# Obfuscation-Resistant Feature Extraction for Macro-based Office Malware Detection

Sergio Imperiale[3,*,†], Marco Morana[1,2,†] and Giuseppe Lo Re[1,2,†]

[1]*Department of Engineering, University of Palermo, Italy*
[2]*Cybersecurity National Lab, CINI - Consorzio Interuniversitario Nazionale per l'Informatica*
[3]*IMT School for Advanced Studies Lucca, Italy*

**Abstract**

Over the years, the widespread adoption of the Microsoft (MS) Office suite as a productivity tool used by millions of users worldwide has attracted the interest of malicious users in exploiting its vulnerabilities. The most known of these concerns documents containing macros, whose construction is becoming increasingly complex in order to avoid detection of malicious, hidden, behaviors. In this context, this paper presents a novel technique that exploits Large Language Models (LLMs) to extract a set of linguistic features that could reveal the presence of malicious code embedded within macros, even in case of obfuscation. The experimental evaluation, conducted on a publicly available dataset of MS Office files, indicates that the proposed system achieves robust detection of obfuscated malicious macros. Moreover, the performances of a lighter, purely statistical, method are also evaluated so as to offer analysts the choice between a high-precision, resource-intensive model, or a more time-efficient alternative.

**Keywords**

Macros Security, Malware Detection, Machine Learning

## 1. Introduction and Related Work

Microsoft (MS) Office documents are among the most widely used and exchanged by users worldwide. It is therefore not surprising that in recent years there has been an increase in the number of cyber attacks that exploit these documents as a vector to transmit malware [1], penetration tools [2], or other threats. In particular, critical documents are those containing Visual Basic for Applications (VBA) macros, i.e., automated routines originally aimed at simplifying the execution of repetitive tasks and the attackers send them to the target as an email attachment, which often looks like an invoice [3] or notice [4]. Indeed, through the use of macros, it is possible to interact with the underlying operating system in order to perform a wide range of potentially dangerous actions, including writing text files, contacting external servers, and executing terminal commands via the ShellExecute function. The binary content of macros is incorporated into the file structure in accordance with the utilized format, i.e., OLE (Object Link and Embedding - Compound Document Format), the established standard in MS Office 97-2003, and OOXML (Office Open XML), the current standard introduced in Office 2007.

Given the implicit and well-known risks that macros pose to user security, anti-malware detectors [5] has gradually incorporated file analysis capabilities to determine the malicious nature of Office documents. In light of this, several approaches have been proposed. For instance, the authors of [6] utilize dynamic analysis to inspect Office files by running them in multiple sandboxes in order to observe the traces of system calls generated during execution. However, it is important to note that this approach is only applicable to OLE files. In [7] an open-source framework that executes VBA macros in a virtualized environment is presented. Secure execution includes monitoring all macro variable values and method calls to reconstruct execution flows and de-obfuscate payloads. This tool, named Oblivion, can automatically *click* dialog boxes that malware inserts to block automatic analyzers, and detects

*Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT*
*Corresponding author.
†These authors contributed equally.
sergio.imperiale@imtlucca.it (S. Imperiale); marco.morana@unipa.it (M. Morana); giuseppe.lore@unipa.it (G. Lo Re)
0009-0005-1276-3587 (S. Imperiale); 0000-0002-5963-6236 (M. Morana); 0000-0002-8217-2230 (G. Lo Re)

suspicious actions such as accessing MS Outlook or saving files. Other strategies involve the analysis of files through the use of Machine Learning (ML) models. For instance, [8] utilizes active learning to conduct static analysis and identify malicious .docx files. However, it is not applicable to the analysis of OLE files because it uses hierarchical structural paths obtained from the XML structure of the files. The authors of [9] propose a classification system for VBA macros based on 15 static features that indicate code obfuscation.

In the context of macros, obfuscation refers to the set of syntactic and semantic transformations applied to code with the aim of reducing its readability and hindering both static analysis and automatic classification. From a technical perspective, these transformations can impact various levels of the program. At the lexical level, obfuscation is characterized by the renaming of variables and functions with high-entropy identifiers, the fragmentation of strings, or the insertion of seemingly irrelevant operations. At the structural level, control flow is often reorganized through non-linear jumps and the introduction of execution paths that are never actually traversed, but alter the complexity of the flow graph, so one of the most used functions in this context is the GoTo function. Data is also often manipulated with alternative encodings, such as base64 or hexadecimal, or through deferred decoding at runtime, in order to hide sensitive information from the most common static analysis techniques.

