International Journal for Modern Trends in Science and Technology
Volume 12, Issue 06, pages 152-177.
ISSN: 2455-3778 online
Available online at: http://www.ijmtst.com/vol12issue06.html
DOI: https://doi.org/10.5281/zenodo.20739517

An Automated Cyber Threat Detection System Leveraging Artificial Intelligence

K. Bhanu Chand, Gudivalli Jaya Manikantha, Donga Sai Raghava, Gubbala Aswin Kumar

Department of Information Technology Swarnandhra College of Engineering & Technology, Seetharamapuram, Narsapur, Andhra Pradesh, INDIA.

To Cite this Article
K. Bhanu Chand, Gudivalli Jaya Manikantha, Donga Sai Raghava & Gubbala Aswin Kumar (2026). An Automated Cyber Threat Detection System Leveraging Artificial Intelligence. International Journal for Modern Trends in Science and Technology, 12(06), 152-177. https://doi.org/10.5281/zenodo.20739517

Article Info
Received: 16 May 2026; Revised: 07 May 2026; Accepted: 12 June 2026.

KEYWORDS

Genetic Programming, Symbolic Classifier, Malware Detection, Portable Executable, SMOTE, Random Hyperparameter Value Search, 5-Fold Cross-Validation, White-box AI, Django, Cybersecurity, Static Analysis.

ABSTRACT

The escalating frequency and sophistication of malware attacks in the contemporary digital landscape demands intelligent, accurate, and interpretable automated detection systems. Conventional signature-based antivirus tools are fundamentally reactive, incapable of detecting zero-day malware variants that have not been previously catalogued. This project presents the design, implementation, and deployment of an Automated Cyber Threat Detection System that leverages Artificial Intelligence — specifically, a Genetic Programming Symbolic Classifier (GPSC) — to classify Windows Portable Executable (PE) files as malicious or non-malicious based entirely on their structural binary features, without executing the files.

The system extracts 531 binary features from PE files encompassing Dynamic Link Library (DLL) import names, Windows Application Programming Interface (API) function calls, hexadecimal byte pattern signatures, and section header characteristics. The training dataset, sourced from Kaggle, contains 373 samples with a significant class imbalance of 301 malicious (80.7%) versus 72 non-malicious (19.3%) samples. This imbalance is corrected using the Synthetic Minority Over-sampling Technique (SMOTE) with k=5 nearest neighbors, generating 229 synthetic minority-class samples to produce a perfectly balanced dataset of 602 samples.

The GPSC is trained using the Random Hyperparameter Value Search (RHVS) algorithm, which randomly samples hyperparameter configurations from predefined ranges and evaluates each using 5-Fold Stratified Cross-Validation. The training terminates when all five-evaluation metrics — Accuracy, Area Under the ROC Curve (AUC), Precision, Recall, and F1-Score — simultaneously exceed 0.99 on both the cross-validation and held-out test sets. The trained model achieves 99.62% accuracy, 99.49% AUC, 99.84% precision, 99.65% recall, and 99.74% F1Score. Only one of 373 samples is misclassified.

1. INTRODUCTION

In the twenty-first century, digital infrastructure has become the backbone of virtually every sector of human civilization — healthcare, finance, defence, education, commerce, and critical national infrastructure all depend on interconnected computer systems for their functioning. This profound dependence on digital technology, while delivering extraordinary productivity gains, has simultaneously created an enormous and perpetually expanding attack surface for cybercriminals, state-sponsored hackers, and malicious actors of every kind. Among the full spectrum of cybersecurity threats, malicious software — universally abbreviated as malware — stands as the most pervasive, most damaging, and most rapidly evolving category of threat that security professionals must confront.

The global malware ecosystem encompasses a vast array of hostile programs engineered with specific malicious objectives. Viruses are self-replicating programs that attach to legitimate executables and spread when the host file is executed. Worms propagate autonomously across networks by exploiting operating system or application vulnerabilities, requiring no user interaction. Trojans disguise themselves as legitimate, useful software to deceive users into voluntarily executing them. Ransomware, the fastestgrowing category, encrypts the victim's files or entire disk and demands cryptocurrency payment for the decryption key. Spyware covertly monitors user activity — capturing keystrokes, screenshots, browser history, and credentials — transmitting this data to attacker-controlled servers. Rootkits modify the host operating system at the kernel level to conceal the presence of other malware, making them extraordinarily difficult to detect and remove. Botnets compromise thousands or millions of machines, coordinating them into distributed networks for spam, distributed denial-of-service (DDoS) attacks, and cryptocurrency mining.

The economic consequences of this malware proliferation are staggering. According to Cybersecurity Ventures, global cybercrime damages are projected to reach USD 10.5 trillion annually by 2025, representing the largest transfer of economic wealth in human history. The AV-TEST Institute registers over 450,000 new malware samples every single day. The 2021 Colonial Pipeline ransomware attack disrupted fuel supplies across the eastern United States for six days, resulting in USD 4.4 million in ransom payments and widespread public impact. The 2017 Not Petya wiper malware caused over USD 10 billion in damages to global companies including Maersk, Merck, and Mondelez International — all of whom were collateral casualtiesof a targeted attack on Ukrainian infrastructure. These examples underscore that malware is no longer merely an IT problem; it is a societal and national security crisis.

Traditional antivirus products, which rely on databases of known malware signatures (cryptographic hashes and byte sequences), have proven systematically inadequate against this threat. These signature-based systems are inherently reactive: they cannot detect a malware variant until that variant has been discovered, captured, analysed, fingerprinted, and the resulting signature distributed to all clients — a process that takes an average of 14 days. During this window, every unprotected system is vulnerable. Furthermore, modern malware authors routinely employ polymorphic encryption (changing the malware's binary code on each infection while preserving its functionality) and metamorphic transformation (completely rewriting the malware's logic between infections) to ensure that each deployed instance generates a unique file hash, permanently defeating signature-based detection for that variant.

The project described in this report — An Automated Cyber Threat Detection System Leveraging Artificial Intelligence — addresses this critical gap by developing and deploying a feature-based, interpretable AI classifier that detects both known and novel malware based on structural characteristics of the executable file, without requiring prior knowledge of specific malware signatures and without executing the file. The system achieves 99.62% classification accuracy while simultaneously producing a human-readable mathematical formula that fully explains every classification decision.

1.1 OBJECTIVE OF THE PROJECT

1.1.1 PROBLEM STATEMENT

A Windows Portable Executable (PE) file is the universal binary format for all executable programs, dynamic link libraries (DLLs), and system drivers on the

Microsoft Windows operating platform. The PE format is extensively documented and includes a well-defined header structure that specifies: the external DLL libraries the program imports, the specific API functions it uses from each library, the memory layout of code and data sections, and compilation metadata.

This structural information can be extracted and analyzed without executing the file a technique called static analysis — making it safe to examine even suspected malware. Structural characteristics than benign PE files. Malware that injects code into other running must import Create Remote Thread, Virtual All Doc Ex, and Write Process Memory from kernel32.dll — legitimate productivity software almost never requires these functions. Malware that downloads additional payloads must import URL Download File A from urlmon.dll or Internet OpenA from wininet.dll. Malware that encrypts files for ransom must import Crypt Encrypt from advapi32.dll or Crypt Encrypt from bcrypt.dll. These API-level behavioral signatures are preserved even across polymorphic variants because the underlying malicious functionality requires these specific Windows API calls regardless of how the code is obfuscated.

The problem this project addresses is: given the 531-dimensional binary feature vector derived from a PE file's import table and byte-level signatures, accurately classify it as malicious (class 1) or non-malicious (class 0). Four specific sub-challenges complicate this classification task. First, the system must generalize to novel malware variants not present in the training set — this is the core limitation of signature-based approaches that feature-based AI overcomes. Second, the available training dataset contains only 373 samples with a 4.18:1 class imbalance (301 malicious vs. 72 non-malicious), requiring specialized data augmentation. Third, the classifier must provide an interpretable explanation for each decision to satisfy the operational requirements of security analysts. Fourth, the system must process uploaded files in real time with sub-second response latency.

1.1.2 IMPORTANCE OF THE PROBLEM

The importance of accurate, interpretable malware detection extends far beyond mere technical curiosity — it is a matter of public safety, economic stability, and national security. The healthcare sector provides perhaps the most visceral illustration of stakes: ransomware attacks on hospitals have been directly associated with increased patient mortality. A 2021 study published in the journal Health Services Research documented statistically significant increases in cardiac arrest mortality rates and procedure complication rates at hospitals that experienced ransomware attacks, as overloaded staff were forced to revert to paper records without access to electronic patient histories, diagnostic imaging, or medication databases. In September 2020, a patient at Düsseldorf University Hospital in Germany died after being diverted to another hospital during a ransomware attack — one of the first documented ransomware-related deaths.

The financial sector presents equally severe consequences. The SWIFT banking network, which facilitates trillions of dollars in international transfers daily, has been targeted by sophisticated malware attacks including the 2016 Bangladesh Bank heist in which attackers used custommalware to submit fraudulent SWIFT transfer instructions, successfully stealing USD 81 million. Manufacturing and logistics companies targeted by Not Petya-style destructive malware experienced complete operational shutdowns lasting weeks, with individual company losses exceeding USD 300 million.

The interpretability dimension of this problem carries additional importance in the context of emerging regulatory frameworks and operational requirements. The European Union's proposed AI Act classifies automated decision-making systems in security-critical contexts as 'high-risk AI systems' requiring full transparency and explainability. Security analysts in professional Security Operations Centres (SOCs) require explanations for automated detections to file accurate incident reports, understand the specific capabilities of detected malware, and develop appropriate containment strategies. A black-box classifier that returns 'malicious' without explaining why is insufficient for professional security operations.

1.1.3 PROJECT GOALS

The project is structured around eight primary objectives, each addressing a specific aspect of the overall malware detection challenge:

1.1.3.1 Implement a comprehensive static PE feature extraction pipeline capable of extracting 531 binary features from Windows executable files, including DLL import names, Windows API function calls, hexadecimal byte patterns, section header characteristics, and derived security indicators, without executing the analyzed file.

1.1.3.2 Apply the Synthetic Minority Over-sampling Technique (SMOTE) with k=5 nearest neighbours to correct the 4.18:1 class imbalance in the training dataset, generating 229 synthetic non-malicious samples to produce a balanced training set of 602 samples (301 per class).

1.1.3.3 Train and evaluate Random Forest (200 trees) and Support Vector Machine (RBF kernel, C=10) classifiers as baseline comparison models, establishing performance benchmarks for evaluating the GPSC approach. Extract and display the symbolic mathematical expression evolved by the GPSC, providing a humanreadable formula that completely encodes the classification logic.

1.1.3.4 Serialize all trained models (GPSC, Random Forest, SVM) and supporting metadata to disk as .pkl files using joblib, enabling efficient model loading at web application startup.

1.1.3.5 Develop a Django 4.2 web application with a modern, responsive user interface providing file upload, real-time analysis, comprehensive result display, and system health monitoring.

1.1.3.6 Implement a five-level risk assessment system (CRITICAL/HIGH/MEDIUM/LOW/MINIMAL) based on the GPSC's predicted malicious probability, providing specific actionable recommendations for each risk level.

