# A Framework for Efficient Pre-Processing of HTTP Requests Using Machine Learning-Based Web Application Firewalls

**Lucas G. Cilento, Paulo S. G. de Mattos Neto and Daniel C. Cunha**[1]

[1]Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE)

{lgc3,psgmn,dcunha}@cin.ufpe.br

***Abstract.*** *The selection of pre-processing methods for requests in web application firewalls (WAFs) that use machine learning is critical in determining performance, latency, and computational resource consumption for detecting web attacks. Among pre-processing techniques, the combination of $N$-gram and term frequency-inverse document frequency (TF-IDF) is one of the most popular alternatives for pre-processing. However, this approach often results in high dimensionality, which can negatively impact latency and resource usage. The key challenge lies in achieving a balance between effective attack detection and resource efficiency. This article proposes a framework for evaluating WAF architectures that preserves detection performance while reducing the number of variables by at least 80%.*

## 1. Introduction

The Hypertext Transfer Protocol (HTTP) is one of the most popular protocols used on the Internet. Millions of clients use the HTTP protocol to communicate with web servers to perform multiple activities, from accessing social networks, streaming, and other general-purpose websites to integrating complex systems using application programming interfaces. Cheap access to virtual private servers and the dissemination of public cloud providers have increased the number of active websites. According to the latest Netcraft report, there are more than a billion active websites [Net 2025]. Most of these sites are publicly available and exposed on the Internet, meaning malicious actors will also have access to them.

The web application firewall (WAF) is a tool capable of parsing and interpreting HTTP requests and searching for suspicious patterns associated with web attacks. These tools can use rules to detect suspicious patterns or machine learning (ML) models to learn how to differentiate between legitimate and suspicious requests. Currently, most enterprise WAF solutions use ML to improve their detection, obtaining great detection rates [Rozenfeld 2024]. One of the main problems regarding ML models in WAF solutions is the increased response time, causing more latency for client applications. High latency is a major issue for all web services, causing a bad user experience, loss of revenue, and other adverse outcomes for businesses. Latency could also become a critical issue for real-time applications such as financial institutions, where loss of credibility, fines, and turnover are some of the worst consequences.

Many factors can impact ML WAF latency, but the main one is the process method used to transform the HTTP request data to feed the models. Approaches such as processing statistical analysis, Markov chains, and embeddings were used as processing methods

[Dhote et al. 2024, Sureda Riera et al. 2020]. Still, the $N$-gram method is chosen by public cloud providers like Cloudflare to process billions of requests daily [Bocharov 2025]. The $N$-gram and the term frequency-inverse document frequency (TF-IDF) methods can capture each feature's importance, but it is not sensitive to feature order. The main issue with this technique is determining the best value for the $N$-gram size, as smaller values may not generate the best features, and larger values significantly increase the consumption of resources and latency.

Some previous work has studied the application of $N$-gram with TF-IDF to parse multiple HTTP payloads, analyzing model performance varying $N$-gram size [Ramezany et al. 2022], [Kumar et al. 2023], [Guo et al. 2023]. However, to the authors' knowledge, no work has yet analyzed the model's performance and prediction time to classify HTTP requests as malicious or legitimate. To address this gap, this work proposes a new framework to analyze ML WAF architectures that use $N$-gram and TF-IDF methods, considering model performance, prediction latency, and resource consumption. We also present the results by analyzing five ML models, using $N$-gram values from one to five and three feature reduction methods. The best results using feature reduction were able to maintain the model performance by decreasing by 99% the number of features created by the $N$-gram and TF-IDF methods.

The remainder of the paper is organized as follows. Section 2 discusses relevant work on our research topics. Section 3 briefly describes the proposed framework. In Section 4, numerical results are shown, while conclusions are drawn in Section 5.

## 2. Related Works

Currently, ML is widely used for detecting and blocking malicious HTTP requests, but the first ML WAF architectures were proposed more than 20 years ago. Kruegel et al. proposed one of the first uses of ML models to detect requests, using feature engineering to create statistical features from arguments [Krügel et al. 2002]. The authors extended their work in [Kruegel and Vigna 2003] and [Kruegel et al. 2005], expanding the number of features designed and using multiple Markov chains to detect anomalies.

The first proposal of using the $N$-gram technique to transform HTTP requests appears in [Song et al. 2009], where Song et al. proposed Spectogram, a framework using $N$-gram and Markov chains to detect web traffic anomalies. In this work, the authors explained the curse of dimensionality for higher values of $N$. To address this issue, Torrano-Gimenez et al. applied a feature reduction method named GeFS to reduce the number of features generated by $N$-gram for $N = 1$ [Torrano-Gimenez et al. 2015]. Babiker et al. propose combining statistical and wrapper methods to reduce the total number of features generated by the $N$-gram method using $N = 1$ [Babiker et al. 2019].

