Unsupervised Hybrid Approach for Intrusion Detection Based on Information System Log Files

Sena Apeke[1], Nadjime Pindra[2], Sèmèvo A. R. M. Ahouandjinou[3]
Laboratoire de Recherche en Science de l'Ingénieur, Universite de Lomé, Togo[1]
Centre d'Excellence Régional pour la Maîtrise de l'Electricité (CERME), Universite de Lomé, Lomé, Togo[1]
Laboratoire d'Analyse, de Modélisations Mathématiques et Applications, Universite de Lomé, Togo[2]
Laboratoire de recherche en Sciences Informatique et Applications, Université d'Abomey-calavi, Bénin[3]

***Abstract*—Intrusion detection in web traffic remains a challenging task due to the high dimensionality, heterogeneity, and imbalance of normal and malicious requests. This study investigates a hybrid anomaly detection framework combining an Autoencoder (AE), Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and Isolation Forest (IF) for intrusion detection in HyperText Transfer Protocol (HTTP) log data. The Autoencoder was first used to learn a compact latent representation of the traffic, after which different unsupervised detection pipelines were evaluated. A key contribution of this work is the analysis of pipeline ordering, specifically comparing AE + DBSCAN + IF, AE + IF, and AE + DBSCAN. The full hybrid pipeline AE + DBSCAN + IF achieved the highest accuracy (0.9803) and strong false positive control, but exhibited extremely poor sensitivity with a recall of 0.0082 and precision of 0.0904, making it too conservative for practical intrusion detection. The intermediate pipeline AE + IF improved sensitivity, reaching a recall of 0.1542 and precision of 0.0536, but at the expense of a higher false positive rate. Surprisingly, the ablation study showed that AE + DBSCAN alone yielded the most balanced results, with accuracy of 0.9670, precision of 0.2606, and recall of 0.4376, clearly outperforming the complete hybrid pipeline in intrusion detection capability. These findings demonstrate that the ordering of anomaly detection stages has a critical impact on IDS performance. In the current setting, Isolation Forest degrades rather than improves detection when combined with DBSCAN. The results suggest that AE + DBSCAN constitutes the most effective and operationally relevant configuration for the studied dataset.**

***Keywords*—Intrusion; detection; DBSCAN; Isolation Forest; Autoencoder; server log files; anomaly**

I. Introduction

In the age of digital transformation, servers form the operational foundation of modern IT infrastructures. They host critical applications, centralize databases, orchestrate network services, and ensure the continuous availability of digital resources that are essential to public and private organizations. Whether for government agencies, universities, hospitals, businesses, or online service providers, server reliability directly impacts business continuity, service quality, and the protection of information assets. This strategic position, however, makes servers prime targets for cyberattacks, particularly malicious intrusions, which aim to compromise the confidentiality, integrity, or availability of information systems [5], [2], [6], [3].

An intrusion can be defined as any unauthorized attempt or action aimed at accessing a system, disrupting its operation, exfiltrating data, disrupting services, or maintaining a covert presence within it. In server environments, these attacks take various forms: privilege escalation, lateral movement, exploitation of software vulnerabilities, execution of malicious commands, installation of malware, Denial-of-Service attacks, or compromise of authentication mechanisms. The consequences can be particularly severe: leakage of sensitive data, unavailability of critical services, financial losses, regulatory noncompliance, damage to institutional reputation, and a breach of user trust [18], [14], [10]. In this context, server security is no longer solely a matter of perimeter defense, but rather of intelligent, continuous, and adaptive monitoring of system events.

The rapidly evolving threat landscape has exposed the limitations of traditional security approaches. Conventional defense mechanisms, such as firewalls, antivirus software, and signature-based intrusion detection systems, remain useful for identifying known behaviors, but they often prove insufficient against new, polymorphic, or stealthy attacks. Signature-based intrusion detection systems (IDS) generally compare observed events against a database of rules or fingerprints corresponding to previously documented attacks. This approach is effective against known threats but becomes less reliable when the attacker employs novel, modified, or deliberately obfuscated techniques [8], [1], [19]. This issue is particularly significant in the case of so-called zero-day attacks, which exploit vulnerabilities that have not yet been documented or patched. Although some security solutions can indirectly detect certain behavioral manifestations of such attacks, their reliable detection remains a major challenge, particularly in complex and highly dynamic environments [8], [1].

At the same time, servers constantly generate massive amounts of data in the form of event logs produced by the operating system, network services, applications, authentication mechanisms, administration tools, and security components. These logs record a wide variety of information: incoming and outgoing connections, system errors, file access, authentication failures, executed commands, application requests, security alerts, and performance anomalies. In theory, these traces represent an extremely rich source for monitoring, auditing, and the early detection of suspicious behavior. In practice, however, leveraging them remains difficult due to several factors: heterogeneous formats, significant noise, lack of uniform structure, redundancy, high volume, and the semantic complexity of the messages [12], [4], [1]. Manual analysis of this data quickly becomes impractical, especially in operational contexts where security teams have limited human resources

and must respond within very short timeframes.

This challenge has led to the emergence of advanced intrusion detection approaches based on machine learning (ML) and deep learning (DL). Unlike purely signature-based approaches, machine learning methods aim to model a system's normal behavior or learn discriminative patterns from historical data in order to identify significant deviations that may indicate malicious activity. These approaches are particularly promising for server log analysis, as they allow for the exploitation of temporal, structural, and contextual patterns that are difficult to capture using static rules [9], [17], [1].

The growing interest in ML and DL techniques in IDS systems can also be attributed to their ability to address the increasing complexity of modern cyberattacks. Models such as decision trees, random forests, support vector machines, autoencoders, recurrent neural networks, convolutional models, and hybrid architectures have demonstrated promising performance in various detection contexts, whether analyzing network traffic, user behavior, or log sequences [6], [9], [1].