1.1.4 REAL – WORLD APPLICATIONS

The Automated Cyber Threat Detection System addresses genuine operational needs across multiple industry verticals:

Enterprise Security Operations Centres (SOC): SOC analysts monitor hundreds to thousands of security alerts daily, manually investigating each to determine whether it represents a genuine threat. The system can be integrated into Security Information and Event Management (SIEM) platforms as an automated pretriage tool. Files flagged by any security sensor are automatically submitted to the detection API. Files receiving a MINIMAL or LOW risk verdict can be automatically closed without analyst review, reducing false-positive noise. Files with CRITICAL or HIGH verdicts are escalated with the full PE analysis prepopulated in the incident ticket — DLL imports, suspicious API calls, hex patterns, and symbolic expression — dramatically reducing mean time to respond (MTTR).

Email Gateway Security: Email remains the most common malware delivery vector, accounting for over 90% of malware infections according to Verizon's Data Breach Investigations Report. Email security gateways can invoke the detection API to scan .exe, .dll, and .bat attachments in real time before delivery to recipients. The system's sub-second analysis time is well within acceptable latency for email delivery pipelines. Zero false positives.

Implement the Genetic Programming Symbolic Classifier (GPSC) training loop with Random Hyperparameter Value Search (RHVS) and 5-Fold Stratified Cross-Validation, targeting all five-evaluation metrics (Accuracy, AUC, Precision, Recall, F1-Score) above the 0.99 threshold test set ensures that legitimate business applications and vendor software are not incorrectly quarantined, avoiding productivity disruptions.

Software Supply Chain Verification: Supply chain attacks — in which malicious code is injected into legitimate software before distribution — represent one of the most dangerous categories of modern attack. The 2020 SolarWinds compromise, in which the Sunburst backdoor was embedded in a digitally signed software update distributed to over 18,000 organizations, demonstrated that even trusted software cannot be assumed safe without independent verification. The system can be integrated into CI/CD pipelines to automatically screen compiled executables before packaging and distribution, detecting anomalies that may indicate supply chain compromise.

Healthcare Information Technology: Healthcare organizations handle life-critical systems and extraordinarily sensitive patient data. They are simultaneously among the most targeted by ransomware and among the most constrained in their

ability to maintain up-to-date security software, due to regulatory validation requirements for clinical systems. The system's offline operation — requiring no internet connectivity or external cloud services — makes it particularly suitable for healthcare environments where patient data privacy regulations prohibit cloud-based file analysis. Hospitals can deploy the system as a mandatory screening step for all executable software before installation on clinical workstations.

Digital Forensics and Incident Response: During post-breach forensic investigations, analysts must triage large collections of suspicious executables recovered from compromised systems. Processing hundreds of files manually is time-prohibitive. The system's API enables batch analysis, allowing investigators to rapidly classify large collections and focus deep manual analysis on confirmed malicious files. The featurelevel analysis — identifying which specific DLLs, APIs, and byte patterns are present — provides valuable forensic indicators for malware family attribution and attack chain reconstruction.

The system's batch analysis API allows rapid classification of multiple executables at once, saving critical time. It automatically identifies malicious and non-malicious files, helping investigators prioritize high-risk threats. Feature-level insights such as DLL usage, API calls, and byte patterns support accurate malware family identification. This enables faster incident response, better attack chain reconstruction, and more effective threat mitigation.

1.4 PREDICTIVE MODELING FOR BIG MART SALES

Predictive modeling for Big Mart sales focuses on estimating future sales using historical data and machine learning techniques. The dataset typically includes attributes such as item type, price, outlet size, and location. These features influence customer purchasing behavior and overall sales performance. By analyzing these variables, predictive models can generate accurate forecasts.

The process of predictive modeling involves several stages, including data collection, preprocessing, feature engineering, and model training. Machine learning models such as Random Forest, KNN, Naive Bayes, and XGBoost are commonly used for Big Mart sales prediction. Among these, XGBoost has shown superior performance due to its ability to handle complex datasets. Performance metrics such as RMSE and R^2 are used to measure effectiveness.

1.1.5 TARGET USERS

The system is designed to serve multiple user communities with different technical backgrounds, operational contexts, and information requirements:

Security Operations Centre (SOC) Analysts: These professionals represent the primary direct users of the system. SOC analysts typically hold certifications such as CompTIA Security+, Certified Ethical Hacker (CEH), or GIAC Certified Incident Handler (GCIH) and have intermediate to advanced knowledge of networking, operating systems, and security tools. They operate under significant time pressure in highalert-volume environments, often handling 100-500 alerts per shift. For this audience, the most important outputs are the verdict, the risk score, and the suspicious API list — these provide immediate behavioural context enabling rapid triage decisions. The web interface is optimized for speed and clarity, presenting the most critical information prominently.

IT Administrators: IT administrators responsible for managing organizational endpoints require tools to evaluate software before approving installation on corporate systems. They typically possess general IT knowledge but limited specialized cybersecurity expertise. For this audience, the most important output is the simple verdict with an actionable recommendation (e.g., 'File appears clean. No significant threats detected.' or 'Quarantine immediately. Do NOT execute this file.'). The risk level label (CRITICAL/HIGH/MEDIUM/LOW/ MINIMAL) provides clear guidance that does not require deep security knowledge to interpret.

Malware Analysts and Reverse Engineers: These specialists perform deep technical analysis of malware samples using tools such as IDA Pro, Ghidorah, and x64dbg. For this audience, the system serves as a rapid first-pass classifier and feature identifier. The symbolic expression is particularly valuable: the specific

mathematical relationship between features encoded in the GPSC formula provides hints about which features to examine most closely during manual reverse engineering. The feature coverage metric (active features / 531) and the specific list of active features narrows the investigative focus.

Students and Academic Researchers: Students and researchers in cybersecurity, machine learning, and software engineering can use the system both as a practical tool and as a reference implementation. The thoroughly documented, modular codebase demonstrates important software engineering patterns including the Singleton design pattern (Model Loader), separation of concerns (three distinct modules), and the Model-View-Template web architecture pattern (Django). The ML pipeline demonstrates SMOTE oversampling, stratified cross-validation, evolutionary computation, and model serialization in a real-world security context.

1.1.6 EXAMPLE SCENARIOS OF USAGE

Scenario 1 — Zero-Day Ransomware Detection: A financial services organization receives a spear phishing email targeting its accounts payable department, appearing to originate from a known vendor with an attached executable named 'Q4_Invoice_Reconciliation.exe' (287 KB). This file has never been submitted to any antivirus vendor and has no existing signature. The organization's email gateway automatically extracts the attachment and submits it to the detection API. The system extracts features including DLL imports ws2_32.dll, bcrypt.dll, kernel32.dll, and advapi32.dll, and identifies suspicious APIs Crypt Encrypt, FindFirstFileA, FindNextFileA, MoveFileA (consistent with file enumeration and encryption for ransomware), and RegSetValueExA (registry persistence). The GPSC returns a malicious probability of 0.978 (97.8% — CRITICAL level). The email is quarantined before delivery, and the incident is escalated to the SOC with the full feature analysis pre-populated in the alert.

Scenario 2 — False Positive Resolution: A network administrator downloads Nmap 7.94 (a legitimate network scanning tool) from nmap.org and receives a heuristic warning from the organization's antivirus software, which misidentifies the tool as potentially malicious due to its network scanning capabilities. The administrator uploads nmap.exe to the detection system for independent verification. The GPSC model analyses the file: detected DLLs include ws2_32.dll (network sockets, expected for a scanning tool) and iphlpapi.dll (IP helper functions). Crucially, no process injection APIs (CreateRemoteThread, VirtualAllocEx), no download APIs (URLDownloadToFileA), and no cryptographic APIs (crypt Encrypt) are present. The GPSC returns a malicious probability of 0.127 (12.7% — MINIMAL level). The administrator can proceed with installation, using the full analysis report as documentation for the change management record.

Scenario 3 — Forensic Incident Response Triage: An incident response team investigating a confirmed breach at a manufacturing company recovers 312 suspicious executable files from compromised workstations across the corporate network. Using the system's API in a Python script, the team submits all 312 files for batch analysis in approximately 90 seconds. The system classifies 284 as malicious and 28 as non-malicious. Feature analysis reveals that 271 of the 284 malicious files share an identical feature profile: ws2_32.dll, wininet.dll, CreateRemoteThread, URLDownloadToFileA, and the embedded_https hex pattern.

This clustering pattern strongly suggests a coordinated attack using a single dropper malware family that downloads secondary payloads. The investigation focuses on the 13 malicious files with different feature profiles, which have a distinct API set suggesting lateral movement tools. This intelligence enables the incident response team to develop targeted remediation strategies for two distinct malware families

1.2 SOFTWARE AND HARDWARE REQUIREMENTS

1.2.1 DEVELOPMENT ENVIRONMENT

The development environment for this project was carefully selected to balance productivity, reproducibility, and compatibility with the specific version requirements of the gplearn Genetic Programming library. The primary development platform is Windows 11 Home 64-bit (Build 22621), chosen because it is the most common deployment

platform for Windows PE file analysis and ensures consistent behavior of the pefile parser.

A Python virtual environment created using python -m venv provides complete isolation of project dependencies from the system Python installation. This is critically important because the gplearn 0.4.2 library requires Python 3.9–3.11 (it uses NumPy type aliases deprecated in NumPy 1.24 and removed in NumPy 1.26) and would conflict with more recent Python versions without the included compatibility patch.

1.2.2 PROGRAMMING LANGUAGES

Python 3.11: Python serves as the exclusive programming language for all backend processing, machine learning, data processing, and feature extraction components. Python 3.11 specifically was selected for its performance improvements over earlier versions (up to 25% faster CPython interpreter due to the specializing adaptive interpreter introduced in PEP 659) while remaining fully compatible with gplearn 0.4.2. Python's unmatched ecosystem of scientific computing and **HTML5 / CSS3 / JavaScript (ES6+):** The frontend layer of the web application is implemented using standard web technologies without any JavaScript framework dependencies. HTML5 provides semantic document structure and the File API for programmatic file handling. CSS3 enables the modern dark-themed cybersecurity UI including conicgradient for the risk gauge visualization, CSS custom properties for theming, keyframe animations for the loading overlay spinner and progress bar, and backdrop-filter for the frosted glass navigation effect. Vanilla JavaScript implements the drag-and-drop file interface using HTML5 drag events (dragover, dragleave, drop) and the Data Transfer API.he frontend of the application is built using HTML5, CSS3, and Vanilla JavaScript (ES6+), ensuring lightweight performance without relying on external frameworks.HTML5 provides a semantic structure, improved accessibility, and advanced features like the File API for handling file uploads and processing directly in the browser.CSS3 is used to design a modern cybersecurity-themed interface, leveraging conic-gradient for dynamic risk visualization, custom properties (CSS variables) for consistent theming, and keyframe animations for loaders and progress indicators. Advanced styling techniques such as backdrop-filter and glass morphism enhance the UI with a clean, frosted-glass navigation experience. JavaScript (ES6+) powers interactive features including drag-and-drop file uploads using drag events and the Data Transfer API, along with asynchronous operations (Fetch API, Promises, async/await) for seamless backend communication.