The $N$-gram and TF-IDF integration is a recent approach for processing HTTP requests using $N$-gram. With this combination, the feature occurrence for all requests is also considered to calculate feature importance. Işiker and Soğukpinar analyzed the accuracy of multiple ML methods with $N$-gram values from one to six, considering both character $N$-gram and word $N$-gram, reaching 99.53% of accuracy score [IŞiker and SoĞukpinar 2021]. Hoang and Nguyen used three ML models to detect injection attacks using only URLs transformed with $N$-gram and TF-IDF, obtaining a 99.68% detection rate [Hoang and Nguyen 2021]. Ramezani et al. propose a framework to cap-

ture malicious payloads using honeypots and use this traffic to train the models, reaching 98.7% accuracy with the SVM model using $N = 2, 4$ [Ramezany et al. 2022]. Kumar and Ponsam investigated the impact of $N$-gram values from one to six and analyzed six ML models, where the decision tree (DT) model had the best score with 99.86% [Kumar et al. 2023].

Some recent works focused on detecting only a single type of attack, creating specialist models [Guo et al. 2023, Zhang et al. 2023, Dong and Li 2024]. Guo et al. applied $N$-gram and TF-IDF to transform a database of structured query language (SQL) and trained an SVM classifier to identify SQL injection queries, using precision and recall to evaluate the model's performance [Guo et al. 2023]. Their results indicate that $N = 3$ is the $N$-gram size with the best results, obtaining 98.67% of precision and 99.12% of recall. Zhang et al. proposed a similar architecture to analyze SQL queries, adding more features using the feature ratio method to extract some characteristics from SQL injection attacks [Zhang et al. 2023]. The authors examined seven ML models using accuracy, precision, and F1-scores, where random forest (RF) models had the best results with 99.64% accuracy, 99.50% precision, and 99.61% F1. Dong and Li proposed an application to detect web shells PHP, also using $N$-gram and TF-IDF to process the payloads into tokens, obtaining an accuracy of 99.64%, precision of 99.12%, recall of 99.34%, and F1-score of 99.24% [Dong and Li 2024].

In summary, using $N$-gram and the TF-IDF method is currently one of the best approaches to processing HTTP requests into tokens for ML detection of malicious activities. Some works have analyzed the results using multiple values of $N$, and others have also analyzed how feature reduction methods can reduce the number of features generated by $N$-grams. However, to the authors' knowledge, no work focuses on how the parameter $N$ affects the model's performance when $N$-gram is combined with TF-IDF and how feature reduction methods can improve the model's performance, latency, and resource consumption. This work proposes a new framework to reduce the model response time and resource consumption, avoiding a loss of detection performance.

## 3. Proposed Framework

To address the curse of dimensionality of the combination of $N$-gram and TF-IDF methods, we propose a framework to analyze the processing pipeline in two steps, as illustrated in Figure 1. The first step evaluates the performance of the ML models using values of $N$ from one to five. It compares their overall performance, training time, prediction time, and the number of features created. The second step uses a fixed value of $N$, which was selected in the previous step. It compares the same criteria, using different feature reduction methods to reduce the number of features created. After the second step, the best value of $N$, the feature reduction method, and the ML model are selected.

The proposed framework focuses on evaluating separately the $N$ value of $N$-gram and TF-IDF methods from the reduction methods and the ML models' hyperparameters. The authors adopted this strategy due to the major impact of the $N$ value on the size of vectorized documents. This drawback of the $N$-gram method also implies more memory consumption and increases training and prediction times for ML models. Having a separate step to analyze the overall performance of ML methods, varying the $N$ value, allows the authors to fix one best value for $N$. The framework could also be used to determine

hyperparameters for other vectorization techniques, and other feature reduction methods more suitable for these vectorization techniques could also be added to the set of reduction methods analyzed in the framework.