From this perspective, intelligent server log analysis emerges as a strategic priority for strengthening the cybersecurity of digital infrastructures. It not only enables earlier detection of abnormal behavior but also provides a better understanding of attack mechanisms, reduces response times, and improves the overall resilience of systems. However, several challenges remain: high variability in server environments, an imbalance between normal and malicious events, computational cost, the need for explainability, false positive rates, and the difficulty of integration into real-world operational environments [1], [12], [10]. This study fits within this context and aims to explore the use of log files for server intrusion detection using modern artificial intelligence techniques. The goal is to propose an approach capable of leveraging these execution traces as a source of actionable knowledge, in order to improve security monitoring, anomaly detection, and, more broadly, the proactive protection of server systems.

Despite recent progress in machine learning and deep learning for intrusion detection, two research gaps remain insufficiently addressed. First, most existing studies rely on benchmark network datasets, whereas real server log files are highly heterogeneous, noisy, redundant, and severely imbalanced. Second, the influence of the ordering of unsupervised anomaly detectors within hybrid pipelines has rarely been investigated. In particular, it remains unclear whether density-based clustering should be applied before or after isolation-based anomaly detection when the input consists of engineered server log features. This work addresses these gaps by evaluating the impact of component ordering in three pipelines: AE+DBSCAN+IF, AE+IF+DBSCAN, and AE+DBSCAN.

The remainder of this study is organized as follows: Section II presents the dataset, the feature engineering process, and the proposed hybrid intrusion detection methodology. Section III reports the experimental results and compares the investigated pipelines. Section IV discusses the main findings, the effect of pipeline ordering, and the limitations of the approach. Finally, Section V concludes the study and outlines future research directions.

II. Methods and Materials

A. Data Collection and Description

This study addresses intrusion detection from server log files in a cybersecurity context. The selected data consist mainly of Apache server access logs, which are semi-structured records generated by web services. Such logs play an important role in threat identification, suspicious behavior analysis, and decision support for protecting critical information systems. Their quality directly influences the ability of information systems to detect unauthorized access attempts, Denial-of-Service (DoS) events [11], and malicious injection attempts [21]. Apache access logs record each HTTP request and include information such as the IP address, timestamp, HTTP method, requested URL, status code, response size, referrer, and user agent. These attributes are useful for identifying suspicious activities, including repeated authentication failures, abnormal status codes, malformed requests, and request peaks related to Distributed Denial-of-Service (DDoS) behavior. The extracted log records are highly imbalanced, as shown in Fig. 1, which presents their temporal distribution between 2021 and 2024.

B. Data Preprocessing

Preprocessing server log files is a demanding step due to the complexity of these files. The first work done was to analyze the raw logs (.log) following the work of [16] and [15] using Python regular expression library to extract eight variables (Table I), the results of the processing of each log file were structured (.csv). Then, an algorithm was developed to eliminate redundant information following the work of [20]. This process, carried out in two phases, made it possible to convert unstructured data into an exploitable basis for intrusion detection. Files were compiled into a single one. After merging all these files, the data set contained 3,558,335 rows and 8 columns, representing HTTP requests recorded between 2021 and 2024. To clearly see the imbalance in the data, a study of the distribution of each variable was carried out. In fact, most of the variables are complex. Fig. 2 summarizes the distributions of the variables Status_code, Referer, User_Agent, and Method. More specifically, Fig. 2a, Fig. 2b, Fig. 2c, and Fig. 2d illustrate the four distributions, respectively. The eight variables defined in Table I are complex variables; to better present a line of information, it was necessary to break down these variables. For example, the IP variable has been split into 8 variables: ip_request_count represents the total number of requests made by an IP in the logs; ip_time_spread represents the duration in seconds between the first and last request of an IP, etc. The URL variable has been split into nine new ones: url_path_depth gives the number of segments in the URL path; url_path_special_chars_count gives the number of special characters in the URL path, this variable reports malformed or encoded URLs, typical attacks; fragment_param_count gives the number of parameters in the fragment (#); etc. This analysis made it possible to have 34 relevant variables which give complete coverage of intrusion signals: temporal (cyclical and linear), IP (intensity, errors, geography), URL (structure, complexity), User-Agent (automation), Referer (navigation), and status (failures). In sum, there are in the dataset $N = 3558335$ observations, each having 34 components, any observation will be noted x_i.

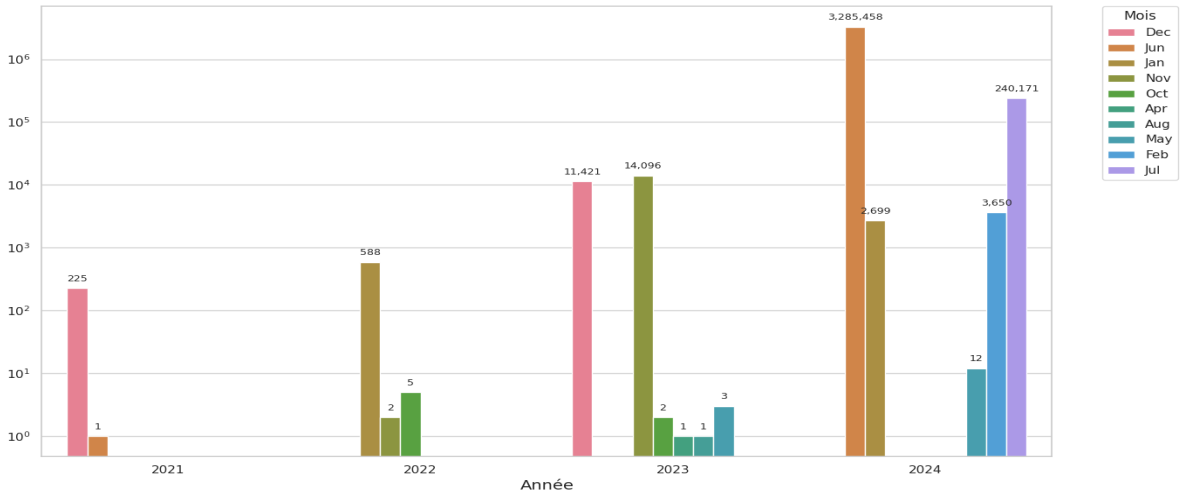
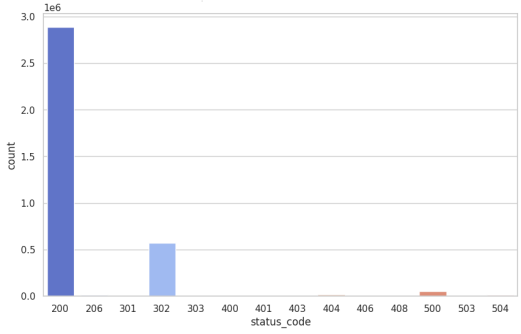
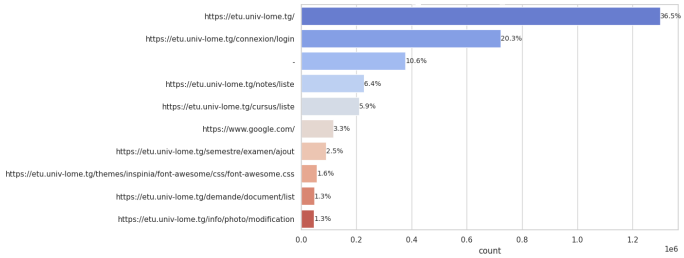