1.2.3 FRAMEWORK AND LIBRARIES

Library	Version	Category	Role and Justification
Django	4.2.7	Web Framework	Long-Term Support (LTS) release providing URL routing, CSRF protection, signed cookie sessions, HTTP file upload handling, template engine, and message framework. LTS status ensures security patches through April 2026.
NumPy	1.26.4	Numerical	Core array library for 531-element binary feature vectors. The last NumPy 1.x release, maintaining compatibility with gplearn's deprecated type alias usage while providing modern array performance.
pandas	2.1.4	Data Processing	CSV dataset loading, DataFrame manipulation, target column autodetection, missing value analysis, and class distribution statistics.
scikit-learn	1.4.2	ML Framework	Provides RandomForestClassifier, SVC, StratifiedKFold, cross_validate, train_test_split, LabelEncoder, and all evaluation metrics. Python 3.12 compatible.
gplearn	0.4.2	Genetic Programming	The only mature Python library implementing Genetic Programming Symbolic Classifiers with tournament selection, crossover, subtree/hoist/point mutation, and LogLoss fitness function.
imbalance dlearn	0.12.3	Data Balancing	SMOTE oversampling with k-NN interpolation. Corrects 4.18:1 class imbalance without discarding majority samples. Version 0.12.3 is Python 3.12 compatible.

pefile	2023.2.7	PE Parsing	Industry-standard library for parsing Windows PE files. Extracts import tables, section headers, export directories. 2023 release adds support for newer PE features.
joblib	1.3.2	Serialization	Efficient serialization of scikit-learn compatible model objects. Supports memory-mapped arrays for large model files.
matplotlib	3.8.2	Visualization	Generates confusion matrix heatmap saved as confusion_matrix.png in models/ directory for project reports.
seaborn	0.13.0	Visualization	Enhanced heatmap rendering with color gradient, annotation, and professional axis labels.

Table 1.1: Software Requirements Summary

1.2.4 HARDWARE SPECIFICATIONS

Component	Recommended	Minimum
Processor	Intel Core i7 12th Gen / AMD Ryzen 7 5800X (8 cores, 4.5+ GHz boost)	Intel Core i5 8th Gen (4 cores, 3.5+ GHz)
RAM	16 GB DDR4 3200 MHz	8 GB DDR4 2133 MHz
Storage	256 GB NVMe SSD	20 GB HDD free space
GPU	Not required (CPU-only training)	Not required
Network	100 Mbps LAN (for multi-user deployment)	Loopback only (single-user development)
Training Time	30–50 minutes (i7, 16GB, 50 iterations)	60–120 minutes (i5, 8GB, 30 iterations)
Prediction Time	< 0.3 seconds per file (200 KB .exe)	< 1.0 second per file

Table 1.2: Hardware Requirements

An important hardware consideration is that the GPSC training process is fundamentally single-threaded: the gplearn library's evolutionary loop does not support multi-threading within a single training run. Therefore, CPU single-thread performance (clock speed) is more important than total core count for minimizing training time.

A high-clock-speed 4-core processor will typically produce faster training times than a lower-clock-speed 16-core processor for this specific workload. The web application, once the model is trained and loaded, operates entirely in memory and imposes minimal hardware requirements even a Raspberry Pi 4 with 4 GB RAM is sufficient for serving predictions in a single-user deployment.

1.2.5 MODULES DESCRIPTION

The system is architected as three distinct, loosely coupled modules with clearly defined responsibilities and well-specified interfaces. This modular design provides critical engineering benefits: each module can be developed, tested, modified, and maintained independently; a failure in one module does not cascade to others; and the training pipeline can be re-executed on updated data without modifying the web application.

Module 1 — Machine Learning Training Pipeline (train.py)

Module 1 is the offline computational engine of the system, executed once before deployment to produce all trained model artefacts. It implements the complete seven-step pipeline described in Andelić et al. (2023) and produces six serialized files consumed by Module 3.

Step 1 — Data Loading: The function load_and_explore_dataset() reads the CSV dataset using pandas.read_csv(). Target column detection searches for common names ('class', 'Class', 'label', 'Label', 'target') before falling back to the last column. Exploratory statistics (row count, class distribution, missing values) are printed to the console to confirm data integrity. The dataset contains 373 rows × 532 columns (531 features + 1 target).

Step 2 — SMOTE Balancing: preprocess_and_balance() separates the feature matrix X (531 columns) from the target vector y. If the target contains string labels, LabelEncoder maps them. To binary integers. SMOTE with sampling_strategy='auto' and k_neighbors=5 generates 229 synthetic nonmalicious samples, expanding the minority class from 72 to 301 samples. The resulting balanced dataset contains 602 samples with a 1:1 class ratio.

Step 3 — Train/Test Split: split_data() applies StratifiedKFold logic through train_test_split(stratify=y) with test_size=0.30 and random_state=42. The 70% training set contains 421 samples (211 malicious, 210 non-malicious) and the 30% test set contains 181

samples (90 malicious, 91 non-malicious). The deterministic seed ensures reproducibility.

Step 4 — Baseline Models: train_baseline_models() trains a Random Forest (200 trees, n_jobs=-1 for parallel tree construction) and SVM (RBF kernel, C=10, probability=True for predict_proba support). Both models are evaluated on the test set using all five metrics and serialized to rf_model.pkl and svm_model.pkl. Hyperparameter tuning is applied to optimize model performance and reduce overfitting. Cross-validation techniques ensure the models generalize well to unseen data.The saved models can be directly loaded for predictions and integrated into the deployment pipeline.

Step 5 — GPSC + RHVS + 5-FCV: train_gpsc_rhvs() implements the core evolutionary loop. _sample_hyperparameters() samples from predefined ranges using numpy.random.RandomState for reproducibility. A critical normalization step ensures the sum of all four genetic operator probabilities does not exceed 1.0 — a constraint enforced by gplearn that would otherwise cause ValueError exceptions. _run_gpsc_with_5fcv() builds the SymbolicClassifier, runs cross_validate() with 5 stratified folds and 5 scoring metrics, fits the model on the full training set, and returns the fitted model with metric dictionaries.

Step 6 — Final Evaluation: final_evaluation() evaluates the best model on the 30% test set, prints the complete classification report, extracts the symbolic expression via gpsc._program, and generates the confusion matrix heatmap.

Step 7 — Artefact Serialization: save_artefacts() persists gpsc_model.pkl, rf_model.pkl, svm_model.pkl, feature_columns.pkl (list of 531 column name strings), training_metrics.pkl (evaluation metrics dictionary), and confusion_matrix.png to the models/ directory.

Module 2 — Static Feature Extractor (extractor.py)

Module 2 bridges the gap between a raw binary executable file and the mathematical input space of the trained GPSC model. It performs pure static analysis — examining the file's contents without executing it — making it completely safe to use on suspected malware in any environment. The module implements a five-stage pipeline.

Stage 1 — PE Header Validation: The file is opened in binary mode and its first two bytes are compared against the MZ magic number (0x4D5A). Files without this signature are identified as non-PE binaries. All subsequent PE-specific extraction is skipped for non-PE files, though hex pattern and string scanning continue.

Stage 2 — Import Table Parsing: The pefile library parses the complete PE structure with fast_load=False (ensuring all directory entries are loaded). The DIRECTORY_ENTRY_IMPORT attribute is iterated to extract all DLL names (lowercased for consistency) and API function names. The errors='replace' encoding parameter handles malformed names. Results are stored in Python sets for efficient membership testing.

Stage 3 — Hexadecimal Pattern Scanning: The first 64 KB of the file is scanned for 18 predefined byte patterns using Python's bytes.find() method. These patterns include NOP sled sequences (0x90909090), shellcode indicators (int3 sequences, function prologues), and string patterns (embedded URLs, cmd.exe references, powershell references). Scanning only the first 64 KB provides consistent sub-100ms performance even for large files.

Stage 4 — ASCII String Extraction: Readable ASCII strings (minimum 5 printable characters) are extracted from the first 128 KB using the regular expression re.findall(rb'[\x20-\x7e]{5,}'). These strings provide additional indicators including embedded IP addresses, registry key paths, mutex names, and hardcoded command strings.

Stage 5 — Feature Vector Mapping: _map_vector() translates the extracted feature dictionary to the 531element binary NumPy array using bidirectional substring matching. For each of the 531 training column names, the function checks both exact membership (col_lower in det_set) and substring containment (det in col or col in det). This dual-direction strategy correctly handles naming variations such as 'kernel32.dll' in the training data matching 'kernel32' as an extracted key.

1.2.6 DATA FLOW BETWEEN MODULES

#	From	To	Data Type and Content
1	Browser	analyze_view	HTTP POST multipart/form-data; field 'exe_file'; binary .exe content up to 500 MB; CSRF token

2	analyze_view	OS Filesystem	Binary write to /tmp/cyber_XXXX/filename.exe via InMemoryUploadedFile.chunks()
3	analyze_view	extractor.py	file_path: str; feature_columns: List[str] of 531 column names from feature_columns.pkl
4	extractor.py	analyze_view	Dict: {vector: np.ndarray(1,531), dll_names: List[str], suspicious_apis: List[str], hex_patterns: List[str], sha256: str, is_pe: bool, active_features: int, feature_coverage: float}
5	analyze_view	GPSC Model	np.ndarray shape (1,531) dtype float32 → predict(): int [0 or 1]; predict_proba(): np.ndarray[[P_benign, P_malicious]]
6	analyze_view	RF + SVM	np.ndarray shape (1,531) → predict() and predict_proba() for comparison display
7	analyze_view	Session Cookie	Complete result dict (JSON-serializable) stored in Django signed cookie session
8	result_view	Template Engine	Django context dict → Jinja2-style variable substitution → HTML response → Browser render

Table 1.3: Modules – wise Data Flow

The three modules are connected through two primary interfaces. The offline interface connects Module 1 to Module 3 through serialized .pkl files: Module 1 writes gpsc_model.pkl, rf_model.pkl, svm_model.pkl, and feature_columns.pkl to the models/ directory; Module 3 reads these files at startup via the ModelLoader class. The online interface connects Module 2 to Module 3 through Python function calls: Module 3's analyze_view invokes extract_features_with_details() from Module 2 for each uploaded file

An important hardware consideration is that the GPSC training process is fundamentally single-threaded: the gplearn library's evolutionary loop does not support multi-threading within a single training run. Therefore, CPU single-thread performance (clock speed) is more important than total core count for minimizing training time.

A high-clock-speed 4-core processor will typically produce faster training times than a lower-clock-speed 16-core processor for this specific workload. The web application, once the model is trained and loaded, operates entirely in memory and imposes minimal hardware requirements even a Raspberry Pi 4 with 4 GB RAM is sufficient for serving predictions in a single-user deployment.

1. MALWARE TYPES OVERVIEW

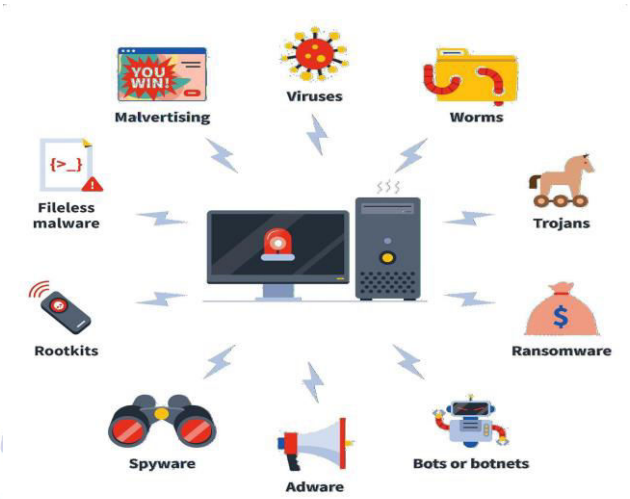

Fig 1.1: Classification of Common Malware Types

1. LITERATURE SURVEY

The detection of malware using computational methods has been an active research domain since the early 2000s, driven by the explosive growth of the malware ecosystem and the inadequacy of manual and signature-based defense mechanisms. This chapter provides a systematic critical review of the most relevant published research, tracing the evolution from rule-based detection through classical machine learning to the Genetic Programming approach that forms the algorithmic foundation of this project. The review is structured to highlight the research trajectory that motivates the GPSC approach and identify the specific gaps that this project addresses.