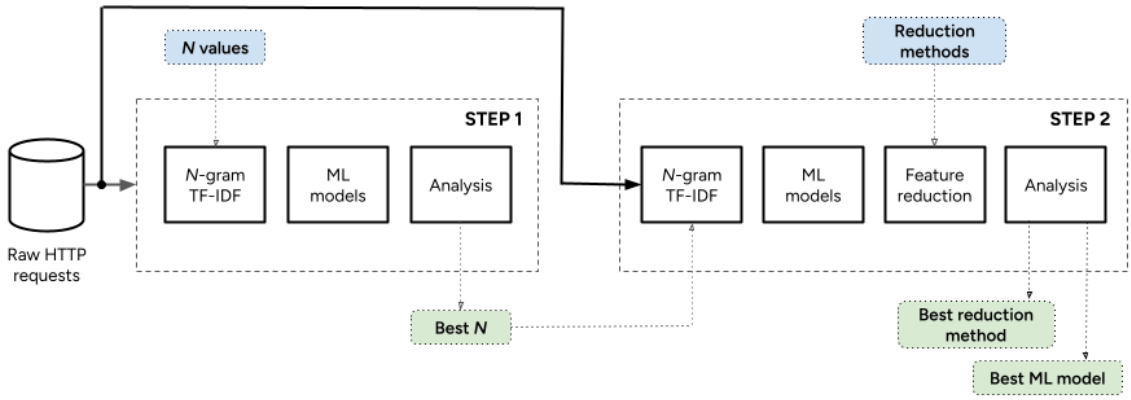


**Figure 1.** Proposed framework steps.

### 3.1. *N*-gram and TF-IDF

$N$-gram is a natural language processing method that creates a sequence of 'grams', parsing the input text using a sliding window of size $N$. The output of $N$-gram is a set of $N$-sized symbols representing the parsed input text. The $N$-gram output creates a set of unique terms that describe the TF-IDF vocabulary. The vocabulary consists of the set of unique terms observed in the training requests and defines the features of the TF-IDF matrix, where each line represents one HTTP request. Figure 2 shows how $N$-gram output defines the TF-IDF features. In this example, three words are provided as input to the N-gram method, which is configured to generate unigrams (1-grams). The output of the N-gram process consists of a set of tokens corresponding to the input words. Subsequently, the vocabulary definition step extracts a set of unique tokens from the N-gram output, forming the vocabulary employed by the TF-IDF method. This TF-IDF vocabulary determines the columns of the resulting TF-IDF matrix.

Two factors must be calculated to generate the TF-IDF matrix. The TF factor indicates the number of occurrences of a given term for one HTTP request. The IDF factor indicates the inverse frequency of a given term for all HTTP requests in the training dataset and is calculated as

$$i(t) = \log\left(\frac{1+R}{1+DF(t)}\right) + 1\,, \tag{1}$$

where $R$ is the number of rows in the TF-IDF matrix (number of requests in the training set), and $DF(t)$ represents the document frequency of $t$-th term (number of requests from the training set with, at least, one occurrence of $t$-th term). Having both TF and IDF factors, the TF-IDF matrix can be generated, as seen in the example scenario shown by Figure 3. Observing the calculation of the values related to the term $W$, it is initially observed that it appears once in the document 'WAF', once in the document 'WEB', and once in the document 'FIREWALL'. For this reason, the term has a TF value of 1.

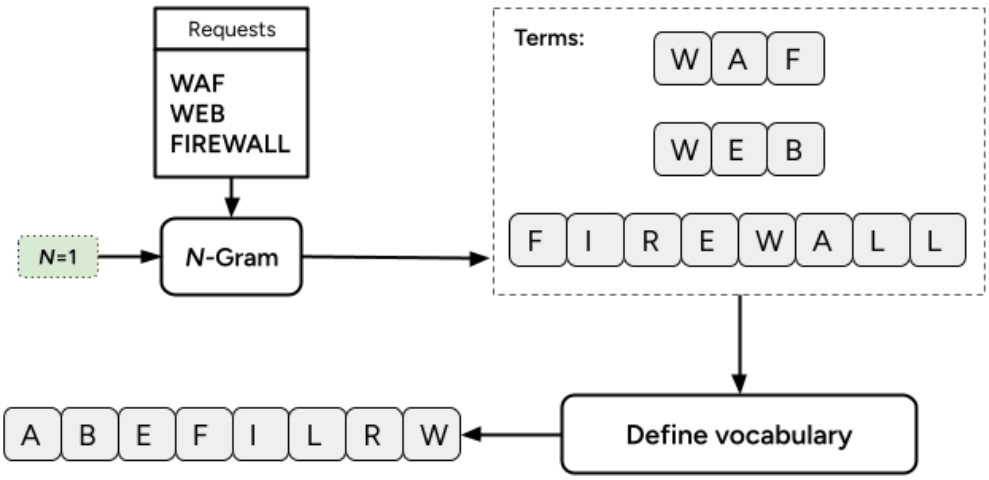


**Figure 2.** Vocabulary definition with $N = 1$. The unique subset of grams generated by $N$-gram defines the vocabulary for TF-IDF.

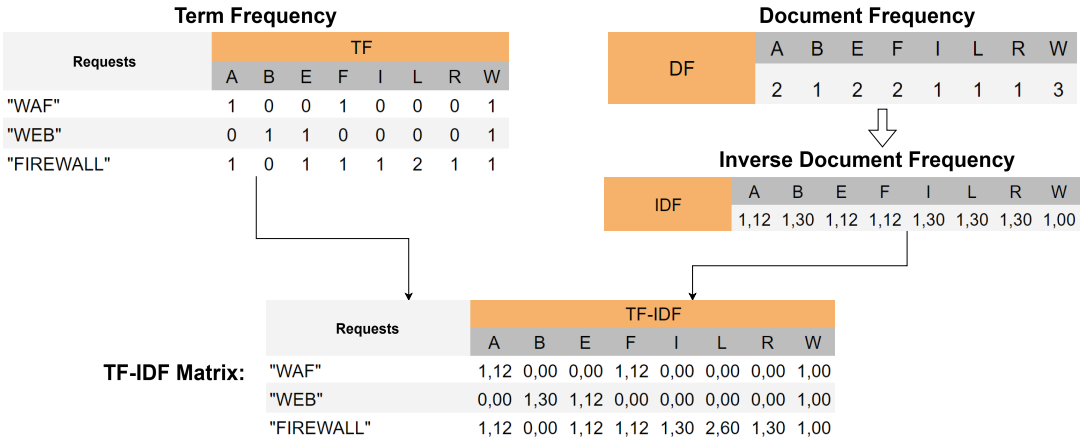


**Figure 3.** Example of TF-IDF matrix generation.

Since it appears in all three documents, the term $W$ also has DF = 3, and applying the IDF function results in an IDF value of 1. The TF-IDF value for the term $W$ in each document is obtained by multiplying the IDF value by each recorded TF value. Since the term $W$ occurs the same number of times in all documents, the TF-IDF distribution for this term (column $W$ in the TF-IDF table) shows no variation. It would not contribute to differentiating the analyzed documents.

