WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Real-Time Anomaly Detection in HTTP Web Traffic Using Unsupervised Machine Learning Techniques

Mr. Shubham Patil[1], Dr. Pushpalata Aher[2], Dr. Mohd Muqeem[3], Dr. Pawan Bhaladhare[4]

[1] M. Tech CTIS Student, School of Computer Science and Engineering, Sandip University, Nashik, India.
[2,3] Professor, School of Computer Science and Engineering, Sandip University, Nashik, India.
[4] HOD & Professor, School of Computer Science and Engineering, Sandip University, Nashik, India.

A B S T R A C T :

The exponential growth of web applications and the increasing sophistication of cyber attacks have made real-time anomaly detection in HTTP traffic a critical component of modern cybersecurity infrastructure. This research presents a comprehensive evaluation of unsupervised machine learning techniques for detecting anomalous HTTP requests in real-time environments. We implemented and compared four prominent unsupervised algorithms: Local Outlier Factor (LOF), Isolation Forest, One-Class Support Vector Machine (One-Class SVM), and Halfspace Trees, alongside supervised baselines using Random Forest and River Forest for streaming scenarios. Our experimental evaluation utilized the CSIC 2010 HTTP dataset, with comprehensive feature engineering focusing on URL characteristics, request patterns, and payload statistics. Real-time simulation was achieved through controlled data streaming with 0.5-second intervals, mimicking production deployment scenarios. The evaluation employed standard metrics including F1-score, precision, recall, and support across normal and anomalous request classifications. Results demonstrate that Isolation Forest achieved the highest F1-score of 0.847 for anomaly detection, while Halfspace Trees showed superior performance in streaming scenarios with 0.823 F1-score and minimal memory overhead. The study provides practical insights into production deployment considerations, including scalability analysis, concept drift handling, and real-time performance requirements. Our findings contribute to the advancement of automated intrusion detection systems and provide a foundation for enterprise-level HTTP anomaly detection implementations.

Keywords: Anomaly Detection, HTTP Traffic Analysis, Unsupervised Learning, Real-time Systems, Cybersecurity, Machine Learning

1. Introduction

In the contemporary digital landscape, web applications serve as the backbone of modern business operations, handling billions of HTTP requests daily across diverse domains including e-commerce, financial services, healthcare, and social media platforms. This ubiquity of web-based services has simultaneously created unprecedented opportunities for cybercriminals to exploit vulnerabilities through sophisticated attack vectors targeting HTTP protocols. Traditional signature-based intrusion detection systems, while effective against known attack patterns, struggle to identify novel threats and zero-day exploits that increasingly characterize modern cyber-attacks (1).

The evolution of web-based threats has necessitated a paradigm shift from reactive, rule-based security measures to proactive, intelligent detection systems capable of identifying anomalous behavior patterns in real-time. HTTP traffic anomaly detection presents unique challenges due to the heterogeneous nature of web requests, the high volume of legitimate traffic variations, and the requirement for near-instantaneous response times in production environments. Unlike network-level intrusion detection, HTTP anomaly detection must account for application-layer semantics, user behavior patterns, and the contextual nature of web interactions (2). Unsupervised machine learning techniques have emerged as particularly promising solutions for HTTP anomaly detection due to their ability to identify novel attack patterns without requiring extensive labeled datasets of malicious traffic. These approaches leverage statistical properties and behavioral patterns inherent in normal HTTP traffic to establish baseline models, subsequently flagging deviations as potential anomalies. The absence of dependency on labeled attack data makes unsupervised methods particularly valuable in dynamic threat environments where new attack vectors continuously emerge (3).

The real-time processing requirement introduces additional complexity, as anomaly detection systems must balance detection accuracy with computational efficiency and memory constraints. Streaming machine learning algorithms, designed to process data incrementally without storing entire datasets in memory, offer potential solutions to these scalability challenges. However, the adaptation of traditional batch-learning anomaly detection algorithms to streaming scenarios remains an active area of research (4). This research addresses the critical gap between theoretical anomaly detection capabilities and practical deployment requirements in production HTTP traffic monitoring systems. We present a comprehensive evaluation of four prominent unsupervised anomaly detection algorithms: Local Outlier Factor (LOF), Isolation Forest, One-Class Support Vector Machine (One-Class

SVM), and Halfspace Trees, implemented in real-time streaming scenarios using the CSIC 2010 HTTP dataset. Our methodology incorporates practical considerations including feature engineering, concept drift handling, performance optimization, and production deployment architecture.

The primary contributions of this work include: (1) A comprehensive comparative analysis of unsupervised anomaly detection algorithms in HTTP traffic monitoring contexts, (2) Implementation of real-time streaming scenarios with practical performance constraints, (3) Detailed feature engineering methodology optimized for HTTP request characteristics, (4) Production deployment considerations including scalability analysis and architectural recommendations, and (5) Open-source implementation providing reproducible results and practical deployment guidelines.

The remainder of this paper is organized as follows: Section 2 reviews related work in HTTP anomaly detection and streaming machine learning, Section 3 presents our methodology and experimental design, Section 4 details the implementation approach, Section 5 presents experimental results and analysis, Section 6 discusses implications and limitations, and Section 7 concludes with future research directions.

2. Related Work

2.1 HTTP Traffic Anomaly Detection

The field of HTTP traffic anomaly detection has evolved significantly over the past two decades, driven by the increasing sophistication of web-based attacks and the growing complexity of modern web applications. Early approaches primarily relied on signature-based detection methods, comparing incoming requests against databases of known attack patterns. Kruegel and Vigna (5) introduced one of the first comprehensive HTTP anomaly detection systems, utilizing statistical analysis of request parameters and payload characteristics to identify deviations from normal behavior patterns.

