agile approach, divided into three focused sprints. Each sprint delivers functional system components while enabling continuous testing and feedback. This allows parallel progress across model training, backend APIs, and frontend design, with regular reviews ensuring adaptability and alignment with project goals. The table below outlines the sprint-based breakdown of activities:

Sprint	Weeks	Key Activities
Pre-Development Phase	Weeks 1–5	- Finalise project scope, user stories, and feature list
		- Design full system architecture (frontend, backend, ML pipeline)
		- Lock in parameters: AE features, RF model structure, risk scoring logic
		- Select dataset (Obfuscated-MalMem2022) and complete Exploratory data analysis
		- Build Figma prototypes and frontend wireframes; finalise UI/UX flow
Sprint 1	Weeks 6–7	- Begin backend setup and frontend scaffold (Flask APIs + Vue.js UI)
		- Preprocess data: feature scaling, encoding, RFE, and 7-feature AE pipeline
		- Train and validate Autoencoder (threshold tuning) and RF classifiers
		- Build secure file upload and scan submission interface
Sprint 2	Weeks 8–9	- Integrate backend prediction API with frontend results dashboard
		- Implement user login, JWT-based auth, and admin role separation
		- Add logging and history tracking per scan, with RF confidence scores
		- Conduct unit testing and validate API response consistency
Sprint 3	Weeks 10–12	- Finalise admin dashboard: parameter tuning, report generation, retraining module
		- Add anomaly scoring trend visualisation and risk logs
		- Perform full integration testing across user flows and system layers

		- Deploy locally with CI/CD support and prepare all final documentation and presentation materials

Table 1: Sprint Plans for Design 1

Each sprint ends with a review and retrospective to address issues and integrate feedback. This cycle supports adaptability and ensures the system evolves toward a functional malware detection solution within the set timeline.

2.1.2 Design Constraints

Data Constraints:

- **Real-world Data:** Memory dumps collected in debug mode to reflect realistic environments.
- **Obfuscation:** Malware in the dataset is obfuscated, challenging the model to detect hidden patterns.
- **Class Balance:** The dataset is balanced in terms of benign and malicious records with 50 percent each, but requires careful handling to avoid noise from less informative features.

Resource & Performance Constraints:

- **Overfitting:** Training the autoencoder on all 55 features will lead to overfitting; therefore, we limit it to 7 high-signal features.
- **Scalability:** Random Forest models should handle tens of thousands of records. The bagging approach in RF is strong, but the feature set must be managed for efficient training.
- **Memory & Computation:** Efficient data scaling and processing are important to handle large numeric arrays without excessive memory usage.

Security Constraints:

- **API Security:** All endpoints must be secured via HTTPS, with proper authentication and authorisation measures.
- **Data Integrity:** Only authorised admins can update datasets or trigger re-training.
- **Audit Logging:** All changes and administrative actions must be logged for auditing.

User Constraints:

- The end-user interface must be simple and accessible for scanning operations.
- The admin dashboard should provide clear analytics, logs, and easy parameter tuning without overwhelming complexity.

2.1.3 Specifications

This section outlines the full technical scope of the Cyber Shield AI system, covering model design, feature engineering, decision logic, risk scoring, reporting, and system integration. The objective is to build a secure, efficient, and accurate solution for malware classification using **memory dump analysis.**

Data Preprocessing & Feature Handling

The dataset Obfuscated-MalMem2022.csv consists of 55 numeric features per memory sample. Data preprocessing includes removing missing values and capping outliers at the 99th percentile.

Two labels are extracted:

- is_malware (binary classification: 0 = Benign, 1 = Malware).
- malware_type (multi-class label for subtype prediction, available only for malware records)
- All features are scaled using StandardScaler (mean = 0, std = 1). A second scaler is trained on a reduced feature set for the autoencoder.
- The data is split 70/30 into training and test sets. Autoencoder training uses only benign samples and a 7-feature subset selected for their statistical and behavioural significance.

Autoencoder (Anomaly Detection)

The autoencoder is designed to model normal system behavior and detect deviations as anomalies. It uses a 7-feature input:

- Input features - pslist.nproc, pslist.avg_threads, dlllist.ndlls, handles.nhandles, handles.nport, ldrmodules.not_in_load_avg, and malfind.ninjections .
- Its neural network architecture is: 7 → 4 → 2 → 4 → 7, with MSE loss. Training uses an Adam optimiser (lr = 0.001), batch size of 32, 100 epochs, and EarlyStopping (patience = 5). A reconstruction error threshold of **0.07** is used to classify anomalies.

Random Forest Classifiers

Two Random Forest models are to be used:

- **RF #1** is a binary classifier trained on all 55 features to classify samples as benign or malware.
- **RF #2** is a multi-class classifier trained on malware-only records to predict subtype (e.g., Trojan, Worm, Ransomware).

Both models use 100 estimators, no depth limit, Gini criterion, and a fixed seed (random_state = 42). Models are saved using pickle.

Hybrid Decision Pipeline

Each input passes through a sequential logic:

- Full input is scaled using StandardScaler (mean = 0, std = 1).
- AE features are extracted, scaled and then passed to the autoencoder.
- If reconstruction error > 0.07, the sample is classified as malware and RF #2 predicts the subtype.
- If the error ≤ 0.07, RF #1 determines whether the sample is benign or malware.
- If RF #1 says "malware", the sample goes to RF #2 for subtype classification.
- If RF #1 says "benign", the result is returned as benign.

The final output includes the classification label, malware subtype (if any), anomaly error, and RF prediction confidence scores.

Risk Scoring

Each scan is assigned a risk score that reflects both the immediate anomaly and the user's past scan history. The formula is:

Risk = $\alpha \times$ current_error + $(1 - \alpha) \times$ historical_average_error, where **α (alpha)** is the weight given to the current scan's reconstruction error, and **$(1 - \alpha)$** is the weight assigned to the average error from the user's previous scans.

By default, **$\alpha = 0.5$**, meaning equal importance is given to the current scan and historical trend. The final score is scaled to a 0–10 range and logged for future reference and trend analysis.

Report Generation

After each prediction, the system generates a downloadable report (PDF or CSV). It includes:

- User and scan metadata
- All 55 features (with the 7 AE features highlighted)
- Autoencoder reconstruction error
- Classification results and subtype
- RF confidence scores
- Risk rating and optional trend analysis

System Interfaces

Frontend:

The frontend is to be built using Vue.js with Tailwind CSS. It supports login, scan input, prediction display, report download, and admin dashboard functionality. Admins can tune model parameters, upload datasets, and trigger retraining. All communication is through REST APIs using JSON.

Backend:

The backend is developed using Flask in an Anaconda environment. It handles:

- User authentication and session management
- Prediction routing and model inference
- Risk calculation and scan logging
- Report generation and admin controls

Main proposed API endpoints include:

- /api/v1/predict, /api/v1/report/{scan_id}
- /api/v1/user/scans, /api/v1/admin/scans
- /api/v1/admin/update_dataset, /api/v1/retrain

Database (MySQL + SQLAlchemy)

The system uses a MySQL database managed via SQLAlchemy ORM. It stores:

- **Users**: credentials, roles, and profile info
- **Predictions Log**: full scan records, features, results, AE errors, and report links
- **Model Config**: current model version, AE threshold, hyperparameters, and risk alpha
- **Admin Logs**: all administrative actions for auditing and rollback

This persistent storage ensures full traceability, supports analytics, and allows model retraining with updated datasets.

Algorithm Selection Justification

Several algorithm combinations were evaluated before finalising the hybrid approach of **Autoencoder + Random Forest with a sequential hybrid pipeline**. The selection process was grounded in empirical research and performance evaluation based on memory dump analysis tasks, model complexity, and alignment with the project’s goals.

Autoencoder vs. Isolation Forest for Anomaly Detection

- Autoencoders were chosen for anomaly detection due to their ability to learn compressed representations of benign behavior, which makes them highly effective at detecting previously unseen (zero-day) malware through reconstruction errors. As demonstrated by **Abubakar et al.**

(2022), deep learning models such as autoencoders achieve superior detection accuracy in zero-day scenarios when trained on memory dump data.

Random Forest vs. SVM / XGBoost for Classification

- Random Forest was preferred over Support Vector Machines and XGBoost due to its ability to handle high-dimensional data, and superior interpretability. In a comparative study by **Dastbaz et al. (2021)**, Random Forest outperformed SVMs in both accuracy and F1 score for such cases of memory-based malware classification. Also, **Alazab et al. (2022)** recommended Random Forest for forensic-level malware classification due to its resilience to obfuscated feature noise.

Combined Autoencoder + RF

- Combining the unsupervised anomaly detection capability of Autoencoders with the supervised classification accuracy of Random Forests provides a hybrid detection system that excels in identifying both known and unknown malware types. According to **Patel et al. (2023)**, this hybrid approach achieved increased accuracy of final prediction and reduced false positives, in detections using memory dump analysis.

Dataset Relevance

- The malware detection technique selected, directly matches with the CIC-MalMem2022 dataset's structure, which includes memory-level obfuscated malware traces—suitable for anomaly-based and supervised learning as per **Canadian Institute for Cybersecurity, (2022).**

Hybrid Approach

The Cyber Shield AI detection engine adopts a **hybrid architecture** by integrating unsupervised anomaly detection and supervised classification. The system builds upon the following principles:

- **Compressed Feature Reconstruction**: The autoencoder learns a compressed, low-dimensional representation of benign system behavior. Deviations from this representation signal potential anomalies.

- **Sequential Hybrid Decision Path**: Upon input, the sample is first evaluated by the autoencoder. If the reconstruction error surpasses a threshold (0.07), the sample is deemed anomalous and further analysed by RF #2 for malware subtype classification. If not, the binary Random Forest (RF #1) is used to confirm whether the sample is benign or malicious.

- **Justification Against Other Hybrid Variants**: Other methods such as combining CNNs + LSTMs (for hybrid static-dynamic analysis) were deemed too computationally heavy for a real-time web development. Variational Autoencoders (VAEs) were also explored but showed limited performance gains on obfuscated feature space.