### 3.2. Feature Reduction

Feature reduction involves selecting or extracting a smaller set of features from a more extensive multidimensional dataset, reducing the amount of resources required to process the database. This approach is helpful to minimize the adverse effects of high $N$-gram values. The feature reduction methods are categorized considering how they reduce the number of features. Feature selection involves selecting a subset of features from the original set and discarding the non-selected features. In contrast, feature extraction or feature projection transforms the original subset of features into a smaller subset using linear or

non-linear techniques. All feature reduction techniques require more time and resources from the pre-processing pipeline to reduce the dimension of the vectorized requests. Also, all feature reduction techniques have some degree of information loss during the reduction process, which can affect the ML model's classification. Therefore, analyzing feature reduction methods is also crucial for improving WAF latency while maintaining the ML model's performance.

Feature selection methods can be divided into three subcategories: filter methods, wrapper methods, and embedded methods. The filter methods calculate the correlation between each feature and the target classes using statistical tests like Chi-square and mutual information (MI). Filter methods are usually faster and less susceptible to overfitting than wrapper and embedded methods [Hamon 2013], and also provide a subset of features independent of the ML model [Guyon and Elisseeff 2003]. Wrapper methods select a subset of features and evaluate the prediction performance of an ML model trained using the selected subset. This process is executed multiple times, searching for an optimal subset of features that maximizes the prediction performance, but this process can become intractable for large sets of features [Guyon and Elisseeff 2003]. Embedded methods utilize ML models that incorporate feature selection into the learning process. This strategy is more efficient than wrapper methods, but it's intrinsically associated with the ML model and still less efficient than filter methods for large sets of features [Guyon and Elisseeff 2003].

Considering the limitations of wrapper and embedded methods, the authors opted to analyze the performance of two filter methods, the Chi-square and mutual information (MI) algorithms, which were also analyzed in [Babiker et al. 2019]. The Chi-square method applies the Chi-square test to obtain the features most correlated with the target class. In contrast, the MI method applies mutual information theory to collect the features that bring more information about the target class.

On the other hand, feature extraction methods project the original database with dimensionality $D$ into a new dataset of dimensionality $d$, where $(d < D)$ and usually $(d << D)$ [van der Maaten et al. 2007]. These methods have a best effort approach to reduce the information loss during the reduction process [van der Maaten et al. 2007]; therefore, extraction methods tend to preserve more information than selection methods. However, extraction methods are more complex to train and fine-tune and consume more time and resources than filter methods [Guyon and Elisseeff 2003]. To add an extraction method to the pool of reduction algorithms analyzed by the proposed framework, the authors selected the classical Principal Component Analysis (PCA) method, which has the advantages of extraction methods and has fewer drawbacks compared to most other extraction methods.

The PCA method reduces the number of features by projecting the database into a new set of linearly independent features. Comparing PCA with non-linear extraction methods, such as tSNE, UMAP, and autoencoders, PCA is limited to linear correlations. However, the combination of $N$-gram and TF-IDF methods at the character level yields multiple linearly dependent features, which makes PCA a suitable method for this scenario. Also, PCA is considerably faster than other nonlinear methods and has fewer parameters to be configured [van der Maaten et al. 2007].

## 4. Results and Discussion

This work uses the proposed framework to analyze the processing pipeline using four ML models: DT, RF, logistic regression (LR), and linear support vector machine (SVM). These models were extensively analyzed in the literature for web attack detection[Babiker et al. 2019, Dong and Li 2024, Torrano-Gimenez et al. 2015], obtaining great results with better latency than more complex models. The accuracy and F1-score were chosen as metrics for comparing and evaluating the performance of the ML models. The training time and classification time were analyzed to investigate the model latency. The classification time considers the average time for the model to classify each request on the test set. For step one of the framework, the number of features was also considered to select the best value of $N$.

We evaluated the framework's performance in detecting malicious payloads using HTTP requests from the CSIC 2010 HTTP dataset [Giménez et al. 2010]. This dataset contains 25,000 malicious and 36,000 legitimate requests, containing all the elements of an HTTP request: method, uniform resource identifier, protocol version, headers, and body. This dataset allows researchers to train ML models on complete HTTP requests. The dataset was split into training, validation, and test sets, with proportions of 60%, 20%, and 20%, respectively. Cross-validation was performed on the training partition to determine the best hyperparameters for the models in each experiment. The pipeline's hyperparameters are defined by observing the model results on the validation set, and the study concludes by evaluating performance using the best hyperparameters on the test partition. Table 1 shows the hyperparameters of the models evaluated in this study.