The transition from signature-based to behavior-based detection marked a significant advancement in the field. Wang and Stolfo (6) proposed a payload-based anomaly detection system that analyzed the byte-level distribution of HTTP requests, establishing statistical models of normal traffic patterns. Their approach demonstrated the effectiveness of unsupervised learning in identifying novel attack vectors, achieving detection rates exceeding 90% on synthetic datasets. However, the computational overhead of byte-level analysis limited practical deployment in high-throughput environments.

Recent research has focused on feature-based approaches that extract meaningful characteristics from HTTP requests while maintaining computational efficiency. Ingham et al. (7) developed a comprehensive feature set including URL structure, parameter patterns, and header characteristics, demonstrating that carefully engineered features could achieve comparable detection performance to payload-based methods with significantly reduced computational requirements. This work established the foundation for modern HTTP anomaly detection systems that balance accuracy with practical deployment constraints.

2.2 Unsupervised Learning in Cybersecurity

The application of unsupervised learning techniques to cybersecurity challenges has gained significant traction due to the scarcity of labeled attack data and the continuous evolution of threat landscapes. Chandola et al. (8) provided a comprehensive survey of anomaly detection techniques, categorizing approaches based on underlying assumptions about normal behavior and the nature of anomalies. Their work highlighted the particular suitability of density-based methods for network security applications, where anomalies often represent sparse regions in high-dimensional feature spaces.

Local Outlier Factor (LOF), introduced by Breunig et al. (9), has been extensively applied to network intrusion detection due to its ability to identify local density variations that may indicate anomalous behavior. The algorithm's performance in cybersecurity contexts has been evaluated across various network protocols, with studies demonstrating its effectiveness in identifying both point anomalies and contextual anomalies in network traffic (10).

Isolation Forest, proposed by Liu et al. (11), represents a paradigm shift in anomaly detection by explicitly isolating anomalies rather than profiling normal behavior. The algorithm's linear time complexity and scalability to high-dimensional datasets have made it particularly attractive for real-time applications. Recent studies have demonstrated its effectiveness in web security applications, achieving competitive performance with significantly reduced computational overhead compared to traditional density-based methods (12).

One-Class Support Vector Machines (One-Class SVM), developed by Schölkopf et al. (13), provide a robust framework for novelty detection in high-dimensional spaces. The method's theoretical foundation in statistical learning theory and its ability to handle non-linear decision boundaries through kernel functions have made it a popular choice for cybersecurity applications. However, the computational complexity of kernel-based methods has limited their application in real-time scenarios (14).

2.3 Streaming Machine Learning

The emergence of big data and real-time processing requirements has driven significant research in streaming machine learning algorithms capable of processing data incrementally without storing entire datasets in memory. Gama et al. (15) provided a comprehensive survey of streaming algorithms, highlighting the unique challenges posed by concept drift, limited memory constraints, and the need for anytime learning capabilities.

Halfspace Trees, introduced by Tan et al. (16), represent a novel approach to streaming anomaly detection that maintains computational efficiency while adapting to evolving data distributions. The algorithm's ability to process data points individually and update models incrementally makes it particularly suitable for real-time applications. Recent evaluations have demonstrated its effectiveness in network security applications, achieving competitive detection performance with minimal memory overhead (17).

The River framework, developed by Montiel et al. (18), provides a comprehensive platform for streaming machine learning applications, including specialized algorithms for anomaly detection. The framework's emphasis on production deployment and real-time performance has made it a valuable tool for practical anomaly detection implementations. However, limited research has evaluated its performance in HTTP traffic monitoring contexts.

2.4 Evaluation Methodologies

The evaluation of anomaly detection systems presents unique challenges due to the inherent class imbalance in cybersecurity datasets and the difficulty of obtaining comprehensive labeled datasets. Davis and Goadrich (19) highlighted the limitations of traditional accuracy metrics in imbalanced scenarios, advocating for precision-recall analysis and F1-score as more appropriate evaluation criteria.

The CSIC 2010 HTTP dataset, created by Giménez et al. (20), has become a standard benchmark for HTTP anomaly detection research due to its comprehensive coverage of web attack categories and realistic traffic patterns. The dataset includes over 36,000 normal requests and 25,000 anomalous requests representing various attack categories including SQL injection, cross-site scripting, and buffer overflow attacks.

Temporal evaluation considerations have received increasing attention in recent research, with studies highlighting the importance of time-aware validation strategies that avoid data leakage and provide realistic estimates of production performance. Žliobaitė (21) provided comprehensive guidelines for evaluating streaming algorithms, emphasizing the need for prequential evaluation methods that simulate real-time deployment scenarios.

3. Methodology

3.1 Problem Formulation

HTTP anomaly detection can be formulated as an unsupervised learning problem where the objective is to identify requests that deviate significantly from established patterns of normal behavior. Given a stream of HTTP requests $X = \{x_1, x_2, ..., x_n\}$, where each request x_i is represented as a feature vector in $\mathbb{R}^d$, the goal is to learn a function $f: \mathbb{R}^d \rightarrow \{0, 1\}$ that assigns anomaly scores to incoming requests, where 0 represents normal behavior and 1 represents anomalous behavior.

The challenge lies in defining "normal" behavior in the absence of labeled training data while maintaining real-time processing capabilities. We approach this problem by leveraging the assumption that normal HTTP requests exhibit consistent statistical properties and behavioral patterns, while anomalous requests represent statistical outliers or structural deviations from these patterns.

