POLITECNICO DI TORINO

Master degree course in Cybersecurity

Master Degree Thesis

HTTP Request Clustering for Automated Threat Detection

Supervisor
prof. Andrea Atzeni
Andrea Menin
Simone Rotondo

Candidate
Lorenzo Ferretti

December 2025

Alla mia famiglia

Summary

This thesis explores whether HTTP sessions reconstructed from server-side logs can be grouped according to behavioral similarities using unsupervised learning. The motivation comes from the limitations of rule-based Web Application Firewalls, which focus on individual requests and struggle to detect threats that develop progressively within a session. By examining sequences of requests instead of isolated events, it becomes possible to detect anomalies, understand usage patterns, and distinguish automated actors from genuine users.

The work begins with an analysis of session identification techniques. Approaches based on IP addresses, User-Agent strings, browser fingerprints, favicon-based identifiers, and cookies are examined. This study adopts a simpler combination of IP addresses and User-Agent strings since the main objective is clustering rather than persistent user tracking.

A central contribution of the thesis is the construction of a feature set designed to describe session behavior. Existing research provides numerical indicators such as request frequency, error rates, distribution of HTTP methods or ratios of HTML to static resources. These classical features are expanded with additional signals derived from traffic inspection, including Sec-Fetch metadata, temporal dynamics, and response content categories. Such features help differentiate normal navigation from scripted actions or stealth scanning activities.

Because clustering techniques are meaningful only when the underlying data exhibit multimodal structure, the thesis evaluates clusterability through the Dip Test. Both dimensionality-reduction and distance-based formulations are examined. The analysis highlights situations where data do not clearly support partitioning, at which preprocessing becomes essential.

Deep clustering is discussed briefly, but the focus remains on Self-Organizing Maps and K-Means, which are appropriate given the moderate dimensionality of the dataset. For SOMs, training parameters, weight interpretation, and methods for evaluating the quality of the map are examined. For K-Means, attention is given to initialization, convexity assumptions, and the need to determine the number of clusters. Internal validation metrics such as the Davies-Bouldin coefficient, the Silhouette score, the Dunn-OWA index, and the WB index are compared, noting that numerical measures do not always reflect expert intuition.

The dataset is obtained from Elasticsearch, and sessions are assembled by grouping requests originating from the same IP address and User-Agent when their timestamps fall within a 60-minute window. Sessions that contain too few interactions or requests blocked due to ASN reputation are discarded. Features are first expressed as ratios and then normalized to avoid dominance by features with larger ranges.

At the current stage, the thesis covers both the exploratory analysis and the unsupervised learning phase. The labeled dataset produced from cluster inspection provides a semantic grounding of the session groups and enables a meaningful interpretation of behavioral patterns. Future work will aim to strengthen the session identification process and to develop supervised classifiers trained on the labeled sessions, with the objective of improving robustness and detection performance and potentially deploying the trained models in a production environment.

Contents

Chapter 1

Introduction

HTTP request clustering refers to the process of grouping user sessions, defined as sequences of requests performed by a user within a specific time window, into clusters based on behavioral similarities. Since clustering is an unsupervised machine learning technique, it can reveal hidden patterns in user behavior. As a result, it becomes possible to gain insights into common usage scenarios, such as identifying navigation paths, spotting unusual behaviors, or detecting potential anomalies in user interactions.

Sicuranext, the company in which this thesis is carried out, currently employs a Web Application Firewall (WAF) whose detection logic relies on predefined rule matching. This mechanism examines requests and responses individually and, as a result, does not take into account the broader session context. Such a limitation encourages the adoption of session-level traffic analysis and unsupervised learning techniques, which can reveal behavioral patterns that extend beyond manually crafted rules.

The objective of this work is to evaluate whether web sessions can be grouped and automatically labeled in order to characterize their intent or functional category, with particular attention to security-oriented classes such as attacks and anomalous behaviors. The analysis is conducted using only information extracted from server-side HTTP logs, for example request headers and aggregated session features. Although the primary focus is the identification of malicious or abnormal activities, the derived labels may also represent legitimate user behavior.

In addition, Sicuranext's WAAP (Web Application & Api Protection) compensates for the absence of native session-tracking capabilities by maintaining stateful information for authenticated users. This enables the application of functionalities such as rate limiting and session-level bans, implemented through proprietary mechanisms that extend and enhance the capabilities of a standard WAF.

After this step is completed and a label is assigned, supervised learning models or neural networks can be used to train a system able to predict session types in real time. This makes the identification of session intent faster and more accurate, which is especially important for detecting and blocking potentially malicious sessions. The proposed methodology also reduces the time needed to manually analyze and classify sessions.

The supervised learning part of this approach could be useful for Sicuranext, as it allows the automation of session classification and improves overall system security. However, before reaching this phase, it is essential to ensure that the labels are both available and correct; otherwise, the model may learn wrong patterns and produce unreliable predictions. It should be noted that the development and implementation of the supervised learning model fall outside the scope of this thesis, which focuses mainly on data preparation and labeling.

In this article [1], the authors analyzed supervised and unsupervised machine learning algorithms applied to intrusion detection systems, outlining their advantages and drawbacks. Although the domain is different from the topic of this thesis, supervised learning has shown promising results. Their analysis also helps identify when each algorithm is most suitable. For example, the Decision Tree algorithm is both fast and interpretable, which is important in cybersecurity: threats must be blocked quickly, and it is necessary to understand why a specific alert is raised.

The Decision Tree algorithm is also suitable for detecting attacks with a WAF [2]. Although this is not the main focus of this thesis, WAF systems must operate very quickly to detect attacks in real time, and the authors reported good results in this regard. They chose traditional machine learning algorithms instead of deep learning models to improve computational cost. Deep learning automatically extracts features from high-dimensional traffic data, while traditional machine learning requires manual feature selection. The authors also noted that they tested only a small subset of the OWASP Top 10 vulnerabilities and that machine learning algorithms can be exposed to new attack types, such as adversarial inputs designed to evade detection.

In this work [3], the authors describe adversarial attacks and present techniques designed to defend against them. This aspect represents a possible improvement for this work; when the supervised learning model is trained, these attacks should be considered.

Deploying a machine learning model in a production environment is more complex than it may appear. In this paper [4], the authors highlight practical challenges and review the strengths and limitations of two popular backend frameworks, Node.js and Python. Moreover, deploying a machine learning algorithm can be expensive in terms of infrastructure, which is an important factor to evaluate before moving forward.

However, this thesis may also serve as a retrospective analysis tool, removing the need to deploy a real-time detection system. Instead, it can be used to evaluate past requests or analyze network traffic to define new static rules for blocking emerging malicious activity. There is also the issue of model retraining, since changes in the volume or distribution of traffic may lead to incorrect session classification.

The implementation is developed in Python, selected for its wide ecosystem of data analysis and machine learning libraries, which supports rapid experimentation with clustering algorithms. The dataset is built by sending queries to Elasticsearch to retrieve aggregated data, where aggregation is handled directly by Elasticsearch. Afterward, an exploratory data analysis is performed to study feature correlations. Different configurations are tested, including one using all features and another excluding highly correlated ones.

To assess whether the dataset is effectively clusterable, a statistical unimodality test known as the Dip Test is applied. However, since this method operates on one-dimensional data, two alternative strategies are considered: (i) reducing dimensionality with Principal Component Analysis (PCA) and performing the test on the first principal component, or (ii) computing pairwise distances and applying the test to these values.

If the data prove to be clusterable, the standard K-Means algorithm is applied. A dedicated section also discusses how internal clustering metrics may diverge from expert intuition, showing the limitations of purely quantitative evaluation methods.

Then, to assign labels to the clusters, an LLM is used, mainly because manually assigning labels is very time-consuming. Large language models have recently shown an ability to capture semantic context; therefore, the suitability of an LLM for this task is examined.

The thesis is organized as follows: Chapter 2 reviews the relevant literature on session identification, feature extraction, exploratory feature analysis, the selection of suitable clustering methods, internal evaluation metrics, and labeling approaches. Chapter 3 presents the core components of Sicuranext's infrastructure and describes the system requirements and semantics. Chapter 4 details the practical implementation of the approach, including the creation of the dataset, the analysis of feature correlations, the application of the Dip Test, the clustering experiments, and the labeling process. Finally, Chapter 5 discusses the results, followed by closing remarks and directions for future work.

Chapter 2

Related Work

To achieve the objectives of this study, the following challenges are addressed:

- Session Identification
- Feature Identification
- Exploratory Feature Analysis
- Selection of Appropriate Clustering Algorithms
- Labeling

These steps are essential for structuring the analysis. For this reason, the current state of the art related to each challenge is reviewed.

2.1 Session Identification

The purpose is to aggregate requests issued by the same user in order to enable further analysis on the resulting sessions. In [5], sessions are reconstructed by grouping consecutive requests originating from the same IP address within a fixed time window of ten minutes. Similarly, [6] adopts a time-based approach but relies on the combination of IP address and User-Agent to identify a user.

Using only the IP address raises the well-known NAT issue: when several devices share the same public address, their requests may be incorrectly merged into a single session. Although combining IP address and User-Agent reduces this risk, it does not eliminate it entirely.

Both approaches can also be easily bypassed. In the first case, rotating the IP address forces the system to create a new session for each request. In the second case, frequently modifying the User-Agent string leads the system to interpret a continuous browsing activity as several independent sessions.

Strengths: Straightforward to deploy, as both the IP address and the User-Agent string are available on the server-side.

Weaknesses: Susceptibility to IP rotation and/or User-Agent rotation.

These issues motivate the investigation of appropriate mitigation techniques. One such approach is browser fingerprinting, which generates a device-specific identifier and assigns incoming data to this identifier rather than relying solely on the IP address or the User-Agent header. This identifier is derived from high-entropy attributes of the client environment that tend to remain stable over time, enabling reliable re-identification of the same device across multiple sessions and, in some cases, across different websites. The survey in [7] offers a comprehensive overview of browser fingerprinting. It outlines the main fingerprinting techniques, examines the associated privacy implications, and reviews existing work on fingerprint randomization. In particular, the authors of [8] and [9] describe how several JavaScript APIs can be leveraged to construct such fingerprints, for example, the `canvas` API, which exposes differences in font rendering; the canvas font list, which reveals installed fonts; `WebRTC`, which can disclose peer information; and the `AudioContext` API, which uncovers hardware-dependent variations in audio processing. Interestingly, even subtle rendering differences, such as those observed in system-provided emojis, can contribute to a browser's uniqueness. These variations may also disclose sensitive information, including the presence of specific software or the absence of certain security patches.

Beyond improving session identification, browser fingerprinting can also assist in detecting automated traffic. Many bots attempt to evade detection by deliberately altering parts of their fingerprint, which often creates inconsistencies among collected attributes [10]. For example, a request may include a User-Agent string identifying the device as an iPhone while reporting the absence of touch capabilities, a combination that is impossible on genuine hardware. Likewise, a device claiming to be mobile may report a screen resolution typical of a desktop system, contradicting the declared platform. Detecting such inconsistencies not only helps distinguish human traffic from evasive bots but also strengthens the reliability of session association.

However, some browsers, such as Brave, already include built-in mechanisms that reduce fingerprinting. In addition, adopting this approach requires modifications to the client source code and the collection of a broad set of client information for fingerprint generation, a process that is technically complex and raises privacy concerns.

Strengths: Provides robust session identification without relying on the IP address or the User-Agent, both of which are relatively easy to bypass.

Weaknesses: Requires substantial development effort and careful attention to detail during implementation.

An alternative method is presented in [11], where the authors track users by exploiting favicons to create persistent identifiers. The favicon cache (F cache) is a dedicated storage area that operates independently of the browser's main cache and is therefore not affected by clearing browsing data such as cache, history, or cookies. The identifier assigned to a user is constructed by redirecting the browser through a sequence of subdomains, each delivering a distinct favicon, which allows information to be encoded in the F cache. An N-bit identifier makes it possible to distinguish among 2^N users and requires N redirections. At the time of publication, Chrome, Safari, Edge, and Brave were all susceptible to this technique. In Firefox, however, the authors reported that the browser did not load favicons from the cache, but instead requested them again, which prevented tracking. This method is capable of identifying users even when they browse through a virtual private network or in private mode.

Although the mechanism is appealing from a theoretical perspective, it is not practical in real deployments because the required sequence of redirections introduces a substantial performance cost.

Strengths: Enables persistent user identification by exploiting a cache mechanism that is unaffected by clearing browsing data and remains effective across private browsing sessions and the use of a virtual private network.

Weaknesses: Relies on multiple sequential redirections, which introduce noticeable performance overhead and limit its practicality in real-world deployments.

A different approach is proposed in [12], where requests are grouped into sessions by relying on the Google Analytics Client ID. When a user visits a website that has Google Analytics enabled and is logged in to their Google account, the platform sets a cookie (typically named `_ga`) that stores this identifier. This value can then be used to associate multiple requests with a single session. Variations in the user's IP address or User-Agent string do not interfere with this process, because the Client ID remains stable across such changes.

Strengths: Provides a stable session identifier that remains consistent across changes in IP address and User-Agent, allowing reliable association of requests without requiring server-side state management.

Weaknesses: Depends on the presence of Google Analytics and the acceptance of the corresponding cookie, which limits applicability in contexts where tracking protection, cookie restrictions, or user consent policies prevent the Client ID from being set or retained.

A methodology closely related to the previous technique is the use of a dedicated cookie. In this approach, the server assigns a unique identifier to the client and stores it in a cookie that is sent with subsequent requests. As long as the cookie remains stored in the browser, the identifier offers a straightforward way to associate multiple interactions with the same user across different requests.

Strengths: Simple to implement and effective for tracking, since the identifier persists across requests and does not rely on network characteristics.

Weaknesses: Loses effectiveness when users delete their cookies, which prevents the identifier from being retained.

In this work, sessions are identified using the combination of IP address and User-Agent. Although this method is known to have limitations, developing a more robust identification mechanism is not the primary focus of the thesis. Nevertheless, it indicates a clear direction for future improvements, since inaccurate session identification would compromise all subsequent analyses. However, it is worth noting that if sessions are similar, they will be grouped into the same cluster; in this sense, the clustering process itself acts as a sort of defense mechanism, mitigating the impact of potential errors in session identification.

2.2 Feature Identification

It is worth remarking that only server-side features are considered, which contribute to simplifying the analysis. The study below is among those applying unsupervised learning to web session characterization. In [6], the authors employ two unsupervised neural network models, namely the Self Organizing Map (SOM) and the Modified Adaptive Resonance Theory 2 (Modified ART2). They select nine features, which are reported below:

1. Click rate, a numerical attribute defined as the number of HTTP requests within a single session.
2. HTML to Image ratio, a numerical attribute defined as the number of HTML page requests divided by the number of image file requests (JPEG and PNG).
3. Percentage of PDF or PS file requests, a numerical attribute defined as the proportion of PDF or PS file requests within a session.
4. Percentage of 4xx error responses, a numerical attribute defined as the proportion of erroneous HTTP requests in a session.
5. Percentage of HTTP requests of type HEAD, a numerical attribute defined as the proportion of HEAD requests in a session.
6. Percentage of requests with unassigned referrers, a numerical attribute defined as the proportion of requests with blank or missing referrer fields within a session.
7. *Robots.txt* file request, a nominal attribute equal to 1 if the *robots.txt* file is requested during a session and 0 otherwise.
8. Standard deviation of requested page depth, a numerical attribute defined as the standard deviation of the depth of the requested pages within a session.

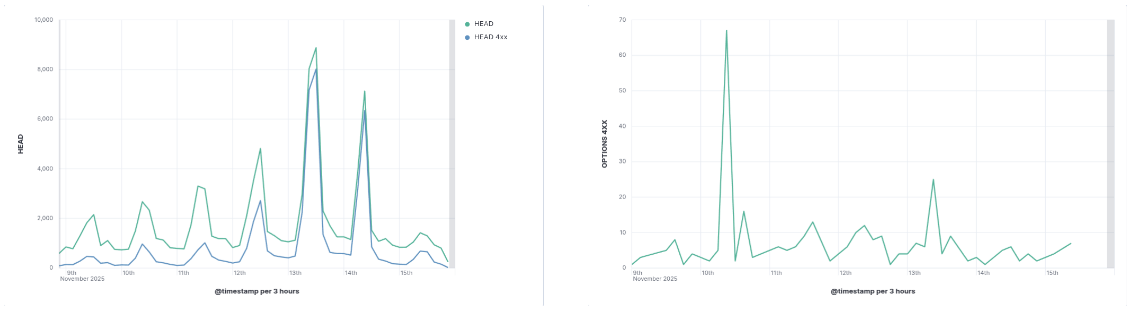
9. Percentage of consecutive sequential HTTP requests, a numerical attribute defined as the proportion of sequential requests for pages belonging to the same directory during a session.