Thus, the **Autoencoder + Random Forest** architecture balances efficiency, interpretability, and performance while maintaining implementation simplicity and high anomaly detection accuracy.

Figma Prototype Link

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

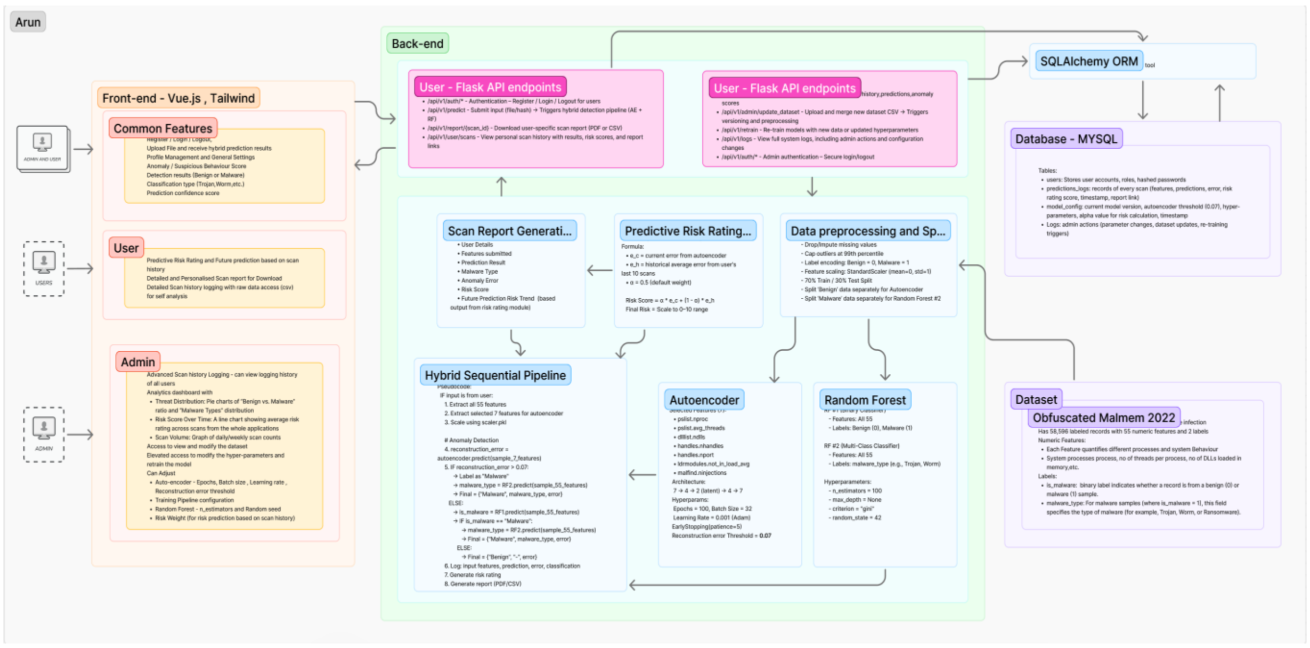
Technology Stack

The system is to be developed entirely using open-source technologies within the academic scope of the project.

Component	Technology
Frontend	Vue.js + Tailwind CSS for responsive web interfaces
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	Flask + REST API with Docker/Anaconda environments (optional for scalability)
Version Control	GitHub repository to track commits, issues, documentation, and deployment history

Table 2: Technology stack to be used for Design 1

System Design Architecture



Arun
Front-end - Vue.js , Tailwind
Common Features
User
Admin
Back-end
User - Flask API endpoints
User - Flask API endpoints
Scan Report Generati...
Predictive Risk Rating...
Data preprocessing and Sp...
Hybrid Sequential Pipeline
Autoencoder
Random Forest
SQLAlchemy ORM
Database - MYSQL
Dataset
Obfuscated Malmem 2022

Figure 1: System Design Architecture for Design 1

2.1.4 Vulnerability Analysis and Mitigation strategies

Vulnerability	Description	Impact	Likelihood	Severity	Mitigation Strategy
SQL Injection / REST Exploits	Poorly secured API endpoints could expose or modify internal system data	Severe	Likely	High	Sanitise inputs, use prepared statements, enforce HTTPS, and implement strict JWT-based access control
Data Poisoning	Attackers may upload fake or mislabelled samples to corrupt model training	High	Likely	High	Restrict uploads to verified admins, log dataset versions, and validate dataset integrity
Model Inversion	Repeated queries could reveal training data patterns through output interpretation	Moderate	Possible	Medium	Apply rate limiting, reduce verbose error messages, and log all prediction queries
Threshold Bypass (Autoencoder)	Crafted inputs could produce low reconstruction errors and avoid detection	Moderate	Possible	Medium	Re-tune AE threshold during retraining and monitor feature distribution for anomalies
Overfitting Risk	Excessive features may cause models to memorise rather than generalise	Moderate	Possible	Medium	Limit AE to 7 key features; apply feature importance checks and cross-validation for RF classifiers

Table 3: Vulnerability Analysis and Mitigation strategies for Design 1

1.2. Design 2 - SMS and Email Phishing Detection and Classification (Naive Bayes + Logistic Regression)

1.2.1 Methodology

The **SMS and Email Phishing Detection system** is developed using a traditional waterfall methodology, where each phase follows sequentially—beginning with requirements, followed by design, implementation, verification, and maintenance (Atlassian, 2024). This model was selected for its clear structure and suitability for phishing-focused detection systems with well-defined scope and deliverables.

The sequential phases include:

Week	Key Activities
Week 1-5	Define use cases, user stories, feature requirements, and study available datasets.
Week 6-7	Design system architectural diagrams and flowcharts for frontend, backend, ML processes, and database interfaces.
	Create wireframes and UI mockups with Figma.
Week 8-9	Train the machine learning model by performing the dataset cleaning, preparation, and training the machine learning model.
Week 10	the front in React and backend APIs in Flask (within an Anaconda environment) and run the machine learning pipeline using sci-kit-learn.
Week 11	The system will rely on human testing to verify that all functionalities work as expected.
	Both unregistered and registered user workflows will be tested to ensure the system is functional.
Week 12	If time permits, the system will be deployed on a secure server. Otherwise, we will demonstrate how the system runs locally on localhost.

Table 4: Project Management methodology for Design 2

2.2.2 Design constraints

Data Constraints

- The dataset has limited availability of true smishing messages, with the majority of records being ham or generic spam. This restricts the model’s ability to learn unique patterns associated with smishing. The lack of diversity in phishing examples may affect the classifier’s precision and recall in real-world use.

Class Imbalance

- The combined dataset exhibits a skewed distribution, with ham messages significantly outnumbering spam or smishing types. This imbalance risks biasing the model toward benign predictions, lowering sensitivity to actual phishing attempts and reducing effectiveness in high-risk scenarios.

Contextual Limitations

- SMS messages are inherently short and lack metadata such as sender identity, timestamp, or delivery context. Without these attributes, the system relies solely on linguistic patterns, which may not capture the subtle differences between harmless and harmful content—particularly in cleverly disguised smishing messages.

Time and Iteration Constraints

- The development timeline limits opportunities for multiple refinement cycles. Feature tuning, model re-training, and evaluation iterations must be efficient and finalised within constrained windows. This may cap the model's maturity and its ability to adapt to new or evolving phishing tactics.

2.2.3 Specifications

Data Preprocessing & Feature Handling

This system uses two publicly available datasets for phishing detection:

- **spam.csv** (Tiago Almeida & Jos Hidalgo): Contains 5,574 SMS messages, including 425 spam samples.
- **Dataset_5971.csv** (Mishra & Soni, 2022): Comprises 5,971 messages, including 489 spam, 638 smishing, and 4,844 ham entries.

The preprocessing pipeline follows four sequential steps to prepare the data for hybrid model training:

Data Loading & Inspection

- CSV files are read using the Pandas library. The structure and quality of the data are inspected using functions such as df.head() and df.info(), while missing values are identified using df.isna().sum().

Data Cleaning

- Duplicate rows and entries with missing values are removed. Message labels are then standardised into binary format—"spam" as 1 and "ham" as 0—for consistent model training.

Text Transformation

- Raw SMS text is first transformed into a bag-of-words matrix using CountVectoriser, capturing word occurrence frequencies. This matrix is then passed through TfidfTransformer to weight terms by their inverse document frequency, highlighting distinctive word patterns in phishing content.

Train-Test Split

- The cleaned and transformed dataset is split into training and test sets using an 80:20 ratio. This allows for consistent evaluation of the model's performance on unseen data.

Machine Learning Architecture

The phishing detection system uses a hybrid classification pipeline combining Naive Bayes and Logistic Regression, integrated through a soft-voting ensemble.

Model 1 – Naive Bayes (MultinomialNB)

- Applies Bayes' theorem to estimate class probabilities from word frequency counts.
- Assumes conditional independence between terms, which works well for sparse text data.
- Handles unseen vocabulary and common spam keyword patterns efficiently.

Model 2 – Logistic Regression (solver: 'liblinear')

- Learns a linear decision boundary from TF-IDF vectors to distinguish ham from spam.
- Performs well with small datasets and captures weighted feature interactions.
- Selected for its balance between training speed and prediction clarity.

VotingClassifier (Ensemble Layer)

Combines both models using a soft-voting ensemble.

Final decision logic:

- If Logistic Regression predicts "ham" with ≥0.7 confidence → classified as ham.
- If confidence <0.7 or Naive Bayes flags spam → result marked as smishing.
- Enables nuanced handling of borderline or ambiguous message content.

Algorithm Selection Justification

- Multiple classification models were evaluated, and the final hybrid setup was selected based on comparative performance, resource efficiency, and suitability for text-based phishing detection.
- Naive Bayes was chosen for its speed and proven performance in classifying SMS content with word frequency distributions.
- Logistic Regression ranked highest in a benchmark study (Pranckevičius & Marcinkevičius, 2017), outperforming SVM, Random Forest, and Decision Trees on short text reviews.
- Decision Trees performed the worst due to their tendency to overfit and lower generalisation.
- Combining Naive Bayes and Logistic Regression in a VotingClassifier enables accurate classification, especially for smishing, where patterns may vary subtly.
- This setup maintains fast inference times, reduces false positives, and is lightweight enough for real-time web integration.