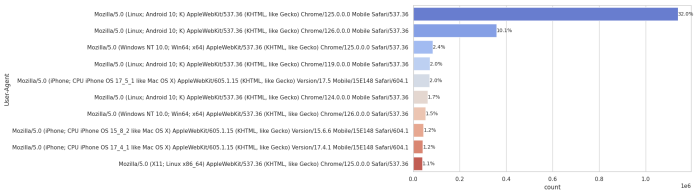
Fig. 1. Temporal distribution of information in the log files (2021–2024).



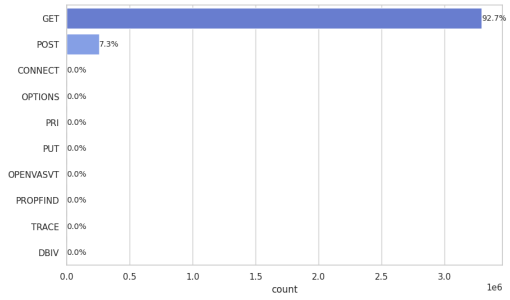
(a) Distribution of the Status_code variable.



(b) Distribution of the Referer variable.



(c) Distribution of the User_Agent variable.



(d) Distribution of the Method variable.

Fig. 2. Distribution of a few variables reflecting the unbalanced aspect of the data.

Table II provides the complete list of engineered features used as an input to the proposed intrusion detection pipelines. These variables summarize temporal behavior, IP activity, URL structure, HTTP response status, User-Agent characteristics, Referer information, and fragment-level patterns.

The overall methodology is summarized in Fig. 3. The workflow starts from raw Apache log files and proceeds through parsing, feature engineering, normalization, latent representation learning with an Autoencoder, evaluation of alternative pipeline orderings, and final performance assessment.

C. Modeling

The relevance of intrusion detection models based on log files requires that the different attack techniques be well represented in the data. The consequence is that very easily, the size of the training data becomes very large. To reduce the computational complexity in training, autoencoders were used initially. This is because autoencoders reduce dimensionality

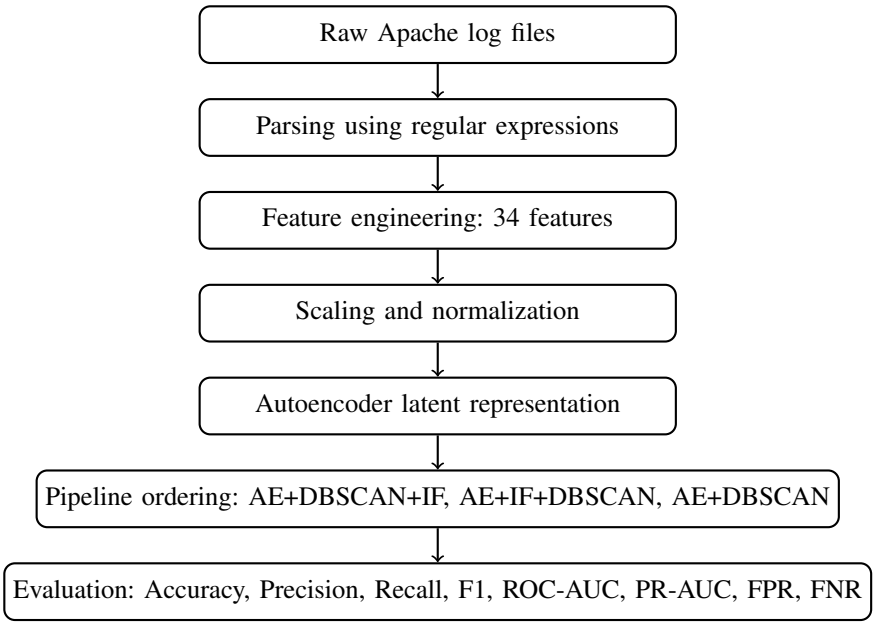


Fig. 3. Workflow of the proposed methodology for server log-based intrusion detection.