In a more recent study [5], the authors detected malicious crawlers as well as attacks such as brute force password cracking. They selected ten features for anomaly detection in a session. The features include:

1. Proportion of abnormal User-Agents, such as spider bots or crawlers
2. Proportion of requests using methods other than GET or POST, such as HEAD, PUT, CONNECT, OPTIONS or PROPFIND
3. Proportion of POST requests
4. Access amount, defined as the total number of requests after excluding those for static pages
5. Access frequency, measured as the number of requests per minute
6. Number of accesses to sensitive files
7. Browser automatic access request ratio, which represents the percentage of visits within a session generated automatically by the browser, for example requests for static resources such as js, css, png or jpeg files. This feature is similar to the HTML to Image ratio used in previous work [6], although it is more general.

The remaining features were the same as those used in the previous study [6]. The authors applied the DBSCAN algorithm, using Euclidean distance as the similarity metric, to identify anomalous web sessions. DBSCAN requires two parameters, MinPts and eps. They set MinPts to 11, corresponding to the ten features plus one, while the value of eps was not specified in their paper.

To assess whether the proposed features provide meaningful information within the context of this thesis, Kibana visualizations can be used to examine their relevance. This approach makes it possible to determine which features actually contribute useful insights to the analysis conducted in this work. The available logs do not include a notion of sessions, which makes it impossible to analyze the features at the session level. The construction of the sessions is therefore performed offline, within the Python scripts that preprocess the data. Nevertheless, examining individual requests can still offer insights into whether the features identified in the previous study remain useful.

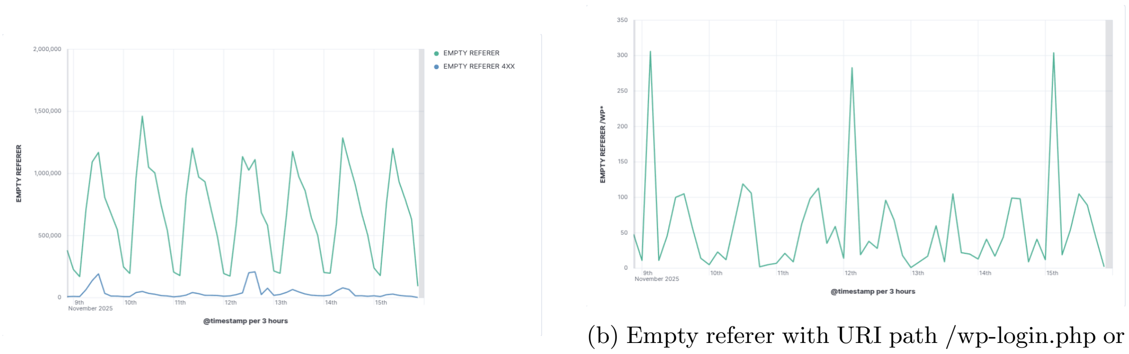


(a) HEAD vs HEAD with response status code 4xx (b) OPTIONS with response status code 4xx

Figure 2.1: Comparison of 4xx responses for HEAD and OPTIONS requests

On the thirteenth and fourteenth of November, nearly all HEAD requests resulted in response codes of the 4xx class, most frequently 403 Forbidden and 404 Not Found. During this period, the volume of HEAD requests was noticeably higher than on the other days of the week. This concentration of erroneous HEAD requests suggests that their occurrence may serve as an indicator of automated threat activity, since attackers often employ HEAD requests to remain covert, retrieving only response headers rather than full content.

A similar pattern is observed in the number of OPTIONS requests on the tenth of November. This indicates that not only HEAD requests but also OPTIONS requests may provide useful information. For this reason, this work focuses more generally on non-GET and non-POST requests, as they can contribute meaningful insights when characterizing potentially automated behaviors.



(a) Empty referer with response status code 4xx

(b) Empty referer with URI path /wp-login.php or /wp/wp-admin/*

Figure 2.2: Comparison of empty referer request patterns

In general, it is possible for requests to appear without an assigned referrer. What is unusual in the Figure 2.2, however, is that at a regular interval of three days (9, 12, and 15 November at 9:00), there is a noticeable increase in requests with no referrer that target either the endpoint `/wp-login.php` or the paths `/wp/wp-admin/*`. When such a periodic pattern emerges, it is typically associated with some form of automated activity. For this reason, the total number of requests with an unassigned referrer may itself represent a useful feature.

2.2.1 Novel Features Introduced

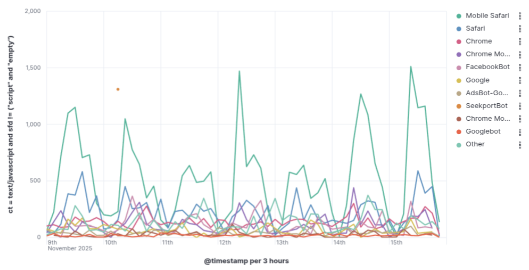
A feature that is absent from the existing research is the type of content contained in each response. This idea is related to what has already been explored, for example features that compute the ratio of HTML to images or that count the number of accesses while excluding requests for static resources. Because many content types exist, this work focuses only on images, fonts, application, and text. In addition, only the main category is taken into account; for instance, responses of type `image/gif` and `image/jpeg` are not distinguished but are grouped under the image category. These features can help characterize user sessions, as several hypotheses can be formulated. For example, a session that contains no image or font requests may be considered suspicious. Although some of these resources might be retrieved from the cache, the absence of such requests can still raise suspicion.

Another feature that is rarely discussed in the literature is the use of the Sec-Fetch headers, which provide additional information on the context of each request. While most studies rely on traditional request headers or server-side information, and some make use of client-side signals to infer navigation behavior as shown for example in this master's thesis [12], the Sec-Fetch headers remain underexplored partly because they are relatively new and not yet uniformly supported across browsers. However, the `Sec-Fetch-*` headers can be leveraged as heuristics to infer whether requests stem from direct user interactions or are automatically generated by scripts.

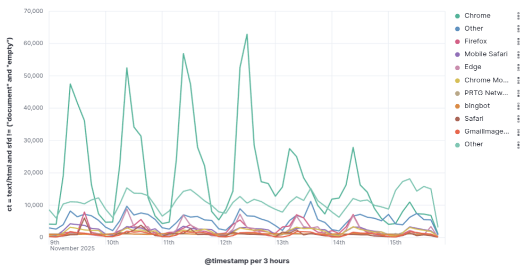
For instance, the `Sec-Fetch-User` header typically appears in requests triggered by explicit user actions such as clicks. The `Sec-Fetch-Site` header, which indicates the relationship between the initiator's origin and the target origin, is often set to `none` when a request originates directly from a user (e.g., typing a URL or selecting a bookmark) rather than from a cross-site or same-site script. Similarly, the `Sec-Fetch-Dest` header specifies the intended destination of a request; when its value is `document`, it commonly signals a top-level navigation event initiated by the user.

It is important to note that this approach is intrinsically heuristic: because these are standard HTTP headers, they can be modified by an attacker. However, when incorporated into a broader detection strategy, they provide valuable indicators for distinguishing legitimate user traffic from automated or scripted requests.

(Information about the `Sec-Fetch-*` headers can be found in the official MDN Web Docs: MDN Fetch Metadata Request Headers)



(a) Responses with content type `text/javascript` where the request `sec-fetch-dest` is not `script` or `empty`.



(b) Responses with content type `text/html` where the request `sec-fetch-dest` is not `document` or `empty`.

Figure 2.3: Comparison of anomalous request patterns based on content type and `sec-fetch-dest` mismatches.

When a browser retrieves a JavaScript file, and provided that the request headers have not been altered, it usually sends the header `Sec-Fetch-Dest: script`. The server response in such cases commonly specifies the content type `text/javascript`. When a response is delivered with the content type set to `text/javascript` but the `Sec-Fetch-Dest` header is absent or indicates a value other than `script/empty`, this can indicate that the request did not originate from a standard browser. Situations of this kind may reveal automated tools or scripted clients and may also correspond to specific device behavior such as certain iOS clients. A detailed analysis shows that the Safari versions reported by these requests, together with the associated iOS versions, are relatively old. This point is illustrated in the figure below.

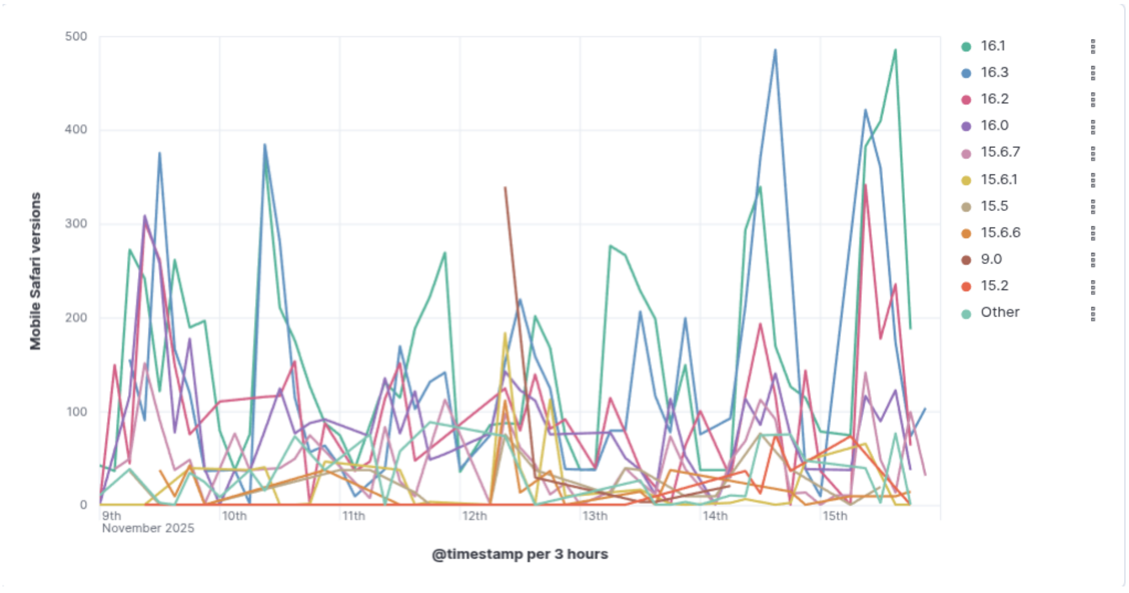


Figure 2.4: Mobile Safari versions

This behavior can be considered legitimate, because the `Sec-Fetch` header is added only starting from version 16.4.

A similar observation applies to the case in which the header `Sec-Fetch-Dest: document` would be expected, as shown in Figure 2.3b. When filtering requests by User-Agent version, it appears that recent versions of Chrome do not send the `Sec-Fetch-Dest` header, however, it also appears that very old versions of Chrome are present. The spike in traffic shown below is very likely produced by an automated process.

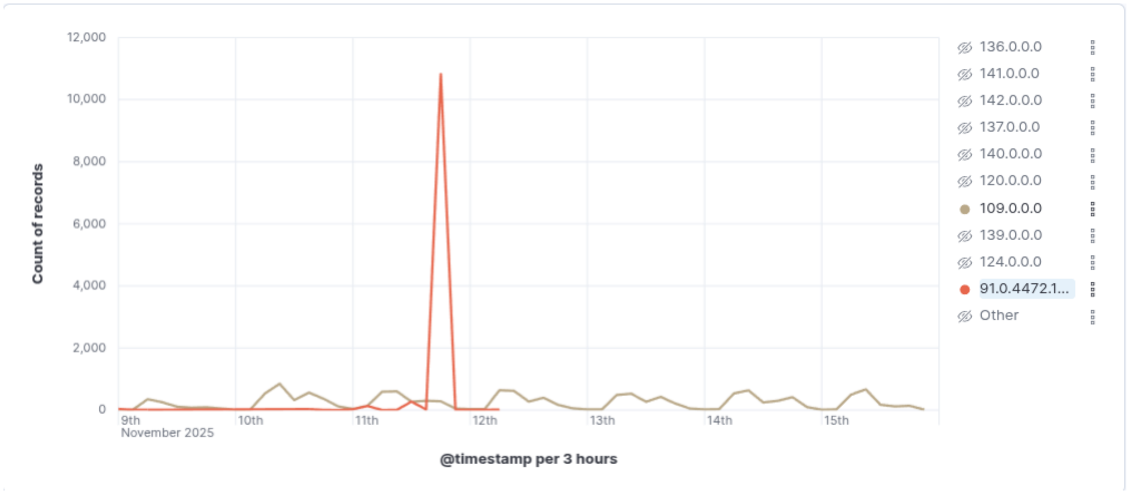


Figure 2.5: Distribution of Chrome versions

The same reasoning applies to the case in which `Sec-Fetch-Dest: image` is present and `Sec-Fetch-User: ?1` is present. Typically, when an image is requested, the `Sec-Fetch-User` header is not set. Nonetheless, as shown in Figure 2.6, there are cases in which it is set. Most of these requests correspond to Chrome version `125.0.0.0`, which is suspicious, because it is a version from the previous year, although it might still be legitimate. Further investigation would be required to determine its nature.

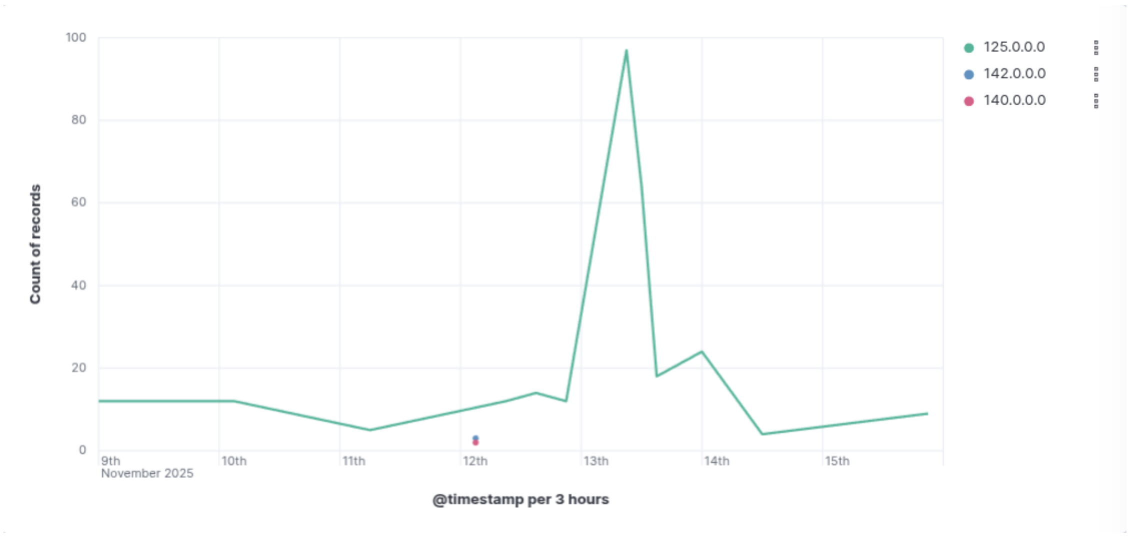


Figure 2.6: `Sec-Fetch-Dest: image` and `Sec-Fetch-User: ?1`

A comparable consideration applies to requests with `Sec-Fetch-Mode: navigate` and `Sec-Fetch-Dest: script`. This combination is suspicious, because navigation requests are normally associated with a destination of `document`, which is usually triggered by direct human interaction. As already mentioned, many of these requests originate from outdated Chrome versions, which suggests the same explanation as in the previous scenarios.

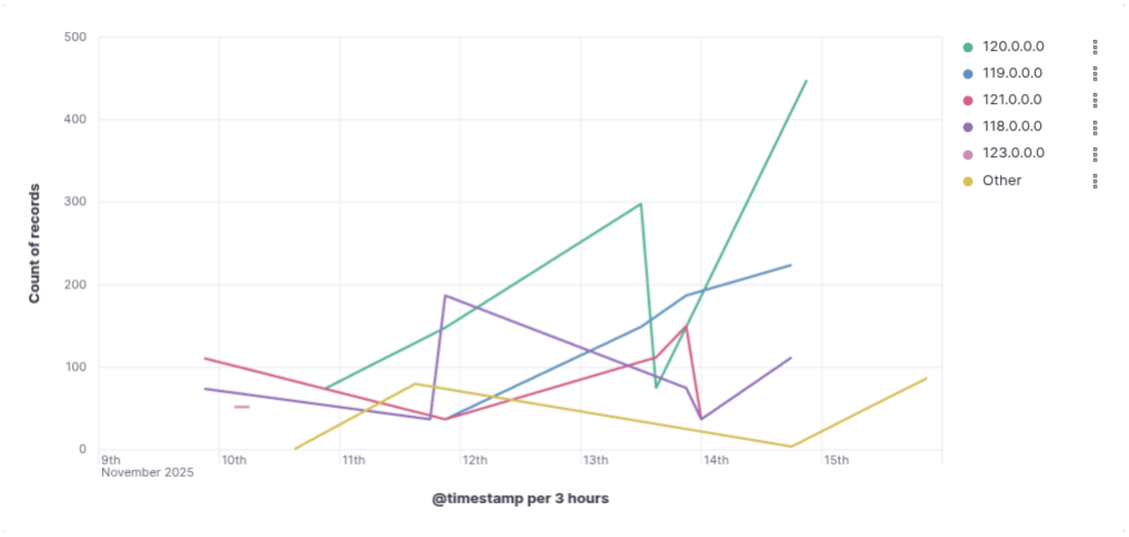


Figure 2.7: `Sec-Fetch-Mode: navigate` and `Sec-Fetch-Dest: script`

Moreover, a third feature requires consideration, namely the average response content length. One hypothesis regarding its usefulness is that, during periods in which attackers extract substantial amounts of data from a website, the size of the returned content increases significantly. In such intervals, the average response content length becomes unusually high, indicating that the retrieved data may be suspicious.