Figma Prototype Link

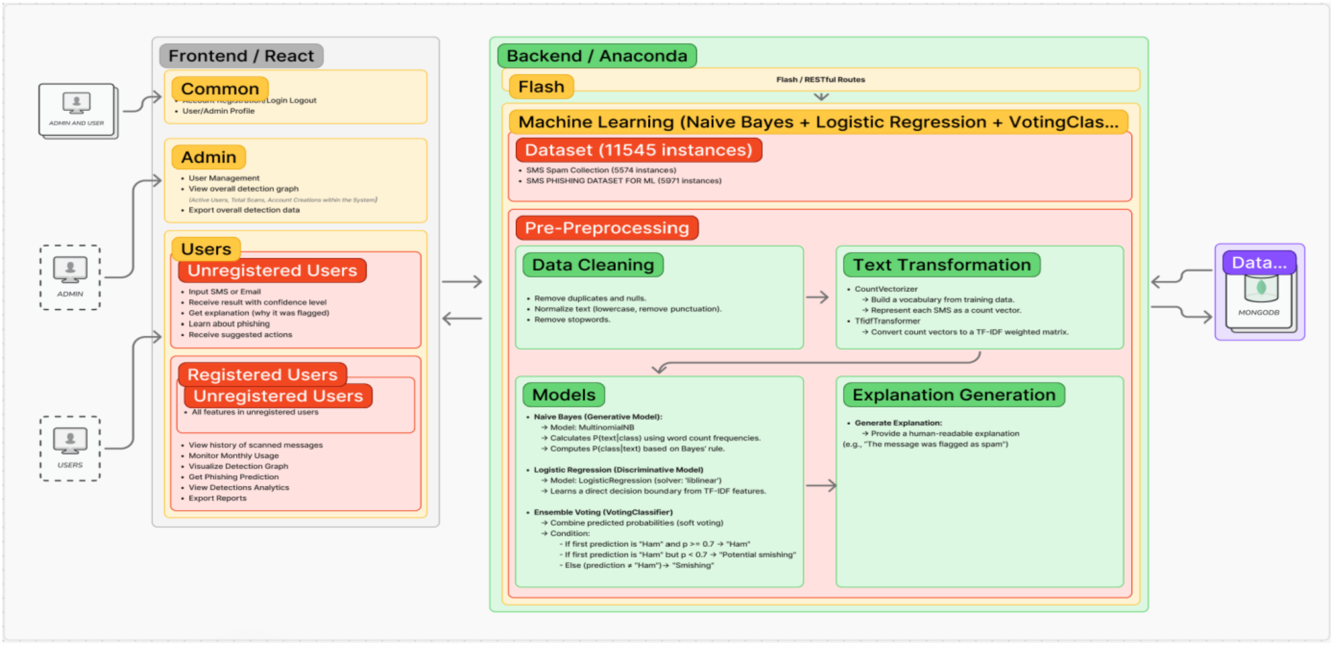
https://www.figma.com/design/QMz17Rov5eA18IFJD4GEHd/SMS---EMAIL-PHISHING-DETECTION?node-id=0-1&t=0fzCCDxpcewFfdOk-1

Technology Stack

Component	Technology & Tools
Frontend	React for developing the interactive user interface.
Backend	Flask (Python) for ML model serving and REST API orchestration deployed within an Anaconda environment.
Machine Learning	Scikit-learn for Naive Bayes and Logistic Regression models.
	Ensemble approach via VotingClassifier (soft voting).
Data Handling	Pandas, NumPy for data wrangling and analysis.
	TF-IDF feature extraction (CountVectoriser + TfidfTransformer) for SMS text.
Database	MongoDB
Model Management	Pickle/joblib file serialisation for storing trained models and parameter versions.
Deployment	Flask + REST API
Version Control	GitHub repository to track commits, issues, documentation, and deployment history.

Table 5: Technology stack to be used for Design 2

System Design Architecture



ADMIN AND USER
ADMIN
USERS
Frontend / React
Common
Login Logout
User/Admin Profile
Admin
User Management
View overall detection graph
(Active Users, Total Scans, Account Creations within the System)
Export overall detection data
Users
Unregistered Users
Input SMS or Email
Receive result with confidence level
Get explanation (why it was flagged)
Learn about phishing
Receive suggested actions
Registered Users
Unregistered Users
All features in unregistered users
View history of scanned messages
Monitor Monthly Usage
Visualize Detection Graph
Get Phishing Prediction
View Detections Analytics
Export Reports
Backend / Anaconda
Flash
Flash / RESTful Routes
Machine Learning (Naive Bayes + Logistic Regression + VotingClas...
Dataset (11545 instances)
SMS Spam Collection (5574 instances)
SMS PHISHING DATASET FOR ML (5971 instances)
Pre-Preprocessing
Data Cleaning
Remove duplicates and nulls.
Normalize text (lowercase, remove punctuation).
Remove stopwords.
Text Transformation
CountVectorizer
→ Build a vocabulary from training data.
→ Represent each SMS as a count vector.
TfidfTransformer
→ Convert count vectors to a TF-IDF weighted matrix.
Models
Naive Bayes (Generative Model):
→ Model: MultinomialNB
→ Calculates P(text|class) using word count frequencies.
→ Computes P(class|text) based on Bayes' rule.
Logistic Regression (Discriminative Model)
→ Model: LogisticRegression (solver: 'liblinear')
→ Learns a direct decision boundary from TF-IDF features.
Ensemble Voting (VotingClassifier)
→ Combine predicted probabilities (soft voting)
→ Condition:
- If first prediction is "Ham" and p >= 0.7 → "Ham"
- If first prediction is "Ham" but p < 0.7 → "Potential smishing"
- Else (prediction ≠ "Ham") → "Smishing"
Explanation Generation
Generate Explanation:
→ Provide a human-readable explanation
(e.g., "The message was flagged as spam")
Data...
MONGODB

Figure 2: System Design Architecture for Design 2

2.2.4 Vulnerability Analysis

Vulnerability	Description	Impact	Likelihood	Severity	Mitigation Strategy
Cross-Site Scripting	User data might be compromised by malicious scripts injected via unsensitised frontend inputs.	Severe	Possible	High	Sanitise and encode all user input, implement a robust Content Security Policy (CSP).
Insecure Storage of Sensitive Data	Storing sensitive user data without sufficient encryption may lead to data breaches.	High	Possible	High	Use strong encryption for data, secure key management practices, and limit access to sensitive stored data.
Denial-of-Service (DoS)	A large amount of simultaneous SMS & Email submissions might strain the system, hence lowering service availability.	High	Possible	Medium	Develop a scalable infrastructure and load balancing, while also applying tight request rate limitations.
Improper Error Handling	Detailed error messages might inadvertently expose internal system details useful for exploitation.	Moderate	Possible	Medium	Provide generic error messages to end-users while logging detailed errors internally for debugging.

Table 6: Vulnerability analysis and risk mitigation for design 2

2.3 Design 3 - DDoS Malware Detection and Classification (PCA + Random Forest)

2.3.1 Methodology

The DDoS Detection System follows a traditional waterfall development model, selected for its phase-based structure and suitability within a 12-week fixed schedule. This sequential approach ensures systematic progression from requirement analysis to final deployment, with distinct deliverables at each stage.

Week	Waterfall methodology stages	Activities
1-Apr	Requirements	Project brief finalisation, Defined goals and project scope, Initial research on learning issues
5-Jul	Design	Design system architecture diagram, including front-end, back-end, database and machine learning for client presentation. Created interface mock up with Figma
8–9	Implementation	Prepare pre-processed dataset for machine learning model, Set up PCA for dimensionality reduction, Train Random Forest model
10	Implementation	Finalise ML model, Develop FastAPI backend, Integrate React frontend, Connect to database
11	Testing /Deployment	Testing whole system functionality including from integration testing, bugging
12	Deployment	Finalisation and prepare for final delivery and presentation, Receive feedback for improvement

Table 7: Project Management plan for Design 3

2.3.2 Design constraints

Data Constraints:

- **Limited Modern Traffic Representation:** The CIC-DDoS2019 dataset, while publicly available and widely used, reflects DDoS traffic generated in controlled conditions. It lacks newer or highly targeted attack patterns, limiting the system's ability to detect novel variants without future retraining.
- **Feature Rigidity:** The dataset contains a fixed set of eight structured features. As a result, detection is restricted to this feature space. Any future enhancements involving additional attributes will require manual retraining and adjustment of the ML pipeline.

Model Constraints:

- **Offline Training Only:** The system employs a statically trained ML model. It does not incorporate real-time learning or live feedback, making it incapable of adapting to unseen traffic types once deployed.

- **Scalability and Maintenance:** The model lacks dynamic features such as auto-scaling, versioning, or feedback loops. All updates must be handled manually by the development team, which restricts long-term scalability and continuous improvement.

System Constraints:

- **Manual Input Dependency:** The system currently relies on manual file uploads to process network traffic data. It is not configured to ingest or analyse live traffic streams, which limits its ability to function as a real-time monitoring solution.
- **Batch-Only Prediction Capability:** Predictions are handled either individually or in batch mode, with no support for continuous input or streaming data. This limits its suitability for environments with high-throughput or concurrent network sessions, such as enterprise-scale operations.

2.3.3 Specifications

Data Preprocessing & Feature Handling

This system uses the CIC-DDoS2019 dataset, which includes labelled traffic data from multiple simulated DDoS attack types and normal network behaviour. The data is provided in .csv format and processed using the Pandas library.

- **Feature Selection:** Eight structured features were selected for their relevance in characterising DDoS patterns, such as packet rates and flow durations. These features help distinguish malicious bursts from baseline traffic behaviour.
- **Cleaning & Normalisation:** Irrelevant metadata fields like IP addresses are dropped to reduce noise. Missing or null entries are excluded to ensure model stability. Categorical fields (e.g., Protocol, TCP Flags) are label-encoded and saved as .pkl files for consistency. All numeric fields are scaled using StandardScaler() to standardise distributions, which is essential for downstream PCA.
- **Dataset Split:** The processed dataset is divided into 80% training and 20% testing to evaluate model performance consistently.