TABLE I. DESCRIPTION OF THE VARIABLES EXTRACTED FROM LOGS

Variable	Description
IP	Client IP address (e.g.: 192.168.1.1). Identifies suspicious sources (e.g. repetitive IPs in attacks DDoS)
Date	Timestamp (e.g.: 14/Mar/2025:10:15:30 +0100). Allows temporal analysis of peaks activity
Method	HTTP method (e.g.: GET, POST). Reveals actions unusual (e.g.: repeated POSTs)
URL	Requested resource (eg: /login.php). Detects access to sensitive pages or injections
Status_code	HTTP code (e.g.: 200, 404). Reports repeated failures (e.g.: 401 for brute force)
Response_size	Response size (e.g.: 1024 bytes). Indicates faults mallies (e.g.: high values for DoS)
Referer	Original URL (e.g.: http:/example.com). Detect the fraudulent or absent referents
User_agent	Client identifier (e.g.: Mozilla/5.0). Spot the bots or malicious tools

by compressing the input data into a lower-dimensional representation, known as a latent space layer, and then reconstructing the original input from this compressed representation. An autoencoder is a neural network that is trained to attempt to copy its input to its output (Fig. 5). Since autoencoders behave like a model solving a regression problem, the error function often used is the mean square error (E_{MSE}). As in machine learning or deep learning regression models, when training, we aim to minimize the cost function.

In this study, the Autoencoder architecture follows a symmetric fully connected structure: 34–64–32–16–8–16–32–64–34. Rectified Linear Unit (ReLU) activations are used in the hidden layers, the bottleneck layer contains eight latent dimensions, and the model is optimized by minimizing the mean square reconstruction error using the Adam optimizer. This architecture is illustrated in Fig. 4.

$$E_{MSE} = \frac{1}{N}\sum_{i=1}^{N} ||x_i - \tilde{x_i}||_2^2 \quad (1)$$

The symbol $||.||_2$ represents the $\mathcal{L}^2$ norm of a vector, and N is the number of the observation in the training dataset. $\tilde{x_i}$ is the decoder output. The more the error tends towards zero, the better the observations are reconstructed.

The proposed intrusion detection framework was designed under a semi-unsupervised anomaly detection paradigm, where the latent space learned by the autoencoder was exploited to improve separability between benign and suspicious traffic behaviors. Tree hybrid pipelines were conceptually evaluated: (1) AE + DBSCAN + IF, (2) AE + IF + DBSCAN and (3) AE+DBSCAN

1) Description of isolation forest: Let $X = \{x_1, \ldots, x_n\}$ be a set of information data files where $x_i \in \mathbb{R}^{34}$, 34 represents the number of characteristics (features) that describe data in this work. An isolation tree (isoTree) is defined as a data structure with an algorithm that is organized as a decision tree [13]. The principle of this algorithm is very simple:

- Randomly select a variable.
- Then carry out a random partitioning of the dataset according to this variable so as to obtain two subsets of data.
- Repeat the previous two steps until a piece of data is isolated.
- Recursively, repeat the previous steps.

isoTree is based on the idea that anomalies, rare and distinct, are isolated more quickly in random decision trees than normal points. For each recording (e.g., Apache log), a

TABLE II. DESCRIPTION OF THE ENGINEERED FEATURES EXTRACTED FROM THE SERVER LOG FILES

Feature	Description
response_size	HTTP response size in bytes.
date_hour_sin	Sine transformation of the request hour (0–23).
date_hour_cos	Cosine transformation of the request hour (0–23).
date_normalized	Elapsed time in seconds since the first request.
ip_request_count	Total number of requests generated by the same IP address.
ip_time_spread	Time elapsed between the first and last request from the same IP address.
ip_subnet_activity	Number of active IP addresses within the same /24 subnet.
ip_error_rate	Proportion of requests returning HTTP error codes (≥ 400).
ip_request_rate	Average request rate per second.
ip_interval_std	Standard deviation of time intervals between consecutive requests.
ip_max_requests_per_sec	Maximum number of requests per second generated by the same IP address.
ip_method_diversity	Number of distinct HTTP methods used by the same IP address.
country	Country of origin of the IP address based on a GeoIP database.
status_code_category	Category of the HTTP response status code.
nbre_unique_urls_per_ip	Number of distinct URLs requested by the same IP address.
url_path_struct_entropy	Structural entropy of the URL path.
url_path_depth	Number of path segments in the requested URL.
nbre_total_url_query_params	Number of query parameters in the requested URL.
url_query_special_chars_count	Number of special characters in the URL query string.
url_path_special_chars_count	Number of special characters in the URL path.
user_agent_length	Length of the User-Agent string.
user_agent_entropy	Character entropy of the User-Agent string.
user_agent_total_special_chars	Number of special characters in the User-Agent string.
referer_is_present	Binary indicator equal to 1 if the Referer header is present and 0 otherwise.
referer_path_struct_entropy	Structural entropy of the Referer path.
referer_path_depth	Number of path segments in the Referer URL.
nbre_total_referer_query_params	Number of query parameters in the Referer URL.
referer_query_special_chars_count	Number of special characters in the Referer query string.
is_bot	Binary label equal to 1 for bot traffic and 0 otherwise.
fragment_param_count	Number of parameters in the URL fragment.
special_char_fragment_count	Number of special characters in the URL fragment.

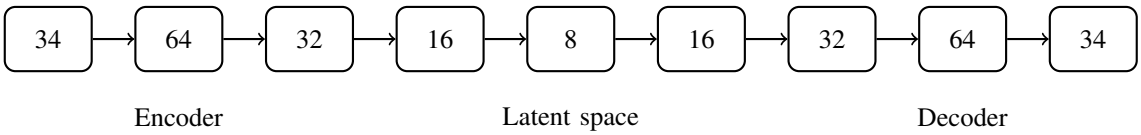


Fig. 4. Architecture of the Autoencoder used for dimensionality reduction. The input layer contains the engineered log features, the bottleneck contains eight latent units, and the decoder reconstructs the original feature vector.

set of trees is constructed by randomly selecting a feature and a cutoff value until each point is isolated. The anomaly score is given by:

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

where,

- $h(x)$: isolation depth of x in a tree (number of divisions to isolate it)
- $E(h(x))$: average of h(x) over T trees
- $c(m) = 2H(m-1) - \frac{2(m-1)}{n}$ if $m > 2$

n is the test set size, m is the size of the sample set, and H is the harmonic number, which can be estimated by: $H(k) = \sum_{i=1}^{k} \frac{1}{i}$, [7].