These new features provide additional signals that go beyond the classical features used in previous studies, and they are presented in Section 4.

2.3 Exploratory feature analysis

A survey covering two decades of research in feature analysis is presented in [13]. Feature selection is a fundamental task in machine learning, as choosing appropriate features reduces dimensionality by removing irrelevant or redundant variables. It also improves model performance and enhances interpretability. However, no single technique is universally optimal. The choice depends on several factors such as dataset size, the presence of missing values, the need for interpretability, and whether the problem is supervised or unsupervised.

In this work, only a limited number of features is used. These features are selected because they provide meaningful information for the task. In the unsupervised setting, several techniques discussed in the aforementioned survey are applied, including correlation analysis, variance analysis, and locality preservation. Another relevant contribution is [14], in which the authors provide a Python library implementing a wide range of feature analysis methods.

This topic will be revisited when the supervised model is developed. At that stage, stronger techniques, as discussed in the cited works, will be employed to identify the most influential features in the model. In the feature analysis conducted in this work, correlation is examined, but it is worth noting that in this type of problem a correlation value around 0.9 does not necessarily justify eliminating a feature, as it may still contribute useful information for detecting suspicious sessions. The article [13] also provides an overview of high-dimensional scenarios, in which the number of features is much greater than the number of samples. In the present work, the dataset contains 87942 samples and 31 features. Therefore, the problem is not considered high dimensional. Consequently, the computational complexity is lower, as no dimensionality reduction preprocessing step is required.

2.4 Clustering Algorithms

The current state of the art in clustering is known as deep clustering. Deep clustering combines deep learning with clustering, and as discussed in [15, 16], these approaches are well suited for handling highly complex data. They are applied in several fields, for example anomaly detection, image analysis, and other domains where non linear relationships are present. These surveys provide an overview of the main techniques.

One interesting aspect is the use of autoencoders for dimensionality reduction, in which the input data are mapped to a latent space and this representation is then used for clustering. Autoencoders are capable of learning non linear structures, whereas classical dimensionality reduction methods, for instance PCA, often struggle. Another introduction to autoencoders and their use in clustering, together with comparisons between clustering on the original data and on the latent representation, is provided in [17].

As noted in Section 2.3, the problem considered in this work is not high dimensional, therefore the use of autoencoders or deep clustering is not necessary. Instead, shallow clustering techniques are adopted. A recent review of these algorithms is presented in [18], which describes different types of methods, including partition based, hierarchical, density based, grid based, and model based approaches. The authors examine the advantages and limitations of each method, taking into account factors such as the number of features and the size of the dataset. In this thesis, K-Means is applied with full awareness of its limitations:

- Sensitive to outliers
- Prone to convergence to local optima
- Requires a predefined number of clusters
- Not well suited for non-convex data

Nevertheless, as noted in the literature, K-Means performs well in many applications, with the exception of domains such as image analysis and bioinformatics, where high-dimensional or complex data structures make alternative clustering approaches more appropriate. Since this is not the case in the present study, K-Means remains a relevant choice. They also discuss internal and external indices for assessing clustering quality. Section 2.4.3 examines these validation metrics in more detail and discusses their limitations.

2.4.1 Dip Test for Multimodality

Clustering is only appropriate when a cluster structure is present in the data, as shown in [19]. Such analysis is independent of any clustering method and should therefore precede the application of clustering algorithms. If the data do not exhibit an inherent cluster structure that can be meaningfully partitioned, clustering may not be suitable for the given data, or the data may need to be reprocessed.

To identify an appropriate clusterability measure, several key properties are considered:

- **Efficiency:** the measure should be computable in polynomial time
- **Algorithm Independence:** the measure should not depend on any specific clustering algorithm
- **Effectiveness:** the measure should accurately identify data as either clusterable or unclusterable

The procedure consists of two main stages: first, the dimensionality of the data is reduced, and second, a statistical test is applied to assess multimodality. The low-dimensional embedding helps determine how readily the dataset can be partitioned into meaningful clusters. When the data are generated from a single bivariate normal distribution, the observations form one cluster, and both the distribution of pairwise distances and the distribution of the first principal component exhibit a single mode. In contrast, if the data come from a mixture of clusters that are well separated, these distributions display multiple modes, reflecting the distinction between within-cluster variation and between-cluster separation.

A common approach for dimensionality reduction is PCA, which projects the data onto orthogonal dimensions that explain most of the original variance. PCA is relatively robust but not well suited for non-linear structures, for which principal curves may be more appropriate. The pairwise-distance approach yields a one-dimensional array but increases the sample size approximately by the square of the original size; thus, for large datasets, it becomes computationally demanding.

Following dimensionality reduction, multimodality tests are applied. These tests generally assume that the data are generated from a unimodal distribution, which serves as the null hypothesis. The p-value indicates how compatible the observed sample is with this hypothesis, in the sense that larger p-values reflect little evidence against unimodality, whereas smaller p-values provide evidence that the data deviate from a single underlying mode. When the p-value is large, the data are likely not separable into distinct clusters. In contrast, a small p-value suggests the presence of multiple modes, hence supporting the existence of several clusters in the population. It is important to note that statistical multimodality tests are designed for data in one dimension, because their asymptotic properties in higher dimensions remain insufficiently understood.

In the study by Adolfsson *et al.*, several multimodality tests are evaluated on both simulated and empirical datasets. Among these methods, the *Dist-Dip* test shows the best performance, and it is the only procedure that recognizes chaining structures in the data, such as single lines, parallel lines, or concentric circles. It is important to note that the previous works [6, 5] do not employ this statistical test to verify whether the data are effectively clusterable, and this represents a crucial step before proceeding with the analysis.

The Hartigan Dip Test [20] evaluates whether a sample originates from a unimodal distribution. The method determines the largest vertical deviation between the empirical cumulative distribution function (ECDF) and the unimodal distribution function that minimizes this deviation. Formally, the dip statistic is defined as

$$\mathrm{DIP} = \max_x |F_n(x) - U(x)|,$$

where $F_n(x)$ denotes the empirical distribution function and $U(x)$ is the unimodal cumulative distribution function that yields the smallest maximum difference.

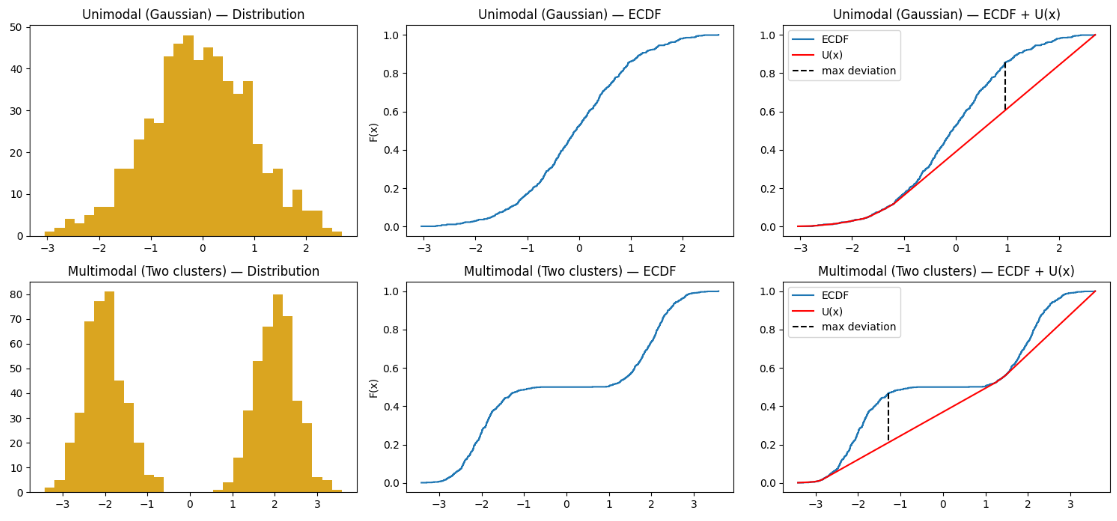


Figure 2.8: Visualization of the Dip Test procedure

As illustrated in Figure 2.8, the ECDF exhibits a flat region, which indicates a departure from unimodality. The distribution $U(x)$ shown in the figure represents only an initial candidate and not the final unimodal approximation. The algorithm iteratively searches for the unimodal cumulative distribution function that is closest to the ECDF. To achieve this, it constructs the Least Concave Majorant (LCM) and the Greatest Convex Minorant (GCM) of the empirical distribution and combines them. The dip statistic corresponds to the maximum vertical distance between the empirical cumulative distribution function and the unimodal cumulative distribution function that minimizes this distance.

To visualize the outcomes of the Dip Test, several datasets are examined, and some of them are found not to be clusterable. The essential point is that when the data are not clusterable, a preprocessing step becomes necessary. For instance, in the Gaussian quantiles dataset, the data are not clusterable, indicating that some form of preprocessing is necessary.

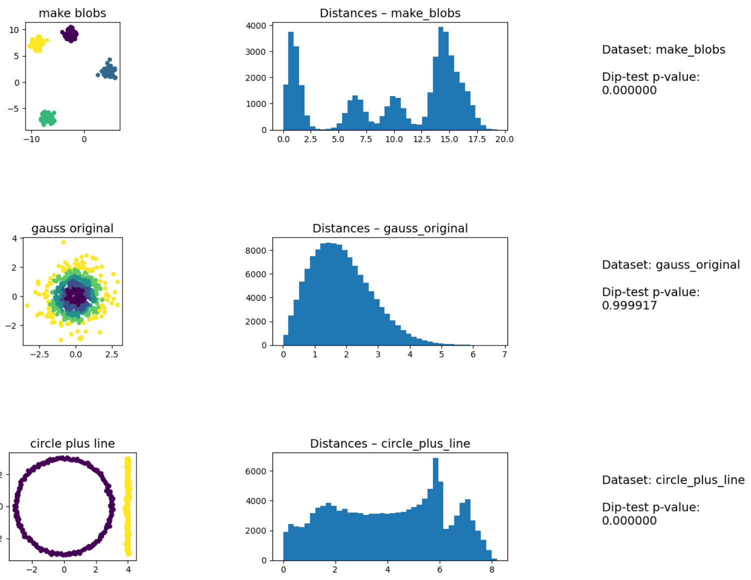


Figure 2.9: Dip Test analysis

However, there are cases in which a dataset is clusterable, although the p-value is greater than 0.005. For example, in the case of the *make_moons* dataset, the Dip Test yields a p-value above this threshold even though the structure is clearly clusterable. As reported in the paper, the distance-based method identifies clusterable data with a success rate of about 95% for both simulated and real datasets. What is particularly important, and explicitly noted by the authors, is that applying the Dip Test to the distance distribution results in a Type I error rate of roughly 1%. This indicates that it is unlikely for the test to classify a dataset as clusterable when it is not. Therefore, even when the Dip Test returns a p-value greater than 0.005, the dataset may still be clusterable, whereas a p-value close to zero provides stronger evidence in favour of clusterability.

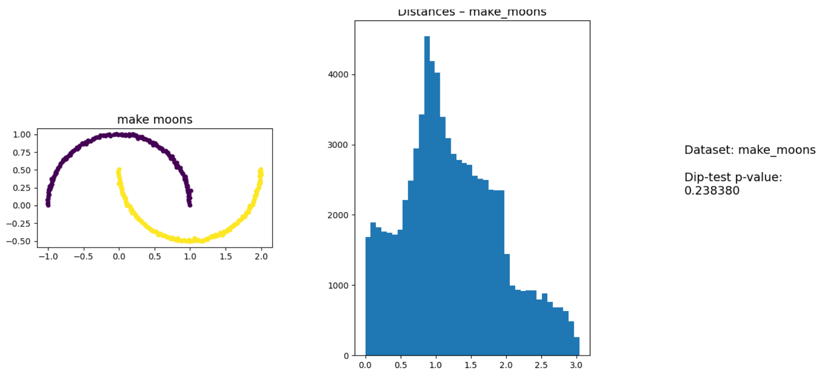


Figure 2.10: Dip Test analysis on the make_moons dataset

2.4.2 Self-Organizing Map

The Self-Organizing Map (SOM), originally proposed by Kohonen [21], is an unsupervised neural algorithm that performs dimensionality reduction and topology-preserving vector quantization. The model consists of a finite set of neurons arranged on a two-dimensional grid (rectangular or square), although higher-dimensional grids are possible.

Let the SOM contain K neurons. Each neuron $i \in \{1, \dots, K\}$ is associated with a prototype (codebook) vector

$$\mathbf{m}_i(t) \in \mathbb{R}^n,$$

where n is the dimensionality of the input space and t denotes the training iteration. The set of all prototypes is denoted

$$\mathcal{M}(t) = \{\mathbf{m}_1(t), \dots, \mathbf{m}_K(t)\}.$$

Let the dataset consist of M vectors

$$\mathbf{x}_s \in \mathbb{R}^n, \qquad s = 1, \dots, M.$$

Each neuron has a fixed position vector $\mathbf{r}_i \in \mathbb{R}^2$ on the map grid. These locations are used to define neighborhood relationships, not to be confused with $\mathbf{m}_i$, which exist in input space.

Best Matching Unit (BMU)

Given an input vector $\mathbf{x}_s$, its best matching prototype (BMU) is the neuron

$$c(s) = \arg\min_i \|\mathbf{x}_s - \mathbf{m}_i(t)\|.$$

Online (Sequential) Training

At each iteration t, one sample is presented and the prototypes are updated. A neighborhood kernel $h_{ci}(t)$ modulates how strongly neuron i reacts to input belonging to BMU c:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \alpha(t)\, h_{c(s)i}(t) \left[\mathbf{x}_s - \mathbf{m}_i(t)\right],$$

where: - $\alpha(t) \in (0,1)$ is the learning rate (monotonically decreasing), - $h_{c(s)i}(t) \in (0,1]$ is the neighborhood function. A common choice for the neighborhood function is a Gaussian kernel:

$$h_{ci}(t) = \exp\left(-\frac{\|\mathbf{r}_c - \mathbf{r}_i\|^2}{2\sigma^2(t)}\right),$$

where $\sigma(t)$ is the neighborhood radius. Typically,

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau}\right),$$

with initial radius σ_0 and time constant τ.

Kohonen suggests that the total number of online training steps should be at least several hundred times the number of neurons, e.g., $500K$. A typical learning rate schedule is linear or exponential decay, for example

$$\alpha(t) = 0.9\left(1 - \frac{t}{1000}\right).$$

Batch Training

The batch SOM update [22] uses the entire dataset at once. After determining the BMU for every sample, the prototype vectors are updated as weighted averages:

$$\mathbf{m}_i^* = \frac{\sum_{s=1}^{M} h_{c(s),i}\,\mathbf{x}_s}{\sum_{s=1}^{M} h_{c(s),i}}.$$

This formulation does not require a learning rate and usually converges faster and more stably than sequential training.

An equivalent and computationally efficient form groups samples by their BMU. Let S_j be the set of samples mapped to neuron j, and $n_j = |S_j|$. Let

$$\mathbf{x}_{\mu,j} = \frac{1}{n_j} \sum_{\mathbf{x}_s \in S_j} \mathbf{x}_s$$

be the mean of those samples. The batch update becomes

$$\mathbf{m}_i^* = \frac{\sum_{j=1}^{K} n_j\, h_{ji}\, \mathbf{x}_{\mu,j}}{\sum_{j=1}^{K} n_j\, h_{ji}}.$$

Initialization

Prototype vectors can be initialized randomly or using the first two principal components (PCA) to accelerate convergence, especially when Euclidean distance is used.

Distance Measures

Kohonen discusses two similarity metrics: - Euclidean distance (dissimilarity), - Cosine similarity via normalized dot product. In high dimensions the difference is often negligible, but with sparse data cosine similarity is computationally advantageous.

Neighborhood Width

The kernel width σ should not decay to zero. Practical guidelines: - Minimum $\sigma \approx 0.5$ grid units. - For large maps, σ may be set to 0.05 of the shorter grid dimension.

Quantization and Topology Preservation

The SOM seeks to minimize the average quantization error between samples and their BMUs, a concept Kohonen refers to as *optimal partitioning*. However, additional indices evaluate preservation of the input topology on the map lattice.

Quantization Error The quantization error (QE) for dataset $\{\mathbf{x}_s\}$ is:

$$\mathrm{QE} = \frac{1}{M} \sum_{s=1}^{M} \left\| \mathbf{x}_s - \mathbf{m}_{c(s)} \right\|.$$

Increasing K typically reduces QE but may impair topology preservation [23].