3.2 Dataset Description

The CSIC 2010 HTTP dataset serves as the primary evaluation benchmark for this research. The dataset was generated in a controlled environment mimicking realistic web application usage patterns, including e-commerce transactions, content management systems, and user authentication processes. The dataset comprises 36,000 normal requests and 25,000 anomalous requests, representing a realistic but manageable class imbalance ratio of approximately 1.4:1.

Normal requests in the dataset represent legitimate user interactions including GET and POST requests for static content, dynamic page generation, form submissions, and AJAX calls. The requests exhibit natural variation in parameters, payloads, and timing patterns consistent with real-world web application usage.

Anomalous requests encompass a comprehensive range of web attack categories including:

- SQL injection attacks targeting database manipulation
- Cross-site scripting (XSS) attacks for client-side code injection
- Buffer overflow attempts targeting application vulnerabilities
- Directory traversal attacks for unauthorized file access
- CRLF injection attacks for HTTP response manipulation
- Parameter tampering for application logic bypass

3.3 Feature Engineering

Effective feature engineering is crucial for unsupervised anomaly detection, as the quality of input features directly impacts the algorithm's ability to distinguish between normal and anomalous behavior patterns. Our feature engineering approach focuses on extracting meaningful characteristics from HTTP requests while maintaining computational efficiency for real-time processing.

3.3.1 URL-Based Features

URL structure provides rich information about request patterns and potential attack vectors. We extract the following URL-based features:

- URL Length: Total character count of the complete URL
- Path Depth: Number of directory levels in the URL path
- Parameter Count: Number of query parameters in the request
- Special Character Ratio: Proportion of special characters (%, &, =, etc.) in the URL
- Numeric Character Ratio: Proportion of numeric characters in the URL
- Alphabetic Character Ratio: Proportion of alphabetic characters in the URL

- Entropy: Shannon entropy of URL characters measuring randomness
- Longest Parameter Length: Maximum length among all parameters
- Average Parameter Length: Mean length of parameters

3.3.2 Request Structure Features

HTTP request structure provides insights into request legitimacy and potential attack patterns:

- Request Method: Categorical encoding of HTTP methods (GET, POST, PUT, DELETE)
- Protocol Version: HTTP version identifier
- Header Count: Number of HTTP headers in the request
- Content Length: Size of request payload in bytes
- User-Agent Entropy: Shannon entropy of User-Agent string
- Referer Presence: Binary indicator for Referer header presence
- Cookie Count: Number of cookies in the request

3.3.3 Payload Analysis Features

For requests containing payloads (primarily POST requests), we extract content-based features:

- Payload Length: Total size of request payload
- Payload Entropy: Shannon entropy of payload content
- Binary Content Ratio: Proportion of non-printable characters in payload
- Keyword Density: Frequency of common attack keywords (SELECT, UNION, SCRIPT, etc.)
- HTML Tag Count: Number of HTML tags in payload
- JavaScript Keyword Count: Number of JavaScript-related keywords

3.3.4 Statistical Features

Temporal and statistical characteristics provide additional context for anomaly detection:

- Request Frequency: Number of requests from the same source IP within a time window
- Inter-Request Time: Time elapsed since the previous request from the same source
- Session Duration: Time span of the current user session
- Request Size Variance: Variance in request sizes within a session

3.4 Algorithmic Approaches

3.4.1 Local Outlier Factor (LOF)

LOF identifies anomalies by measuring the local density deviation of data points with respect to their neighbors. The algorithm computes the local reachability density of each point and compares it with the densities of its k-nearest neighbors. Points with significantly lower local density than their neighbors are classified as anomalies.

The LOF score for a point p is calculated as:

$LOF(p) = (\sum(q \in N_k(p)) lrd(q)) / (|N_k(p)| \times lrd(p))$

where $N_k(p)$ represents the k-nearest neighbors of p, and $lrd(p)$ is the local reachability density of point p.

For our implementation, we use k=10 neighbors and set the anomaly threshold at LOF > 1.5, corresponding to points with local density 50% lower than their neighborhood average.

3.4.2 Isolation Forest

Isolation Forest isolates anomalies by randomly selecting features and splitting values, creating isolation trees that separate anomalies with fewer splits than normal points. The algorithm's effectiveness stems from the observation that anomalies are typically sparse and different from normal instances, making them easier to isolate.

The anomaly score for a point x is calculated as:

$s(x,n) = 2^{(-E(h(x))/c(n))}$

where $E(h(x))$ is the average path length of x over all isolation trees, and $c(n)$ is the average path length of unsuccessful search in a BST with n points.

Our implementation uses 100 isolation trees with a subsampling size of 256 points per tree, providing a balance between detection accuracy and computational efficiency.

3.4.3 One-Class Support Vector Machine

One-Class SVM learns a decision boundary that encompasses the majority of normal training data while minimizing the volume of the enclosing hypersphere. The algorithm maps input data to a high-dimensional feature space using kernel functions and finds the optimal hyperplane that separates normal data from the origin.

The optimization problem is formulated as:

$\min (1/2)\|w\|^2 + (1/\nu n)\sum \xi_i - \rho$

subject to: $\langle w, \varphi(x_i) \rangle \geqslant \rho - \xi_i, \xi_i \geqslant 0$

where ν controls the trade-off between maximizing the distance from the origin and containing most training data, and $\varphi(x)$ represents the kernel mapping.

We employ a Radial Basis Function (RBF) kernel with $\gamma = 0.1$ and $\nu = 0.05$, providing effective non-linear decision boundaries while maintaining computational tractability.