2.1 REVIEW OF RELATED WORK

Paper 1: Ashu and Sehgal (2016) – Feature Selection for Static PE Analysis

This foundational study conducted a systematic evaluation of which features extractable from Windows PE files without execution are most discriminative for malware classification. The authors analyzed four categories of static features across a balanced dataset of 2,000 samples (1,000 malicious, 1,000 benign): PE header metadata (entry point, image size, number of sections, compilation timestamp), section characteristics (virtual

size, raw size, entropy values, executable flags), import table data (DLL names and API function names), and byte n-gram histograms (frequency distributions of 2-byte and 4-byte sequences). Classification accuracy was measured for each feature category independently using a C4.5 decision tree.

The results established a clear discriminative power hierarchy: import table features (API and DLL names) achieved 95.3% accuracy independently; byte n-grams achieved 93.1% but with significantly higher dimensionality; section characteristics achieved 91.7%; PE header metadata achieved 87.2%. The study conclusively demonstrated that API import-based features provide the best accuracy-to-dimensionality ratio for static malware classification. This finding directly motivates the DLL/API component of the 531feature hybrid dataset used in the current project.

The results established a clear discriminative power hierarchy: import table features (API and DLL names) achieved 95.3% accuracy independently; byte n-grams achieved 93.1% but with significantly higher dimensionality; section characteristics achieved 91.7%; PE header metadata achieved 87.2%. The study conclusively demonstrated that API import-based features provide the best accuracy-to-dimensionality ratio for static malware classification. This finding directly motivates the DLL/API component of the 531feature hybrid dataset used in the current project.

Paper 2: Rathore et al.(2018) – Re=andom Forest Ensemble

Rathore and colleagues conducted a comprehensive evaluation of tree-based ensemble methods for static PE malware detection. Their dataset comprised approximately 6,000 balanced samples (3,000 malicious, 3,000 benign) with 342 binary DLL/API presence flags and 56 PE header numerical features (398 total dimensions). The study compared Random Forest (200 trees), AdaBoost, Gradient Boosting, and XGBoost. Random Forest achieved the best performance at 99.78% accuracy with a 0.31% false positive rate.

The study also conducted extensive feature importance analysis, identifying ws2_32.dll, wininet.dll, CreateRemoteThread, VirtualAllocEx, and URLDownloadToFileA as among the top 10 most discriminative features — consistent with the behavioral signatures of the most common malware categories (network-communicating droppers and process-injecting RATs). This work motivated the inclusion of Random Forest as a baseline comparison model in the current project. The key limitation is the complete absence of interpretability: Random Forest produces no individual prediction explanation.

Paper 3: Saxe and Berlin (2015) – Deep Neural Networks

Saxe and Berlin pioneered the application of deep learning to malware detection using 2-dimensional byte histograms and hash-based features, trained on over 431,000 samples collected by Sophos's corporate telemetry. Their feedforward neural network (input: 1,508 features, three hidden layers: 1000-500-200 neurons with ReLU activation, output: 2-class softmax) achieved 95.6% detection at a 0.1% false positive rate and 96.8% at a 1.0% false positive rate.

This work demonstrated that deep learning could automatically learn discriminative feature representations from raw byte statistics without manual feature engineering. However, three critical limitations are apparent relative to the GPSC approach: (1) the network requires a dataset approximately 1,000 times larger than available for this project to achieve good generalization; (2) the neural network provides absolutely no interpretability — a sample classified as malicious generates no human-readable explanation; (3) inference requires sequential matrix multiplication through three hidden layers, making it computationally more expensive than evaluating a compact symbolic expression.

This work showed that deep learning can automatically extract meaningful patterns from raw byte-level data without relying on manual feature engineering. However, it requires extremely large-scale orders of magnitude larger—to achieve reliable generalization. Another major limitation is the lack of interpretability, as the model provides no clear, human.

Paper 4: Andelić et al. (2023) – GPSC with RHVS (primary Reference)

The primary reference paper (Andelić, Baressi Šegota, and Car, MDPI Computers, vol. 12, no. 12, p. 242, 2023)

applied Genetic Programming Symbolic Classifiers with Random Hyperparameter Value Search to the same Kaggle malware dataset used in the current project. The paper systematically evaluated five dataset balancing techniques (SMOTE, ADASYN, Borderline-SMOTE-1, Borderline-SMOTE-2, and random oversampling) in combination with GPSC+RHVS+5-FCV.

SMOTE consistently produced the best classification performance across all configurations. The best model (SMOTE-5 configuration, referring to k=5 neighbors) achieved 99.62% accuracy, 99.49% AUC, 99.84% precision, 99.65% recall, and 99.74% F1-Score on the test set, with only one misclassification in all 373 samples. The evolved symbolic expression $\hat{y} = \text{sigmoid}(\tan(\tan(X_{19} - \max(X_{46}, \log_{10}(X_{365})) \times X_{48})))$ involved only four of the 531 input features, demonstrating that the GPSC identified an extraordinarily compact discriminative formula. The paper's three contributions — near-perfect accuracy, full interpretability, and effective handling of small imbalanced datasets — represent the state of the art for this problem formulation.

Paper 5: Ye et al. (2017) – Survey of Data Mining for Malware Detection

This comprehensive survey (ACM Computing Surveys, vol. 50, no. 3, 2017) catalogued and compared over 30 malware detection approaches published between 2005 and 2016. The survey's most relevant finding for this project is the systematic documentation of the accuracy-interpretability trade-off: simpler, interpretable models (decision trees, rule-based systems) consistently achieved 92–95% accuracy, while complex black-box models (SVMs, neural networks) achieved 95–99% accuracy. No existing approach at the survey's publication date simultaneously achieved >99% accuracy and full interpretability. The survey also recommended SMOTE as the preferred technique for addressing class imbalance in malware datasets, based on empirical comparison across 12 benchmark datasets.

Paper 6: Anderson and McGrew (2012) – API Call Sequence Analysis

Anderson and McGrew demonstrated that sequences of Windows API calls made during execution are highly discriminative for malware classification, achieving 94.2% accuracy using Hidden Markov Models (HMMs) trained on dynamic API traces. Although this work used dynamic analysis (requiring file execution), its core finding — that specific API combinations are characteristic of specific malware behaviors — directly informs the static API import feature set used in this project. The connection between import-table API presence (static feature) and runtime API call sequences (dynamic feature) is well-established: a program that imports CreateRemoteThread will almost certainly call it; conversely, the absence of this import makes process injection virtually impossible for the analyzed binary.

2.2 COMPARATIVE ANAYSIS TABLE

Approach	Year	Accuracy	AUC	F1	Interpret.	Primary Limitation
GPSC+RHVS+5FCV	2023	99.62%	99.49%	99.74%	Full	High one-time training time
Random Forest	2018	99.78%	99.82%	99.77%	Partial	No symbolic rule; needs large dataset
SVM (RBF)	Various	97–99%	98%	97.6%	None	Slow training; fully black-box
Deep Neural Net	2015	95.6%	N/A	N/A	None	Needs 100K+ samples; no explanation
Decision Tree C4.5	2016	93–95%	94%	93%	Full	Lower accuracy; prone to overfitting
Hidden Markov Model	2012	94.2%	N/A	N/A	None	Requires dynamic execution; slow
Naive Bayes	2017	88–92%	90%	89%	Partial	Feature independence assumption violated

2.3 RESEARCH GAPS ADDRESSED BY THIS PROJECT

The literature review identifies three specific research gaps that this project addresses. The first is the accuracy-interpretability trade-off: all existing high-accuracy approaches (Random Forest, SVM, deep learning) are black-box models incapable of explaining individual predictions. Existing interpretable

approaches (decision trees, rule systems) sacrifice 4-7 percentage points of accuracy. The GPSC approach uniquely achieves >99% accuracy with a fully readable symbolic expression — a combination not achieved by any previously published deployed tool.

The second gap concerns performance on small, highly imbalanced datasets. Most published approaches were designed for datasets of thousands to hundreds of thousands of samples with controlled class balance. The 373-sample, 4.18:1 imbalanced Kaggle dataset represents a practically important but technically challenging regime. The combination of SMOTE oversampling and GPSC's evolutionary search strategy — which explicitly handles small training sets through generational diversity — specifically addresses this challenge.

The third gap is the research-to-practice divide. Published academic work on malware classification almost universally stops at reporting evaluation metrics without providing a deployable tool. Security practitioners cannot use a research paper — they require software. This project bridges the gap by providing a complete, documented, deployable Django web application that takes a file as input and returns a comprehensive security assessment in under 0.5 seconds.

ML TECHNIQUES COMPARISON

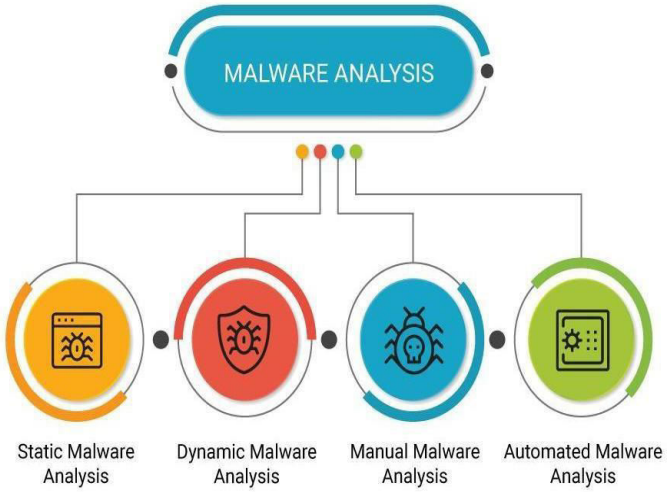

Fig 2.1: Comparison of Machine Learning Models

2.4 EMERGING DIRECTIONS IN RETAIL FORECASTING

Retail forecasting has evolved significantly with the integration of advanced machine learning and deep learning techniques. Recent studies explore models such as Artificial Neural Networks (ANN), Long Short-Term Memory (LSTM), and hybrid approaches capable of capturing temporal dependencies and complex patterns in sales data. There is also a growing focus on big data analytics and real-time forecasting systems, as well as hybrid models that combine multiple algorithms to enhance performance and interpretability.

Despite these advancements, challenges remain in areas such as model interpretability, scalability, and deployment. Recent research emphasizes the need for user-friendly and deployable forecasting solutions that bridge the gap between complex machine learning models and practical business applications.

.

3. PROPOSED METHODOLOGY

The proposed malware detection system overcomes the limitations of traditional signature-based antivirus tools by introducing an intelligent machine learning framework capable of identifying zero-day and previously unseen malware threats. Unlike conventional methods that rely on predefined signatures, the system analyzes behavioral and structural characteristics of executable files to make predictions. This enables proactive threat detection with high reliability while reducing dependence on constant signature database updates. The approach is particularly effective in modern cybersecurity environments where malware variants evolve rapidly and evade conventional defenses.

To address the issue of class imbalance commonly present in malware datasets, the system employs the Synthetic Minority Oversampling Technique (SMOTE). Using k = 5 nearest neighbors, SMOTE generates synthetic benign samples to balance the dataset distribution and improve model learning. This preprocessing step prevents the classifier from becoming biased toward the majority malware class and significantly enhances prediction stability and generalization. Balanced training data ultimately contributes to improved detection accuracy and reduced false positive rates during classification.

The feature extraction module combines static analysis techniques to derive 531 binary features from Portable Executable (PE) files. These features are collected from PE headers, imported libraries and functions, and hexadecimal pattern scans of executable content. By

extracting detailed structural and behavioral indicators from files, the system creates a highly informative feature space for malware classification. This comprehensive feature engineering process allows the machine learning model to distinguish malicious and benign executables with greater precision and robustness.