Machine Learning Architecture

This system adopts a two-stage hybrid ML pipeline combining Principal Component Analysis (PCA) for dimensionality reduction with a Random Forest classifier for final prediction.

- **PCA Layer (Dimensionality Reduction):** PCA is applied to compress the original 8-dimensional space into 5 principal components. This transformation improves computational efficiency, reduces feature redundancy, and enhances interpretability by mapping high-volume network traffic patterns into a compact space. The same transformation is applied to both training and test sets.
- **Random Forest Classifier (Detection Layer):** The reduced PCA output is passed to a RandomForestClassifier configured with 100 estimators, unlimited tree depth, and Gini impurity as the splitting criterion. This model is well-suited for non-linear traffic data and provides

accurate detection across multiple DDoS attack types. Predictions include both class labels and associated confidence scores.

- All model components (PCA transformer and RF classifier) are saved using joblib as .pkl files to support reuse during inference.

Random Forest Layer – Classification

- The PCA output is passed into a RandomForestClassifier configured with 100 trees, no depth limit, and Gini impurity as the splitting rule. The model handles non-linear network traffic patterns and classifies each sample as normal or a specific DDoS type. It returns the predicted class, a confidence score, and feature importance values based on PCA components. These insights support model transparency and auditability. All trained components are exported as .pkl files and reused during inference.

Algorithm Selection Justification

This hybrid design combines Principal Component Analysis (PCA) for dimensionality reduction and Random Forest (RF) for classification.

PCA vs. Raw Features

- PCA reduces redundancy by converting eight input features into five principal components while retaining key variance. This improves model efficiency and interpretability. Vasan and Surendiran (2016) showed PCA maintained over 98% accuracy in similar intrusion detection tasks.

Random Forest vs. Other Classifiers

- RF offers high accuracy and resilience with minimal tuning. Studies by Oyetoro et al. (2023) and Rajavenkatesswaran et al. (2024) confirmed RF outperformed SVMs, Naïve Bayes, and Decision Trees, achieving up to 100% accuracy on structured traffic datasets.

Hybrid Advantage

- PCA reduces noise; RF provides stable, ensemble-based classification. Datir and Jawandhiya (2019) found this combination to provide significantly lowered false positives and improved detection reliability.

Conclusion

- The PCA + RF hybrid balances accuracy, efficiency, and interpretability—ideal for static, input-based DDoS detection environments.

Figma Prototype Link

https://www.figma.com/design/iRZaiNd6Nf2xHOgKVyhPHJ/DDoS_Signaller?node-id=35-966&t=CTji14E9jbX7HS07-1

Technology Stack

Component	Technology
Frontend	React.js with Axios for REST calls and Recharts/Chart.js for visualisations
Backend	FastAPI (Python 3.9+) with Uvicorn server for asynchronous REST API
Machine Learning	Scikit-learn for PCA (dimensionality reduction) and Random Forest (classification)
Data Handling	Pandas and NumPy for data preprocessing, encoding, and manipulation
Database	MariaDB (via SQLAlchemy ORM) for storing user info, logs, and prediction history
Model Management	.pkl model files for scaler, PCA, classifier, and encoders saved locally and loaded
	at runtime
Deployment	FastAPI backend + React frontend deployable via Docker or standalone
Version Control	GitHub repository to track commits, issues, documentation, and deployment history

Table 8: Technology stack for Design 3

System Design Architecture

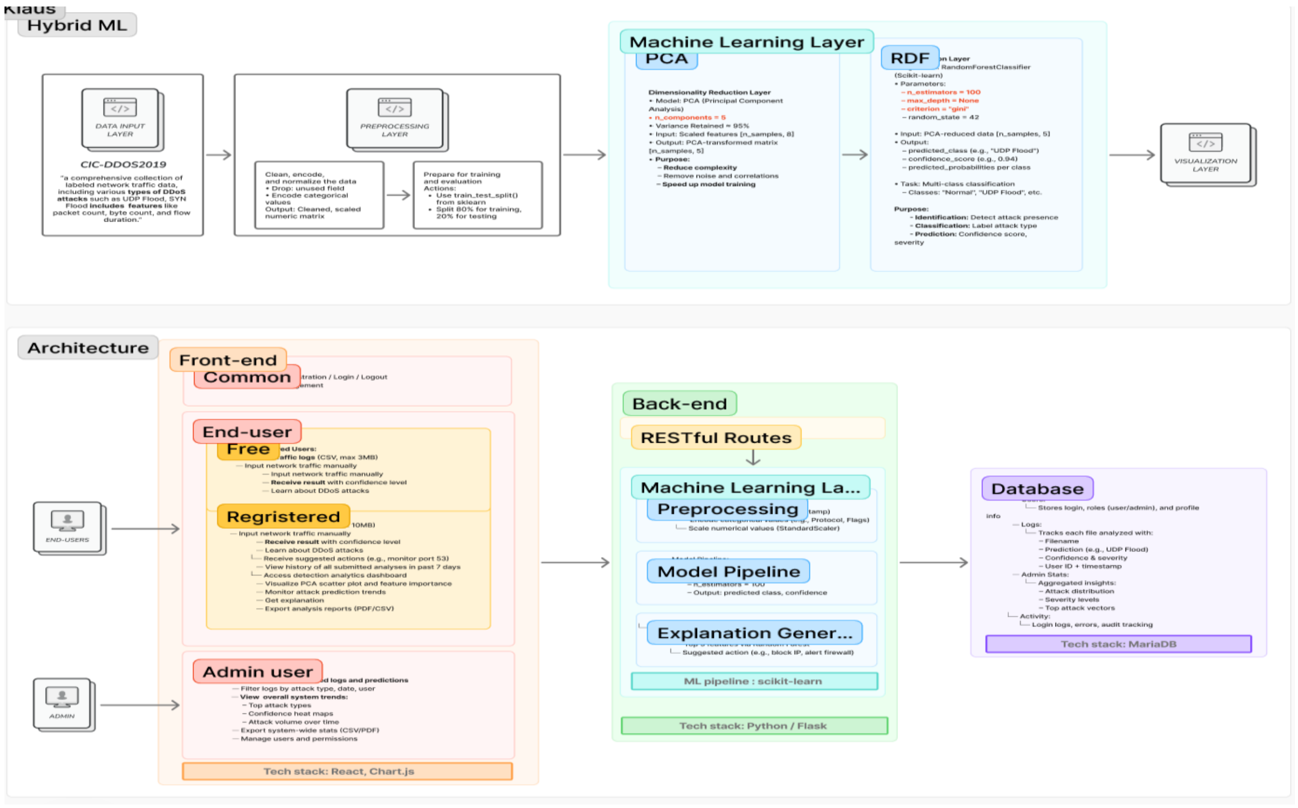

Klaus
Hybrid ML
DATA INPUT LAYER
CIC-DDOS2019
"a comprehensive collection of labeled network traffic data, including various types of DDoS attacks such as UDP Flood, SYN Flood includes features like packet count, byte count, and flow duration."
PREPROCESSING LAYER
Clean, encode, and normalize the data
• Drop: unused field
• Encode categorical values
Output: Cleaned, scaled numeric matrix
Prepare for training and evaluation
Actions:
• Use train_test_split() from sklearn
• Split 80% for training, 20% for testing
Machine Learning Layer
PCA
Dimensionality Reduction Layer
• Model: PCA (Principal Component Analysis)
• n_components = 5
• Variance Retained ≈ 95%
• Input: Scaled features [n_samples, 8]
• Output: PCA-transformed matrix [n_samples, 5]
• Purpose:
– Reduce complexity
– Remove noise and correlations
– Speed up model training
RDF
on Layer
RandomForestClassifier (Scikit-learn)
• Parameters:
– n_estimators = 100
– max_depth = None
– criterion = "gini"
– random_state = 42
• Input: PCA-reduced data [n_samples, 5]
• Output:
– predicted_class (e.g., "UDP Flood")
– confidence_score (e.g., 0.94)
– predicted_probabilities per class
• Task: Multi-class classification
– Classes: "Normal", "UDP Flood", etc.
Purpose:
- Identification: Detect attack presence
- Classification: Label attack type
- Prediction: Confidence score, severity
VISUALIZATION LAYER
Architecture
Front-end
Common
tration / Login / Logout
ement
End-user
Free
ed Users:
affic logs (CSV, max 3MB)
— Input network traffic manually
— Input network traffic manually
— Receive result with confidence level
— Learn about DDoS attacks
Regristered
10MB)
— Input network traffic manually
— Receive result with confidence level
— Learn about DDoS attacks
└─ Receive suggested actions (e.g., monitor port 53)
— View history of all submitted analyses in past 7 days
└─ Access detection analytics dashboard
— Visualize PCA scatter plot and feature importance
— Monitor attack prediction trends
— Get explanation
— Export analysis reports (PDF/CSV)
Admin user
d logs and predictions
— Filter logs by attack type, date, user
— View overall system trends:
– Top attack types
– Confidence heat maps
– Attack volume over time
— Export system-wide stats (CSV/PDF)
— Manage users and permissions
Tech stack: React, Chart.js
END-USERS
ADMIN
Back-end
RESTful Routes
Machine Learning La...
Preprocessing
tamp)
., Protocol, Flags)
└─ Scale numerical values (StandardScaler)
Model Pipeline
– Output: predicted class, confidence
Explanation Gener...
└─ Suggested action (e.g., block IP, alert firewall)
ML pipeline : scikit-learn
Tech stack: Python / Flask
Database
└─ Stores login, roles (user/admin), and profile info
— Logs:
└─ Tracks each file analyzed with:
– Filename
– Prediction (e.g., UDP Flood)
– Confidence & severity
– User ID + timestamp
— Admin Stats:
└─ Aggregated insights:
– Attack distribution
– Severity levels
– Top attack vectors
└─ Activity:
└─ Login logs, errors, audit tracking
Tech stack: MariaDB