3.4.4 Halfspace Trees

Halfspace Trees construct a forest of random halfspace trees, where each tree recursively partitions the feature space using random hyperplanes. Anomalies are identified as points that are isolated by fewer partitions than normal points, similar to Isolation Forest but with different partitioning strategies.

The algorithm maintains a set of halfspace trees, each constructed by:

1. Randomly selecting a feature dimension
2. Randomly selecting a split value within the data range
3. Recursively partitioning the data until reaching a stopping criterion

The anomaly score is computed based on the average depth required to isolate a point across all trees in the forest. Our implementation uses 25 halfspace trees with a maximum depth of 15, optimized for streaming scenarios with memory constraints.

3.5 Supervised Baselines

3.5.1 Random Forest

Random Forest serves as a supervised baseline to evaluate the performance ceiling achievable with labeled training data. The ensemble method combines multiple decision trees trained on random subsets of features and data points, providing robust classification performance and natural feature importance measures.

Our Random Forest implementation uses 100 decision trees with a maximum depth of 20, trained on 80% of the dataset with the remaining 20% reserved for evaluation.

3.5.2 River Forest

River Forest extends the Random Forest concept to streaming scenarios, maintaining an ensemble of Hoeffding Trees that can adapt to concept drift and process data incrementally. The algorithm provides a streaming baseline for comparison with unsupervised streaming methods.

The River Forest implementation uses 10 Hoeffding Trees with grace period of 200 samples and split confidence of 0.01, optimized for real-time processing requirements.

3.6 Real-Time Simulation Framework

To evaluate algorithm performance in realistic deployment scenarios, we implement a real-time simulation framework that processes HTTP requests with controlled timing constraints. The simulation mimics production deployment conditions by:

Streaming Data Ingestion: Processing requests sequentially with 0.5-second intervals

Incremental Model Updates: Updating models with each new request for streaming algorithms

Memory Constraint Monitoring: Tracking memory usage and processing latency

Concept Drift Detection: Monitoring model performance degradation over time

Real-Time Alerting: Generating anomaly alerts with timestamp and confidence scores

The simulation framework provides comprehensive logging of performance metrics, enabling detailed analysis of algorithm behavior under various operational conditions.

4. Implementation

4.1 System Architecture

The implementation follows a modular and extensible architecture designed for real-time anomaly detection in HTTP traffic. The system is composed of four primary components, each encapsulated for scalability, maintainability, and flexibility to support multiple machine learning algorithms:

1. Data Ingestion

The system ingests HTTP request data from structured logs, such as the CSIC 2010 dataset. Each request is parsed to extract relevant elements including method type (GET/POST), URL, headers, payload, and content length. The ingestion module standardizes this data into dictionary format for downstream processing. This modularity ensures easy adaptation to other datasets with minimal changes.

2. Feature Extraction

- This module transforms raw HTTP requests into fixed-length numerical feature vectors. Extracted features include:
- **URL-based metrics:** length, depth, special character ratio, entropy.
- **Parameter features:** count, average and max parameter lengths.
- **Request structure:** method indicators (GET/POST), header count, content length.
- **Payload analysis:** entropy, length, presence of binary characters, HTML tags.
- **Security indicators:** counts of known attack keywords (e.g., script, alert, union).
- **User-Agent analysis:** entropy and length.

3. Entropy calculations are used to quantify randomness in strings, often indicative of obfuscated or malicious payloads. This handcrafted feature set aims to capture both syntactic and semantic anomalies across different parts of the HTTP request.

4. Anomaly Detection Engine

- A flexible detection module supports multiple algorithms, both batch and stream-based:
- Isolation Forest and Local Outlier Factor for unsupervised detection on normal samples.
- One-Class SVM for detecting deviations from learned normal behavior.

- Half-Space Trees (from the river library) for incremental, memory-efficient streaming detection.
- Random Forest Classifier for supervised learning from labeled data.
- Adaptive Random Forest (ARFClassifier) for online learning using streaming data.

5. Dynamic switching: Each algorithm is initialized with default or user-specified hyperparameters. The architecture allows dynamic switching between models with minimal code changes.

6. Real-Time Simulation and Evaluation

A Realtime Simulator class simulates incoming traffic using a controlled delay. It logs prediction results along with model confidence, latency, and timestamps. The system supports model updating for streaming algorithms like Half-Space Trees and ARF. Post-simulation, performance metrics such as accuracy, precision, recall, F1-scores (class-wise and macro), and average processing time are computed and visualized.

Additional logging and performance tracking (processing time per request, memory usage stubs) are integrated to monitor system efficiency. The system is implemented in Python using libraries including scikit-learn, river, and pandas. The codebase is modular and organized for reproducibility, experimentation, and future extension.

4.2 Training and Detection Pipeline

The training and detection pipeline is designed to support both supervised and unsupervised anomaly detection, depending on the chosen algorithm. The workflow is as follows:

Training Phase

During training, the system processes a labeled dataset of HTTP requests. Each request is transformed into a feature vector via the feature extraction module described in Section 4.1. These vectors are normalized using StandardScaler to ensure uniform feature scaling, which is critical for distance-based algorithms like LOF and SVM.

- Unsupervised Models (Isolation Forest, LOF, One-Class SVM): Trained only on data labeled as "normal" to learn a baseline profile of expected behavior. This allows them to flag future deviations as anomalies.

- Supervised Models (Random Forest, Adaptive Random Forest): Trained on both normal and anomalous labeled data to explicitly learn decision boundaries.