Topographic Error The topographic error (TE) [23] measures whether the first- and second-best matching units of each sample are adjacent:

$$\mathrm{TE} = \frac{1}{M}\sum_{s=1}^{M} \mathbb{I}\Big[\delta\Big(c^{(1)}(s),\, c^{(2)}(s)\Big) > 1\Big],$$

where $\delta(\cdot,\cdot)$ denotes graph distance on the map grid, and $c^{(1)}, c^{(2)}$ are the first and second BMU indices.

Topographic Product The topographic product (TP) [23, 24] quantifies map dimensionality suitability. Let - $d(\cdot,\cdot)$ denote input-space distance, - $\delta(\cdot,\cdot)$ denote grid distance. Define for neuron j the k-th nearest neighbors: - $n_k^{\delta}(j)$: nearest in grid space, - $n_k^{d}(j)$: nearest in input space. Define ratios

$$Q_1(j,k) = \frac{d(\mathbf{m}_j, \mathbf{m}_{n_k^{\delta}(j)})}{d(\mathbf{m}_j, \mathbf{m}_{n_k^{d}(j)})}, \qquad Q_2(j,k) = \frac{\delta(j, n_k^{\delta}(j))}{\delta(j, n_k^{d}(j))}.$$

Their geometric mean is

$$P_3(j,k) = \left[\prod_{l=1}^{k} Q_1(j,l)\, Q_2(j,l)\right]^{\frac{1}{2k}}.$$

The topographic product is

$$\mathrm{TP} = \frac{1}{K(K-1)} \sum_{j=1}^{K}\sum_{k=1}^{K-1} \log P_3(j,k).$$

Negative values ($\mathrm{TP} < 0$) indicate insufficient map dimensionality; positive values ($\mathrm{TP} > 0$) indicate excessive dimensionality. This measure is reliable primarily for nearly linear data.

Clustering on SOM Prototypes

Several works [25, 26] cluster data indirectly by clustering SOM prototypes instead of the full dataset. Since $K \ll M$, this approach offers reduced computational cost and noise smoothing because prototypes represent local averages.

This method is effective only if the SOM has preserved the data topology; otherwise, clustering the prototypes deviates from clustering the raw data. Map visualization often uses distance-based methods such as the U-matrix, which encodes inter-prototype distances through colors.

The optimal cluster count can be determined using validity indices such as the Davies-Bouldin Index (DBI), though this should be interpreted as guidance rather than strict ground truth. SOM map size is frequently chosen heuristically as

$$K \approx 5\sqrt{M}.$$

2.4.3 Internal Validation Indices

In an unsupervised learning problem, where the true class labels are unknown, only internal validation indices can be used to assess the quality of the clustering. These indices evaluate the clustering structure based on intrinsic properties of the data, such as cohesion (how close the points within a cluster are) and separation (how distinct different clusters are), without relying on any external ground truth labels.

In [27], the authors provide an extensive review of both classical and recently proposed clustering validation indices, including a summary table of their computational complexity, advantages, and limitations. The newly proposed WB index is the only one that identifies the correct number of clusters in all the datasets tested. Although the way the WB index is applied differs from the approach used in this work, an additional evaluation is carried out to verify whether it also performs well in this context.

WB Index

According to [28], this family of indices, which is based on sums of squares, shows promising results in determining the number of clusters. These indices rely on the sum of squares within clusters (SSW), and the sum of squares between clusters (SSB).

$$\mathrm{SSW} = \sum_{i=1}^{N} \|x_i - c_{p_i}\|^2 \qquad \mathrm{SSB} = \sum_{i=1}^{M} n_i \left\|c_i - \bar{X}\right\|^2$$

$X = \{x_1, \ldots, x_N\}$	dataset of N points
$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} x_i$	global mean
$C = \{c_1, \ldots, c_M\}$	set of M cluster centroids
c_i = centroid of the i-th cluster	
p_i = cluster index (label) of point x_i	
n_i = number of points in the i-th cluster	

The WB index is calculated as:

$$\mathrm{WB} = M\,\frac{\mathrm{SSW}}{\mathrm{SSB}} \tag{2.1}$$

Unfortunately, no Python implementation of the WB index appears to be available in the literature, therefore the following code provides the implementation.

```
def wb_index(self, data: np.ndarray, labels: np.ndarray) -> float:
    n_clusters = len(np.unique(labels))
    overall_mean = np.mean(data, axis=0)

    ssw = 0
    ssb = 0
    for k in np.unique(labels):
        cluster_points = data[labels == k]
        cluster_mean = np.mean(cluster_points, axis=0)
        ssw += np.sum((cluster_points - cluster_mean) ** 2)
        ssb += len(cluster_points) * np.sum((cluster_mean - overall_mean) **
            2)

    wb = n_clusters * (ssw / ssb)
    return wb
```

The optimal value for this metric is the minimum. However, when the data have more than two dimensions, the metric rarely identifies a clear minimum. Since the dataset used in this work contains 31 features, this metric is not suitable. Moreover, it is designed for data that follow a Gaussian distribution, which is the same limitation already discussed for K-Means.

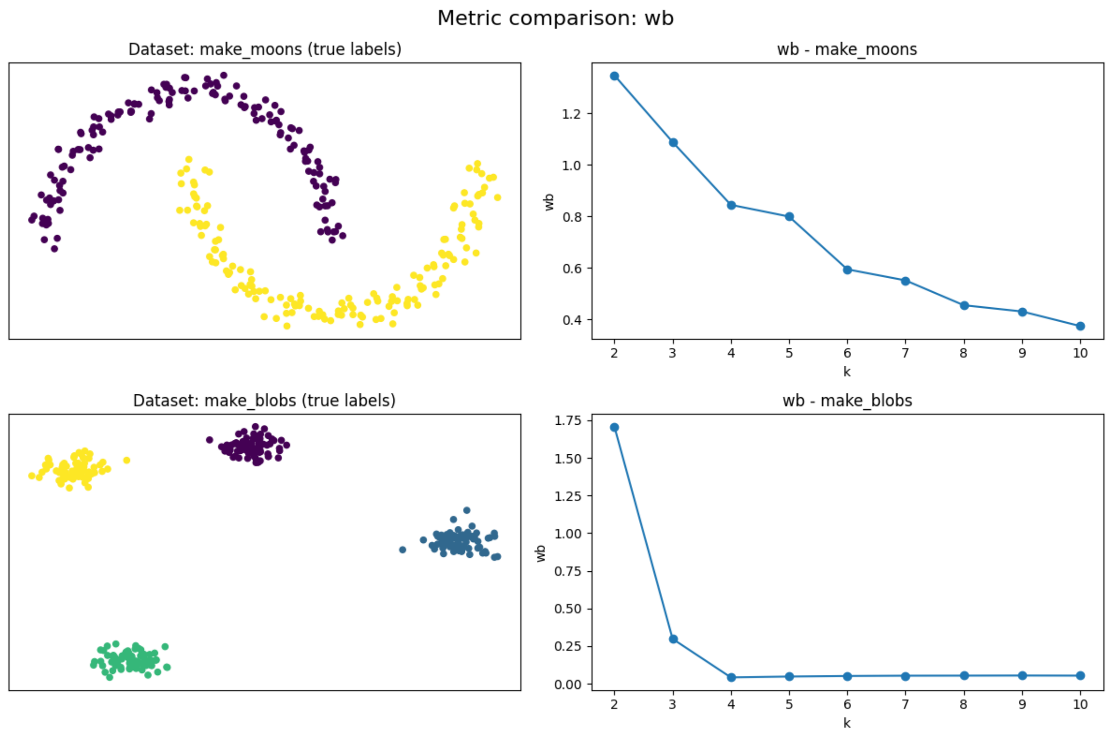


Figure 2.11: WB Index

As shown in the plot, the metric has difficulties determining the exact number of clusters in the *make_moons* dataset, while it performs well on the *make_blobs* dataset.

Silhouette Coefficient

The Silhouette coefficient [29] measures the consistency of clustering results by quantifying how similar a data point is to other points within its own cluster (cohesion) compared to points in other clusters (separation). For a data point i, it is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{2.2}$$

where:

- $a(i)$ is the average distance between i and all other points in the same cluster (intra-cluster cohesion).
- $b(i)$ is the minimum average distance between i and all points in any other cluster (inter-cluster separation).

The Silhouette value ranges from -1 to 1, where values close to 1 indicate that a sample is well matched to its own cluster and poorly matched to neighboring clusters. The overall Silhouette score is computed as the mean of $s(i)$ across all data points. The computational complexity of the Silhouette index is $O(n^2)$, where n is the number of data points.

As noted by Rousseeuw, the Silhouette coefficient is most informative when the clusters are compact and well separated, and when their structure is close to spherical.

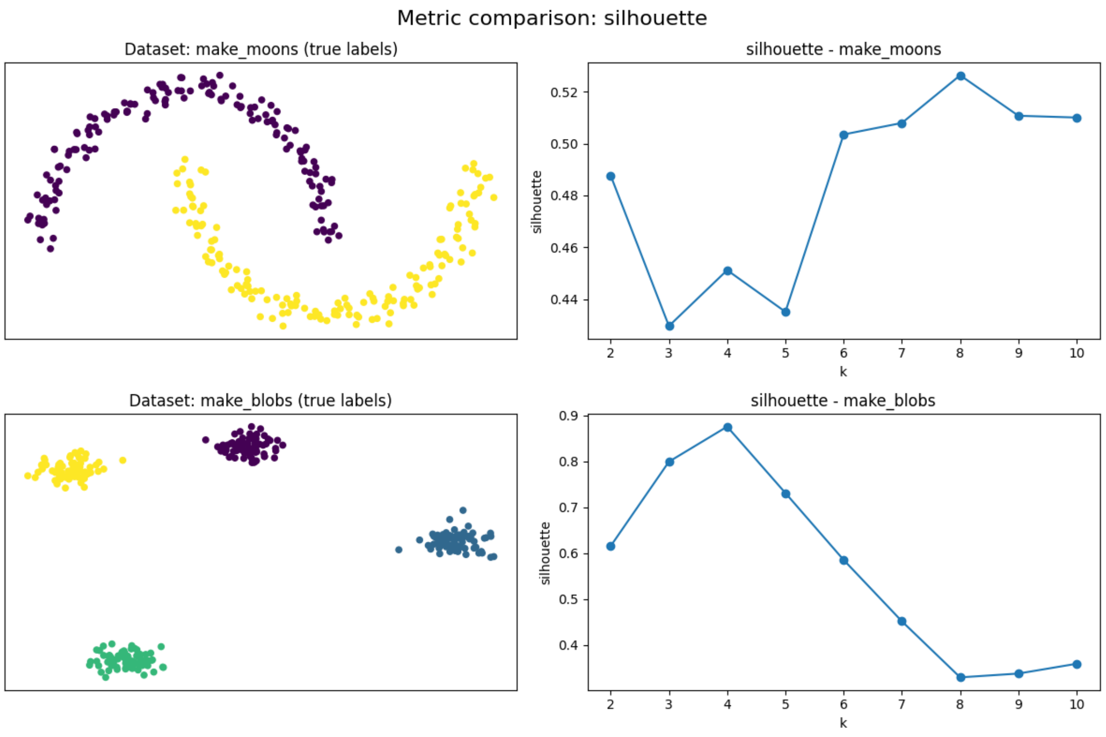


Figure 2.12: Silhouette

This figure illustrates that the Silhouette score struggles to detect the appropriate number of clusters in the *make_moons* dataset, while it provides a clear indication of the optimal partitioning for the *make_blobs* dataset.

Davies–Bouldin Index

The Davies–Bouldin Index (DBI) [30] is another widely used internal evaluation metric. It quantifies the average similarity between each cluster and the most similar one to it, where the similarity is defined as a function of the ratio between within-cluster scatter and inter-cluster separation. It is defined as:

$$DBI = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{S_i + S_j}{M_{ij}}\right) \tag{2.3}$$

where:

- k is the number of clusters.
- S_i is the average distance between each point in cluster i and the centroid of cluster i (intra-cluster scatter).
- M_{ij} is the distance between the centroids of clusters i and j (inter-cluster separation).

A lower DBI value indicates a better clustering result, as it reflects low intra-cluster dispersion and high inter-cluster separation. Its computational complexity is $O(n)$, making it more efficient than the Silhouette coefficient.

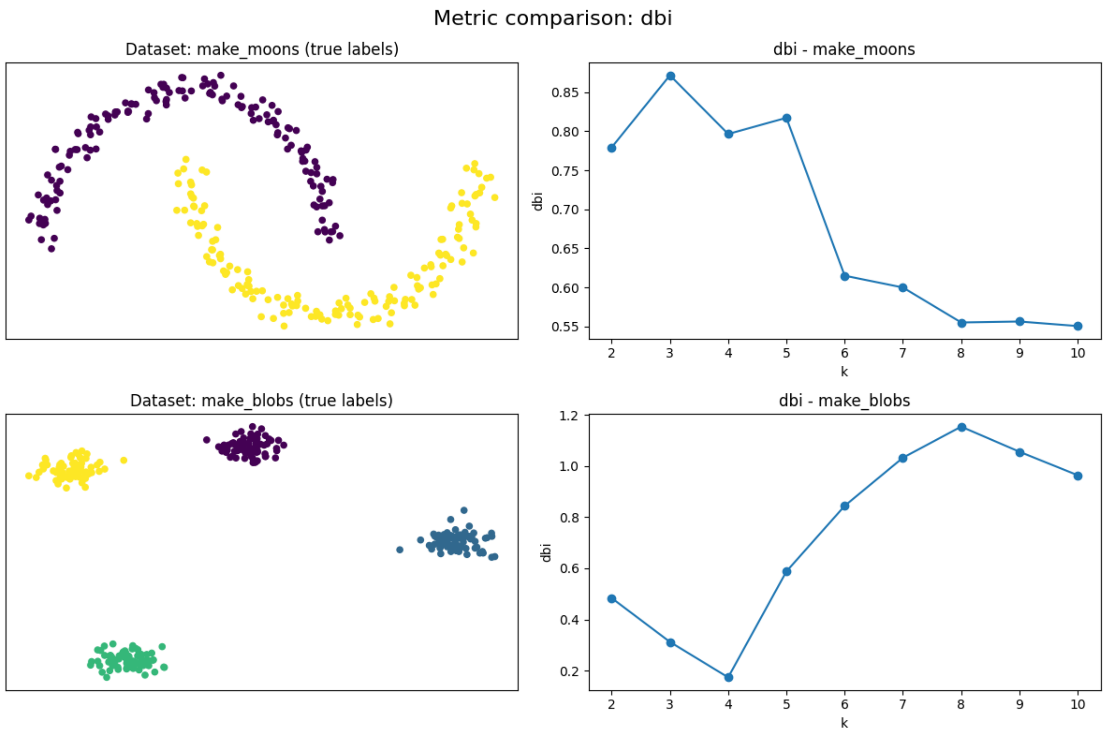


Figure 2.13: Davies-Bouldin Index (DBI)

The same reasoning applied to the Silhouette score also applies to the Davies-Bouldin Index, since both metrics evaluate the quality of the clustering based on intra-cluster cohesion and inter-cluster separation, although they do so through different formulations.

These observations align with the findings of Gagolewski *et al.* [31], who show that several widely used internal validity indices, including the Silhouette and the Davies–Bouldin coefficient, may favor clusterings that do not correspond to expert judgement, particularly when they are used as objective functions to be maximized. In their study, Gagolewski *et al.* also introduce more robust alternatives to classical validity measures, for example OWA-based generalized Dunn variants, which are available in the `genieclust` library [32].

DuNN-OWA

Let $X = \{x_1, \dots, x_n\}$ and let $C(i)$ denote the cluster label of x_i. For each i, let $NN_M(i)$ be the set of its M nearest neighbours. The DuNN-OWA index is

$$\mathrm{DuNN}(C) = \frac{\mathrm{OWA}_s(\{\ \|x_i - x_j\| : C(i) \neq C(j),\ i \in NN_M(j) \text{ or } j \in NN_M(i)\ \})}{\mathrm{OWA}_c(\{\ \|x_i - x_j\| : C(i) = C(j),\ i \in NN_M(j) \text{ or } j \in NN_M(i)\ \})}. \tag{2.4}$$

Only distances in the M-nearest neighbour graph are used: the numerator reflects inter-cluster separation and the denominator reflects intra-cluster compactness. OWA operators may be chosen as Min, Max, or Mean, while smooth variants such as SMin and SMax downweight extreme values, reducing the influence of isolated outliers.

To avoid degenerate ratios, if one of the distance sets is empty, default values can be used (e.g. $\mathrm{OWA}_s = 0$ or $\mathrm{OWA}_c = +\infty$), since $\mathrm{DuNN}(C) = 0$ indicates no inter-cluster separation and $\mathrm{DuNN}(C) = +\infty$ indicates perfect separation with no internal connectivity.

This metric shows good performance on the *make_moons* and *make_blobs* datasets.

The optimal clustering corresponds to the highest DuNN-OWA score.