If the score s is close to 1, x is most likely an anomaly, but if s is less than 0.5, x is probably normal. Isolation Forest has the ability to detect global anomalies without relying on density or distance comparison. However, it is not effective in detecting local anomalies.

2) Description of DBSCAN: DBSCAN is a well-known unsupervised clustering algorithm. It was proposed in 1996 by [7]. Given points and an integer k, the algorithm aims to divide the points into k homogeneous and compact groups, called clusters. Given points and an integer k, the algorithm aims to divide the points into k groups, called clusters, which are homogeneous and compact. It is a simple algorithm that defines clusters using local density estimation. It can be divided into 4 steps: For each observation, we look at the number of points at most one distance ϵ from it. This area is called the ϵ-neighborhood of the observation. If an observation has at least a certain number of neighbors including itself, it is considered a “core" observation. In this case, a high-density observation was detected. All observations in the vicinity of a core observation belong to the same cluster. There may be core observations close to each other. As a result, from close to close, a long sequence of core observations was obtained that form a single cluster. Any observation that is not a core observation and has no core observation in its vicinity is considered an “anomaly". Two main parameters are used in DBSCAN: epsilon (ϵ), which defines the maximum distance between two points to be considered neighbors, and MinPts, which specifies the minimum number of points required to form a dense region. Note also the choice of a distance to be used (usually the Euclidean distance). Given below is the

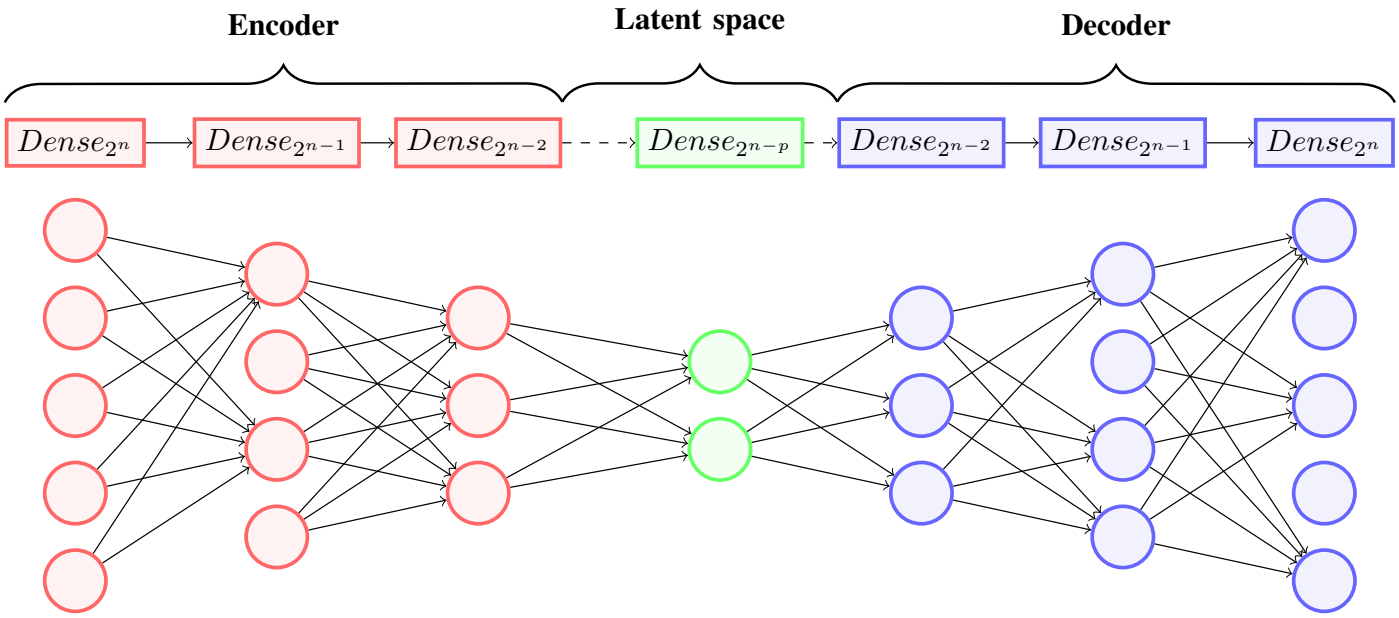


Fig. 5. Example of the diagram of an Autoencoder. $\mathbf{Dense_{2^n}}$ is a dense layer like in keras and 2^n is the number of units. $n = 7$ and $p = 4$ in the context of this study.

algorithm of DBSCAN.

1) **Initialization**
   - Consider the dataset $D = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^{34}$.
   - Set parameters: neighborhood radius ε and *MinPts*.
   - Mark all points in D as *unvisited*.
2) **Loop over points**
   - For each point $x_i \in D$:
     - If x_i is already visited, move on to the next point.
     - Otherwise, mark x_i as *visited*.
3) **Neighborhood calculation**
   - Calculate the neighborhood of x_i:

$$N_\varepsilon(x_i) = \{x_j \in D \mid ||x_j - x_i|| \leq \varepsilon\}.$$

   - If $|N_\varepsilon(x_i)| <$ MinPts, then :
     - Mark x_i as *noise*.
     - Continue with the next point.
4) **Create a cluster**
   - Create a new cluster C.
   - Add x_i to the cluster C.
5) **Cluster expansion**
   - For each point $p \in N_\varepsilon(x_i)$:
     - If p is not visited:
       - Mark p as *visited*.
       - Calculate its neighborhood $N_\varepsilon(p)$.
       - If $|N_\varepsilon(p)| \geq$ MinPts, extend the neighborhood:

$$N_\varepsilon(x_i) \leftarrow N_\varepsilon(x_i) \cup N_\varepsilon(p).$$

     - If p doesn't belong to any cluster, add it to C.
6) **End of algorithm**
   - Repeat until all points in D have been visited.
   - Return:

$$\{C_1, C_2, \ldots, C_K\}, \qquad \text{Noise} = D \setminus \bigcup_{k=1}^{K} C_k.$$

3) Goal: In this study, three methods were developed: first, the AE+DBSCAN pipeline was trained; next, the AE+DBSCAN pipeline; and finally, the AE+IF+DBSCAN pipeline. The objective was to determine which ordering of density-based clustering and anomaly isolation is more appropriate for network intrusion detection.