### 4.1. Step 1: $N$ value analysis

The first step of the proposed framework analyzes the impact of $N$ value on the ML performance and prediction latency. Also, the number of features created by $N$-gram combined with TF-IDF is also examined. The training partition was used to train the ML models using $N$ values from one to five, while the validation dataset was used to investigate the model's performance and latency.

Table 2 exhibits the accuracy and F1-score for the CSIC validation set considering $N = \{1, 2, 3, 4, 5\}$ and the four ML models mentioned at the beginning of the Section. The performance differences between the values of $N = 1$ and $N = 2$ are significant, and in all evaluated models, a performance improvement was observed for both values of $N$. However, the difference in performance between the models becomes less significant,

**Table 1. Hyperparameters of the models analyzed in this work.**

| Model | Hyperparameter | Values |
|---|---|---|
| Decision Tree | Depth | 10, 20, 40, 60, 80 |
| Logistic Regression | Penalty | L1, L2 |
| | $C$ | 1, 5, 10, 20, 40 |
| Random Forest | # Estimators | 10, 50, 100, 200, 400 |
| Linear SVM | $C$ | 1, 5, 10, 20, 40 |

**Table 2. Accuracy and F1-score for DT, LR, RF, and linear SVM models considering CSIC validation set and $N = \{1, 2, 3, 4, 5\}$.**

| Model | $N$-gram | Accuracy (%) | F1-score (%) |
|---|---|---|---|
| Decision Tree | 1 | 83.8 | 78.6 |
| | 2 | 97.0 | 96.4 |
| | 3 | **97.8** | **97.4** |
| | 4 | 97.3 | 96.7 |
| | 5 | 95.9 | 95.1 |
| Logistic Regression | 1 | 80.4 | 73.8 |
| | 2 | 97.3 | 96.7 |
| | 3 | 98.3 | 97.9 |
| | 4 | **98.8** | **98.5** |
| | 5 | 98.7 | 98.4 |
| Random Forest | 1 | 88.4 | 85.7 |
| | 2 | **97.7** | **97.2** |
| | 3 | 97.5 | 97.0 |
| | 4 | 95.9 | 95.1 |
| | 5 | 52.7 | 54.9 |
| Linear SVM | 1 | 80.5 | 73.7 |
| | 2 | 97.4 | 96.8 |
| | 3 | 98.6 | 98.3 |
| | 4 | 98.9 | 98.7 |
| | 5 | **99.0** | **98.9** |

starting from $N = 2$. The DT and RF models performed best when $N < 3$, while the LR model performed best at $N = 4$, showing only a slight improvement compared to $N = 3$. The linear SVM model demonstrated a steady improvement as the value of $N$ increased, reaching its optimal performance at $N = 5$. In fact, the best linear SVM performance with $N = 5$ outperformed all the other models assessed, with the LR model coming in second.

Figure 4 shows the classification time for the validation set. As can be seen, the linear SVM model has the lowest average classification times, followed closely by LR. The classification time difference between the linear SVM model and the LR and DT models is minor for values of $N = 1$ and $N = 3$ but increases from $N = 4$ onward. The RF model shows slower classification times for requests. Observing the results presented in Figure 4, it becomes evident that the classification times worsen as the value of $N$ in the $N$-gram increases.

Increasing the value of $N$ degrades time performance and requires significantly more memory to store the vectorized requests. Table 3 presents the number of features created by the TF-IDF method using $N$-grams for each value of $N$, along with the memory required to store the entire vectorized dataset. To calculate the amount of memory needed to store the whole vectorized dataset for a given $N$ value, multiply the total number of entries in the dataset by the number of features generated through the $N$-gram technique. Each value in the table occupies eight bytes of memory, which is the default size for the float type in Python.

The number of generated features increases exponentially with the increase of $N$, requiring more memory to store the processed requests and more time to classify