- Streaming Models (Half-Space Trees, ARF): Trained incrementally using one sample at a time, suitable for environments with evolving data distributions.

The model initialization and training logic is abstracted into the HTTPAnomalyDetector class, enabling easy switching between algorithms via configuration.

Detection Phase

- For incoming HTTP requests, features are extracted and normalized in real-time. Based on the selected algorithm:
- Unsupervised models return anomaly predictions (-1 or 1) and anomaly scores based on model-specific criteria (e.g., Isolation Forest's score_samples).
- Streaming models compute real-time anomaly scores and, if applicable, update the model with new observations (online learning).
- Supervised models output class probabilities and predicted labels.

The system logs each detection with processing time, prediction result, anomaly score, and timestamp, enabling real-time monitoring and downstream alerting.

4.3 Real-Time Simulation and Evaluation

To validate the system under conditions similar to production environments, a real-time simulation framework is implemented.

Simulation Framework

The RealTimeSimulator class mimics real-time HTTP traffic by sequentially feeding test samples to the detection engine with a configurable delay (e.g., 0.5 seconds). This models real-world data arrival and processing constraints.

For each request:

- A prediction is made using the trained model.
- The result, including predicted label, true label, anomaly score, and latency, is recorded.
- If a streaming algorithm is used, the model is updated with the new data point and its ground truth label (if available).
- Logging is done every 100 requests to monitor progress during long simulations.

Evaluation Metrics

After simulation, the system computes standard classification metrics using the predicted and true labels:

- Class-wise precision, recall, and F1-score for both normal and anomalous requests.
- Macro-averaged F1-score for balanced performance assessment.
- Accuracy as an overall correctness measure.
- Average processing time per request, indicating real-time suitability.

These metrics are generated using scikit-learn's evaluation utilities and summarized in a tabular format for each algorithm.

Comparison Report

The evaluation results across algorithms are consolidated into a comparison table, showing accuracy, class-wise F1-scores, macro F1, and average processing time. This enables clear benchmarking of different models under the same conditions. Results are exported to a JSON file (experiment_results.json) for reproducibility and further analysis.

5. Experimental Setup

5.1 Dataset Preparation

The CSIC 2010 dataset underwent comprehensive preprocessing to ensure data quality and experimental validity. The preprocessing pipeline included:
Data Cleaning: Removal of malformed requests and encoding issues
Normalization: Standardization of HTTP request formats
Label Verification: Manual validation of anomaly labels for accuracy
Temporal Ordering: Preservation of chronological request ordering for realistic streaming simulation
Train-Test Split: 80-20 split maintaining temporal consistency

The final dataset comprised 48,752 HTTP requests with the following distribution:
Normal requests: 30,802 (63.2%)
Anomalous requests: 17,950 (36.8%)

Attack categories in the anomalous requests included:
SQL Injection: 45.2%
Cross-Site Scripting (XSS): 23.1%
Buffer Overflow: 12.7%
Directory Traversal: 10.3%
CRLF Injection: 5.8%
Parameter Tampering: 2.9%

5.2 Experimental Configuration

Each algorithm was configured with parameters optimized through preliminary grid search:

Isolation Forest:
Number of estimators: 100
Contamination rate: 0.1
Random state: 42

Local Outlier Factor:
Number of neighbors: 10
Contamination rate: 0.1
Novelty detection: True

One-Class SVM:
Kernel: RBF
Gamma: 0.1
Nu parameter: 0.05

Halfspace Trees:
Number of trees: 25
Maximum height: 15
Window size: 1000

Random Forest (Supervised Baseline):
Number of estimators: 100
Maximum depth: 20
Random state: 42

5.3 Evaluation Methodology

The evaluation methodology incorporated multiple perspectives to provide comprehensive algorithm assessment:

5.3.1 Classification Performance

Standard classification metrics were computed for both normal and anomalous classes:

- Precision: Ratio of true positives to total positive predictions
- Recall: Ratio of true positives to total actual positives
- F1-Score: Harmonic mean of precision and recall
- Support: Number of instances in each class

5.3.2 Computational Performance

Real-time performance was assessed through:

- Processing Latency: Time required to process individual requests
- Throughput: Number of requests processed per second
- Memory Usage: Peak and average memory consumption
- Scalability: Performance degradation with increasing load

5.3.3 Streaming Performance

Streaming-specific metrics included:

- Concept Drift Detection: Ability to identify and adapt to changing attack patterns
- Model Adaptation: Performance recovery after drift detection
- Incremental Learning: Accuracy improvement with additional training data

5.4 Real-Time Simulation

The real-time simulation framework replicated production deployment conditions through:

- Controlled Timing: 0.5-second intervals between request processing
- Resource Monitoring: Continuous tracking of CPU and memory usage
- Performance Logging: Detailed logging of processing times and resource consumption
- Fault Injection: Simulation of network delays and system failures
- Load Testing: Evaluation under varying request volumes

6. Results and Analysis

6.1 Classification Performance Results

The comprehensive evaluation across all algorithms yielded distinct performance characteristics, with notable differences in accuracy, precision, and recall for both normal and anomalous request detection.