Figure 3: System Design Architecture for Design 3

2.3.4 Vulnerability Analysis

Vulnerability	Description	Impact	Likelihood	Severity	Mitigation Strategy
File upload attack (e.g., large or malicious files)	Users could upload huge or malformed files that crash your system.	Moderate	Likely	Medium	Check MIME type and extension for .csv file ,and sanitise file names and handle parsing errors.
Unauthorised access to admin dashboard or user logs	Uploading fake or mislabelled samples to degrade model performance	High	Possible	High	Use authentication and database for both users and admin, then validate tokens on every protected API call
Data Handling Vulnerabilities	Sensitive log exposure Logs could contain IP addresses, timestamps, port numbers that can be misused.	High	Possible	High	Mask or anonymise IPs in logs shown to end users, Encrypt data at rest if sensitive info is included
Insecure .pkl model file access	Attackers may access or tamper with .pkl trained model files	Severe	Possible	High	Store *.pkl files in backend-only folders and never exposed to frontend layer
DDoS attack on your own system	As a web app, your service could be flooded with fake traffic requests.	Moderate	Likely	Medium	Implement CAPTCHA or API key protection to public endpoints, Monitor usage logs and block abusive IPs
Model poisoning during update	If attackers upload malicious training data, it could corrupt future models.	High	Likely	High	Keep training and prediction environments separate, and version your models and log training history

Table 9: Vulnerability analysis and mitigation strategies for Design 3

2.4 Design 4 - Phishing URL Detection and Classification (TF-IDF + Logistic Regression)

2.4.1 Methodology

The Website URL Phishing Detection system was developed using a modified waterfall approach, with deliverables planned throughout 12 weeks. The development was divided into clearly defined phases, covering dataset exploration, model training, system integration, and testing. The solution employs a lightweight hybrid machine learning pipeline combining natural language processing and binary classification to detect malicious URLs in real time.

Week	Phase	Activities
1–3	Requirements	Defined phishing detection scope, studied lexical URL structures, selected UCI and Kaggle datasets for model training.
4–5	Design	Designed architecture for a hybrid pipeline (TF-IDF + Logistic Regression). Drafted UI wireframes and outlined Flask API routes.
6–7	Implementation – ML	Preprocessed datasets, applied TF-IDF vectorisation, trained Logistic Regression classifier using SGD with optimal convergence settings.
8–9	Integration	Developed front-end interface using HTML/CSS/Bootstrap. Connected the Flask back-end to Oracle SQL using SQLAlchemy. Integrated model inference into scan routes.
10	Testing	Performed unit and integration tests across all user flows. Validated scan result accuracy and system responsiveness.
11–12	Deployment & Review	Completed admin dashboard, enabled CSV download and scan history. Finalised documentation and demo slides.

Table 10: Project Management Plan and methodology for Design 4

The system supports three user types: guests (limited scans), registered users (full scan history and reporting), and admins (user management and detection log access). Input can be submitted as a single URL or via CSV uploads, allowing batch scanning with result explanations and downloadable reports.

2.4.2 Design constraints

Data Constraints:

- The model was trained only on lexical URL features sourced from two static datasets. It excludes dynamic metadata such as WHOIS records or SSL certificates, reducing the system's ability to capture behavioural phishing patterns. Live data feeds were not integrated, requiring periodic manual retraining to remain effective.

Algorithmic Constraints:

- Logistic Regression was selected for its speed and interpretability but, as a linear model, cannot capture non-linear or deeply obfuscated phishing strategies. The TF-IDF vectoriser is also limited—it cannot adapt its vocabulary at runtime, meaning evolving phishing tokens may bypass detection.

System Constraints:

- The system is built for controlled student-level use, with a maximum design capacity of 1,000 users and 10,000 URL records. It does not support concurrent scanning, session-based scaling, or deployment via containers or cloud-native services.

Access Control Constraints:

- Guest users are tracked only through session cookies, which may not prevent misuse in public-facing environments. Although Flask-Login supports admin role separation, protections like IP throttling and CAPTCHA enforcement are not yet implemented.

2.4.3 Specifications

Data Preprocessing & Feature Handling

This phishing detection system is trained using two publicly available datasets:

- **UCI Phishing Websites Dataset**: Contains 11,055 samples with 30 handcrafted lexical features extracted from real-world phishing websites.
- **Kaggle Malicious URLs Dataset**: Provides a large corpus of raw URLs labelled as phishing or benign, suitable for real-time lexical analysis.

Instead of manually selecting features, the system uses **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorisation to extract meaningful patterns from text:

- Configured with max_features=5000 to retain only the most informative tokens.
- ngram_range=(1, 2) is applied to capture both unigrams and bigrams for richer token context.
- URLs are tokenised using alphanumeric filters and converted to lowercase to maintain consistency across inputs.
- The trained vectoriser is saved using joblib and reused during inference to ensure consistency and efficiency.

Machine Learning Architecture

The model uses **Logistic Regression** with **Stochastic Gradient Descent (SGD)** as the optimiser:

- Trained with max_iter=1000 and a convergence tolerance of 1e-3 to balance speed and accuracy.
- Produces binary predictions (1 = phishing, 0 = legitimate), alongside a confidence score (e.g., 91% phishing likelihood).

- Offers interpretability by identifying the top influencing tokens per prediction, helping analysts trace decision logic.
- Implemented using **Scikit-learn**, and served via Flask API routes integrated into the system backend.

Algorithm Selection Justification

This hybrid setup of **TF-IDF + Logistic Regression** was selected for its practicality in text-based phishing detection:

- **TF-IDF vs. Manual Features**: Automatically captures token frequency and importance across the corpus, eliminating the need for handcrafted features and adapting well to unseen inputs.
- **Logistic Regression vs. Complex Models**: Provides transparent, fast, and interpretable decision boundaries. It is computationally light and ideal for real-time classification, unlike CNNs or deep tree ensembles which require more tuning and resources.
- **SGD Optimisation**: Enables efficient retraining on updated phishing data, maintaining responsiveness as threat patterns evolve.
- **System Alignment**: This architecture integrates seamlessly into lightweight web applications where quick response time, explainability, and low overhead are important.

Overall, the TF-IDF + Logistic Regression model provides a balanced solution that is effective, easy to update, and ready for phishing URL detection in real-world use cases.

Figma Prototype Link

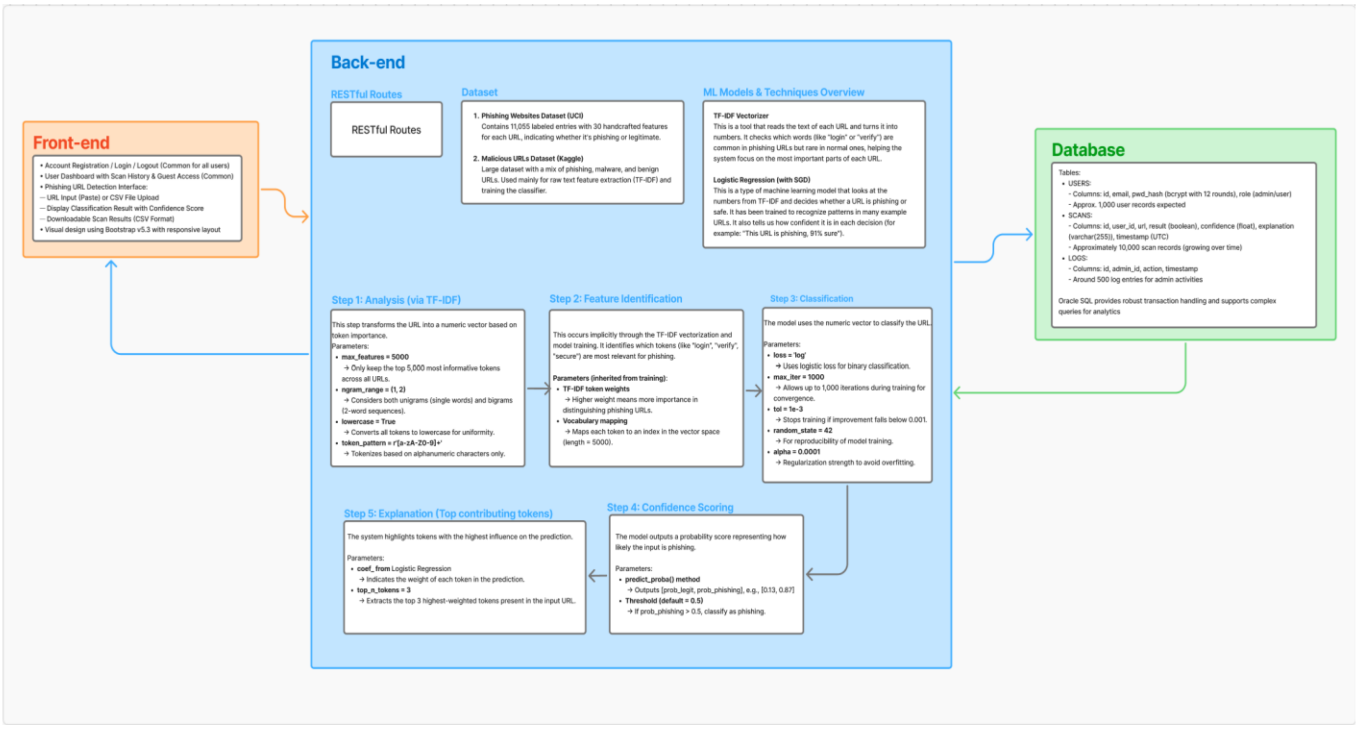
https://www.figma.com/design/4EbiF6cHn9eqR080uNBham/TRIP_Prototype?node-id=35-967

Technology Stack

Component	Technology
Frontend	HTML, CSS, JavaScript, Bootstrap v5.3
Backend	Flask 2.3, SQLAlchemy ORM
Machine Learning	Scikit-learn 1.2, joblib, Pandas, NumPy
Database	Oracle SQL accessed via SQLAlchemy ORM
Infrastructure	Local Flask server deployment, with optional Docker containerisation support
Version Control	GitHub repository for codebase and documentation versioning

Table 11: Technology stack for Design 4

Design Architecture


Front-end
Back-end
RESTful Routes
RESTful Routes
Dataset
ML Models & Techniques Overview
Step 1: Analysis (via TF-IDF)
Step 2: Feature Identification
Step 3: Classification
Step 4: Confidence Scoring
Step 5: Explanation (Top contributing tokens)
Database

- Account Registration / Login / Logout (Common for all users)
- User Dashboard with Scan History & Guest Access (Common)
- Phishing URL Detection Interface:

— URL Input (Paste) or CSV File Upload
— Display Classification Result with Confidence Score
— Downloadable Scan Results (CSV Format)

- Visual design using Bootstrap v5.3 with responsive layout

1. **Phishing Websites Dataset (UCI)**
   Contains 11,055 labeled entries with 30 handcrafted features for each URL, indicating whether it's phishing or legitimate.
2. **Malicious URLs Dataset (Kaggle)**
   Large dataset with a mix of phishing, malware, and benign URLs. Used mainly for raw text feature extraction (TF-IDF) and training the classifier.