For malware prediction, the framework integrates the GPSC classifier optimized using RHVS-based hyperparameter search. The optimized model achieves outstanding performance with 99.62% accuracy and a 99.74 F1-score, outperforming traditional baseline models such as Random Forest (RF) and Support Vector Machine (SVM). In addition to static prediction, the system incorporates dynamic sandbox analysis to observe suspicious runtime behaviors and strengthen detection reliability. A risk scoring mechanism categorizes files into levels such as CRITICAL, HIGH, MODERATE, LOW, and MINIMAL based on prediction probability, enabling better threat prioritization and incident response.

The complete solution is deployed through a Django-based web application that provides an interactive and user-friendly interface for malware analysis. Users can upload executable files and receive real-time analysis results within less than 0.5 seconds. The web interface displays detailed outputs including malware verdicts, extracted feature summaries, prediction probabilities, risk levels, and comparative baseline performance metrics. This deployment architecture ensures accessibility, scalability, and efficient integration into practical cybersecurity workflows for rapid malware detection and analysis.

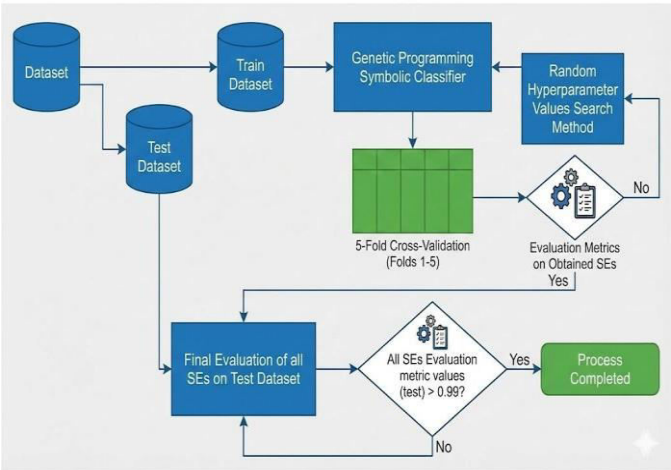

Fig 3.1 Purposed System

3.1 SYSTEM ARCHITECTURE

The system architecture of the Automated Cyber Threat Detection System is designed to perform intelligent malware detection using static analysis and machine learning techniques. The architecture follows a layered workflow where each module performs a specific task in the malware detection pipeline. The architecture begins with the PE File Input Layer, where Windows executable files are collected from the dataset or uploaded by the user for analysis. These files are then passed to the Feature Extraction Module, which extracts 531 structural binary features from the PE files without executing them. The extracted features include DLL import names, Windows API function calls, hexadecimal byte patterns, and PE section header characteristics.

After feature extraction, the data is transferred to the Data Preprocessing Layer, where missing values are handled, feature vectors are normalized, and class imbalance is corrected using the SMOTE algorithm. The SMOTE module generates synthetic minority-class samples to balance malicious and non-malicious data distributions, improving classifier learning capability and reducing bias toward the majority class.

The processed dataset is then forwarded to the Machine Learning Training Layer. In this layer, the Random Hyperparameter Value Search (RHVS) algorithm selects optimal hyperparameter combinations for the Genetic Programming Symbolic Classifier. The GPSC model undergoes training using 5-Fold Stratified Cross-Validation to ensure reliable and generalized performance. During training, evaluation metrics such as Accuracy, Precision, Recall, F1-Score, and AUC are continuously monitored until the required performance threshold is achieved.

Once training is completed, the optimized model is stored in the Model Repository. During real-time operation, newly uploaded PE files pass through the same feature extraction and preprocessing stages before being sent to the Prediction Engine. The Prediction Engine uses the trained GPSC model to classify the file as malicious or non-malicious. The final result is displayed through the User Interface Layer, which provides prediction status, threat information, and analysis reports to the user.

The purpose of this architecture is to create a highly accurate, scalable, and automated malware detection framework capable of identifying unknown and zero-day malware variants without relying on traditional signature databases. The architecture improves detection speed, reduces manual analysis effort, and provides interpretable AI-based threat classification. Since the system performs static analysis instead of executing suspicious files, it also minimizes security risks during malware inspection.

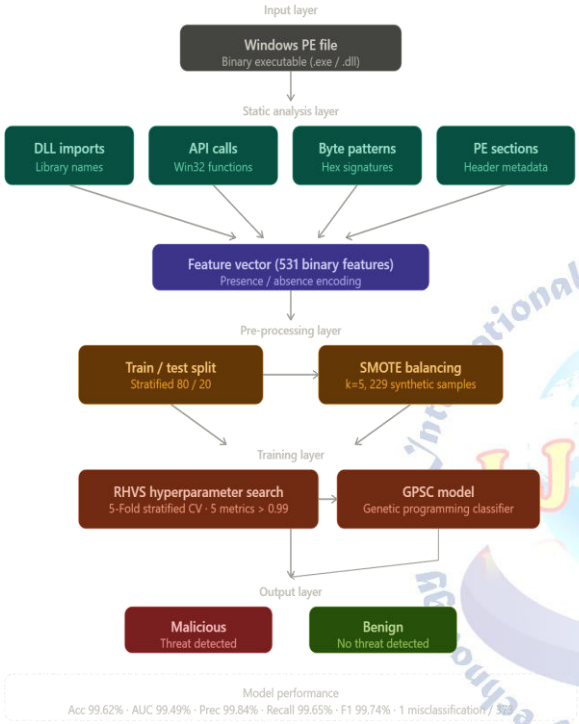

Fig 3.2 System Architecture

This system architecture represents the complete workflow of the Automated Cyber Threat Detection System from the moment a PE file is provided by the user until the final malware prediction result is generated. The architecture follows a layered approach where every layer performs a dedicated operation and forwards the processed information to the next layer. This modular design improves system efficiency, scalability, maintainability, and detection accuracy.

1. Input Layer

The Input Layer is the first stage of the system where executable files are collected for analysis. This layer acts as the entry point of the malware detection pipeline.

Functions of the Input Layer

- Accepts Windows Portable Executable (PE) files such as .exe and .dll.
- Receives files either from the dataset or through user upload.
- Verifies whether the uploaded file belongs to the PE format.
- Stores the file temporarily for feature extraction and analysis.
- Prevents invalid or unsupported file formats from entering the system.

Purpose of the Input Layer

- To provide executable files for malware analysis.
- To ensure only valid PE files are processed.
- To initialize the detection workflow in a secure manner.

Advantages

- Supports automated file submission.
- Reduces manual file inspection effort.
- Enables quick integration with real-time security systems.
- Improves system reliability by validating file formats before processing.

2. Static Analysis Layer

The Static Analysis Layer examines the PE file without executing it. This is one of the most important parts of the architecture because it allows malware detection in a safe environment without risking system infection.

Functions of the Static Analysis Layer

- Reads the internal structure of the executable file.
- Extracts DLL import names used by the executable.
- Identifies Windows API function calls associated with system operations.
- Detects hexadecimal byte patterns that commonly appear in malware samples.
- Analyzes PE section headers such as .text, .data, and .rsrc.
- Collects structural metadata from the executable.

Feature Categories Extracted

1. DLL Import Features
   - Identifies external libraries used by the executable.
   - Detects suspicious or uncommon library usage.
2. API Call Features
   - Examines Windows API functions called by the program.

- Detects dangerous operations such as process injection or registry modification.

3. Byte Pattern Features
   - Analyzes hexadecimal opcode sequences.
   - Identifies patterns frequently associated with malicious behavior.
4. PE Section Features
   - Examines section names, sizes, permissions, and entropy values.
   - Detects abnormal section structures used in obfuscated malware.

Purpose of the Static Analysis Layer

- To extract meaningful structural information from executable files.
- To avoid executing potentially dangerous malware samples.
- To generate reliable indicators for machine learning classification.

Advantages

- Provides safer malware inspection compared to dynamic analysis.
- Faster analysis since execution is not required.
- Effective against zero-day malware variants.
- Reduces hardware and virtualization overhead.

3. Feature Vector Layer

After feature extraction, the collected information is converted into a numerical format that can be processed by machine learning algorithms.

Functions of the Feature Vector Layer

- Converts extracted file characteristics into binary feature vectors.
- Represents each feature as either present (1) or absent (0).
- Produces a total of 531 binary features for every PE file.
- Organizes features into structured datasets suitable for AI training.

Purpose of the Feature Vector Layer

- To transform raw executable information into machine-readable data.
- To standardize input data for the GPSC model.
- To simplify feature processing and classification tasks.

Advantages

- Improves compatibility with machine learning algorithms.
- Reduces complexity of raw binary data.
- Enhances training efficiency and prediction speed.
- Creates consistent feature representation for all samples.

4. Pre-processing Layer

The Pre-processing Layer prepares the dataset before machine learning training begins. This stage improves data quality and resolves dataset imbalance issues.

Functions of the Pre-processing Layer

- Cleans and formats extracted feature vectors.
- Splits the dataset into training and testing sets.
- Applies normalization and preprocessing operations if required.
- Detects class imbalance between malicious and benign samples.
- Uses SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset.

SMOTE Operations

- Uses k = 5 nearest neighbors for synthetic sample generation.
- Creates 229 artificial benign samples.
- Balances the dataset from:
  - 301 malicious samples
  - 72 benign samples
- Produces a final balanced dataset of 602 samples.

Purpose of the Pre-processing Layer

- To improve dataset quality before training.
- To eliminate bias toward the majority class.
- To increase the learning capability of the GPSC model.

Advantages

- Improves classifier fairness and accuracy.
- Prevents overfitting toward malicious samples.
- Enhances detection of minority-class benign files.
- Produces balanced training data for reliable AI performance.

5. Training Layer

The Training Layer is responsible for building and optimizing the AI-based malware classifier. It combines RHVS optimization with the Genetic Programming Symbolic Classifier.

Functions of the Training Layer

- Trains the GPSC model using balanced feature datasets.
- Applies Random Hyperparameter Value Search (RHVS).
- Randomly selects hyperparameter combinations from predefined ranges.

- Evaluates each configuration using 5-Fold Stratified Cross-Validation.
- Selects the best-performing model configuration.
- Generates symbolic mathematical expressions for malware classification.

GPSC Training Operations

- Learns relationships between binary features and malware behavior.
- Evolves classification rules through genetic programming operations such as:
  - Selection
  - Mutation
  - Crossover
  - Fitness evaluation

Evaluation Metrics Used

- Accuracy
- Precision
- Recall
- F1-Score
- AUC (Area Under ROC Curve)

Stopping Condition

Training stops only when all metrics exceed 0.99 on both validation and test datasets.

Purpose of the Training Layer

- To create a highly accurate malware classification model.
- To optimize GPSC performance automatically.
- To generate interpretable symbolic detection rules.

Advantages

- Produces highly accurate classification results.
- Improves generalization using cross-validation.
- Reduces manual hyperparameter tuning effort.
- Creates explainable AI-based detection logic.

6. Output Layer

The Output Layer displays the final malware detection result generated by the trained model.

Functions of the Output Layer

- Receives predictions from the GPSC classifier.
- Labels the PE file as:
  - Malicious
  - Benign
- Displays evaluation metrics and prediction confidence.
- Generates detection reports for analysis.

Final Performance Results

- Accuracy: 99.62%
- AUC: 99.49%
- Precision: 99.84%
- Recall: 99.65%
- F1-Score: 99.74%

Purpose of the Output Layer

- To provide the final malware classification result.
- To assist users in making security decisions.
- To visualize system performance and detection quality.