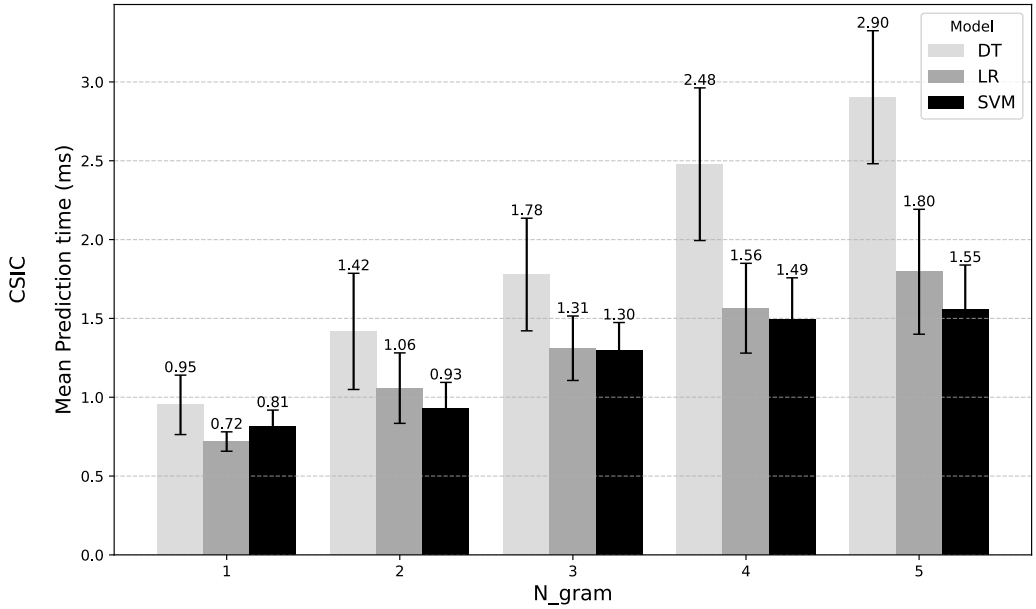


**Figure 4.** Mean prediction times for CSIC validation set. RF model has the higher mean prediction times, with 25.98 ms for $N = 1$, 26.62 ms for $N = 2$, 26.70 ms for $N = 3$, 34.31 ms for $N = 4$ and 48.59 ms for $N = 5$.

them as legitimate or malicious. For $N > 3$, the number of features reaches hundreds of thousands, surpassing the maximum amount of requests in the CSIC dataset and requiring a high memory consumption to store all the processed requests, increasing the resource consumption and cost to maintain the WAF solution. Also, most ML models found the best accuracy and F1-score on $N \leq 4$, except the linear SVM one, as can be seen in Table 2. Based on the results, we assume that $N = 3$ is the optimal value for generating features using $N$-grams and TF-IDF.

### 4.2. Step 2: Feature reduction analysis

After selecting the best $N$ value, this step of the proposed framework analyzes the best thresholds to reduce the number of features using three feature reduction methods: Chi-square, MI, and PCA. Only the model performance using accuracy and F1-score was investigated in this step to determine the best thresholds for each feature reduction method. After defining the best threshold, the performance and latency of the pipelines using feature reduction are compared with the pipeline without feature reduction. The number of

**Table 3. Number of generated features and required memory for the TF-IDF method using $N$-grams.**

| N | CSIC | |
|---|---|---|
| | # Features | Memory (MB) |
| 1 | 52 | 24.2 |
| 2 | 1,896 | 833.3 |
| 3 | 27,438 | 12,480.0 |
| 4 | 179,093 | 81,480.0 |
| 5 | 1,190,752 | 541,760.0 |

features selected or extracted using the reduction methods varies from 500 to 6,000, and these thresholds were determined using feature variance from the training partition.

Table 4 exhibits the number of used features, accuracy, and F1-score for each ML model and feature reduction method evaluated on the CSIC validation dataset, considering $N = 3$. The Chi-Square method showed the best accuracy and F1-score for all the models evaluated. The MI method yielded results closer to those obtained using the Chi-Square method. For most models, the performance difference observed between these methods is less than 0.5%. The linear SVM model performed best using the Chi-Square method to select the 6,000 most important features. The second-best performance belongs to the RF model, using the Chi-Square method to select the 1,000 most important features.

**Table 4. Number of used features, accuracy, and F1-score for each reduction method (Chi-square, mutual information, and PCA) and ML model, considering the CSIC validation set and $N = 3$.**

| Model | Method | # Features | Accuracy (%) | F1-score (%) |
|---|---|---|---|---|
| Decision Tree | Chi-square | 6,000 | **97.7** | **97.2** |
| | Mutual information | 4,000 | 97.5 | 96.9 |
| | PCA | 1,000 | 92.9 | 91.1 |
| Logistic Regression | Chi-square | 6,000 | **97.3** | **96.8** |
| | Mutual information | 6,000 | 97.0 | 96.3 |
| | PCA | 2,000 | 94.9 | 93.7 |
| Random Forest | Chi-square | 1,000 | **98.1** | **97.6** |
| | Mutual information | 4,000 | 98.0 | **97.6** |
| | PCA | 500 | 96.9 | 96.2 |
| Linear SVM | Chi-square | 6,000 | **98.4** | **98.1** |
| | Mutual information | 2,000 | 97.5 | 97.0 |
| | PCA | 2,000 | 96.2 | 95.4 |