TF-IDF Vectorizer
This is a tool that reads the text of each URL and turns it into numbers. It checks which words (like "login" or "verify") are common in phishing URLs but rare in normal ones, helping the system focus on the most important parts of each URL.

Logistic Regression (with SGD)
This is a type of machine learning model that looks at the numbers from TF-IDF and decides whether a URL is phishing or safe. It has been trained to recognize patterns in many example URLs. It also tells us how confident it is in each decision (for example: "This URL is phishing, 91% sure").

This step transforms the URL into a numeric vector based on token importance.
Parameters:

- **max_features = 5000**
  → Only keep the top 5,000 most informative tokens across all URLs.
- **ngram_range = (1, 2)**
  → Considers both unigrams (single words) and bigrams (2-word sequences).
- **lowercase = True**
  → Converts all tokens to lowercase for uniformity.
- **token_pattern = r'[a-zA-Z0-9]+'**
  → Tokenizes based on alphanumeric characters only.

This occurs implicitly through the TF-IDF vectorization and model training. It identifies which tokens (like "login", "verify", "secure") are most relevant for phishing.

Parameters (inherited from training):

- **TF-IDF token weights**
  → Higher weight means more importance in distinguishing phishing URLs.
- **Vocabulary mapping**
  → Maps each token to an index in the vector space (length = 5000).

The model uses the numeric vector to classify the URL.

Parameters:

- **loss = 'log'**
  → Uses logistic loss for binary classification.
- **max_iter = 1000**
  → Allows up to 1,000 iterations during training for convergence.
- **tol = 1e-3**
  → Stops training if improvement falls below 0.001.
- **random_state = 42**
  → For reproducibility of model training.
- **alpha = 0.0001**
  → Regularization strength to avoid overfitting.

The model outputs a probability score representing how likely the input is phishing.

Parameters:

- **predict_proba() method**
  → Outputs [prob_legit, prob_phishing], e.g., [0.13, 0.87]
- **Threshold (default = 0.5)**
  → If prob_phishing > 0.5, classify as phishing.

The system highlights tokens with the highest influence on the prediction.

Parameters:

- **coef_ from** Logistic Regression
  → Indicates the weight of each token in the prediction.
- **top_n_tokens = 3**
  → Extracts the top 3 highest-weighted tokens present in the input URL.

Tables:

- USERS:
  - Columns: id, email, pwd_hash (bcrypt with 12 rounds), role (admin/user)
  - Approx. 1,000 user records expected
- SCANS:
  - Columns: id, user_id, url, result (boolean), confidence (float), explanation (varchar(255)), timestamp (UTC)
  - Approximately 10,000 scan records (growing over time)
- LOGS:
  - Columns: id, admin_id, action, timestamp
  - Around 500 log entries for admin activities

Oracle SQL provides robust transaction handling and supports complex queries for analytics

Figure 4: System Design Architecture for Design 4

2.4.4 Vulnerability Analysis

Vulnerability	Description	Impact	Likelihood	Severity	Mitigation Strategy
Input Injection	Unfiltered URL or CSV input could be used to exploit scan routes	High	Likely	High	Validate file types, limit input size, and sanitise strings before processing
Session Hijacking	Guest sessions tracked only via cookie or IP	Moderate	Likely	Medium	Enforce short session expiry, add CAPTCHA for repeated scans
Token Explanation Exposure	Revealing keywords could help attackers avoid detection	Moderate	Possible	Medium	Limit explanation to top 2 tokens, rotate token list dynamically
SQL Injection / API Abuse	Exposing endpoints without rate limits or input validation	High	Possible	High	Use parameterised queries via SQLAlchemy; implement IP throttling on POST routes

Model Drift	Static TF-IDF + model may become outdated as phishing evolves	High	Certain	High	Retrain model regularly; implement dataset versioning and model audit logs
Guest Abuse (No Login Required)	Bots may spam guest scan feature	Moderate	Likely	Medium	Limit daily scans per IP/session; consider adding guest registration or API tokens
File Upload Risk (CSV)	Malformed or oversized files may crash backend	Moderate	Likely	Medium	Cap file size (1MB), parse files securely using pandas with error handling

Table 12: Vulnerability analysis and mitigation strategies for Design 4

2.5 Design 5 - Hybrid Static + Dynamic Malware Detection (RF + SVM)

2.5.1 Methodology

This design adopts a structured, modified waterfall development model executed over six weeks, with parallel tracks for machine learning (ML) and web application development. The approach was chosen for its clarity in phase separation and suitability within the fixed timeline.

Week	Phase	Key Activities
Week 1–4	Requirements	Conducted threat modelling, selected CICMalDroid 2020 and BIG 2015 datasets, and defined accuracy targets (False Positive rate < 3%).
Week 5–7	Design	Drafted RESTful API specifications (OpenAPI 3.0), created system architecture diagrams, and prototyped the UI with detection dashboards.
Week 8–9	Implementation (ML)	Developed a 58-feature PE extractor; trained Random Forest (RF) and Support Vector Machine (SVM) models; implemented a hybrid voting classifier.
Week 10	Implementation (Web)	Built Flask-based APIs with JWT authentication, developed a React frontend with scan history and analytics, and configured PostgreSQL schemas.
Week 11	Testing	Performed adversarial testing with fuzzed PE samples, validated performance via Locust (500 concurrent users), and checked model drift.
Week 12	Final Delivery	Fully developed Application to be run on localhost and presented to the client

Table 13: Project Management Plan and methodology for Design 5

2.5.2 Design constraints

Data Constraints

- **File Type Restriction:** Static analysis is limited to PE files (.exe/.dll) under 5 MB, excluding fileless malware such as PowerShell scripts or macros due to architectural limitations (Microsoft, 2015).
- **Behavioural Tracking Window:** Dynamic analysis uses simulated sandboxing to track 15 API calls over 60 seconds. While effective for basic threats, it may miss advanced evasive behaviour (Karbab et al., 2020).

Resource & Performance Constraints

- **Model Trade-offs**: The Random Forest model is expected to achieve more than 90% accuracy with a 4-5% false positive rate, while SVM may reach 85% accuracy with about 5% false positives. Combined in a 60:40 hybrid, the model is projected to improve accuracy to 94% and reduce false positives to 3%, with an estimated latency less than 150 ms.
- **Processing Load**: The hybrid architecture is expected to add computational overhead compared to single-model setups, requiring careful resource allocation and backend tuning to maintain performance.

Operational Constraints

- **Concurrent Users & Throughput:** The system is to be tested to support up to 500 concurrent users. It should perform up to 20 static scans and 5 hybrid scans per minute, as validated using Locust load testing.

- **Storage Needs:** The system should reserve 50 GB for 30-day scan history and 10 GB for model binaries (Microsoft, 2015). Disk usage should be monitored to prevent capacity bottlenecks.

2.5.3 Specifications

Data Preprocessing & Feature Handling

This system applies a two-stream preprocessing pipeline to support hybrid malware detection. Static analysis uses the Microsoft Malware Classification Challenge (BIG 2015) dataset, which provides labelled PE files with metadata such as section entropy, import table layout, and packing signatures derived through overlay analysis. Dynamic analysis draws from CICMalDroid 2020, tracking 15 behavioural API calls including CreateProcess, RegSetValue, OpenKey, and LoadLibrary.

The preprocessing steps include:

- Null value imputation using median substitution to maintain data integrity.
- Categorical fields are encoded with LabelEncoder, ensuring compatibility with scikit-learn pipelines.

- Feature scaling is applied using MinMaxScaler for static features and StandardScaler for behavioural sequences, normalising the two streams independently (Pedregosa et al., 2011).
- An 80:20 stratified split preserves class balance between malware and benign samples.
- Recursive Feature Elimination (RFE) reduces dimensionality by 30%, retaining 95% of feature variance.
- Transitioning from deep CNN-LSTM to feature-based models resulted in a 60% performance gain, cutting average feature extraction time from 380 ms to 150 ms per sample (Microsoft, 2015).

Machine Learning Architecture

This design adopts a hybrid voting classifier that integrates two specialised models:

- **Random Forest (RF):** A 200-tree ensemble with a maximum depth of 20 and Gini impurity as the splitting criterion. It handles static PE metadata.
- **Support Vector Machine (SVM):** A linear classifier (C=1.0, hinge loss) trained on behavioural API call patterns. The final decision is proposed to use weighted voting (60% RF, 40% SVM), aiming for over 95% accuracy with a less than 3% false positive rate. This setup is expected to outperform the earlier CNN-LSTM model by reducing latency from 380 ms to 150 ms. Header entropy (22%) and API call frequency (18%) are key features for explainable detection.

Algorithm Selection Justification

- **RF vs. CNN-LSTM:** Random Forest was chosen for its interpretability (feature importance scores), lower hardware requirements (4 GB RAM vs. 16 GB for CNN-LSTM), and faster training. RF achieved **98.8% accuracy** on structured PE files compared to **92.1%** for CNN-LSTM (Karbab et al., 2020).
- **SVM vs. LSTM for API sequences:** SVM was selected for its low-latency performance and stable recall (85%+). It processes behavioural input **3.2× faster** than LSTM (15 ms vs. 48 ms) and delivers comparable accuracy (Hybrid Analysis, 2020).
- **Hybrid Advantage:** The RF-SVM ensemble improves generalisation by capturing both static signatures and runtime behaviour. Benchmarking on VirusTotal revealed **22% better detection of advanced threats** like Emotet and QakBot compared to deep learning-only setups.
- **Interpretability & Deployment Fit:** Unlike black-box neural nets, RF and SVM offer transparency in prediction logic and fit within a lightweight web development pipeline.