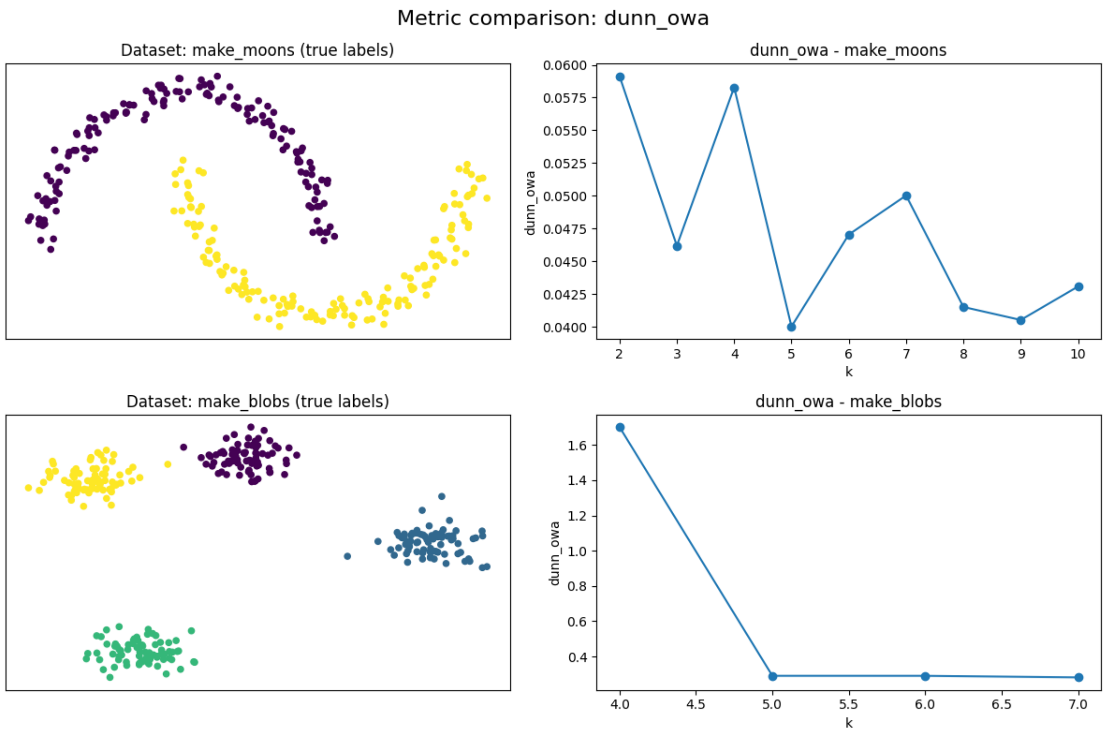


Figure 2.14: DuNN-OWA index values

2.5 Labeling

In the study [33], the authors propose a technique for performing clustering by relying exclusively on large language models. The method consists of two main stages.

In the first stage, the objective is to identify potential labels for the dataset. Due to the context window limitations of the models, the dataset is divided into batches, each batch is processed to assign preliminary labels to its samples, and the resulting candidate labels are then aggregated. This aggregation step is necessary because different batches may produce labels with distinct terms but equivalent meanings.

In the second stage, the approach performs a classification task. Each text sample is assigned to one of the previously determined labels, which enables the clustering process. The authors also evaluate the token cost associated with the method and provide the prompts employed in their experiments.

This article focuses on text clustering, however, it explores several approaches [34]. The authors investigate how different large language models generate embeddings for clustering tasks. Multiple models are tested and compared with classical embedding methods. Six clustering techniques are evaluated, one of which is K-Means, and the GPT based model achieves the highest performance. Furthermore, when using GPT embeddings, even a simple algorithm such as K-Means performs well.

Text clustering is not the primary focus of this thesis. However, large language models (LLMs) have evolved at a rapid pace, with increasingly promising benchmark results. In this work, we investigate whether LLMs can effectively handle contextual information, process sessions, and interpret HTTP requests. Their performance in these scenarios will be evaluated in the course of this thesis.

Chapter 3

Infrastructure and Operational Context

3.1 Infrastructure Requirements and Architectural Justification

Before describing the Sicuranext infrastructure, it is important to explain the requirements that shaped its design. Since Sicuranext's Web Application & API Protection (WAAP) is a managed service, the infrastructure must meet several technical and operational needs:

- **Complete TLS termination and inspection:** Sicuranext must decrypt incoming traffic to apply WAF rules, plugins, and logging. TLS termination must be secure, compliant with standards, and able to scale automatically during periods of high traffic
- **Elastic scalability and fault tolerance:** The system should adapt automatically to changes in traffic, without manual intervention. It must include automatic scaling, health checks, and failover mechanisms to guarantee availability
- **Layered security:** Security rules must act at different points of the HTTP lifecycle. L7 protections (WAF, rate-limiting, anomaly detection) must be enforced before forwarding requests, while the outgoing response must also be inspected to avoid data leaks
- **Performance and latency constraints:** Security should not create excessive delays. Cache mechanisms and geographically distributed entry points should reduce response time and avoid unnecessary computation
- **Audit and traceability:** Every request and response must be logged with enough context to allow forensic analysis, behavior tracking, and debugging. Logs should be indexed, searchable, and usable with analytical tools
- **Vendor independence and sustainability:** Cloud services simplify operations, but the infrastructure must remain portable. Sicuranext must be able to reduce cloud dependency and migrate components without redesigning the entire system

These requirements lead to a WAAP architecture based on Kong Gateway, strengthened by ModSecurity, and supported by distributed caching and centralized logging. Each element balances security, performance, and operational cost.

3.1.1 Infrastructure Overview

The diagram below shows the main components of the infrastructure and the flow of requests and responses:

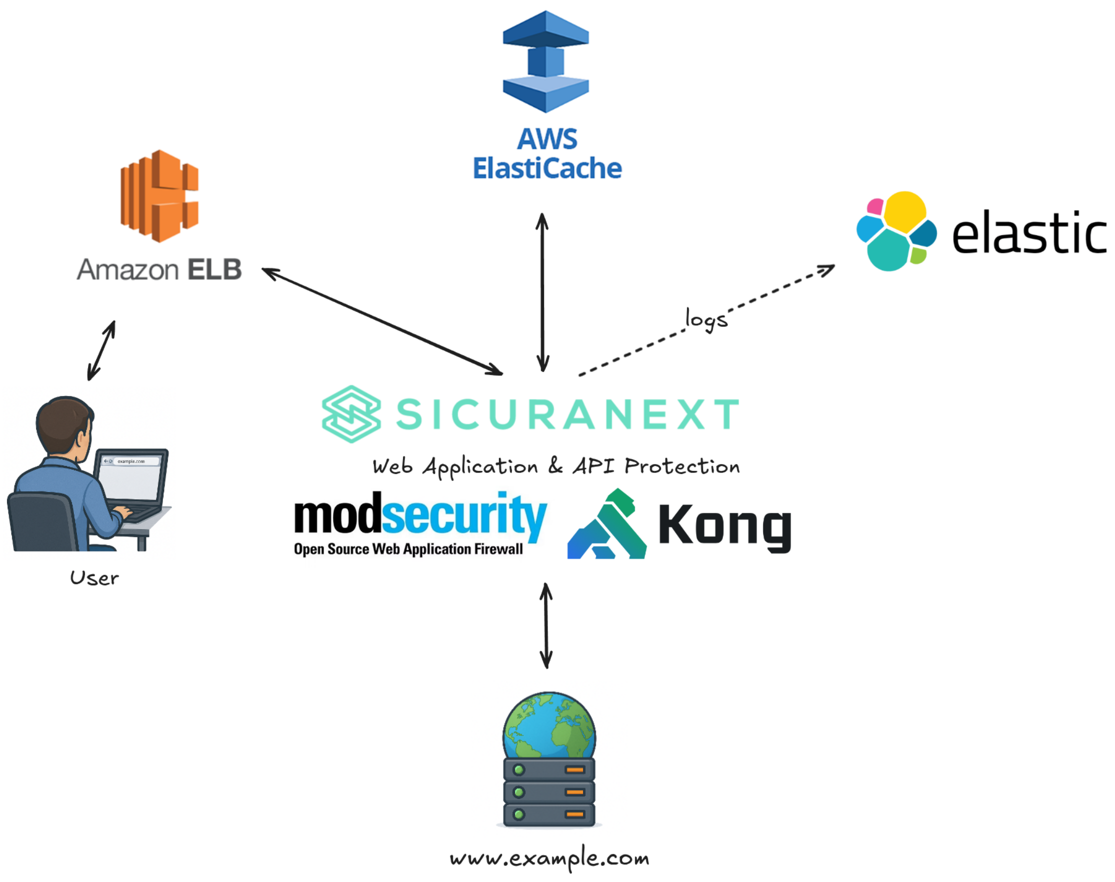


Figure 3.1: Sicuranext Infrastructure

Assuming Sicuranext protects the website *example.com*, the request and response flow is as follows:

- When a user visits *example.com*, the request is received by the Amazon Elastic Load Balancer (ELB). ELB distributes traffic and manages TLS certificates to decrypt it.
  ELB was chosen for three main reasons. First, AWS handles load balancing, scaling, and health checks. This allows the infrastructure to scale without extra work from Sicuranext. Second, ELB supports AWS Global Accelerator, which provides static IPs distributed across more than 20 AWS regions. Requests are served from the closest region, reducing latency. Third, AWS Certificate Manager (ACM) integrates with ELB and allows the creation of trusted TLS certificates at no additional cost.

  Alternatives to ELB: Sicuranext could use Kong Gateway, OpenResty, or Nginx for load balancing, but these options would require more engineering and maintenance. The company plans to replace ELB with Kong Gateway in the future to reduce AWS dependency and support smaller clients.

  Alternatives to ACM: Let's Encrypt could generate certificates and import them into ACM or use them directly in Kong. Sicuranext plans to use Let's Encrypt with Kong to automate ACME challenges and manage TLS certificates per service.

- After TLS termination, the decrypted request is forwarded to the WAAP layer, where Kong Gateway operates. Kong integrates the ModSecurity WAF with the OWASP Core Rule Set (CRS) and custom rules.
  ModSecurity is one of the few open-source WAFs that reliably work with Nginx. Other

open-source options such as Coraza still show stability issues. Service-based solutions such as AWS WAF exist, but they do not offer the flexibility or features required by Sicuranext.

- Kong Gateway also supports plugins. In this setup, one plugin blocks requests from malicious Autonomous System Numbers (ASNs). An AS controls a group of IP address ranges. Blocking individual IPs is easy to bypass since attackers can switch to another IP from the same AS. Blocking the whole AS is more effective, but it must be used carefully to avoid affecting legitimate users. Only ASes with a history of abuse should be blocked.
- If the request is valid, it is forwarded to the protected application (*example.com*). The application generates a response.
- On the return path, ModSecurity inspects the response again. If the content is static, it may be stored in AWS ElastiCache (Redis) to improve performance for future requests. Sicuranext chose Amazon ElastiCache to deploy a managed Redis cluster, reducing operational complexity. In the future, the company plans to host its own Redis cluster to lower AWS costs.

  Alternatives: Other possible technologies include Memcached, KeyDB, or Apache Cassandra. Redis was selected because of its maturity and strong Lua module support in OpenResty.
- When a second request for the same content is made, ModSecurity still performs its checks, but cached content may bypass some Kong plugins, such as the ASN filter. This improves performance while maintaining a reasonable level of security.
- All requests and responses are logged asynchronously and stored in Elasticsearch. Sicuranext uses Elasticsearch for storage and Kibana for visualization. Other valid alternatives include OpenSearch, Apache Solr, or MongoDB.

The dataset for analysis will be downloaded from Elasticsearch. Blocking based on ASNs will be important during the preprocessing described in Chapter 4.

Chapter 4

Implementation Approach

4.1 Dataset Creation

The construction of the dataset required defining which information could identify a client in a stable manner during a session. As discussed earlier, relying on a single feature such as IP address or User-Agent is not sufficiently reliable because of NAT, IP rotation, and User-Agent rotation (see Section 2.1).

For this reason, the dataset creation relies on the combination of both identifiers. This approach reduces the risk of grouping unrelated clients within the same session or splitting a single client into multiple artificial ones.

User activity is not constant. It may contain short bursts of requests followed by periods of inactivity and then resume. To capture this pattern, the requests are aggregated per minute. This representation preserves the chronological order of actions and separates active periods from those without traffic. It also prevents Elasticsearch from returning a single large document per client, which would be difficult to process. At this stage, minute buckets provide only a temporal view of the traffic and do not define the logical session.

To extract the data, Elasticsearch is queried through a composite aggregation over the IP and the User-Agent. The composite aggregation returns groups of client keys in batches, which avoids loading all records into memory at once. Within each key, a date histogram with a fixed interval of one minute collects the requests occurring during that time range. The query used is shown below:

```
"size": 0,
"query": {
    "bool": {
        "must": [
            {"range": {"@timestamp": {"gte": start_time, "lte": end_time}}},
            {"term": {"octofence.service-id": service_id}}
        ]
    }
},
"aggs": {
    "all_clients": {
        "composite": {
            "size": 100,
            "sources": [
                {"real_client_ip": {"terms": {"field": "real_client_ip"}}},
                {"user_agent": {"terms": {"field":
                    "request.headers.user-agent"}}}
            ]
        },
        "aggs": {
            "per_minute": {
                "date_histogram": {
                    "field": "@timestamp",
                    "fixed_interval": "1m",
                    "min_doc_count": 1
                }
            }
        }
    }
}
```

An example of the response is:

```
"key": {
    "real_client_ip": "X.X.X.X",
    "user_agent": "Mozilla/5.0 (X11; Linux i686) ..."
},
"doc_count": 1,
"per_minute": {
    "buckets": [
        {
            "key_as_string": "2025-07-31T19:24:00.000Z",
            "key": 1753989840000,
            "doc_count": 1
        },
        {
            "key_as_string": "2025-07-31T19:31:00.000Z",
            "key": 1753990260000,
            "doc_count": 1
        }
    ]
}
```

This output shows two requests issued by the same client. They appear in two different minute buckets because they were sent seven minutes apart. However, this does not imply that they correspond to two distinct sessions. The minute buckets are only a representation of the temporal distribution of activity.

The actual session reconstruction is performed in Python. All consecutive requests generated by the same $(\text{IP}, \text{User-Agent})$ pair are assigned to the same session as long as the temporal gap between them is less than one hour. If the gap exceeds one hour, a new session is created. In the previous example, the two requests are separated by seven minutes, therefore they are assigned to the same session. This logic reflects typical user behaviour: brief pauses are common, whereas longer periods of inactivity usually indicate that the interaction has ended.

The one-hour threshold was selected primarily because Sicuranext also protects e-commerce platforms. In this context, users often take more time before generating the next request. They may read product pages, compare alternatives, or leave the page open while deciding what to purchase. Shorter time limits would fragment this behaviour into many separate sessions, even though it corresponds to a single visit. The one-hour window maintains slow navigation within the same session and prevents artificial interruptions in the user journey.

Elasticsearch cannot directly manage session information because the logs contain no field that defines it. The system stores only individual HTTP requests, each with its own timestamp and metadata. Consequently, Elasticsearch can group data by fields or time windows, but it cannot detect when a user stops browsing and resumes later. Since the concept of *session* does not exist in the raw data, the session boundaries must be computed externally in Python after the aggregated records have been retrieved.

4.1.1 Features

The feature set contains 31 elements. They are organised by purpose, and their identification is based on the considerations already discussed in Section 2.2. The intention is not to introduce features only because they are technically available, but to select those that can provide information useful to distinguish normal behaviour from suspicious traffic. For this reason, the dataset includes both features already explored in previous research and new ones tailored to the context of this work.

The first group captures basic traffic volume and request types:

- total_number_requests
- get
- post
- non_get_post
- unassigned_referer

These features describe the overall structure of the session. For example, an unusual number of non-GET requests is often associated with automated scripts or brute force attacks rather than interactive browsing. Similarly, a missing or undefined `referer` may indicate non-standard navigation patterns or direct URL probing, which are less common in regular user behavior.

The `Sec-Fetch-*` headers are used to enrich this view. As presented earlier, they contain browser-generated information about the context and origin of a request. This information makes it possible to approximate actions that are likely initiated by a human user, as opposed to requests issued by the browser in the background or by automated tools. This distinction is important because many raw HTTP requests do not reflect real user intent. For example, loading images, fonts, or JavaScript files may produce dozens of requests even if the user is only visiting a single page.

Using this filtering logic, two behavioral metrics are derived: `max_requests_per_minute` and `max_page_ratio`. They are computed only from requests that match one of the following values: `sec-fetch-user = ?1`, `sec-fetch-site = none`, `sec-fetch-dest = document`, or `sec-fetch-mode = navigate`. These values correlate with active navigation events, such as opening a page or clicking a link, and therefore increase the reliability of the measurements.

The first metric captures bursts of interaction: `max_requests_per_minute`. Without filtering, this metric would largely reflect how many static assets the browser retrieves. Instead, in this work, it focuses on requests that are more closely aligned with user intent. The second metric measures concentration:

$$\texttt{max\_page\_ratio} = \frac{\text{requests for the most frequent page}}{\text{total requests}}$$

A high value suggests repetitive interactions with a specific endpoint, which may be legitimate (for example a dashboard refresh) or malicious (for example a brute force login attempt). Previous studies proposed similar indicators [6, 5], but in this thesis the use of the `Sec-Fetch` heuristic increases their interpretability in real-user contexts.