III. Results

The dataset was partitioned into three subsets in order to ensure a robust evaluation of the intrusion detection pipeline: training set 70% of the data, validation set 10% of the data and test set 20% of the data. The training subset was primarily used to learn the latent representation of the normal traffic through the Autoencoder (AE) and to calibrate the anomaly detection stages. The validation subset was used for threshold selection and intermediate tuning, while the test subset was strictly reserved for final performance evaluation.

Table III summarizes the main performance indicators obtained for the evaluated pipelines. In addition to accuracy, precision, and recall, the table reports the F1-score derived from precision and recall. Additional confusion matrices, ROC-AUC, PR-AUC, false positive rate (FPR), and false negative rate (FNR) values are available for the implemented experimental runs and will be systematically extended in future comparative experiments.

TABLE III. MAIN METRICS OBTAINED FOR THE EVALUATED ANOMALY DETECTION PIPELINES.

Pipeline	Accuracy	Precision	Recall	F1-score
AE+DBSCAN+IF	0.9803	0.0904	0.0082	0.0150
AE+IF	0.9347	0.0536	0.1542	0.0796
AE+DBSCAN	0.9670	0.2606	0.4376	0.3267

A. Pipeline: AE + DBSCAN + IF

After training, testing, and validating the AE+DBSCAN+IF pipeline, an accuracy of 0.9803 was achieved, followed by a precision of 0.0904 and a recall of 0.0082. This configuration exhibits extremely high accuracy and excellent control of false positives, indicating that the model is highly effective at modeling normal traffic behavior. However, this comes at the cost of a dramatic loss in sensitivity, as reflected by the extremely low recall (below 1%). This suggests that the model fails to detect the vast majority of intrusions. This behavior can be explained by the interaction between DBSCAN and Isolation Forest. DBSCAN, when applied to the latent space, constructs a very large and dense region of normality, effectively absorbing a significant portion of the data, including subtle anomalous patterns. As a consequence, many malicious samples are either classified as part of normal clusters or rejected as noise before reaching the final detection stage. Subsequently, Isolation Forest operates on a reduced and already filtered subset of data, with thresholds that are too strict to recover the missed anomalies. This results in a pipeline that is overly conservative, prioritizing specificity over sensitivity. Overall, while the pipeline achieves excellent false positive control, it is unsuitable for intrusion detection, where missing attacks is more critical than raising occasional false alarms.

B. Pipeline: AE + IF + DBSCAN

After training, testing, and validating the AE+IF+DBSCAN pipeline, an accuracy of 0.93 was achieved, followed by a precision of 0.05 and a recall of 0.15.

This configuration significantly improves recall compared to the previous one, demonstrating a better ability to detect anomalous traffic. By applying Isolation Forest before DBSCAN, the model first identifies suspicious samples based on global isolation properties in the latent space. This step ensures that anomalies are not prematurely absorbed into dense normal regions. DBSCAN is then applied only to these suspicious points, enabling the identification of coherent anomalous structures and improving the interpretability of the results. This ordering is more aligned with the nature of intrusion detection, where anomalies are often sparse and irregular rather than densely clustered. However, this improved sensitivity comes at the cost of an increased number of false positives, as reflected by the lower precision. The pipeline becomes more reactive but less selective. This configuration provides a better balance between detection and false alarm control than AE + DBSCAN + IF, but still requires careful threshold tuning and post-processing mechanisms to be operationally viable.

C. Pipeline: AE + DBSCAN

The AE + DBSCAN pipeline yields the most surprising and insightful results in this study, with an accuracy of 0.9670, a precision of 0.2606, and a recall of 0.4376. Contrary to expectations, removing Isolation Forest leads to a substantial improvement in recall and overall detection performance, with recall reaching approximately 44% and precision significantly higher than in the other configurations. This indicates that DBSCAN alone, when applied in the latent space learned by the Autoencoder, is capable of effectively distinguishing between normal and anomalous traffic patterns. The latent representation appears to preserve sufficient structure to allow density-based clustering to separate benign and malicious behaviors without requiring an additional anomaly isolation stage. The degradation observed when adding Isolation Forest suggests that, in the current configuration, IF introduces an overly restrictive filtering stage, removing or suppressing useful anomaly signals identified by DBSCAN. Instead of enhancing detection, it acts as a bottleneck that reduces the model's sensitivity. This result highlights a critical insight: additional model complexity does not necessarily improve performance, and in some cases, it can even degrade it. The AE + DBSCAN combination appears to provide a strong baseline for intrusion detection in this context.

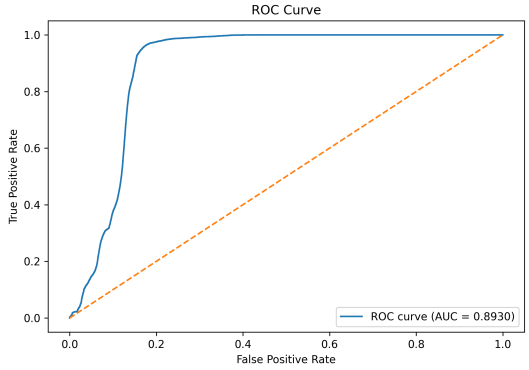


Fig. 6. Receiver Operating Characteristic (ROC) curve showing the true positive rate as a function of the false positive rate.