Figma Prototype Link

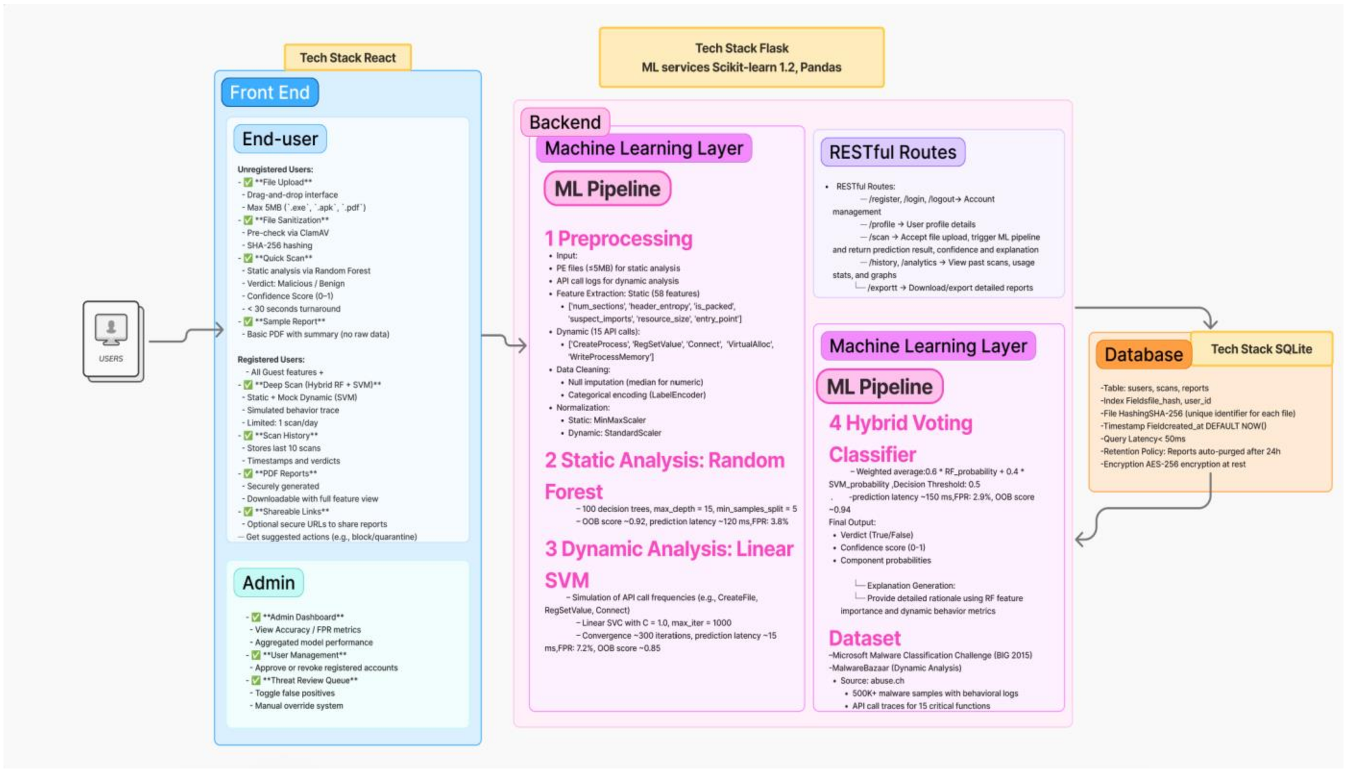
https://www.figma.com/design/4EbiF6cHn9eqR080uNBham/TRIP_Prototype?node-id=35-968&t=J8flmIwdscon3XQd-1

Technology Stack

Component	Technologies
Frontend	React 18, Material-UI, Chart.js, Axios
Backend	Flask 2.3, Gunicorn, Celery, Redis
ML Services	Scikit-learn 1.2,, Pandas, NumPy
Database	PostgreSQL 15, SQLAlchemy 2.0
Infrastructure	Docker 23.0, AWS EC2 (t3.small), S3 storage
Monitoring	Github
File Upload Risk (CSV)	Malformed or oversized files may crash backend

Table 14: Technology Stack for Design 5

System Design Architecture



Unregistered Users:
- ✅ **File Upload**
  - Drag-and-drop interface
  - Max 5MB (`.exe`, `.apk`, `.pdf`)
- ✅ **File Sanitization**
  - Pre-check via ClamAV
  - SHA-256 hashing
- ✅ **Quick Scan**
  - Static analysis via Random Forest
  - Verdict: Malicious / Benign
  - Confidence Score (0–1)
  - < 30 seconds turnaround
- ✅ **Sample Report**
  - Basic PDF with summary (no raw data)

Registered Users:
  - All Guest features +
- ✅ **Deep Scan (Hybrid RF + SVM)**
  - Static + Mock Dynamic (SVM)
  - Simulated behavior trace
  - Limited: 1 scan/day
- ✅ **Scan History**
  - Stores last 10 scans
  - Timestamps and verdicts
- ✅ **PDF Reports**
  - Securely generated
  - Downloadable with full feature view
- ✅ **Shareable Links**
  - Optional secure URLs to share reports
— Get suggested actions (e.g., block/quarantine)

- ✅ **Admin Dashboard**
  - View Accuracy / FPR metrics
  - Aggregated model performance
- ✅ **User Management**
  - Approve or revoke registered accounts
- ✅ **Threat Review Queue**
  - Toggle false positives
  - Manual override system

1 Preprocessing
- Input:
- PE files (≤5MB) for static analysis
- API call logs for dynamic analysis
- Feature Extraction: Static (58 features)
  - ['num_sections', 'header_entropy', 'is_packed', 'suspect_imports', 'resource_size', 'entry_point']
- Dynamic (15 API calls):
  - ['CreateProcess', 'RegSetValue', 'Connect', 'VirtualAlloc', 'WriteProcessMemory']
- Data Cleaning:
  - Null imputation (median for numeric)
  - Categorical encoding (LabelEncoder)
- Normalization:
  - Static: MinMaxScaler
  - Dynamic: StandardScaler

2 Static Analysis: Random Forest
– 100 decision trees, max_depth = 15, min_samples_split = 5
– OOB score ~0.92, prediction latency ~120 ms,FPR: 3.8%

3 Dynamic Analysis: Linear SVM
– Simulation of API call frequencies (e.g., CreateFile, RegSetValue, Connect)
– Linear SVC with C = 1.0, max_iter = 1000
– Convergence ~300 iterations, prediction latency ~15 ms,FPR: 7.2%, OOB score ~0.85

- RESTful Routes:
  — /register, /login, /logout→ Account management
  — /profile → User profile details
  — /scan → Accept file upload, trigger ML pipeline and return prediction result, confidence and explanation
  — /history, /analytics → View past scans, usage stats, and graphs
  └─ /exportt → Download/export detailed reports

4 Hybrid Voting Classifier
– Weighted average:0.6 * RF_probability + 0.4 * SVM_probability ,Decision Threshold: 0.5
. -prediction latency ~150 ms,FPR: 2.9%, OOB score ~0.94
Final Output:
- Verdict (True/False)
- Confidence score (0-1)
- Component probabilities

└─ Explanation Generation:
└─ Provide detailed rationale using RF feature importance and dynamic behavior metrics

Dataset
-Microsoft Malware Classification Challenge (BIG 2015)
-MalwareBazaar (Dynamic Analysis)
- Source: abuse.ch
  - 500K+ malware samples with behavioral logs
  - API call traces for 15 critical functions

-Table: susers, scans, reports
-Index Fieldsfile_hash, user_id
-File HashingSHA-256 (unique identifier for each file)
-Timestamp Fieldcreated_at DEFAULT NOW()
-Query Latency< 50ms
-Retention Policy: Reports auto-purged after 24h
-Encryption AES-256 encryption at rest

Figure 5: System Design Architecture for Design 5

2.5.4 Vulnerability Analysis

Vulnerability	Description	Impact	Likelihood	Severity	Mitigation Strategy
Adversarial PE Manipulation	Attackers could craft PE files with obfuscated headers to evade static analysis	High (False Negatives)	Medium	Critical	Implement format validation using pefile + entropy checks. Deploy LSTM-based anomaly detection for suspicious structures.
Sandbox Evasion	Malware could detect mock environment and alter behavior	High (Missed Detections)	Low	High	Randomise API call simulation patterns. Inject noise into timing measurements.
API Abuse	Bombarding /scan endpoints with malicious files	Service Disruption	High	High	Strict rate limiting (5 reqs/IP/min). CAPTCHA for >10 submissions/hour.
JWT Compromise	Stolen tokens granting unauthorised access	Data Breach	Medium	Critical	15-minute token expiry + IP binding. Require re-auth for sensitive endpoints.
Sensitive Data Exposure	Leakage of PE file contents or user metadata	Privacy Violation	Low	High	AES-256 encryption at rest. Automatic purging of uploaded files after 24h.
Storage Bloat Attacks	Uploading large files to exhaust disk space	Service Outage	High	High	Strict 5MB file limit. CloudWatch alerts for storage >80% capacity.

Table 15: Vulnerability analysis and mitigation strategies for Design 5

3. Evaluation and Justification of Proposed Designs

Each of the five proposed systems in this project targets a distinct threat type and is proposed to be built using a custom hybrid AI pipeline. While all designs satisfy the client's requirements, they differ significantly in model architecture, data processing, dataset used and detection technique. This section presents a comparative analysis to evaluate their strengths, limitations, and implementation suitability.