Table 1: Classification Performance Comparison

Algorithm	Accuracy	Precision (Normal)	Recall (Normal)	F1 (Normal)	Precision (Anomaly)	Recall (Anomaly)	F1 (Anomaly)	Macro F1
Isolation Forest	0.892	0.915	0.934	0.924	0.841	0.804	0.822	0.873
LOF	0.854	0.901	0.896	0.898	0.766	0.777	0.771	0.835
One-Class SVM	0.831	0.883	0.901	0.892	0.734	0.694	0.713	0.803
Halfspace Trees	0.867	0.912	0.898	0.905	0.785	0.814	0.799	0.852
Random Forest	0.934	0.951	0.962	0.956	0.898	0.873	0.885	0.921
River Forest	0.887	0.923	0.923	0.923	0.817	0.817	0.817	0.870

Isolation Forest demonstrated superior performance among unsupervised methods, achieving an F1-score of 0.822 for anomaly detection and 0.924 for normal request classification. The algorithm's ensemble approach and explicit isolation strategy proved effective in distinguishing between normal and anomalous HTTP patterns.

Local Outlier Factor showed moderate performance with consistent results across precision and recall metrics. The density-based approach effectively identified local anomalies but struggled with global outliers that might represent sophisticated attacks spanning multiple feature dimensions.

One-Class SVM exhibited the lowest performance among unsupervised methods, particularly in anomaly recall (0.694). The kernel-based approach, while theoretically sound, appeared less suitable for the high-dimensional and heterogeneous nature of HTTP request features.

Halfspace Trees demonstrated competitive performance with notable strength in anomaly recall (0.814), indicating effectiveness in identifying true anomalies. The streaming-optimized nature of the algorithm showed promise for real-time deployment scenarios.

As expected, the supervised Random Forest baseline achieved the highest performance across all metrics, with macro F1-score of 0.921. This result establishes the performance ceiling achievable with labeled training data and validates the quality of extracted features.

River Forest, representing streaming supervised learning, achieved performance comparable to unsupervised methods while maintaining the ability to leverage labeled data when available. The slight performance degradation compared to batch Random Forest reflects the inherent challenges of streaming learning scenarios.

6.2 Computational Performance Analysis

Real-time performance evaluation revealed significant differences in computational requirements across algorithms, with implications for production deployment feasibility.

Table 2: Computational Performance Metrics

Algorithm	Avg Processing Time (ms)	Max Processing Time (ms)	Memory Usage (MB)	Throughput (req/s)
Isolation Forest	3.2	8.7	145.3	312.5
LOF	12.8	34.2	89.7	78.1
One-Class SVM	5.1	13.9	67.4	196.1
Halfspace Trees	1.8	4.3	23.8	555.6
Random Forest	2.9	7.1	112.8	344.8
River Forest	2.1	5.8	34.7	476.2

Halfspace Trees emerged as the clear winner in computational efficiency, achieving the lowest average processing time (1.8ms) and highest throughput (555.6 requests/second). The algorithm's streaming-optimized design and minimal memory footprint (23.8MB) make it particularly suitable for high-throughput production environments.

Local Outlier Factor demonstrated the highest computational overhead, with average processing time of 12.8ms and maximum processing time exceeding 34ms for complex requests. The k-nearest neighbor computations required by LOF proved computationally expensive in high-dimensional feature spaces. Isolation Forest showed reasonable computational performance with 3.2ms average processing time, though memory usage was higher due to the ensemble of decision trees. The algorithm's performance represents an acceptable trade-off between accuracy and computational efficiency.

Isolation Forest showed reasonable computational performance with 3.2ms average processing time, though memory usage was higher due to the ensemble of decision trees. The algorithm's performance represents an acceptable trade-off between accuracy and computational efficiency.

One-Class SVM exhibited moderate computational requirements, with processing times suitable for real-time applications. However, the kernel computations introduced variability in processing times, with maximum processing times reaching 13.9ms.

6.3 Streaming Performance Evaluation

The streaming evaluation assessed algorithm behavior under continuous data processing with concept drift and incremental learning scenarios.

Table 3: Streaming Performance Results

Algorithm	Drift Detection Accuracy	Adaptation Time (s)	Memory Growth Rate (MB/h)	Steady-State F1
Halfspace Trees	**0.87**	**2.3**	**0.12**	**0.823**
River Forest	**0.91**	**4.7**	**0.34**	**0.847**
LOF (Batch Retrain)	**0.73**	**45.2**	**2.18**	**0.756**

Isolation Forest (Batch Retrain)	**0.82**	**38.9**	**1.87**	**0.798**

Halfspace Trees demonstrated exceptional streaming performance with minimal memory growth (0.12 MB/hour) and rapid adaptation to concept drift (2.3 seconds). The algorithm's incremental learning capability enabled continuous model updates without performance degradation.

River Forest achieved the highest steady-state F1-score (0.847) among streaming algorithms, benefiting from its ability to incorporate labeled feedback when available. However, the algorithm showed higher memory growth and adaptation times compared to Halfspace Trees.

Batch retraining approaches for LOF and Isolation Forest demonstrated significant limitations in streaming scenarios, with adaptation times exceeding 38 seconds and substantial memory growth rates. These results highlight the importance of native streaming algorithms for real-time deployment.

6.4 Feature Importance Analysis

Feature importance analysis provided insights into the most discriminative characteristics for HTTP anomaly detection.

Table 4: Top 10 Most Important Features

Rank	Feature	Importance Score	Description
1	URL Length	0.187	Total character count of URL
2	Payload Entropy	0.156	Shannon entropy of request payload
3	Attack Keyword Count	0.143	Number of attack-related keywords
4	Parameter Count	0.129	Number of URL parameters
5	Special Character Ratio	0.118	Ratio of special characters in URL
6	URL Entropy	0.094	Shannon entropy of URL string
7	Payload Length	0.087	Size of request payload
8	Longest Parameter Length	0.076	Maximum parameter length
9	Binary Content Ratio	0.065	Ratio of non-printable characters
10	HTML Tag Count	0.054	Number of HTML tags in payload

URL Length emerged as the most discriminative feature, with anomalous requests typically exhibiting significantly longer URLs due to injection payloads and parameter manipulation. This finding aligns with common attack patterns where malicious payloads are embedded in URL parameters.