**Table 5. Number of used features, accuracy, and F1-score for no feature reduction, as well as each feature reduction method (Chi-square, mutual information, and PCA), and ML model, considering the CSIC test dataset and $N = 3$.**

| Model | Method | # Features | Accuracy (%) | F1-score (%) |
|---|---|---|---|---|
| Decision Tree | No feature reduction | 27,438 | **98.1** | **97.7** |
| | Chi-square | 6,000 | **98.1** | **97.7** |
| | Mutual information | 4,000 | **98.1** | **97.7** |
| | PCA | 1,000 | **98.1** | **97.7** |
| Logistic Regression | No feature reduction | 27,438 | **98.6** | **98.3** |
| | Chi-square | 6,000 | 98.5 | 98.2 |
| | Mutual information | 6,000 | 98.5 | 98.2 |
| | PCA | 2,000 | 98.5 | 98.2 |
| Random Forest | No feature reduction | 27,438 | **97.8** | **97.3** |
| | Chi-square | 1,000 | **97.8** | **97.3** |
| | Mutual information | 4,000 | **97.8** | **97.3** |
| | PCA | 500 | **97.8** | **97.3** |
| Linear SVM | No feature reduction | 27,438 | 98.5 | 98.2 |
| | Chi-square | 6,000 | 98.7 | 98.5 |
| | Mutual information | 2,000 | 98.7 | 98.5 |
| | PCA | 2,000 | **99.0** | **98.8** |

After defining the optimal hyperparameters for each feature reduction method and ML model, a comparison is conducted between the performance and latency of models with and without feature reduction. The models were trained using all the available training and validation datasets. Table 5 shows the number of used features, accuracy, and F1-score for no feature reduction, as well as each feature reduction method (Chi-square, mutual information, and PCA), and ML model, considering the CSIC test dataset. Again, it was assumed that $N = 3$. The DT and RF models demonstrated identical accuracy and F1-score across all feature reduction methods applied and in the test set without any feature reduction. For the linear SVM method, the best performance was achieved using the PCA method, which yielded the highest results among all evaluated models. Among the models, only the LR one showed worse results when feature reduction methods were applied, with the accuracy dropping by 0.1 across all considered reduction methods.

Figure 5 illustrates the total predicted time for the complete pipeline of the evaluated models. This includes the effects of the feature reduction methods and a baseline model that does not use any reduction with $N = 3$. The total predicted time for the entire pipeline is calculated as the sum of the HTTP request processing time and the predicted time taken by the ML model. The Chi-Square and MI methods were the reduction techniques that kept the total prediction time equal to or lower than the baseline's time, except for the RF model. The PCA method had a longer prediction time than the baseline and other feature reduction methods. The DT model excels among various models with the lowest classification time compared to all reduction feature methods and the baseline.

In terms of model performance, the linear SVM demonstrated the highest accuracy and F1-score at 99.0% and 98.8%, respectively, after applying PCA to reduce the feature space to 2,000 variables. However, the Chi-Square and MI methods yielded better prediction times. Among these, the MI method achieved the lowest prediction latency while maintaining the same number of features as PCA. Although its performance metrics were slightly lower, the accuracy and F1-score were only 0.3% below those achieved with PCA.

The DT model combined with MI exhibited the lowest prediction latency, achieving approximately 50 milliseconds faster classification than the linear SVM model. However, in terms of predictive performance, the linear SVM model outperformed DT, with an accuracy improvement of 0.9% and a 1.1% increase in the F1-score.

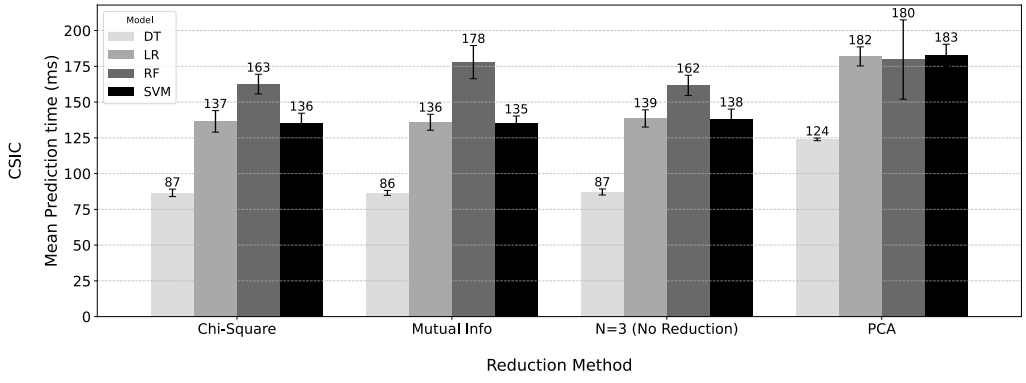


**Figure 5.** Total predicted time for the entire pipeline of the models considering the CSIC test dataset and $N = 3$.

## 5. Conclusion and Future Work

This paper proposes a framework for investigating the optimal value of $N$ in the $N$-gram and TF-IDF combination, as well as identifying the best reduction method to maintain the performance of ML models while reducing latency and resource consumption. The results show that the processing pipelines successfully reduce latency without compromising the performance of the ML models.

The framework organizes hyperparameter tuning for request processing into two distinct phases. The first phase evaluates the impact of the $N$ parameter in the $N$-gram method on performance, classification latency, and memory usage. The second phase examines the combination of feature reduction techniques and ML models, assessing their effectiveness based on performance metrics and classification latency.

The results from the first phase show a significant improvement in model performance when increasing from $N = 1$ to $N = 3$. However, for values of $N > 3$, the gains in performance become marginal. In addition, classification latency and memory consumption continue to rise with higher $N$ values. This increase leads to a point where the negative effects of high dimensionality outweigh the slight performance improvements.