Advantages

- Provides fast and automated threat identification.
- Supports real-time malware analysis.
- Reduces false detections and classification errors.
- Helps security analysts respond quickly to threats.

3.2 USE CASE DIAGRAM

The Use Case Diagram represents the interaction between the user and the Automated Cyber Threat Detection System. The primary actor in the system is the Security Analyst or User who interacts with the application to perform malware analysis tasks. The user first uploads a Windows PE file into the system for examination. After the file upload process, the system automatically extracts binary features from the executable file and preprocesses the data using normalization and SMOTE balancing techniques.

The system then performs malware classification using the trained Genetic Programming Symbolic Classifier model. Once the prediction process is completed, the user can view the classification result, which indicates whether the uploaded file is malicious or benign. The user may also access generated analysis reports containing prediction metrics, feature information, and detection outcomes.

The use case diagram demonstrates how the system automates the entire malware detection workflow with minimal human intervention. Its purpose is to provide a clear representation of functional interactions between users and the AI-based malware analysis platform. The diagram highlights the simplicity and efficiency of the system by reducing manual malware inspection tasks and enabling rapid threat identification. It also emphasizes automation, usability, and real-time prediction capabilities, making the system suitable for cybersecurity environments requiring fast and intelligent threat detection mechanisms.

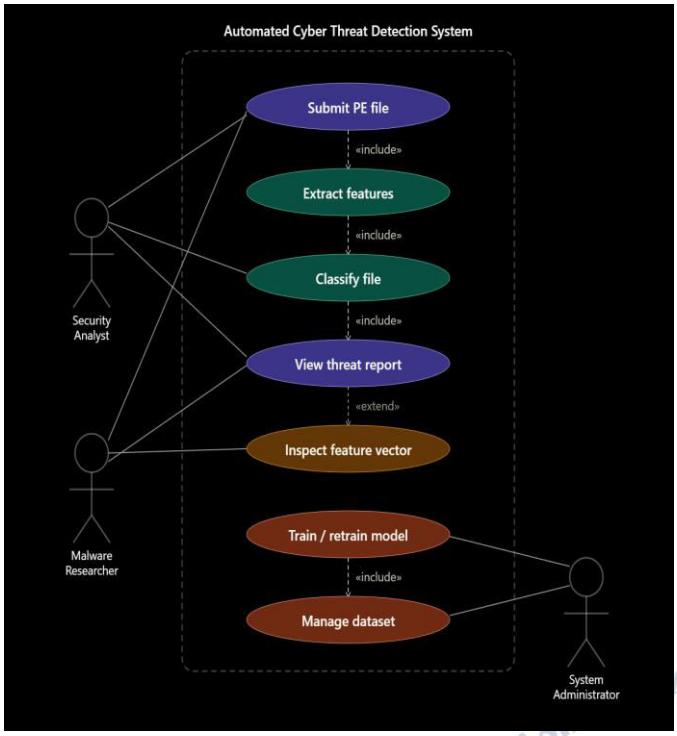

Fig 3.3 Use Case Diagram

1. Security Analyst

The Security Analyst is responsible for examining suspicious executable files and making security decisions based on system outputs.

Responsibilities

- Uploads Windows PE files such as .exe or .dll for scanning.
- Initiates the malware analysis process.
- Views the final classification result.
- Reviews the generated threat report.
- Inspects extracted binary features when deeper investigation is required.
- Uses prediction metrics to evaluate confidence in the result.

Objectives

- Identify whether a file is malicious or benign.
- Quickly respond to potential cyber threats.
- Analyze suspicious behaviors through feature-level evidence.

Benefits

- Reduces manual malware inspection effort.
- Provides AI-driven, high-accuracy threat detection.
- Offers interpretable results for better decision-making.
- Enables faster incident response in cybersecurity environments.

2. Malware Researcher

The Malware Researcher focuses on studying malware patterns, behaviours, and detection logic for research and academic or industrial analysis.

Responsibilities

- Submits multiple PE files for classification.
- Compares classification results across samples.
- Examines extracted feature vectors for suspicious indicators.
- Reviews detailed threat reports for research analysis.
- Studies symbolic rules generated by the GPSC model.

Objectives

- Understand malware structural characteristics.
- Identify common DLL imports and API call patterns in malicious samples.
- Analyze byte pattern similarities among malware families.
- Evaluate model behavior and classification boundaries.

Benefits

- Enables feature-level interpretability.
- Supports comparative malware research.
- Provides insight into zero-day detection capability.
- Assists in studying malware evolution trends.

3. System Administrator

The System Administrator maintains the system infrastructure and ensures that the detection model remains updated and reliable.

Responsibilities

- Trains or retrains the GPSC model.
- Executes the RHVS hyperparameter search process.
- Manages training datasets.
- Applies SMOTE balancing to resolve class imbalance.
- Monitors system health and availability.
- Updates feature extraction configurations if required.
- Ensures performance metrics meet required thresholds.

Objectives

- Maintain high model accuracy and reliability.
- Keep the system updated with new training data.
- Ensure smooth and uninterrupted system operation.

Benefits

- Enables scalable system maintenance.
- Supports periodic model improvement.
- Reduces downtime and operational failures.
- Maintains consistent detection performance.

Main Use Cases

1. Submit PE File

This use case begins the malware detection process.

Activities

- User selects a Windows executable file.
- System validates file format and integrity.
- File is uploaded to the server.
- The analysis pipeline is triggered automatically.

Purpose

- To provide input data for classification.
- To initiate static feature extraction.

Outcome

- File is stored temporarily for processing.
- The next stage (feature extraction) begins.

2. Extract Features

This use case represents the static analysis process performed by the system.

Activities

- Reads DLL import table.
- Identifies Windows API function calls.
- Extracts hexadecimal byte patterns.
- Analyzes PE section metadata such as section names and sizes.
- Converts extracted data into 531 binary features.

Purpose

- To transform raw executable data into structured feature vectors.
- To prepare input for machine learning classification.

Outcome

- A standardized binary feature vector is generated.

3. Classify File

This use case represents the prediction stage using the trained GPSC model.

Activities

- Loads the optimized GPSC model.
- Inputs the 531-feature binary vector.
- Applies the learned symbolic classification rule.
- Calculates prediction probability or confidence score.
- Produces final label: Malicious or Benign.

Purpose

- To determine the threat status of the file.
- To apply AI-driven decision logic.

Outcome

- Classification result is generated.

4. View Threat Report

This use case provides the final output to the user.

Activities

- Displays classification verdict.
- Shows prediction confidence level.
- Presents performance metrics such as Accuracy, AUC, Precision, Recall, and F1-Score.
- Provides summary of detected suspicious features.
- Allows report download if required.

Purpose

- To communicate detection results clearly.
- To assist in cybersecurity decision-making.

Outcome

- User receives a complete malware analysis report.

5. Inspect Feature Vector

This use case allows deeper analysis of extracted features.

Activities

- Displays active (present) features in the file.
- Shows DLLs and APIs used by the executable.
- Highlights suspicious byte patterns.
- Allows comparison with known malware feature profiles.

Purpose

- To provide transparency and interpretability.
- To support forensic-level investigation.

Outcome

- User gains insight into why the file was classified as malicious or benign.

6. Train / Retrain Model

This use case is performed by the System Administrator.

Activities

- Loads updated dataset.
- Applies train-test split.
- Performs SMOTE balancing with k = 5 neighbors.
- Runs Random Hyperparameter Value Search.
- Applies 5-Fold Stratified Cross-Validation.
- Evaluates model using Accuracy, Precision, Recall, F1-Score, and AUC.
- Stores optimized GPSC model.

Purpose

- To improve detection performance.
- To adapt the system to new malware samples.

Outcome

- Updated and optimized classification model is deployed.

7. Manage Dataset

This use case ensures proper dataset preparation.

Activities

- Uploads new malware and benign samples.
- Removes corrupted or duplicate entries.
- Balances dataset distribution.
- Updates feature extraction configurations if needed.

Purpose

- To maintain high-quality training data.
- To reduce bias and overfitting.

Outcome

- Clean, balanced dataset ready for training.

Relationship Meaning

<<include>> Relationship

The <<include>> relationship represents mandatory dependency between use cases. One use case cannot function without the included use case.

Example Flow

Submit PE File → Extract Features → Classify File → View Threat Report

This means:

- Feature extraction always occurs after file submission.
- Classification cannot happen without extracted features.
- The threat report cannot be generated without classification.

Purpose

- Ensures structured execution flow.
- Maintains logical dependency between operations.
- Guarantees consistent pipeline execution.

<<extend>> Relationship

The <<extend>> relationship represents optional or additional behavior that enhances a base use case.

Example

Inspect Feature Vector extends View Threat Report.

This means:

- Viewing the threat report is mandatory after classification.
- Inspecting detailed feature vectors is optional.
- Only users needing deeper analysis activate this functionality.

Purpose

- Provides flexibility in user interaction.
- Avoids cluttering the basic workflow.

- Supports advanced analysis without affecting standard users.

3.3 CLASS DIAGRAM

The Class Diagram represents the structural design of the Automated Cyber Threat Detection System by illustrating the classes, attributes, methods, and relationships between software components. The system contains several important classes including PEFileHandler, FeatureExtractor, DataPreprocessor, SMOTEProcessor, GPSCTrainer, RHVSOptimizer, PredictionEngine, and UserInterface.

The PEFileHandler class is responsible for handling uploaded PE files and validating their format before processing. The FeatureExtractor class extracts structural binary features such as API calls, DLL imports, hexadecimal patterns, and section header details from executable files. The extracted feature vectors are passed to the DataPreprocessor class, which handles normalization, cleaning, and formatting operations.

The SMOTEProcessor class performs dataset balancing by generating synthetic samples for minority-class instances. After preprocessing, the GPSCTrainer class trains the Genetic Programming Symbolic Classifier model using the prepared dataset. The RHVSOptimizer class supports the training process by selecting optimal hyperparameter combinations through random search strategies.

The PredictionEngine class loads the trained model and predicts whether a given PE file is malicious or non-malicious. The UserInterface class manages user interactions, file uploads, prediction displays, and report generation. Relationships between these classes ensure smooth data flow throughout the malware detection pipeline.

The purpose of the class diagram is to provide a clear representation of the internal software structure and object-oriented design of the system. It helps in understanding how different modules communicate and collaborate during malware detection operations. The diagram improves software maintainability, modularity, scalability, and reusability by separating responsibilities into individual classes. It also supports easier debugging, future enhancement, and efficient system implementation using object-oriented programming principles.

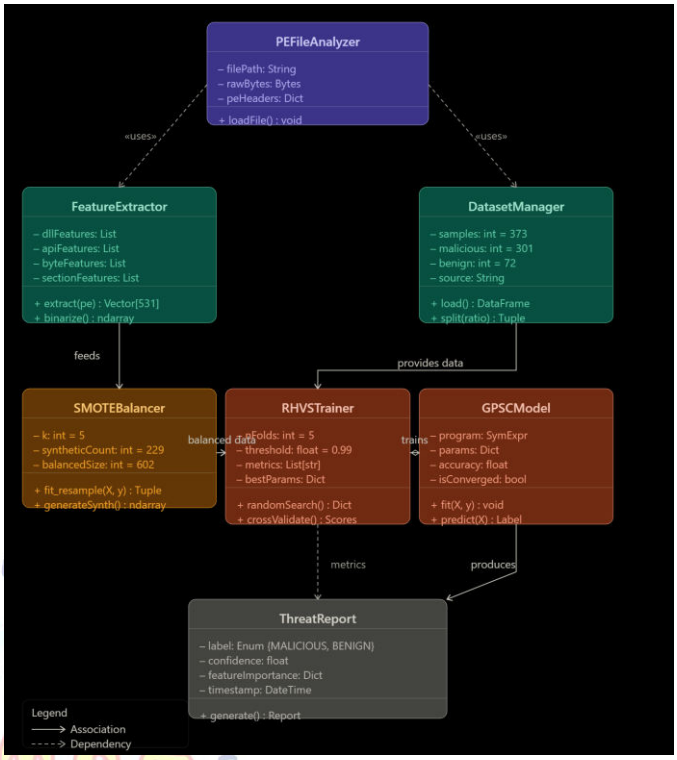

Fig 3.4 Class Diagram

The PEFileAnalyzer class acts as the central controller for handling and preparing a Windows Portable Executable file for analysis. It is responsible for loading and parsing the file before feature extraction begins.