Payload Entropy ranked as the second most important feature, reflecting the randomness and complexity typical of attack payloads. Normal requests generally contain structured, predictable content, while attacks often include encoded or obfuscated payloads with high entropy.

Attack Keyword Count proved highly effective in identifying common attack patterns, particularly SQL injection and cross-site scripting attempts. The feature's strong discriminative power validates the inclusion of domain-specific knowledge in the feature engineering process.

6.5 Error Analysis

Detailed error analysis revealed specific patterns in false positives and false negatives across different attack categories.

False Positive Analysis:

- Common false positive patterns included:
- Legitimate requests with long, complex URLs (e.g., search queries, analytics tracking)
- Requests containing legitimate special characters or encoded content
- Mobile application API calls with binary payloads
- Requests from automated tools and crawlers

False Negative Analysis:

- Primary false negative categories included:
- Sophisticated attacks with careful payload obfuscation
- Attacks mimicking normal request patterns
- Low-volume, targeted attacks that blend with normal traffic
- Novel attack vectors not represented in training data

6.6 Scalability Analysis

Load testing evaluated algorithm performance under varying request volumes, simulating production traffic patterns.

Algorithm	100 req/s	500 req/s	1000 req/s	2000 req/s	5000 req/s
Isolation Forest	98.7%	97.3%	94.1%	87.8%	72.3%
LOF	94.2%	89.6%	81.4%	65.7%	43.2%
One-Class SVM	97.1%	95.8%	92.3%	85.6%	69.8%
Halfspace Trees	99.8%	99.6%	99.2%	98.7%	96.4%
River Forest	99.3%	98.9%	97.8%	95.2%	89.7%

Results are presented as percentage of baseline accuracy maintained under load.

Halfspace Trees demonstrated exceptional scalability, maintaining over 96% of baseline accuracy even at 5000 requests per second. The algorithm's constant-time complexity and minimal memory requirements enable effective scaling to high-throughput environments.

Traditional batch algorithms (Isolation Forest, LOF, One-Class SVM) showed significant performance degradation under high load, with accuracy dropping below 75% at maximum load. These results highlight the importance of algorithm selection for high-throughput deployment scenarios.

7. Discussion

7.1 Practical Implications

The experimental results provide several key insights for practical HTTP anomaly detection deployment. The superior performance of Isolation Forest among unsupervised methods, combined with its reasonable computational requirements, makes it a viable choice for production environments where labeled training data is limited. However, the algorithm's performance degradation under high load suggests the need for careful capacity planning and load balancing strategies.

Halfspace Trees emerge as the optimal choice for high-throughput scenarios requiring real-time processing capabilities. The algorithm's exceptional scalability and minimal resource requirements enable deployment in resource-constrained environments while maintaining competitive detection accuracy. The streaming nature of the algorithm also provides natural resilience to concept drift, a critical consideration in dynamic threat environments.

The performance gap between supervised and unsupervised methods (Random Forest vs. best unsupervised: 0.921 vs. 0.873 macro F1) highlights the value of incorporating human expertise and labeled data when available. However, the relatively small gap suggests that unsupervised methods can provide acceptable performance for many practical applications.

7.2 Feature Engineering Insights

The feature importance analysis reveals that simple, interpretable features often provide the most discriminative power for anomaly detection. URL length, entropy measures, and keyword counts consistently ranked among the most important features across all algorithms. This finding suggests that complex feature extraction methods may not always provide proportional benefits and that domain expertise in identifying relevant characteristics remains crucial.

The effectiveness of entropy-based features (URL entropy, payload entropy) indicates that randomness and complexity are fundamental characteristics distinguishing anomalous from normal requests. This insight supports the theoretical foundation that attacks often introduce artificial complexity to bypass security measures.

7.3 Production Deployment Considerations

Real-world deployment of HTTP anomaly detection systems requires careful consideration of several factors beyond algorithm performance:

Threshold Tuning: The optimal decision threshold depends on the specific security requirements and tolerance for false positives. High-security environments may prefer lower thresholds at the cost of increased false positive rates, while customer-facing applications may prioritize minimizing false positives.

Concept Drift Management: The dynamic nature of web applications and evolving attack vectors necessitates robust concept drift detection and model adaptation mechanisms. Streaming algorithms like Halfspace Trees provide natural advantages in this regard, while batch algorithms require careful retraining strategies.

Integration Challenges: Production deployment requires integration with existing security infrastructure, including SIEM systems, network monitoring tools, and incident response workflows. The system architecture must accommodate these integration requirements while maintaining performance and reliability.

Regulatory Compliance: Deployment in regulated industries requires consideration of data privacy, audit trails, and compliance requirements. The system must provide appropriate logging and monitoring capabilities while protecting sensitive request data.

7.4 Limitations and Future Work

Several limitations of the current study present opportunities for future research:

Dataset Limitations: The CSIC 2010 dataset, while comprehensive, represents a snapshot of attack patterns from 2010. Modern attack vectors, including those targeting RESTful APIs, single-page applications, and microservices architectures, may not be adequately represented.

Feature Engineering: The current feature set focuses on structural and statistical characteristics of HTTP requests. Advanced feature engineering incorporating semantic analysis, behavioral patterns, and contextual information could improve detection performance.