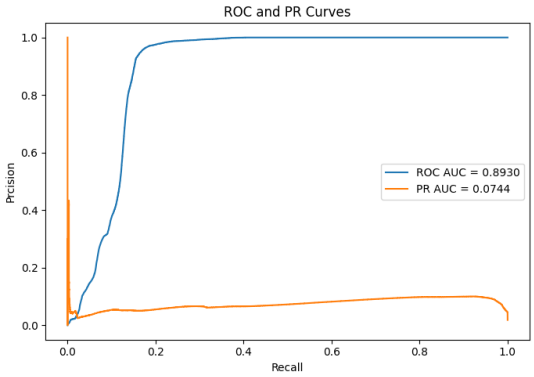


Fig. 7. Combined ROC and Precision–Recall curves used to evaluate the detection behavior under class imbalance.

The resulting figures provide a complementary and consistent view of the system's performance. The ROC curve (Fig. 6) shows only moderate overall separation between normal traffic and malicious traffic, indicating that the anomaly score partially distinguishes intrusions but without a clear boundary. This trend is confirmed by the Precision–Recall

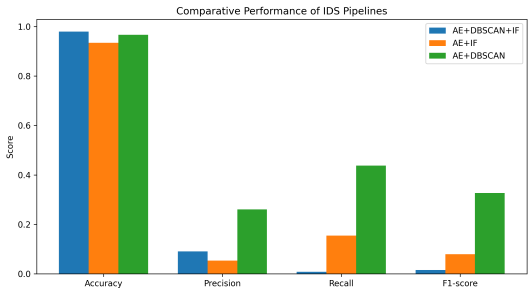


Fig. 8. Comparative performance of the evaluated IDS pipelines in terms of accuracy, precision, recall, and F1-score.

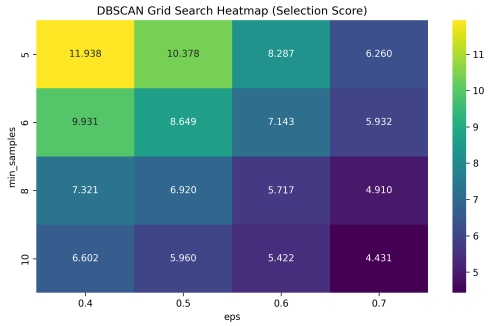


Fig. 9. Heatmap of the DBSCAN grid search showing the selection score for different values of ε and MinPts.

curve (Fig. 7), which is more relevant in a highly imbalanced IDS context. It highlights that an increase in sensitivity is quickly accompanied by a drop in precision, reflecting a delicate trade-off between detection and false positives. The comparative bar chart of the pipelines (Fig. 8) is particularly informative. It clearly shows that AE+DBSCAN currently offers the best balance between recall and precision, while AE+DBSCAN+IF primarily maximizes accuracy by becoming excessively conservative. This confirms that adding Isolation Forest, in its current configuration, degrades detection rather than improving it. Finally, the heatmap of the DBSCAN selection score (Fig. 9) shows that the clustering performance is highly dependent on the (eps, min_samples) pair, with a well defined optimal range. This figure underscores the importance of fine-tuning DBSCAN for the overall quality of the pipeline.

IV. Discussion

The results of this study provide several important insights into the design of hybrid intrusion detection systems based on representation learning and unsupervised anomaly detection. While combining multiple models is often assumed to improve performance, the ablation analysis clearly demonstrates that this assumption does not always hold. The most striking result is that the AE + DBSCAN configuration outperforms the full hybrid pipeline (AE + DBSCAN + IF) in terms of recall and overall detection capability. This suggests that the latent representation learned by the Autoencoder is already sufficiently structured to allow DBSCAN to effectively separate normal and anomalous behaviors. In this context, DBSCAN acts not only as a clustering algorithm but also as a powerful anomaly detector based on density estimation. The degradation observed when adding Isolation Forest highlights a critical issue: the introduction of an additional filtering stage can inadvertently remove meaningful anomaly signals. In the current setup, Isolation Forest appears to be too restrictive, either due to an insufficient contamination parameter or to thresholds that are not adapted to the distribution of the latent space. As a result, many anomalies identified by DBSCAN are suppressed, leading to a dramatic drop in recall.

More specifically, the ablation study indicates that Isolation Forest does not improve the detection capability in the current configuration. After DBSCAN has modeled the dense region of normal traffic, Isolation Forest operates on a reduced subset containing mostly borderline observations. Under these conditions, the anomaly thresholds become overly restrictive and many suspicious samples identified by DBSCAN are rejected, leading to a substantial decrease in recall. This behavior suggests that the current Isolation Forest configuration favors specificity at the expense of sensitivity. Another important factor is the behavior of DBSCAN itself. The results suggest that DBSCAN is modeling a very large region of normality, likely due to a relatively high epsilon value or insufficient cluster density constraints. This can lead to the absorption of borderline anomalies into normal clusters. However, despite this limitation, DBSCAN alone still achieves significantly better recall than the hybrid pipeline, indicating that the main performance bottleneck lies in the Isolation Forest stage rather than in the clustering step. A broader comparison with recent baseline methods such as One-Class Support Vector Machine, Local Outlier Factor, Random Forest, and recent deep learning-based IDS models is currently being conducted as part of an extended version of this work. This additional study will provide a more comprehensive positioning of the proposed approach against supervised and unsupervised IDS baselines.

From an operational perspective, the key challenge remains the trade-off between recall and false positives. While AE + DBSCAN provides strong detection capabilities, it may generate more false alarms. Several strategies can be implemented to address this issue without sacrificing recall:

- Threshold calibration: Instead of using fixed thresholds, adaptive thresholds based on quantiles or validation distributions can be used to dynamically adjust sensitivity.
- Temporal aggregation: Anomalies can be aggregated over time windows (e.g., per IP or session). Persistent anomalies are more likely to correspond to real attacks.
- Post-classification filtering: A lightweight supervised classifier can be applied to suspicious samples to filter false positives.
- Cluster validation: Clusters can be validated based on additional criteria such as stability, temporal coherence, or feature entropy.
- Explainability-based filtering: Feature importance or explainability methods (e.g., SHAP) can help identify whether an anomaly is meaningful or spurious.