The following features represent the distribution of `Sec-Fetch` values over the session:

- sfd_image (sec-fetch-dest)
- sfd_document (sec-fetch-dest)
- sfd_script (sec-fetch-dest)
- sfd_font (sec-fetch-dest)
- sfm_cors (sec-fetch-mode)

- sfm_navigate (sec-fetch-mode)
- sfm_no_cors (sec-fetch-mode)
- sfm_same_origin (sec-fetch-mode)
- sfs_cross_site (sec-fetch-site)
- sfs_same_origin (sec-fetch-site)
- sfs_same_site (sec-fetch-site)
- sfs_none (sec-fetch-site)
- sfu_?1 (sec-fetch-user)
- xml_http_requests
- max_requests_per_minute
- max_page_ratio

These indicators describe the "shape" of a session. Normal browsing tends to mix document loads, script retrievals, and embedded resources from related origins. On the contrary, automated tools often show simpler patterns: repeated navigation to a single endpoint, an unusual ratio of cross-site requests, or an absence of resources associated with page rendering.

Response content types were previously addressed in Section 2.2. They are useful to distinguish sessions originating from webpages, which typically return `text/html`, from those caused by programmatic APIs, which frequently return `application/json`. Only the top-level category is considered:

- ct_image
- ct_font
- ct_application
- ct_text

HTTP error codes provide further insight. Instead of merging all 4xx responses into one group, they are separated, because each type explains a different behaviour. Scanning activities usually produce many `404 Not Found` responses, as the attacker explores unknown paths. In contrast, attempts to bypass access controls are more likely to produce `403 Forbidden`. This separation has more diagnostic value than the single 4xx indicator used in several previous works:

- error_status_code_401
- error_status_code_403
- error_status_code_404
- error_status_code_other

The `avg_response_content_length` feature was already introduced in Section 2.2. The `time_interval` measures the duration of the session, which helps separate short automated attacks from longer activities typical of human users. When considered together, these features provide multiple perspectives on the session and support the model in identifying behaviour that deviates from normal patterns:

- avg_response_content_length
- time_interval

4.1.2 Pre-Processing

Before clustering, the raw dataset contains 37 features. This initial set includes identification fields such as `real_client_ip` and `user_agent`, temporal boundaries such as `start_time` and `end_time`, and operational indicators such as `num_asn_blocks`. The last feature records how many requests belonging to the session were blocked because the ASN of the client had a poor reputation. The role of ASN and its impact on traffic quality are described in Chapter 3.

The rationale behind this first filtering step is that sessions originating from untrusted ASNs have a high probability of being malicious or at least not representative of normal user behavior. Keeping them in the dataset would introduce noise and affect the interpretation of the clustering output. Therefore, if a session contains at least one blocked request, the entire session is excluded. Once the filtering is done, the `num_asn_blocks` feature has no remaining purpose and is removed.

A second exclusion rule is applied based on session size. Sessions containing fewer than five requests do not provide enough information to derive meaningful behaviour. Such short traces often correspond to accidental visits, crawler probes, or incomplete navigation, and including them would weaken the model rather than improve it. For this reason, they are discarded.

The next transformation concerns feature scaling relative to traffic volume. Most indicators represent counts, and large sessions would naturally show larger values even if their behaviour is normal. To avoid bias toward users who simply interact more, all features except `total_number_requests`, `time_interval`, `max_requests_per_minute`, `max_page_ratio`, and `avg_response_content_length` are divided by the total request count. This step converts raw frequencies into ratios and makes sessions comparable regardless of size.

After normalization, feature redundancy is inspected through the correlation matrix (Section 2.3). Highly correlated variables generally reflect the same underlying phenomenon, therefore they provide limited additional information. Their inclusion increases the dimensionality of the dataset without necessarily improving the separation between clusters, and in distance-based models this may even distort the results. Nevertheless, in certain contexts such variables can capture subtle or complementary aspects that contribute to distinguishing one group from another, so their relevance ultimately depends on the specific analytical scenario.

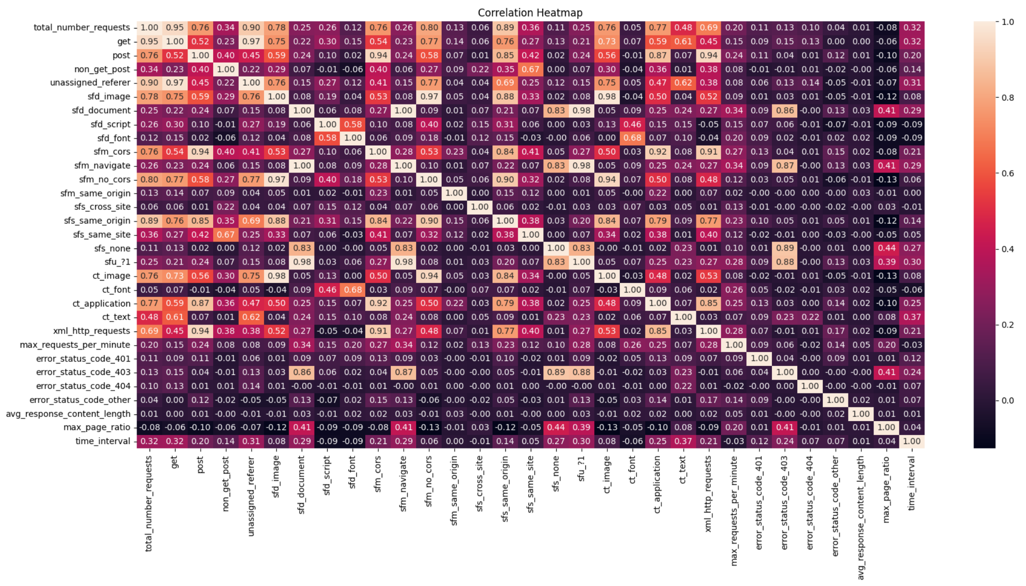


Figure 4.1: Correlation Matrix

As shown in Figure 4.1, the features `get`, `unassigned_referer`, `sfm_cors`, `sfm_navigate`, `sfm_no_cors`, `sfu_?1`, `ct_image`, `ct_application` and `xml_http_requests` exhibit strong linear correlations, with coefficients of at least 0.9. A reduced dataset is therefore produced where these features are removed to evaluate whether clustering performance improves. This step is not only a dimensionality reduction process, but also a way to assess which signals are actually necessary to represent distinct behaviours.

Finally, features are standardized using the Z-score transformation,

$$z = \frac{x - \mu}{\sigma},$$

so that variables with large numeric ranges do not dominate the clustering algorithm. Standardization ensures that all features contribute with equal weight and that distance metrics reflect differences in behavior rather than scale.

4.2 Dip Test for Multimodality

It is necessary to understand whether the dataset is likely to contain more than one underlying group. The Dip Test is a statistical tool that helps detect multimodality by analysing the distribution of pairwise distances between samples (see Section 2.4.1). If the distance distribution contains more than one significant peak, the dataset is unlikely to be homogeneous and a single-cluster model would not be appropriate. This step therefore acts as a preliminary validation that can prevent misleading interpretations in later stages.

In practice, computing the full pairwise distance matrix for this dataset is not feasible. With 87,942 samples and 31 features, evaluating all combinations would generate $\frac{87942 \times 87941}{2}$ distance values, approximately 3.87 billion entries. Since each `float64` element requires 8 bytes, the total memory usage would reach about 28.81 GB. The workstation used in this project has 16 GB of RAM, so a full calculation cannot be performed.

A possible workaround is to apply the Dip Test to a lower dimensional representation. One common solution is to run PCA and apply the test only to the first principal component. This approach assumes that most of the structure of the dataset is captured by a small number of linear components. If the data were approximately linear, the first two or three principal components should explain most of the variance.

For this reason, PCA is used to explore whether the dataset behaves linearly:

```
def _explore_dataset_linearity(self):
    def plot_cumulative_variance(explained_variance_ratio: np.ndarray):
        plt.figure(figsize=(8, 5))
        plt.plot(
            range(1, len(explained_variance_ratio) + 1),
            explained_variance_ratio,
            marker="o",
            linestyle="--",
        )
        plt.title("Elbow Method for PCA")
        plt.xlabel("Number of Principal Components")
        plt.ylabel("Cumulative Explained Variance")
        plt.grid(True)
        plt.show()

    X = self._X
    pca = PCA(n_components=0.9)
    pca.fit(X)
    eigenvalues = pca.explained_variance_ratio_
    explained_variance_ratio = np.cumsum(eigenvalues)
    logging.info(f"Eigenvalues: {np.round(eigenvalues, 5)}")
    plot_cumulative_variance(explained_variance_ratio=explained_variance_ratio)
```

The expected outcome, if the dataset were roughly linear, is that only the first few eigenvalues would be significantly larger than zero. Equivalently, two or three principal components should capture around 90% of the variance. However, in this case 16 components are needed to reach this threshold, which indicates that the information is distributed across many directions:

```
Eigenvalues: [0.24604 0.1277 0.09541 0.06468 0.04382 0.03985 0.03947 0.03472
              0.03413 0.0319 0.02983 0.02965 0.02731 0.02537 0.02475 0.02021]
```

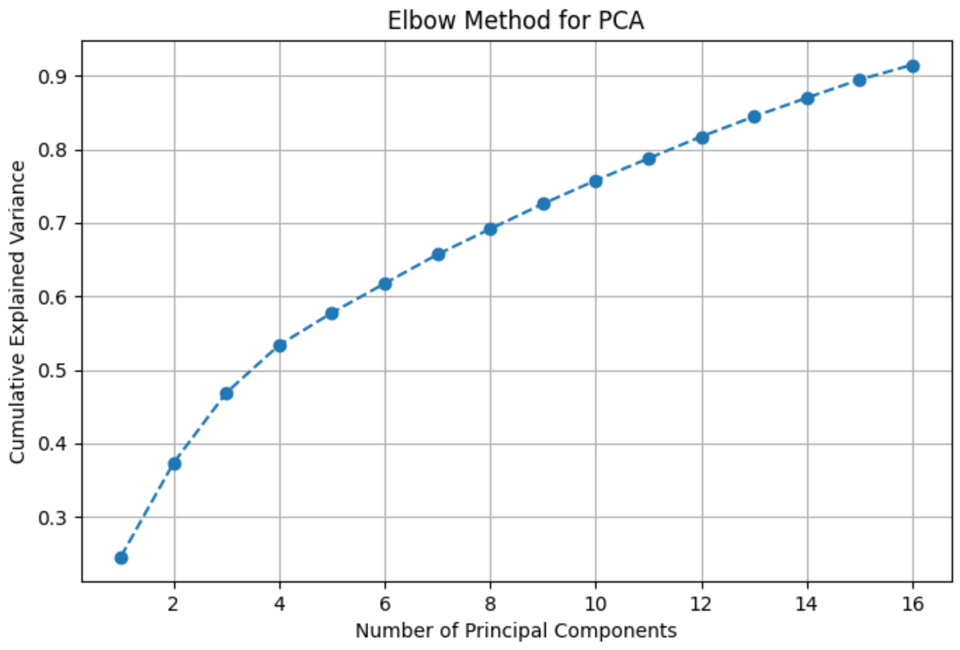


Figure 4.2: PCA Explained Variance

Since the dataset does not exhibit a clear linear structure, applying the Dip Test only to the first principal component would ignore most of the information. This would produce results that are not reliable and could misrepresent the real geometry of the data. For this reason, the PCA–Dip approach is discarded.

The alternative is to keep the original feature space but reduce the number of samples in a meaningful way. A naive sampling method, such as randomly selecting a subset of sessions, would risk removing relevant behaviors and modifying the underlying distribution. The goal is not just to reduce the volume of data, but to preserve the relative distances between samples so that the Dip Test remains interpretable.

To achieve this, a SOM is adopted. When correctly trained, a SOM preserves the topology of the input space: samples that are close in the original dataset remain close on the resulting grid. This allows replacing individual sessions with the weight vectors of SOM neurons. These weight vectors keep the original dimensionality, so this procedure is not a form of dimensionality reduction. Instead, it compresses the dataset into a smaller set of representative prototypes, reducing the number of points on which the Dip Test is applied while maintaining the global structure of the data.

4.3 Self-Organizing Map

Before setting the parameters, it is important to clarify the role of the SOM within the proposed solution. The SOM is not employed to predict or classify. Instead, it provides a low-dimensional representation of the data, where sessions that behave in a similar way should be mapped close to one another. The quality of this representation depends on the ability of the map to preserve distances from the original space and reflect meaningful neighbourhood relations.

From a design perspective, the SOM parameters must support these goals. Therefore, the selection of the map size, neighbourhood radius, distance metric and training policy is not arbitrary. They affect how the model learns the structure of the dataset and how stable the resulting representations are.

The Python library used in this thesis to construct the SOM is **MiniSom** [35]. The SOM has several parameters that must be defined, such as:

- **x** and **y**: the dimensions of the map. In this work, $x = y$, resulting in a square map.
- **input_len**: the dimensionality of the input vectors (i.e., the number of features).
- **sigma**: the initial spread of the neighborhood function.
- **learning_rate**: the initial learning rate.
- **decay_function**: the method used to reduce the learning rate over time.
- **neighborhood_function**: the kernel used to define how neighboring neurons are influenced during training.
- **topology**: the map topology (e.g., rectangular or hexagonal).
- **activation_distance**: the metric used to compute distances when assigning input vectors to neurons.
- **random_seed**: random initialization seed to ensure reproducibility.
- **sigma_decay_function**: the method used to update the neighborhood radius during training.

Starting from the dimensional perspective, the topographic product does not provide substantial information, as already discussed in Section 2.4.2. Since the dataset is not linear, the topographic product oscillates between small negative and positive values, often remaining close to zero. Therefore, this metric cannot be used to determine the optimal size of the map. Moreover, because the computation of the topographic product is computationally demanding, only the map sizes of 10, 15, 20, 25, 30, 35, and 40 were evaluated. A common rule of thumb to define the map size is $x = y = 5\sqrt{M}$, where M denotes the number of data samples; in this case $M = 39$. This rule is adopted in the present work, as it yields satisfactory results in terms of quantization error and topographic error. These two errors were already examined in Section 2.4.2, and lower values indicate better performance.

Unfortunately, in version 2.3.5 of MiniSom, the batch training mode is not available, and the function named `train_batch` in that version does not correspond to Kohonen's batch training algorithm is not available. Since the author of the Self Organizing Map, Kohonen, introduced this algorithm in 2013, in this thesis the batch version is implemented so that it is possible to evaluate whether it performs better than the standard online training. It should be noted that the batch algorithm does not require a learning rate.

Regarding the remaining parameters, such as `sigma`, the `learning_rate` in the case of standard training, the `neighborhood_function` and the `topology`, they remain unchanged. This choice is mainly motivated by the heuristics provided by Kohonen, according to which the initial `sigma` should be approximately one half of the map side length in the case of a quadratic topology, while for the online method the `learning_rate` should start at a relatively high value. The Gaussian neighborhood function is the classical formulation discussed in the original works. These recommendations are described in Kohonen's articles. What is optimized through Optuna in order to determine the best configuration are the `decay_function`, the `sigma_decay_function` and the `activation_distance`.

The hyperparameter tuning procedure operates as follows. The two algorithms are executed under different training configurations. In the online training setting, the number of iterations is set to 1000, and the batch size is equal to 2. Regarding the initialization of the model parameters, the online training approach employs random weight initialization, whereas the batch training approach uses initialization based on principal component analysis (PCA).

It is important to note that a single iteration of the batch training procedure is not equivalent to one iteration of the online training procedure, since the batch update internally cycles through the entire dataset. After multiple runs, the configuration that produces the best performance is selected, and the corresponding hyperparameters are retained. Eventually, both algorithms converged toward the same optimal set of parameters, namely:

```
{'decay_function': 'asymptotic_decay', 'sigma_decay_function':
    'inverse_decay_to_one', 'activation_distance': 'cosine'}
```

Both training methods effectively train the map using the appropriate number of iterations. In the case of online training, the number of iterations is $500 \cdot x \cdot y$ where x and y are the dimensions of the map. In the case of batch training, the number of iterations is 20. These values are not chosen arbitrarily, they are suggested in the work of Kohonen.