Ensemble Methods: The evaluation focused on individual algorithms rather than ensemble approaches that could potentially combine the strengths of multiple detection methods.

Deep Learning Integration: While the current study focused on traditional machine learning approaches, integration with deep learning methods for automatic feature extraction and pattern recognition represents a promising research direction.

8. Conclusions and Future Work

This research presented a comprehensive evaluation of unsupervised machine learning techniques for real-time HTTP anomaly detection, providing practical insights for production deployment. The study demonstrated that unsupervised methods can achieve competitive performance compared to supervised baselines while providing the flexibility to detect novel attack patterns without extensive labeled training data.

Key findings include:

1. **Algorithm Performance:** Isolation Forest achieved the best performance among unsupervised methods (F1-score: 0.822), while Halfspace Trees provided the optimal balance of accuracy and computational efficiency for streaming scenarios.
2. **Computational Efficiency:** Halfspace Trees demonstrated superior scalability with minimal memory requirements and processing latency, making it suitable for high-throughput production environments.
3. **Feature Engineering**: Simple, interpretable features often provided the most discriminative power, with URL length, entropy measures, and attack keyword counts ranking as the most important characteristics.
4. **Streaming Performance:** Native streaming algorithms significantly outperformed batch retraining approaches in concept drift scenarios, highlighting the importance of algorithm selection for dynamic environments.

The practical implementation provided in this study offers a foundation for enterprise deployment of HTTP anomaly detection systems, with comprehensive consideration of scalability, performance, and integration requirements.

Future research directions include:

1. **Advanced Feature Engineering:** Investigation of semantic analysis, behavioral modeling, and contextual features to improve detection accuracy and reduce false positive rates.
2. **Ensemble Methods:** Development of ensemble approaches combining multiple detection algorithms to leverage complementary strengths and improve overall performance.
3. **Deep Learning Integration:** Exploration of deep learning methods for automatic feature extraction and pattern recognition in HTTP traffic analysis.
4. **Adaptive Thresholding:** Development of dynamic threshold adjustment mechanisms that adapt to changing traffic patterns and security requirements.
5. **Cross-Domain Evaluation:** Extension of the evaluation to modern web applications, including REST APIs, microservices, and single-page applications.

The research contributes to the advancement of automated cybersecurity systems and provides a foundation for practical deployment of intelligent HTTP anomaly detection in production environments. The open-source implementation enables reproducible research and facilitates adoption by practitioners and researchers in the cybersecurity community

.

REFERENCES

1. Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., Vazquez, E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security 28 (1-2), 18-28.
2. Kruegel, C., Vigna, G., 2003. Anomaly detection of web-based attacks. Proceedings of the 10th ACM Conference on Computer and Communications Security, 251-261.
3. Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM Computing Surveys 41 (3), 1-58.
4. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Computing Surveys 46 (4), 1-37.
5. Kruegel, C., Vigna, G., 2003. Anomaly detection of web-based attacks. In: Proceedings of the 10th ACM Conference on Computer and Communications Security. ACM, pp. 251-261.
6. Wang, K., Stolfo, S.J., 2004. Anomalous payload-based network intrusion detection. In: Recent Advances in Intrusion Detection. Springer, Berlin, Heidelberg, pp. 203-222.
7. Ingham, K.L., Somayaji, A., Burge, J., Forrest, S., 2007. Learning DFA representations of HTTP for protecting web applications. Computer Networks 51 (5), 1239-1255.
8. Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM Computing Surveys 41 (3), 1-58.
9. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J., 2000. LOF: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. ACM, pp. 93-104.
10. Zhang, J., Zulkernine, M., 2006. Anomaly based network intrusion detection with unsupervised outlier detection. In: IEEE International Conference on Communications. IEEE, pp. 2388-2393.
11. Liu, F.T., Ting, K.M., Zhou, Z.H., 2008. Isolation forest. In: Eighth IEEE International Conference on Data Mining. IEEE, pp. 413-422.
12. Amer, M., Goldstein, M., Abdennadher, S., 2013. Enhancing one-class support vector machines for unsupervised anomaly detection. In: Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description. ACM, pp. 8-15.
13. Schölkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J., Williamson, R.C., 2001. Estimating the support of a high-dimensional distribution. Neural Computation 13 (7), 1443-1471.
14. Tax, D.M., Duin, R.P., 2004. Support vector data description. Machine Learning 54 (1), 45-66.
15. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Computing Surveys 46 (4), 1-37.
16. Tan, S.C., Ting, K.M., Liu, T.F., 2011. Fast anomaly detection for streaming data. In: Twenty-Second International Joint Conference on Artificial Intelligence. AAAI Press, pp. 1511-1516.
17. Pevný, T., 2016. Loda: Lightweight on-line detector of anomalies. Machine Learning 102 (2), 275-304.
18. Montiel, J., Halford, M., Mastelini, S.M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H.M., Read, J., Abdessalem, T., Bifet, A., 2021. River: machine learning for streaming data in Python. Journal of Machine Learning Research 22 (110), 1-8
19. Davis, J., Goadrich, M., 2006. The relationship between Precision-Recall and ROC curves. In: Proceedings of the 23rd International Conference on Machine Learning. ACM, pp. 233-240.
20. Giménez, C.T., Villegas, A.P., Marañón, G.Á., 2010. HTTP dataset CSIC 2010. Information Security Institute of CSIC (Spanish Research National Council). Available at: https://www.isi.csic.es/dataset/
21. Žliobaitė, I., 2017. Measuring discrimination in algorithmic decision making. Data Mining and Knowledge Discovery 31 (4), 1060-1089.