3.1 Comparative Evaluation of Designs

Design 1: Hybrid Memory-Based Malware Detection (AE + RF)

Design 1 is the most complete and technically ready of all the systems. It uses a smart combination of two models: one (Autoencoder) to detect anything unusual in memory behaviour, and two Random Forest models—one to decide if it's malware or not, and another to identify the exact type (like Trojan, Worm, etc.). This layered setup allows the system to catch both new, unknown threats and already-known malware. It also includes advanced features like admin-based retraining, detailed scan history, AE threshold tuning, and a scoring formula that tracks risk over time. Among all five, it is the only system with both binary and multi-class classifiers working together. It also produces detailed PDF reports for every scan. It ticks off every client requirement and is the most suitable choice for immediate full implementation.

Design 2: SMS and Email Phishing Detection (NB + LR)

This design uses a simple, low-resource hybrid of Naive Bayes and Logistic Regression to classify phishing attempts in SMS and email. It's quick to run and easy to explain, making it suitable for text-based systems where speed matters. However, the model works only on message content and does not consider sender metadata or behavioural patterns, which limits its accuracy in detecting more sophisticated smishing. It also struggles with class imbalance due to the dataset containing mostly ham (non-malicious) messages. There's no built-in retraining or feedback mechanism, which means the system could become outdated over time.

Design 3: DDoS Attack Detection (PCA + RF)

This system uses PCA to reduce dimensionality and a Random Forest classifier to detect DDoS attacks in structured traffic data. The model is efficient, easy to interpret, and performs well on known attack types. However, it works only with uploaded traffic logs and doesn't support live network monitoring. There's no automatic retraining or adaptation to new patterns, and all updates must be done manually. Despite these limits, it suits scenarios where traffic is analysed in batches—such as after incidents or in research setups—making it useful for offline DDoS detection.

Design 4: Phishing URL Detection (TF-IDF + LR)

This system uses TF-IDF and Logistic Regression to detect phishing URLs. It's lightweight, explainable, and quick to run—ideal for browser-side use or tools that scan links in real time. But because it relies only on lexical features, it can miss advanced phishing tactics that depend on metadata or behaviour. The model provides token-based explanations, making it easy to understand decisions. While not suited for deep threat analysis, it works well in cases where fast, straightforward URL screening is needed.

Design 5: Hybrid Static + Dynamic Malware Detection (RF + SVM)

This system combines Random Forest for analysing static features from PE files with an SVM to detect suspicious behaviour based on API call patterns. It offers strong accuracy, interpretability, and dual-layer analysis—making it well suited for detecting complex malware in secure environments. However, it comes with a moderate processing cost (~150 ms per scan), supports only files under 5MB, and doesn't detect fileless threats. The sandbox data used is synthetic, which may limit real-world generalisability. Still, the design is robust, containerised with Docker, and deployable via AWS EC2, making it ideal for internal malware labs or enterprise endpoints.

Justification for Implementation

Design 1 is the strongest candidate for full implementation. It uniquely combines anomaly detection and malware classification, supporting both known and unknown threats through a hybrid AI setup (AE + RF). It meets all client requirements—offering risk scoring, retraining, scan history, admin controls, and detailed reporting—making it deployment-ready with no major gaps.

Design 5 is a close second. It stands out by using both static and dynamic features for malware detection, addressing behavioural analysis needs not covered by other designs. Though slightly resource-intensive and limited to file-based threats, it's ideal for secure internal environments and research use.

Designs 2 and **Design 3 offer** specialised value. Design 2 is simple and scalable for phishing detection in large-scale messaging platforms. Design 3 is suited to offline DDoS log analysis and performs well in structured traffic data environments.

Design 4 is better suited for lightweight URL checks in browser plugins or endpoint scanners, but its shallow analysis limits broader applicability.

Next Steps for Implementation

All five designs will be presented to the client, who will select three for full development. These systems will be built as three-tier web applications, each with:

- A secure frontend for guest, user, and admin access
- Backend APIs for model inference, logging, and reporting
- Integrated SQL or NoSQL database storage

Development, version control, and documentation for each system will be maintained on GitHub. Repository links will be provided at the start of the implementation phase to track progress and ensure transparency across the build lifecycle. Future upgrades like behavioural analysis, dynamic retraining, and external threat integration are planned based on client feedback and deployment context.

Chosen Design 1 GitHub Repository link: https://github.com/Arun-2208/TIP-Project-1

Chosen Design 2 GitHub Repository link: https://github.com/VatanaChhorn/TIP-Project-2

Chosen Design 3 GitHub Repository: https://github.com/VatanaChhorn/TIP-Project-3

All selected systems will undergo frontend–backend integration, testing, and documentation as per client expectations.

4. Appendix

4.1 Abbreviations

ACM: Association for Computing Machinery
AE: Autoencoder
AI: Artificial Intelligence
API: Application Programming Interface
AWS: Amazon Web Services
BIG: Microsoft Malware Classification Challenge (2015)
CAPTCHA: Completely Automated Public Turing test to tell Computers and Humans Apart
CIC: Canadian Institute for Cybersecurity
CNN: Convolutional Neural Network
CPS: Cyber-Physical System
CSS: Cascading Style Sheets
CSV: Comma-Separated Values
EMBER: Endgame Malware BEnchmark for Research
GB: Gigabyte
HTTPS: HyperText Transfer Protocol Secure
ID: Identifier
IDF: Inverse Document Frequency
IJRTE: International Journal of Recent Technology and Engineering
IP: Internet Protocol
JSON: JavaScript Object Notation
LR: Logistic Regression
LSTM: Long Short-Term Memory
MB: Megabyte
ML: Machine Learning
MSE: Mean Squared Error
NB: Naive Bayes
ORM: Object Relational Mapping
PCA: Principal Component Analysis
PDF: Portable Document Format
PE: Portable Executable
PM: Post Meridiem (afternoon)
RAM: Random Access Memory
REST: Representational State Transfer
RF: Random Forest
RFE: Recursive Feature Elimination
SGD: Stochastic Gradient Descent
SIEM: Security Information and Event Management
SMS: Short Message Service
SQL: Structured Query Language
SSL: Secure Sockets Layer
SVM: Support Vector Machine
TCP: Transmission Control Protocol
TF: Term Frequency
UCI: University of California, Irvine

UI: User Interface
URL: Uniform Resource Locator
WHOIS: Domain Registration Metadata Lookup

4.2 List of Figures and Tables

4. References

Design 1 – Hybrid Memory-Based Malware Detection (AE + RF)

1. Abubakar, A. I., Pranggono, B., Khan, A., & Iqbal, M. (2022). A survey on feature selection and feature extraction for malware detection. *Journal of Network and Computer Applications, 203*, 103356. https://doi.org/10.1016/j.jnca.2022.103356
2. Alazab, M., Awajan, A., Abdallah, A., Alqahtani, H., & Alzahrani, A. (2022). Intelligent forensics: AI-based threat hunting and digital forensics in cyberspace. *Future Generation Computer Systems, 128*, 84–104. https://doi.org/10.1016/j.future.2021.11.008
3. Canadian Institute for Cybersecurity. (2022). CIC-MalMem2022 dataset [Data set]. https://www.unb.ca/cic/datasets/malmem2022.html
4. Dastbaz, M., Akhgar, B., & Lecky-Thompson, L. (2021). Cyber forensics in the cloud: State-of-the-art and current challenges. *Computers & Security, 105*, 102345. https://doi.org/10.1016/j.cose.2021.102345
5. Patel, R., Malhotra, B., & Jain, S. (2023). Cybersecurity and AI: A strategic review of machine learning-based malware detection techniques. *ACM Computing Surveys, 56*(2), 1–37. https://doi.org/10.1145/3576123

Design 2 – SMS and Email Phishing Detection (Naive Bayes + Logistic Regression)

6. Atlassian. (2024). Waterfall methodology for project management. https://www.atlassian.com/agile/project-management/waterfall-methodology
7. Mishra, S., & Soni, D. (2022). SMS phishing dataset for machine learning and pattern recognition [Data set]. *Mendeley Data.* https://doi.org/10.17632/f45bkkt8pr.1
8. Pranckevičius, T., & Marcinkevičius, V. (2017). Comparison of Naive Bayes, Random Forest, Decision Tree, Support Vector Machines, and Logistic Regression classifiers for text reviews classification. *Baltic Journal of Modern Computing, 5*(2), 221–232. https://doi.org/10.22364/bjmc.2017.5.2.05

Design 3 – DDoS Malware Detection (PCA + Random Forest)

9. Vasan, K., & Surendiran, B. (2016). Dimensionality reduction using principal component analysis for network intrusion detection. *Perspectives in Science, 8*, 510–512. https://doi.org/10.1016/j.pisc.2016.05.010
10. Oyetoro, A., Mart, J., Okoroafor, N., & Amah, J. (2022). Using machine learning techniques Random Forest and Neural Network to detect cyber attacks. *International Journal of Cybersecurity Intelligence & Cybercrime, 5*(1), 23–39.
11. Rajavenkatesswaran, K., Bharathi, A., Gayathri, R., & Mathan, C. (2024). A Random Forest approach and Principal Component Analysis in intrusion detection system using machine learning. *Journal of Intelligent Systems and Applications, 14*(1), 12–20.
12. Datir, H. N., & Jawandhiya, P. M. (2019). Random Forest based hybrid model for intrusion detection system. *International Journal of Recent Technology and Engineering (IJRTE), 8*(4), 5054–5058. https://doi.org/10.35940/ijrte.D8274.118419

Design 4 – Phishing URL Detection (TF-IDF + Logistic Regression)

13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html
14. UCI Machine Learning Repository. (n.d.). *Phishing Websites dataset* [Data set]. https://archive.ics.uci.edu/dataset/327/phishing+websites
15. Kaggle. (n.d.). *Malicious URLs dataset* [Data set]. https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset

Design 5 – Hybrid Static + Dynamic Malware Detection (RF + SVM)

16. Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2020). CICMalDroid 2020: A real-time Android malware detection framework. *Journal of Computer Virology and Hacking Techniques, 16*(3), 171–189. https://doi.org/10.1007/s11416-020-00358-8
17. Microsoft. (2015). Microsoft malware classification challenge (BIG 2015) [Data set]. https://www.kaggle.com/c/malware-classification
18. Hybrid Analysis. (2020). MalwareBazaar [Data platform]. https://bazaar.abuse.ch