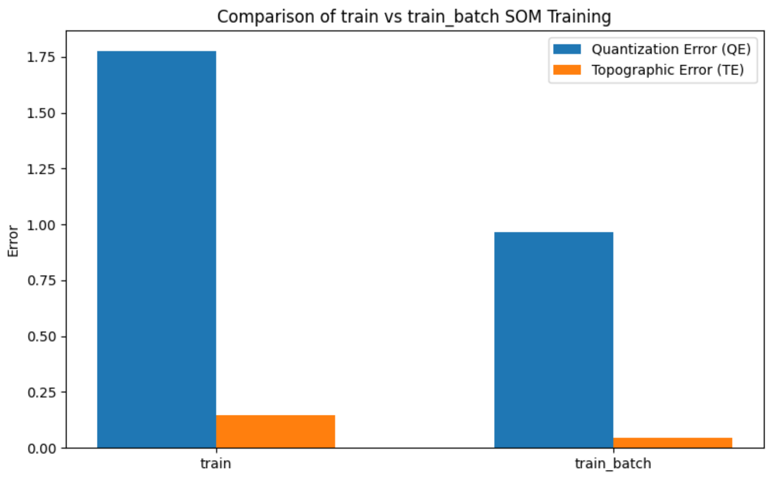


Figure 4.3: Online Training versus Batch Training

As shown in Figure 4.3, batch training achieves consistently very good results, and it performs better than online training.

```
[train] QE = 1.77805 | TE = 0.14707
[train batch] QE = 0.96502 | TE = 0.04284
```

To assess whether the obtained results are satisfactory, the quantization error (QE) and the topographic error (TE) should be as low as possible (see Section 2.4.2).

Before applying any clustering method, it is necessary to verify whether the weights are likely to form distinct groups. To this end, the Dip Test is employed to assess the null hypothesis of unimodality. The procedure was presented in Section 2.4.1. The resulting p-value is:

```
dip p-value: 0.02567051939734566
```

Since the obtained p-value is greater than the conventional significance threshold of 0.05, the null hypothesis of unimodality cannot be rejected. Nevertheless, as mentioned earlier, the author observes a success rate of 95%, therefore this dataset is not discarded and is still taken into account.

Indeed, as shown below, the distribution of the pairwise distances, together with the empirical cumulative distribution function (ECDF), displays a unimodal pattern, since no evident flat regions are present.

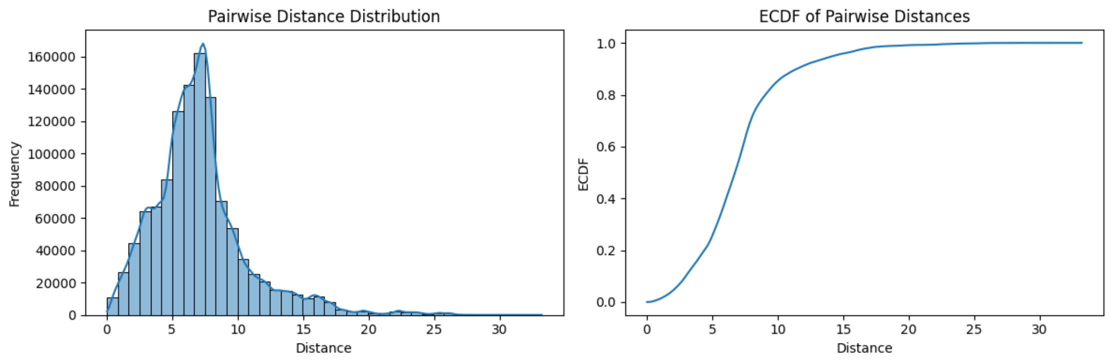


Figure 4.4: Distribution of the pairwise distances and ECDF of SOM weights

At this stage, a new SOM map was trained on the dataset after the correlated features had been removed, and this approach appears to yield better results. A secondary, smaller mode emerges in the resulting distribution, which may indicate a tendency toward multimodality.

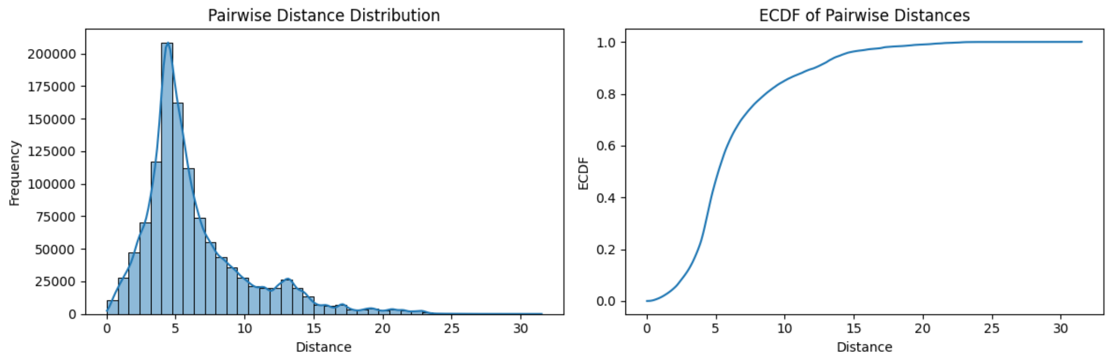


Figure 4.5: Distribution of the pairwise distances and ECDF of SOM weights with correlated features removed

To verify this result, the *p*-value of the Dip Test is reported below:

```
dip p-value: 0.0
```

4.3.1 K-Means

K-Means was employed to cluster the SOM weights. As noted in Section 2.4, this algorithm has certain limitations; however, it remains relatively simple compared to alternatives like DBSCAN, which is also reviewed in [18].

The main challenge with DBSCAN lies in properly tuning the hyperparameters $minPts$ and *eps*. The automatic parameter determination method proposed in [36] was applied, but the algorithm identified only a single cluster in both distinct datasets. For the dataset containing all features, the results were:

```
Auto-tuned DBSCAN params: eps=11.7067, MinPts=32
```

In the dataset without highly correlated features, the parameters were:

```
Auto-tuned DBSCAN params: eps=10.6948, MinPts=2
```

These results suggest that DBSCAN may struggle with the current dimensions of the datasets. While it might perform well with lower dimensionality (e.g., up to 10 dimensions), it proved ineffective in our specific cases involving 22 and 31 features.

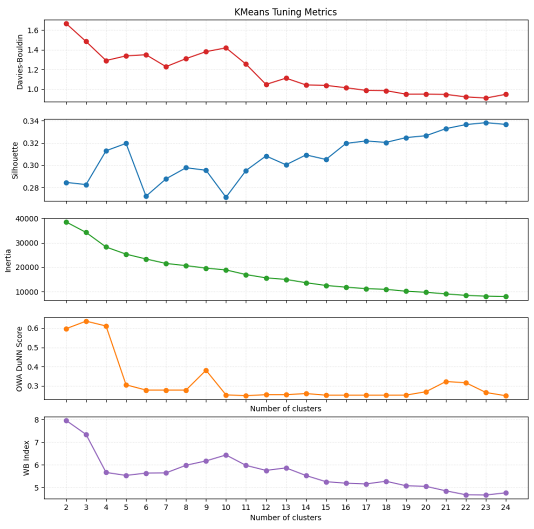
4.3.2 Results

The clustering results may also be examined, even though these indices have already been discussed in detail in Section 2.4.3, where it was shown that they do not always coincide with expert judgement.

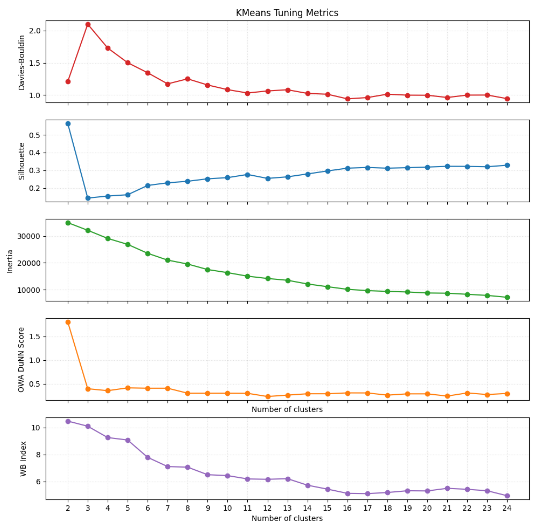
In the present case, the validation indices do not provide consistent indications. The best values of both the Silhouette coefficient and the Davies–Bouldin index are obtained at $k = 23$, whereas the DuNN OWA method identifies $k = 3$ as the optimal number of clusters.

In contrast, for the SOM trained with correlated features removed, the silhouette score and the DuNN–OWA metric are coherent. These metrics have performed reliably in experiments using the *make_moons* and *make_blobs* datasets available in scikit–learn. Moreover, The silhouette value is greater than 0.5, which indicates satisfactory clustering quality.

The SOM trained without the correlated features performs better than the SOM trained using all features, the map is reported below (Figure 4.7). The interpretation of the SOM map is that the neurons shown in black are very distant from the neighbouring neurons, which indicates that they are different, and possibly that the sessions belonging to them behave differently.



(a) Internal metrics for SOM weights



(b) Internal metrics for SOM weights with correlated features removed

Figure 4.6: Comparison of internal metrics for SOM weights before and after removing correlated features

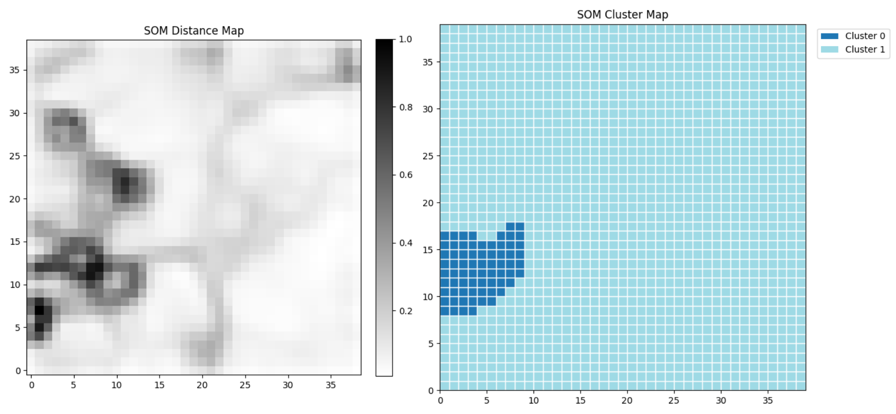


Figure 4.7: SOM map with correlated features removed

Two distinct clusters are expected to exhibit different behaviours. It is plausible that one cluster represents human navigation, or at least traffic patterns that mimic human navigation, while the other appears to be automated. To assess this hypothesis, a subset of neurons is analysed manually.

```
Neuron: (0, 10)
user_agent Go-http-client/1.1
start_time 2025-11-16T11:32:00.000Z
end_time 2025-11-16T11:51:00.000Z
total_number_requests 59
get 59
unassigned_referer 59
error_status_code_403 59
time_interval 19.0

user_agent python-requests/2.31.0
start_time 2025-11-13T21:20:00.000Z
end_time 2025-11-13T21:21:00.000Z
total_number_requests 51
get 43
post 8
unassigned_referer 51
error_status_code_403 51
time_interval 1.0

Neuron: (1, 10)
user_agent Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Ge...
start_time 2025-11-16T05:31:00.000Z
end_time 2025-11-16T05:57:00.000Z
total_number_requests 8
get 8
unassigned_referer 7
ct_text 3
error_status_code_403 6
avg_response_content_length 8371.333333
time_interval 26.0
```

These sessions are generated by automated scripts. The first two can be identified from the user agent, while the third can be inferred from unassigned referrers and from the fact that the number of HTTP 403 errors is almost equal to the total number of requests. It is also possible to understand that it is an automated script because it does not include the Sec–Fetch headers and it is likely not a genuine Mozilla client, but a script that modifies the User–Agent. This highlights the importance of these new features, which help distinguish legitimate traffic from automated traffic.

In contrast, a normal navigation process is observed. Almost all of the Secure Fetch (Sec–Fetch) headers are present, only a single HTTP 404 error occurs, and the total number of requests is 151 (neuron $(35, 0)$).

```
user_agent Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...
start_time 2025-11-12T07:35:00.000Z
end_time 2025-11-12T07:46:00.000Z
total_number_requests 151
get 117
post 34
unassigned_referer 114
sfd_image 114
sfd_document 1
sfm_cors 35
```

```
sfm_navigate 1
sfm_no_cors 115
sfs_same_origin 150
sfs_none 1
sfu_?1 1
ct_image 114
ct_font 0
ct_application 35
ct_text 2
xml_http_requests 35
max_requests_per_minute 2.0
error_status_code_404 1
avg_response_content_length 1048.925926
max_page_ratio 0.006623
time_interval 11.0
```

When the number of clusters increases, the manual inspection of the results becomes increasingly time consuming. As already noted, clustering evaluation metrics do not provide conclusive assessments, therefore additional analyses are required. Since $k = 23$ produces the most coherent structure in the clustering of the SOM weights using the dataset that includes all features, an additional experiment is carried out with a large language model (LLM) in order to determine whether it can interpret the underlying context and distinguish between benign and malicious sessions.

4.4 LLM

4.4.1 Design Rationale for the Labeling Approach

The goal of the labeling phase is to provide a meaningful interpretation of the clusters produced by K-Means and the Self-Organizing Map. A cluster without an explanation is difficult to use in any practical scenario. For this reason, the design of the labeling method must consider both the structure of the data and the way in which patterns emerge at the session level.

The first design decision concerns the nature of the labels. Instead of focusing on very specific features (for example, large numbers of image requests or repeated failures), the labels should describe a higher-level intent or behaviour. This abstraction makes the results easier to compare across different clusters and different datasets. It also supports future work, since behavioral labels can be reused even if the underlying raw features change.

A second design choice is the use of representative samples rather than full session histories. Full inspection of every session would be impractical, especially when the number of neurons grows. The model is therefore asked to infer the dominant behaviour from a limited number of sessions per cluster and a few raw HTTP requests for each session. This reduces the computational cost, but it also creates a trade-off: some contextual information is removed, and small variations may become invisible. The design accepts this limitation because the objective is to produce an initial hypothesis, not to provide a definitive classification.

The third design element consists of treating previously assigned labels as part of the system state. When a label has already been assigned, it should be reused if a similar pattern emerges. This choice is motivated by the need for consistency, since assigning independent names to similar clusters would increase ambiguity and hinder comparison. Reusing existing labels encourages the model to connect current evidence with previously acquired knowledge. New labels are introduced only when the observed behaviour clearly differs from the existing descriptions. This principle has already been addressed in [33].

Finally, the method assumes that automatic labeling cannot fully replace expert judgment. The model is used to reduce the manual workload and to highlight potential behavioural patterns, but the final evaluation remains a human task. This is important because the system may extract superficial signals from the data, or it may generalize based on incomplete evidence. Therefore, the design explicitly includes a validation phase in which the analyst compares session statistics, raw requests, and known traffic patterns to decide whether the assigned label is credible.

Since the map containing all the features shows the best values of the Silhouette score and the Davies–Bouldin Index at $k = 23$, the K-Means algorithm is executed with $k = 23$. The resulting map is presented below.

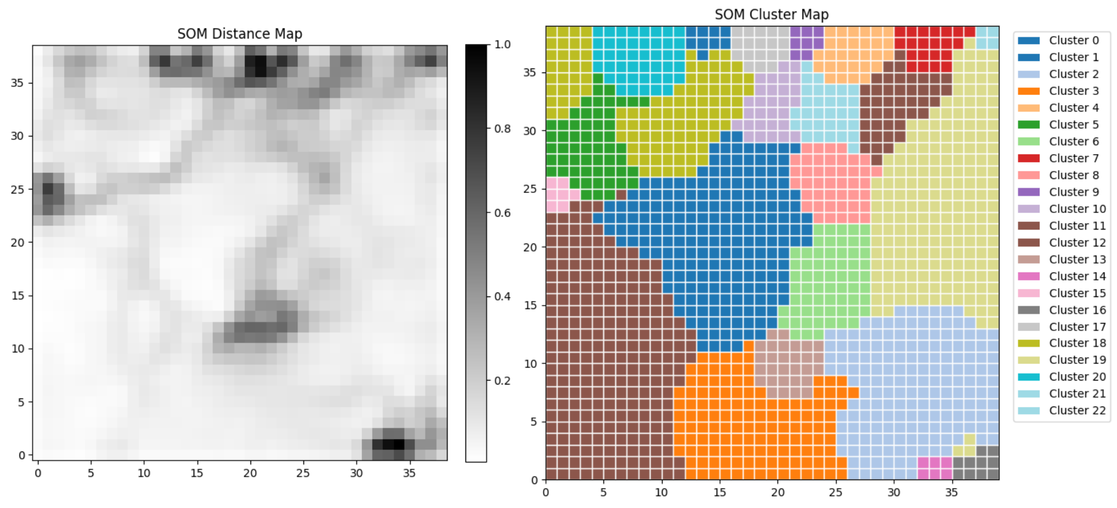


Figure 4.8: SOM Map

The neurons display clear coloration, which makes it possible to observe that neurons located farther away, represented by darker colors, form the boundaries of the clustering. This effect does not appear in the previous example, where only two clusters are present. In the current case, manually examining each cluster is very time-consuming, particularly because a large number of neurons must be inspected.