V. Conclusion

This study investigated whether the ordering of DBSCAN and Isolation Forest affects intrusion detection performance on server log files. The results show that the ordering of unsupervised components has a substantial impact on detection quality. The AE+DBSCAN+IF pipeline achieved the highest accuracy (0.9803) but produced an extremely low recall (0.0082), making it unsuitable for practical IDS deployment. The AE+IF configuration improved sensitivity but remained less precise. The AE+DBSCAN pipeline achieved the best compromise, with an accuracy of 0.9670, a precision of 0.2606, and a recall of 0.4376. These findings answer the initial research question by showing that adding Isolation Forest after DBSCAN can degrade detection performance when thresholds are too restrictive. Future work will investigate adaptive thresholding, temporal aggregation, explainable anomaly filtering, and a broader comparison with recent IDS baselines.

Acknowledgment

We would like to thank the Laboratoire de Recherche en Science de l'Ingénieur (LARSI) at the University of Lomé, Togo, for providing us with the data used in this study.

References

[1] Alaa, A., et al.: A comprehensive survey on intrusion detection systems with advances in machine learning, deep learning and emerging cybersecurity challenges. Discover Artificial Intelligence (2025)

[2] Bundala, N.N.: Understanding cybercrime modus operandi: Techniques, psychological tricks, and countermeasures. Asian Journal of Research in Computer Science **17**(12), 234–251 (Dec 2024). https://doi.org/10.9734/ajrcos/2024/v17i12541, https://journalajrcos.com/index.php/AJRCOS/article/view/541

[3] Chen, D., Wawrzynski, P., Lv, Z.: Cyber security in smart cities: A review of deep learning-based applications and case studies. Sustainable Cities and Society **66**, 102655 (2021)

[4] Chen, Z., Liu, J., Gu, W., Su, Y., Lyu, M.R.: Experience report: Deep learning-based system log analysis for anomaly detection. arXiv preprint arXiv:2107.05908 (2021)

[5] Durgaraju, S., Vel, D.V.T., Madathala, H.: The evolution of cyber threats and defenses: A review of innovations and challenges. In: 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI). pp. 117–123 (2025). https://doi.org/10.1109/ICMCSI64620.2025.10883230

[6] Elsayed, R.A., Hamada, R.A., Abdalla, M.I., Elsaid, S.A.: Securing iot and sdn systems using deep-learning based automatic intrusion detection. Ain Shams Engineering Journal **14**(10), 102211 (2023)

[7] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. p. 226–231 (1996)

[8] Holm, H.: Signature based intrusion detection for zero-day attacks: (not) a closed chapter? In: 2014 47th Hawaii International Conference on System Sciences. pp. 4895–4904 (2014). https://doi.org/10.1109/HICSS.2014.600

[9] Kasongo, S.M.: A deep learning technique for intrusion detection system using a recurrent neural networks based framework. Computer Communications **199**, 113–125 (2023)

[10] Kazimierczak, M., Habib, N., Chan, J.H., Thanapattheerakul, T.: Impact of ai on the cyber kill chain: A systematic review. Heliyon **10**(24), e40699 (2024)

[11] Kumar, G.: Denial of service attacks–an updated perspective. Systems science & control engineering **4**(1), 285–294 (2016)

[12] Landauer, M., Onder, S., Skopik, F., Wurzenberger, M.: Deep learning for anomaly detection in log data: A survey. arXiv preprint arXiv:2207.03820 (2022)

[13] Liu, F.T., Ting, K., Zhou, Z.H.: Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery From Data - TKDD **6**, 1–39 (2012). https://doi.org/10.1145/2133360.2133363

[14] Maddikunta, P.K.R., Pham, Q.V., Prabadevi, B., Deepa, N., Dev, K., Gadekallu, T.R., et al.: Industry 5.0: A survey on enabling technologies and potential applications. Journal of Industrial Information Integration **26**, 100257 (2022)

[15] Sedki, I., Hamou-Lhadj, A., Ait-Mohamed, O.: Awsom-lp: An effective log parsing technique using pattern recognition and frequency analysis (2021), https://arxiv.org/abs/2110.15473

[16] Sedki, I., Hamou-Lhadj, A., Ait-Mohamed, O., Shehab, M.A.: An effective approach for parsing large log files. In: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 1–12 (2022). https://doi.org/10.1109/ICSME55016.2022.00009

[17] Shivhare, I., Purohit, J., Jogani, V., Attari, S., Chandane, M.: Intrusion detection: A deep learning approach. arXiv preprint arXiv:2306.07601 (2023)

[18] Xu, L.D., Xu, E.L., Li, L.: Industry 4.0: state of the art and future trends. International Journal of Production Research **56**(8), 2941–2962 (2018)

[19] Xu, Z., Wu, Y., Wang, S., Gao, J., Qiu, T., Wang, Z., Wan, H., Zhao, X.: Deep learning-based intrusion detection systems: A survey. arXiv preprint arXiv:2504.07839 (2025)

[20] Yeh, W.C., Forghani-elahabad, M.: An efficient algorithm for sorting and duplicate elimination by using logarithmic prime numbers. Big Data and Cognitive Computing **8**(9) (2024). https://doi.org/10.3390/bdcc8090096, https://www.mdpi.com/2504-2289/8/9/96

[21] Zou, T., Bretas, A.S., Aljohani, N., Bretas, N.G.: Malicious data injection attacks: A relaxed physics model based strategy for real-time monitoring. In: 2019 North American Power Symposium (NAPS). pp. 1–6 (2019). https://doi.org/10.1109/NAPS46351.2019.9000397