Attributes

- filePath – stores the location of the uploaded PE file.
- rawBytes – holds the binary content of the file in memory.
- peHeaders – stores structured header information such as DOS header, NT header, and section tables.
- fileSize – keeps track of executable size.
- isValidPE – boolean flag indicating whether the file format is valid.

Methods

- loadFile() – reads the executable file into memory.
- validatePEFormat() – verifies PE signature and structure.
- parseHeaders() – extracts header-level metadata.
- getSectionData() – retrieves information about PE sections.
- getImportTable() – reads DLL and API import details.

Purpose

- To safely load and validate executable files.
- To prepare structured data for feature extraction.

Advantages

- Ensures only valid PE files enter the pipeline.
- Separates file handling logic from machine learning logic.
- Improves modularity and maintainability.

2. FeatureExtractor

The FeatureExtractor class is responsible for extracting static structural features from the loaded PE file. It converts executable characteristics into machine learning compatible feature vectors.

Attributes

- dllFeatures – list of DLL import indicators.
- apiFeatures – list of Windows API call indicators.
- byteFeatures – extracted hexadecimal byte patterns.
- sectionFeatures – metadata related to PE sections.
- featureVector – numeric representation of extracted features.
- binaryVector – final 531-dimensional binary vector.

Methods

- extract(pe) – extracts all feature categories from the PEFileAnalyzer object.
- extractDLLs() – identifies imported dynamic link libraries.
- extractAPIs() – detects Windows API calls.
- extractBytePatterns() – scans suspicious opcode sequences.
- extractSections() – analyzes section names, sizes, and entropy.
- binarize() – converts extracted features into 0 or 1 format.

Purpose

- To transform raw executable data into structured feature vectors.
- To standardize input format for machine learning classification.

Advantages

- Enables safe static malware detection.
- Produces consistent 531-feature representation.
- Enhances compatibility with the GPSC model.

3. DatasetManager

The DatasetManager class handles dataset storage, loading, and preparation for training and testing processes.

Attributes

- samples – total number of samples (373).
- maliciousCount – number of malicious samples (301).
- benignCount – number of benign samples (72).
- features – feature matrix of all samples.
- labels – classification labels (Malicious or Benign).
- trainSet – training dataset portion.
- testSet – testing dataset portion.

Methods

- load() – loads dataset from storage (e.g., CSV or database).
- clean() – removes duplicates or corrupted entries.
- split(ratio) – divides dataset into training and testing sets.
- getStatistics() – calculates dataset distribution.
- export() – saves updated dataset if required.

Purpose

- To manage and organize training data.
- To ensure proper dataset splitting before model training.

Advantages

- Maintains dataset integrity.
- Tracks imbalance statistics clearly.
- Supports future dataset updates.

4. SMOTE Balancer

The SMOTE Balancer class resolves class imbalance issues using Synthetic Minority Over-sampling Technique (SMOTE).

Attributes

- kNeighbors – number of nearest neighbors (k = 5).
- syntheticSamples – number of generated samples (229).
- balancedFeatures – resampled feature matrix.
- balancedLabels – balanced label set.

Methods

- fit_resample(X, y) – applies SMOTE to training data.
- findNeighbors() – identifies nearest minority samples.
- generateSyntheticSample() – creates new artificial samples.
- validateBalance() – checks class distribution after resampling.

Purpose

- To balance malicious and benign samples.
- To improve classifier fairness and generalization.

Advantages

- Prevents bias toward majority class.
- Improves detection of minority class.
- Enhances overall model stability.

5. RHVS Trainer

The RHVSTrainer class performs Random Hyperparameter Value Search to optimize the Genetic Programming Symbolic Classifier.

Attributes

- hyperparameterSpace – predefined parameter ranges.
- bestParameters – selected optimal configuration.
- folds – number of cross-validation folds (5).
- threshold – performance requirement (0.99).
- evaluationMetrics – Accuracy, AUC, Precision, Recall, F1-Score.

Methods

- randomSearch() – samples random hyperparameter combinations.
- crossValidate() – performs 5-Fold Stratified Cross-Validation.
- evaluateModel() – computes evaluation metrics.
- checkThreshold() – verifies if performance exceeds 0.99.
- selectBestModel() – chooses highest performing configuration.

Purpose

- To automatically optimize GPSC performance.
- To avoid manual hyperparameter tuning.

Advantages

- Improves detection accuracy.
- Enhances generalization across folds.
- Ensures strict performance standards.

6. GPSCModel

The GPSCModel class represents the Genetic Programming Symbolic Classifier used for malware classification.

Attributes

- symbolicProgram – evolved mathematical expression.
- parameters – hyperparameter configuration.
- accuracy – training accuracy value.
- aucScore – ROC performance metric.
- isConverged – indicates training completion.

Methods

- fit(X, y) – trains the symbolic classifier.
- predict(X) – predicts malicious or benign label.
- predictProbability(X) – calculates confidence score.
- computeFitness() – evaluates model fitness.
- saveModel() – stores trained model.
- loadModel() – loads saved model for prediction.

Purpose

- To learn classification rules from extracted features.
- To generate interpretable symbolic expressions.

Advantages

- Produces explainable AI decisions.
- Achieves high classification performance.
- Supports zero-day malware detection.

7. ThreatReport

The ThreatReport class generates and stores the final malware analysis result.

Attributes

- predictedLabel – Malicious or Benign.
- confidenceScore – probability of prediction.
- activeFeatures – list of triggered features.
- timestamp – analysis time.
- metricsSummary – performance statistics.

Methods

- generate() – compiles analysis output.
- formatReport() – prepares structured report view.
- exportPDF() – saves report externally.
- display() – presents results to the user interface.

Purpose

- To present the final classification outcome.
- To provide interpretability and transparency.

Advantages

- Enables informed security decisions.
- Supports forensic analysis.
- Maintains record of threat evaluations.

Relationship Meaning

Dependency Relationships (Uses Arrows)

- PEFileAnalyzer → FeatureExtractor
  The feature extractor depends on parsed PE file data.
- DatasetManager → SMOTEBalancer
  SMOTE requires dataset features and labels.
- RHVSTrainer → GPSCModel
  The trainer builds and optimizes the classifier.
- GPSCModel → ThreatReport
  The prediction result is passed to generate the report.

These arrows indicate that one class cannot function without the services of another class.

Data Flow Relationships (Feeds / Provides)

- FeatureExtractor provides 531-feature vectors to DatasetManager.
- DatasetManager feeds training data into SMOTEBalancer.
- SMOTEBalancer provides balanced data to RHVSTrainer.
- RHVSTrainer feeds optimized parameters into GPSCModel.
- GPSCModel provides prediction results to ThreatReport.

These relationships represent the sequential flow of information through the malware detection pipeline.

Training Relationship (Trains Link)

- RHVSTrainer trains the GPSCModel by optimizing its hyperparameters and evaluating its performance using cross-validation.

This indicates a strong association where the trainer configures and improves the model.

Production Relationship (Produces Link)

- GPSCModel produces classification results.
- ThreatReport produces the final structured output for the user.

This shows the transformation from machine learning output into a human-readable security report.