To address this issue, large language models (LLMs) can assist in automating the labeling process. In this work, only three clusters are analyzed, and for each cluster five sessions are considered. For every session, five raw requests are randomly selected and examined. This procedure serves merely as a preliminary test, since a full analysis would require an excessive number of tokens and represents a promising direction for improvement, possibly by employing a local LLM. Below, the system prompt is defined, as it is an important component when working with large language models:

```
system_prompt = """
Self-Organizing Map Neuron Labeling - Behavioral/Intent Categories

You are an analyst working with a Self-Organizing Map (SOM) that clusters web
    sessions based on HTTP traffic patterns. Each neuron in the SOM
    represents a group of sessions with similar characteristics. Your task is
    to assign a meaningful, high-level behavioral label to a single neuron
    based on the data provided.

You will receive:
- Some sessions mapped to the same neuron, each with summarized statistics.
- For one session, a sample of some raw HTTP requests.
- A list of previously assigned labels and their descriptions (to avoid
    duplicates or near-duplicates).

Labeling guidelines:
```

```
- Focus on the main purpose, behavioral pattern, or security-relevant signals
    reflected by these sessions.
- Do not simply summarize atomic features (such as "onlyimage" or "manyget").
    Instead, infer the higher-level intent or behavior suggested by the
    features.
- Consider whether the sessions resemble typical human browsing, automated
    bots, crawlers, background tasks, or other patterns of interest for
    traffic and security analysis.
- If the behavior matches or is very similar to an existing label (including
    its description), reuse that label and description exactly. Only create a
    new label and description if no existing label is suitable.
- Descriptions must be a single brief sentence and reused if similar patterns
    arise.

The very first labeling task will not include any existing labels and will be
    used to define the initial label set.

Expected output format:
Return a JSON object exactly as follows (no explanations, no extra text):
{
"label": <one-word category label>,
"description": <very brief explanation, suitable for cluster analysis, of why
    this label was chosen>
}
"""
```

An example of the user prompt is provided below. In the actual system, empty fields or fields containing 0 are included, even though they are omitted here:

```
--- Session Summary ---
{
  "total_number_requests": 525,
  "get": 525,
  "unassigned_referer": 525,
  "ct_application": 241,
  "ct_text": 284,
  "max_requests_per_minute": 0.0,
  "error_status_code_401": 284,
  "time_interval": 281.0
}
--- Raw Requests Sample ---
[
  {
    "method": "GET",
    "uri_path": "[REDACTED]",
    "content-type": "application/json",
    "http_code": 200
  },
  {
    "method": "GET",
    "uri_path": "[REDACTED]",
    "content-type": "text/html",
    "http_code": 401
  },
  {
    "method": "GET",
    "uri_path": "[REDACTED]",
    "content-type": "text/html",
    "http_code": 401
  },
  {
    "method": "GET",
    "uri_path": "[REDACTED]",
    "content-type": "application/json",
    "http_code": 200
  },
  {
    "method": "GET",
    "uri_path": "[REDACTED]",
    "content-type": "application/json",
    "http_code": 200
  }
]
```

Only three clusters are labeled by the LLM. The labels and their descriptions are the following:

```
{
  "12": {
    "label": "pageview",
    "description": "Multiple GET-only requests for same-origin images, styles
        and scripts with occasional navigations, consistent with normal
        browser page loads."
  },
  "0": {
    "label": "pageview",
    "description": "Multiple GET-only requests for same-origin images, styles
        and scripts with occasional navigations, consistent with normal
        browser page loads."
  },
  "1": {
    "label": "api-probing",
    "description": "Repeated unauthenticated GET requests to API endpoints
        with many 401 responses and no referer, consistent with automated API
        scanning or probing."
  }
}
```

Then, it is possible to manually analyze these clusters and evaluate whether both the clustering and the labeling are appropriate.

For cluster 1, the neurons are located at (13,38) and (13,37), the remaining neurons are empty, and the results are consistent with the labels and the description.

```
user_agent axios/1.13.2
start_time 2025-11-14T06:20:00.000Z
end_time 2025-11-14T11:30:00.000Z
total_number_requests 58
get 58
unassigned_referer 58
ct_text 58
error_status_code_401 58
time_interval 310.0

user_agent Mozilla/5.0 (Linux; Android 10; K) AppleWebKit...
start_time 2025-11-12T12:55:00.000Z
end_time 2025-11-12T13:18:00.000Z
total_number_requests 8
```

```
get 5
post 3
unassigned_referer 4
ct_application 4
ct_text 4
xml_http_requests 4
error_status_code_401 3
time_interval 23.0
```

In the first case, the User-Agent information helps to identify the type of session, and it is also relevant because the total number of requests is equal to the number of HTTP 401 errors. The second session was grouped into this cluster due to the presence of HTTP 401 errors. However, this evidence alone does not allow us to conclude that the session is malicious. Some fields, such as `ct_application` and `xml_http_requests`, appear consistent with legitimate activity. It is possible that the attacker modified the headers. While this is a session in the cluster 0, labeled as pageview:

```
user_agent Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...
start_time 2025-11-12T17:13:00.000Z
end_time 2025-11-12T17:16:00.000Z
total_number_requests 34
get 34
unassigned_referer 28
sfd_image 11
sfd_document 2
sfd_script 12
sfd_font 2
sfm_cors 2
sfm_navigate 2
sfm_no_cors 30
sfs_cross_site 2
sfs_same_origin 32
sfu_?1 2
ct_image 11
ct_application 2
ct_text 21
max_requests_per_minute 1.0
max_page_ratio 0.058824
time_interval 3.0
```

The presence of the `Sec-Fetch-*` headers suggests that the request originates from a real browser operated by a human user. In particular, the value `Sec-Fetch-Mode: navigate` denotes that the request triggers a navigation action, which is typically associated with a deliberate visit to the page.

Here, another type of session is observed. Until now, only sessions containing errors have been considered malicious. However, another cluster includes this session. It is likely to be malicious, because it contains eight requests that are neither GET nor POST, and the requests do not include any `Sec-Fetch` headers.

```
user_agent Mozilla/5.0 AppleWebKit/537.36...
start_time 2025-11-15T13:45:00.000Z
end_time 2025-11-15T16:45:00.000Z
total_number_requests 16
get 8
non_get_post 8
unassigned_referer 16
ct_text 16
time_interval 180.0
```

Chapter 5

Conclusion

The aim of this thesis was to investigate whether HTTP sessions reconstructed from server-side logs can be clustered according to their behavioral properties. The study focused on patterns that emerge when multiple requests are considered as a collective sequence, instead of isolated events. This shift from request-level inspection to session-level analysis allows a better understanding of user behavior and helps distinguish genuine navigation from automated or suspicious interactions.

The first contribution of this work concerns the reconstruction of sessions from raw traffic data. Since Sicuranext logs do not contain any explicit session identifier, the use of IP address and User-Agent was adopted as a practical compromise. This method is not perfect, but it was sufficient to perform exploratory analysis, to extract features, and to evaluate the feasibility of clustering. The second contribution lies in the construction of a feature set designed to characterize sessions. Alongside well-known indicators such as request frequency or ratio of HTTP methods, additional signals were introduced, including Sec-Fetch headers and response content categories. These features provided valuable context and helped capture differences between human-driven requests and automated behavior.

To assess whether clustering was meaningful, the Dip Test was applied. Although this statistical test operates in one dimension, it enabled the identification of multimodal structure through dimensionality reduction or distance-based formulations.

Classical clustering method, mainly K-Means, was adopted due to the moderate dimensionality of the dataset. While this method has limitations, it produced coherent clusters that could be interpreted in practical contexts. Finally, the semantic interpretation of clusters confirmed the presence of distinct behavioral groups and provided initial labels that may be used in subsequent supervised learning tasks.

Closing Remarks

The results of this study demonstrate that unsupervised learning can extract meaningful behavioral patterns from HTTP sessions using only server-side information. Even without advanced session identification or deep learning techniques, the selected models were able to highlight clear differences between types of user activity. Feature selection played a central role: the combination of classical statistical indicators and less explored metadata, such as Sec-Fetch headers, helped distinguish genuine navigation flows from automated or stealth interactions. Although the work does not deliver a production-ready system, it lays a solid foundation for future development and shows that clustering is a valid first step toward automated threat detection.

Future Improvements

Several directions may enhance the effectiveness of this approach. First, the session identification method should be refined. Reliance on IP and User-Agent introduces fragmentation and mixing effects, especially in the presence of NAT, VPNs, or User-Agent rotation. Techniques such as browser fingerprinting, dedicated cookies, or persistent identifiers could provide more stable session boundaries.

Second, the clustering stage may benefit from more advanced models. Deep clustering, autoencoders, or contrastive representation learning could better capture non-linear patterns and reduce noise in highly heterogeneous traffic. These approaches may also help deal with edge cases where classical methods show instability.

Third, the labeling process could be reinforced. While manual labeling is time-consuming, and cloud-based LLMs raise privacy and security concerns, a local large language model could be fine-tuned on representative sessions and used to produce labels without exposing sensitive traffic data outside the protected environment. This would help automate semantic interpretation while preserving confidentiality and regulatory compliance.

Finally, the labeled clusters derived in this work provide a starting point for supervised learning. Models such as decision trees or gradient boosting could be trained to classify sessions in real time. When doing so, latency, interpretability, and robustness against adversarial manipulation must be considered, since threats often evolve to evade detection.

Deploying the approach in a controlled production environment would also provide essential feedback. Factors such as model drift, changing traffic patterns, and operational constraints cannot be fully evaluated offline. Continuous monitoring, retraining strategies, and integration with existing logging or alerting systems are necessary to maintain the long-term effectiveness of session-based detection.

Bibliography

[1] A. Kumar and J. A. Gutierrez, "Impact of machine learning on intrusion detection systems for the protection of critical infrastructure", Information, vol. 16, no. 7, 2025, DOI 10.3390/info16070515

[2] M. E. Durmuşkaya and S. Bayraklı, "Web application firewall based on machine learning models", PeerJ Computer Science, vol. 11, 2025, p. e2975, DOI 10.7717/peerj-cs.2975

[3] O. Ibitoye, R. Abou-Khamis, M. ElShehaby, A. Matrawy, and M. Shafiq, "The threat of adversarial attacks against machine learning in network security: A survey", Journal of Electronics and Electrical Engineering, 01 2025, DOI 10.37256/jeee.4120255738

[4] K. Mani and A. K. Shenoy, "Machine learning models in web applications: A comprehensive review", ICT Express, 2025, DOI https://doi.org/10.1016/j.icte.2025.09.001

[5] Y. Sun, Y. Xie, W. Wang, S. Zhang, J. Gao, and Y. Chen, "Wsad: An unsupervised web session anomaly detection method", 2020 16th International Conference on Mobility, Sensing and Networking (MSN), 2020, pp. 735–739, DOI 10.1109/MSN50589.2020.00125

[6] D. Stevanovic, N. Vlajic, and A. An, "Unsupervised clustering of web sessions to detect malicious and non-malicious website users", Procedia Computer Science, vol. 5, 2011, pp. 123–131, DOI https://doi.org/10.1016/j.procs.2011.07.018. The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011)

[7] D. Zhang, J. Zhang, Y. Bu, B. Chen, C. Sun, and T. Wang, "A survey of browser fingerprint research and application", Wireless Communications and Mobile Computing, vol. 2022, no. 1, 2022, p. 3363335, DOI https://doi.org/10.1155/2022/3363335

[8] S. Li and Y. Cao, "Who touched my browser fingerprint? a large-scale measurement study and classification of fingerprint dynamics", Proceedings of the ACM Internet Measurement Conference, New York, NY, USA, 2020, pp. 370–385, DOI 10.1145/3419394.3423614

[9] M. S. M. S. Annamalai, I. Bilogrevic, and E. D. Cristofaro, "Beyond the crawl: Unmasking browser fingerprinting in real user interactions", 2025

[10] H. Venugopalan, S. Munir, S. Ahmed, T. Wang, S. T. King, and Z. Shafiq, "Fp-inconsistent: Measurement and analysis of fingerprint inconsistencies in evasive bot traffic", 2025

[11] K. Solomos, J. Kristoff, C. Kanich, and J. Polakis, "Tales of favicons and caches: Persistent tracking in modern browsers", Proceedings of the Network and Distributed System Security Symposium (NDSS), 01 2021, DOI 10.14722/ndss.2021.24202

[12] P. C. Pereira, "Web bot detection using biometric features and unsupervised outlier detection", master's thesis, Universidade Do Porto, October 2022. Available at `https://repositorio-aberto.up.pt/bitstream/10216/145527/2/592224.pdf`

[13] D. Theng and K. K. Bhoyar, "Feature selection techniques for machine learning: a survey of more than two decades of research", Knowl. Inf. Syst., vol. 66, December 2023, pp. 1575–1637, DOI 10.1007/s10115-023-02010-5

[14] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective", ACM Computing Surveys (CSUR), vol. 50, no. 6, 2018, p. 94

[15] X. Wei, Z. Zhang, H. Huang, and Y. Zhou, "An overview on deep clustering", Neurocomputing, vol. 590, 2024, p. 127761, DOI https://doi.org/10.1016/j.neucom.2024.127761

[16] Y. Ren, J. Pu, Z. Yang, J. Xu, G. Li, X. Pu, P. S. Yu, and L. He, "Deep clustering: A comprehensive survey", IEEE Transactions on Neural Networks and Learning Systems, vol. 36, no. 4, 2025, pp. 5858–5878, DOI 10.1109/TNNLS.2024.3403155

[17] U. Michelucci, "An introduction to autoencoders", 2022

[18] H. Yin, A. Aryani, S. Petrie, A. Nambissan, A. Astudillo, and S. Cao, "A rapid review of clustering algorithms", 2024

[19] A. Adolfsson, M. Ackerman, and N. C. Brownstein, "To cluster, or not to cluster: An analysis of clusterability methods", Pattern Recognition, vol. 88, April 2019, pp. 13–26, DOI 10.1016/j.patcog.2018.10.026

[20] J. A. Hartigan and P. M. Hartigan, "The Dip Test of Unimodality", The Annals of Statistics, vol. 13, no. 1, 1985, pp. 70 – 84, DOI 10.1214/aos/1176346577

[21] T. Kohonen, "The self-organizing map", Proceedings of the IEEE, vol. 78, no. 9, 1990, pp. 1464–1480, DOI 10.1109/5.58325

[22] T. Kohonen, "Essentials of the self-organizing map", Neural Networks, vol. 37, 2013, pp. 52–65, DOI https://doi.org/10.1016/j.neunet.2012.09.018. Twenty-fifth Anniversay Commemorative Issue

[23] G. Polzlbauer, "Survey and comparison of quality measures for self-organizing maps", Proceedings of the Fifth Workshop on Data Analysis (WDA04), 01 2004

[24] F. Forest, M. Lebbah, H. Azzag, and J. Lacaille, "A survey and implementation of performance metrics for self-organized maps", 2020

[25] J. Vesanto and E. Alhoniemi, "Clustering of the self-organizing map", IEEE Transactions on Neural Networks, vol. 11, no. 3, 2000, pp. 586–600, DOI 10.1109/72.846731

[26] M. Pacella and M. Blaco, "On the use of self-organizing map for text clustering in engineering change process analysis: A case study", Computational Intelligence and Neuroscience, vol. 2016, 01 2016, pp. 1–11, DOI 10.1155/2016/5139574

[27] B. A. Hassan, N. B. Tayfor, A. A. Hassan, A. M. Ahmed, T. A. Rashid, and N. N. Abdalla, "From a-to-z review of clustering validation indices", Neurocomputing, vol. 601, October 2024, p. 128198, DOI 10.1016/j.neucom.2024.128198

[28] Q. Zhao and P. FrÃ¤nti, "Wb-index: A sum-of-squares based index for cluster validity", Data & Knowledge Engineering, vol. 92, 2014, pp. 77–89, DOI 10.1016/j.datak.2014.07.008

[29] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis", Journal of Computational and Applied Mathematics, vol. 20, 1987, pp. 53–65, DOI https://doi.org/10.1016/0377-0427(87)90125-7

[30] D. L. Davies and D. W. Bouldin, "A cluster separation measure", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2, 1979, pp. 224–227, DOI 10.1109/TPAMI.1979.4766909

[31] M. Gagolewski, M. Bartoszuk, and A. Cena, "Are cluster validity measures (in) valid?", Information Sciences, vol. 581, dec 2021, pp. 620–636, DOI 10.1016/j.ins.2021.10.004

[32] M. Gagolewski, "genieclust: Fast and robust hierarchical clustering", SoftwareX, vol. 15, 2021, p. 100722, DOI https://doi.org/10.1016/j.softx.2021.100722

[33] C. Huang and G. He, "Text clustering as classification with llms", 2025

[34] I. Keraghel, S. Morbieu, and M. Nadif, "Beyond words: A comparative analysis of llm embeddings for effective clustering", Advances in Intelligent Data Analysis XXII (I. Miliou, N. Piatkowski, and P. Papapetrou, eds.), Cham, 2024, pp. 205–216

[35] G. Vettigli, "Minisom: minimalistic and numpy-based implementation of the self organizing map", 2018

[36] A. Starczewski, P. Goetzen, and M. J. Er, "A new method for automatic determining of the dbscan parameters", Journal of Artificial Intelligence and Soft Computing Research, vol. 10, no. 3, 2020, pp. 209–221, DOI 10.2478/jaiscr-2020-0014