The results from the second phase indicate that the performance of the models, measured in terms of accuracy and F1-score, is similar mainly whether feature reduction is applied or not. The linear SVM model consistently achieves the highest scores among the evaluated methods. Regarding classification latency, the feature selection methods exhibit latency comparable to the baseline (no feature reduction) and perform better than the PCA-based approach. Among all the models assessed, the DT model exhibited the lowest classification latency, followed by LR and linear SVM, both of which demonstrated similar latency across all feature reduction configurations.

For future work, this framework could be expanded to evaluate the effectiveness of alternative tokenization techniques. Emerging preprocessing architectures, such as word embeddings, are quickly developing and becoming increasingly popular. These methods may provide competitive or superior performance compared to traditional approaches like $N$-grams and TF-IDF.

## References

(2025). February 2025 web server survey. Available at `https://www.netcraft.com/blog/february-2025-web-server-survey/` [Accessed at: March 31, 2025].

Babiker, M., Karaarslan, E., and Hoşcan, Y. (2019). A hybrid feature-selection approach for finding the digital evidence of web application attacks. *Turkish Journal of Electrical Engineering and Computer Sciences*, 27:4102–4117.

Bocharov, A. (2025). Making WAF ML models go brrr: Saving decades of processing time. Available at `https://blog.cloudflare.com/making-waf-ai-models-go-brr/` [Accessed at: March 31, 2025].

Dhote, S., Singh, S., Student, A., and Raigar, D. (2024). A comprehensive survey of ml-based wafs with signature and anomaly detection. *Strad Research*, 11:54–60.

Dong, C. and Li, D. (2024). AST-DF: A new webshell detection method based on abstract syntax tree and deep forest. *Electronics*, 13(8):1482.

Giménez, C. T., Villegas, A. P., and Marañón, G. A. (2010). HTTP data set CSIC 2010. *Information Security Institute of CSIC (Spanish Research National Council)*, 64.

Guo, Z., Li, Q., Li, X., Xiao, M., Hu, R., Jiang, Y., Zhao, L., Du, H., and Chen, Q. (2023). SQL injection detection method based on N-gram and TFIDF. In *2023 International Seminar on Computer Science and Engineering Technology (SCSET)*, pages 204–207. IEEE.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182.

Hamon, J. (2013). *Optimisation combinatoire pour la sélection de variables en régression en grande dimension : Application en génétique animale*. Theses, Université des Sciences et Technologie de Lille - Lille I.

Hoang, X. D. and Nguyen, T. H. (2021). Detecting common web attacks based on supervised machine learning using web logs. *Journal of Theoretical and Applied Information Technology*, 99(6).

IŞiker, B. and SoĞukpinar, İ. (2021). Machine learning based web application firewall. In *2021 2nd International Informatics and Software Engineering Conference (IISEC)*, pages 1–6. IEEE.

Kruegel, C. and Vigna, G. (2003). Anomaly detection of web-based attacks. In *Proc. of the 10th ACM Conference on Computer and Communications Security*, pages 251–261.

Kruegel, C., Vigna, G., and Robertson, W. (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738.

Krügel, C., Tøth, T., and Kirda, E. (2002). Service specific anomaly detection for network intrusion detection. In *Proc. of the ACM Symposium on Applied Computing*, pages 201–208.

Kumar, H. et al. (2023). Securing web application using web application firewall (WAF) and machine learning. In *2023 First International Conference on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI)*, pages 1–8. IEEE.

Ramezany, S., Setthawong, R., and Tanprasert, T. (2022). A machine learning-based malicious payload detection and classification framework for new web attacks. In *2022 19th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 1–4.

Rozenfeld, B. (2024). Best WAF solutions in 2024-2025: Real-world comparison. Available at `https://www.openappsec.io/post/best-waf-solutions-in-2024-2025-real-world-comparison` [Accessed at: March 31, 2025].

Song, Y., Keromytis, A. D., and Stolfo, S. (2009). Spectrogram: A mixture-of-Markov-chains model for anomaly detection in web traffic. In *Proc. of the 16th Annual Network and Distributed System Security (NDSS) Symposium*, pages 1–15.

Sureda Riera, T., Bermejo Higuera, J.-R., Bermejo Higuera, J., Martínez Herraiz, J.-J., and Sicilia Montalvo, J.-A. (2020). Prevention and fighting against web attacks through anomaly detection technology: A systematic review. *Sustainability*, 12(12).

Torrano-Gimenez, C., Nguyen, H. T., Alvarez, G., and Franke, K. (2015). Combining expert knowledge with automatic feature extraction for reliable web attack detection. *Security and Communication Networks*, 8(16):2750–2767.

van der Maaten, L., Postma, E., and Herik, H. (2007). Dimensionality reduction: A comparative review. *Journal of Machine Learning Research - JMLR*, 10.

Zhang, S., Li, Y., and Jiang, Q. (2023). Feature ratio method: A payload feature extraction and detection approach for SQL injection attacks. In *2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, pages 172–175. IEEE.