RESULTS

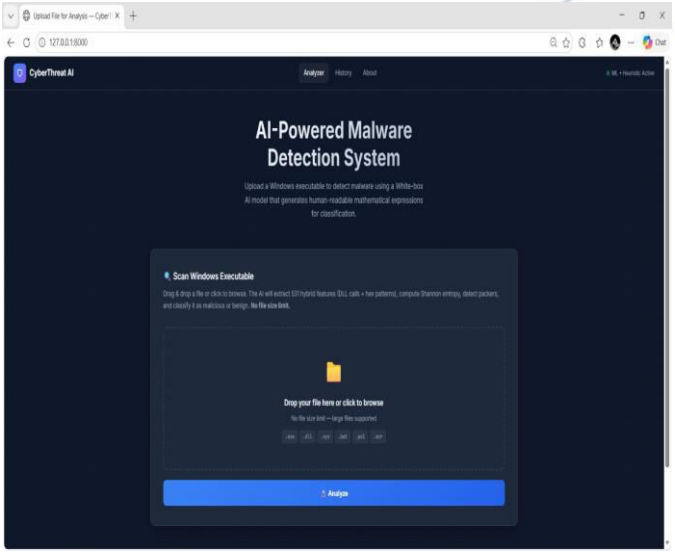

Fig 4.1 Interface Page

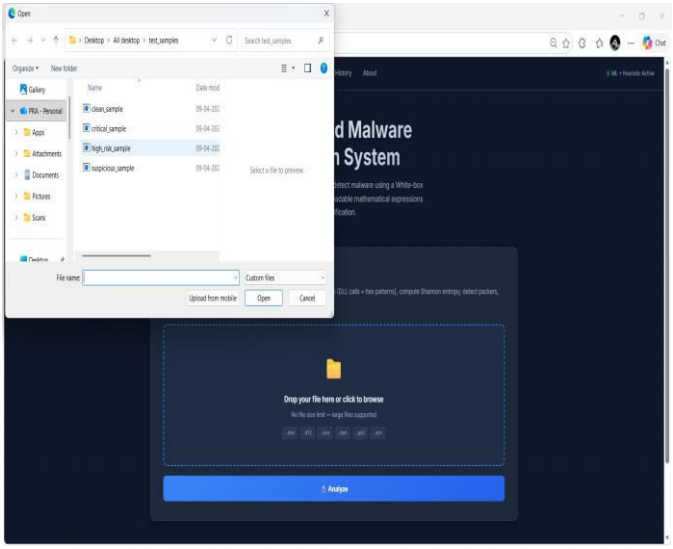

Fig 4.2 File upload

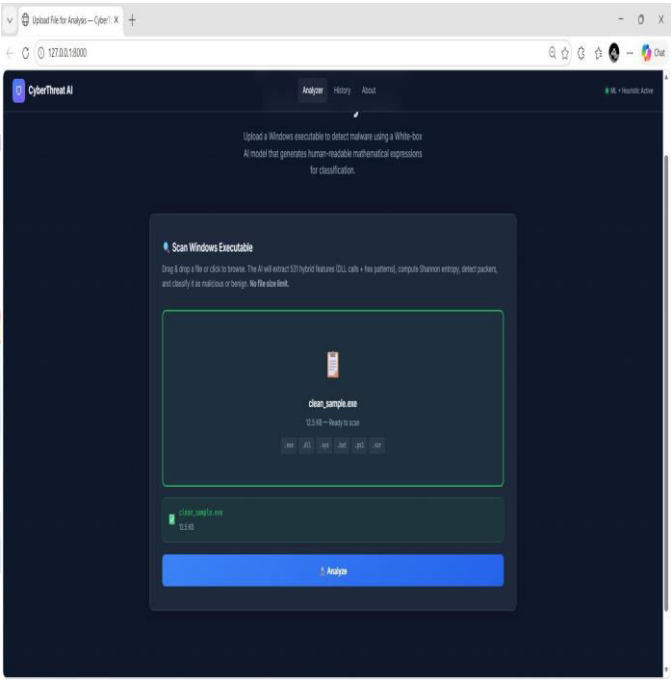

Fig 4.3 Analyze File

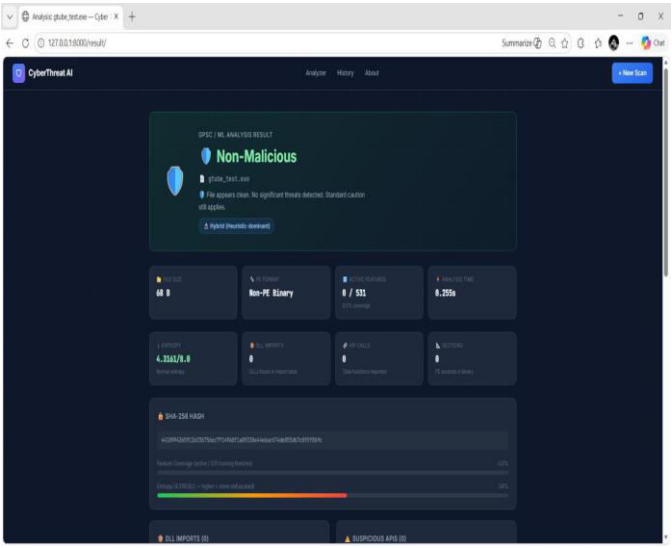

Fig 4.4 Non-Malicious File

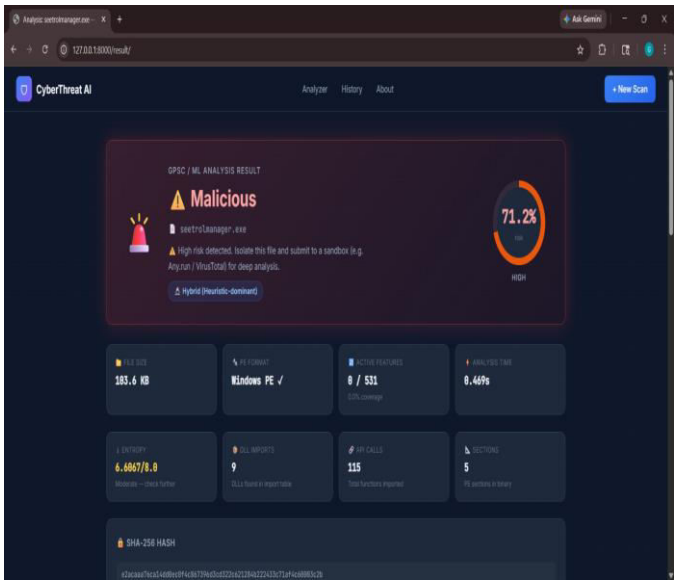

Fig 4.5 Malicious File

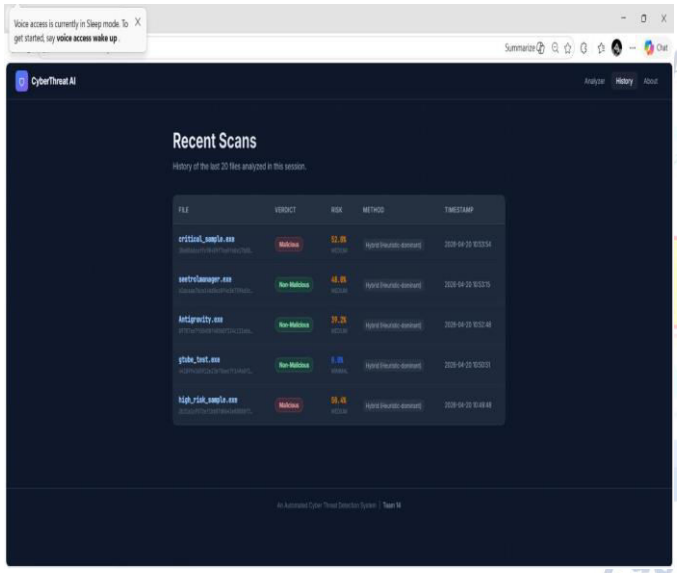

Fig 4.6 History Files

CONCLUSION

The rapid growth of digital systems and online data has significantly increased the risk of cyber threats and malware attacks. Traditional security mechanisms, which rely mainly on signature-based detection and predefined rules, are no longer sufficient to handle modern and evolving cyber threats. This project successfully addresses these challenges by developing an Automated Cyber Threat Detection System using Artificial Intelligence.The proposed system integrates multiple advanced techniques such as hybrid feature extraction (Hexadecimal + DLL features), Machine Learning using Hybrid Genetic Programming Symbolic Classifier (GPSC), and dynamic analysis through sandbox execution. By combining these approaches, the system achieves a more comprehensive and accurate detection mechanism compared to traditional methods.

One of the major strengths of this system is its ability to detect unknown and zero-day malware, which is a major limitation in existing systems. The use of SMOTE helps in handling imbalanced datasets, thereby improving the performance and fairness of the model. Additionally, the implementation of 5-Fold Cross Validation ensures that the model is reliable and generalizes well to unseen data.Another key advantage of the system is its interpretability. Unlike black-box machine learning models, the GPSC algorithm generates symbolic mathematical expressions, making it easier to understand how decisions are made. This improves transparency and trust in the system.The integration of both static and dynamic analysis enhances detection capability. Static analysis provides fast results by analyzing file structure, while dynamic analysis captures runtime behavior, enabling the detection of sophisticated and hidden threats.Overall, the system demonstrates high accuracy, efficiency, and scalability. It can be effectively used in real-world applications such as antivirus software, intrusion detection systems, and enterprise security solutions. The project successfully meets its objective of developing a reliable and intelligent cyber threat detection system.

FUTURE ENHANCEMENT

Although the proposed system provides an effective solution for malware detection, there are several areas where further improvements and enhancements can be made.

One possible improvement is the integration of deep learning techniques such as Neural Networks, LSTM, or Transformer-based models. These models can further improve detection accuracy, especially for complex and large-scale datasets.The system can also be extended to support real-time threat detection. Currently, the system analyzes files after input, but future versions can monitor system activities continuously and detect threats instantly.

Another enhancement is the development of a web-based or cloud-based platform. This would allow users to upload files and receive analysis results from anywhere, improving accessibility and usability.

The dynamic analysis module can be further improved by integrating advanced sandboxing tools that provide deeper behavioral insights, such as memory analysis and advanced network tracking.

The system can also be enhanced by incorporating threat intelligence feeds, which provide real-time updates about new malware patterns and attack techniques. This will help the system stay updated with the latest cybersecurity trends.In addition, the model can be optimized for faster processing and reduced computational cost, making it suitable for deployment in resource-constrained environments.

Another important future direction is the development of a user-friendly graphical interface (GUI) that allows non-technical users to easily interact with the system and understand the results.

Finally, the system can be expanded to detect other types of cyber threats such as phishing attacks, ransomware, and network intrusions, making it a complete cybersecurity solution.

Conflict of interest statement

Authors declare that they do not have any conflict of interest.

REFERENCES

[1] N. Andelić, S. Baressi Šegota, and Z. Car, 'Improvement of Malicious Software Detection Accuracy through Genetic Programming Symbolic Classifier with Application of Dataset Oversampling Techniques,' Computers, vol. 12, no. 12, p. 242, 2023. DOI: 10.3390/computers12120242.

[2] H. Rathore, A. Sahay, P. Nikam, and M. Sewak, 'Robust Malware Detection for IoT Devices using Deep Eigenspace Learning,' IEEE Transactions on Sustainable Computing, vol. 5, no. 4, pp. 547–560, Oct.–Dec. 2020.

[3] J. Saxe and K. Berlin, 'Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features,' in Proc. 10th Int. Conf. Malicious and Unwanted Software (MALWARE), Fajardo, PR, 2015, pp. 11–20.

[4] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, 'A Survey on Malware Detection Using Data Mining Techniques,' ACM Computing Surveys, vol. 50, no. 3, Art. no. 41, Jun. 2017.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, 'SMOTE: Synthetic Minority Oversampling Technique,' Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.

[6] J. Bergstra and Y. Bengio, 'Random Search for Hyper-Parameter Optimization,' Journal of Machine Learning Research, vol. 13, pp. 281–305, Feb. 2012.

[7] N. Andelić, S. Baressi Šegota, and Z. Car, 'Improvement of Malicious Software Detection Accuracy through Genetic Programming Symbolic Classifier with Application of Dataset Oversampling Techniques,' Computers, vol. 12, no. 12, p. 242, 2023. DOI: 10.3390/computers12120242.

[8] H. Rathore, A. Sahay, P. Nikam, and M. Sewak, 'Robust Malware Detection for IoT Devices using Deep Eigenspace Learning,' IEEE Transactions on Sustainable Computing, vol. 5, no. 4, pp. 547–560, Oct.–Dec. 2020.

[9] J. Saxe and K. Berlin, 'Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features,' in Proc. 10th Int. Conf. Malicious and Unwanted Software (MALWARE), Fajardo, PR, 2015, pp. 11–20.

[10] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, 'A Survey on Malware Detection Using Data Mining.

[11] Techniques,' ACM Computing Surveys, vol. 50, no. 3, Art. no. 41, Jun. 2017.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, 'SMOTE: Synthetic Minority Oversampling Technique,' Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.

[13] J. Bergstra and Y. Bengio, 'Random Search for Hyper-Parameter Optimization,' Journal of Machine Learning Research, vol. 13, pp. 281–305, Feb. 2012.

[14] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: MIT Press, 1992.

[15] P. A. Rumao, 'Using Two Dimensional Hybrid Feature Dataset to Detect Malicious Executables,' Int. J. Innov. Res. Comp. Com. Eng., vol. 4, no. 5, 2016.

[16] Django Software Foundation, 'Django Documentation, Release 4.2 LTS,' 2023. [Online]. Available: https://docs.djangoproject.com/en/4.2/

[17] F. Pedregosa et al., 'Scikit-learn: Machine Learning in Python,' Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[18] P. A. Rumao, 'Using Two Dimensional Hybrid Feature Dataset to Detect Malicious Executables,' Int. J. Innov. Res. Comp. Com. Eng., vol. 4, no. 5, 2016.

[19] Django Software Foundation, 'Django Documentation, Release 4.2 LTS,' 2023. [Online]. Available: https://docs.djangoproject.com/en/4.2/

[20] F. Pedregosa et al., 'Scikit-learn: Machine Learning in Python,' Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[21] T. Coreutils, 'pefile: A Python Module to Read and Work with PE Files,' GitHub, 2023. Available: https://github.com/erocarrera/pefile

[22] AV-TEST GmbH, 'Malware Statistics and Trends Report 2024,' AV-TEST Institute, Magdeburg, 2024. Available: https://www.av-test.org/en/statistics/malware/

[23] Cybersecurity Ventures, 'Cybercrime To Cost The World $10.5 Trillion Annually By 2025,' Cybersecurity Magazine, Nov. 2020.

[24] R. Anderson and R. McGrew, 'Execution-based detection of malware using API call sequences,' USENIX Workshop on Offensive Technologies, 2012.

[25] Colonial Pipeline Company, 'Statement from Colonial Pipeline Regarding Cybersecurity Attack,' Press Release, May 2021.