In particular, in [9], four obfuscation techniques are classified and adapted the characteristics detected from studies on malicious JavaScript code used in PDF file macros. It is evident that this static approach hinders the framework's ability to adapt to emerging obfuscation techniques, potentially leading to obsolescence or anachronism. In [10] is presented a hybrid static feature extractor that includes the extraction of features related to obfuscation techniques and features related to the use of suspicious keywords, which are statistically very common in obfuscation. This indicates that, compared to the use of obfuscation features or keywords alone, the combination increases robustness and speed of evolution. However, the authors do not evaluate embedding obfuscation, which could evade the model. In [11], on the other hand, authors analyzed the use of linguistic models for malware detection, proposing an approach based on Latent Semantic Indexing (LSI) to represent VBA macros. Words are extracted from the VBA code, and an efficient linguistic model is constructed with LSI, reducing the vector size compared to the complete bag-of-words. The LSI model is then employed to generate features that are subsequently fed into an ML classifier. However, the method remains static and does not take into account additional structural information. Additionally, it does not directly address heavily obfuscated macros and one of the authors, in [12], suggest the implementation of more advanced linguistic models, such as SCDV and Doc2Vec. On the other hand, in [13] a hybrid system that exploits two modes of documents: the text of the VBA macro and the embedded deceptive images is presented. For each document, they extract the macro code, processed with a text neural network, and the document image preview, processed with a visual CNN such as MobileNetV3.

More recently, there has been an emergence of new methods based on adversarial machine learning, which has led to the creation of macros designed to evade classifiers based on statistical or machine learning models. In this scenario, the attacker doesn't just complicate the code's structure; instead, introducing perturbations targeted at the distribution of features that the model uses to make predictions or adding innocuous calls, while preserving the malicious behavior of the software [14]. In the scenario depicted so far, many of the existing methods remain static, particularly in the face of sophisticated obfuscation and AML attacks. Furthermore, linguistic methods reliant on a specific set of words, languages, or static bag-of-words can be vulnerable to attacks once the adversary becomes aware of the set of languages or words employed by the model.

To address these challenges, this paper proposes a novel feature-extraction approach that leverages Large Language Models (LLMs) to identify linguistic features capable of capturing relevant information from obfuscated-macros code. The feature extraction process itself is adaptable and can be updated or replaced as LLMs evolve or as new, more robust, techniques become available in response to increasingly sophisticated attacks. The resulting set of features can be seamlessly integrated into any machine learning model, ensuring flexibility and long-term applicability when used to detect whether a VBA code is malicious or not, even if it is opportunely obfuscated.

A series of experiments conducted on a publicly available dataset have demonstrated the efficacy of

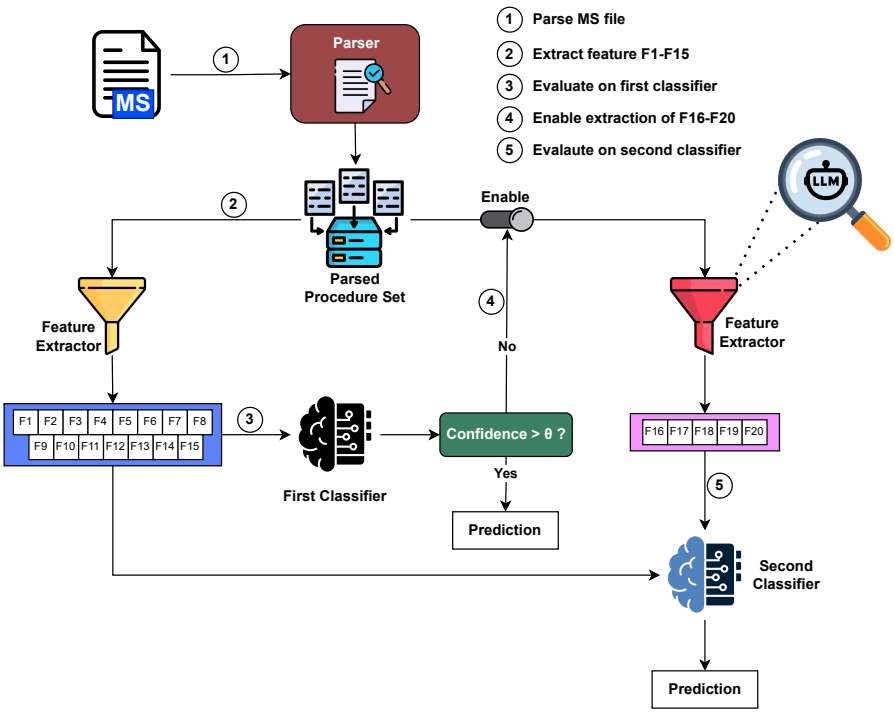


**Figure 1:** System overview.

the proposed strategy, thereby validating its capacity to address the challenges faced by models that has the goal to detect obfuscation or AML attacks.

The remainder of this paper is organized as follows: Section 2 details the architecture and components of the proposed system. Section 3 evaluates the system's performance through a set of experiments and metrics, while Section 4 concludes with insights and future directions.

## 2. System Overview

The general schema behind a malicious macro detection system involves analyzing the content of an Office file to detect the presence of macros, read their content, extract numerical statistical features, and use them as input of a ML-based classifier to assess the maliciousness of the file's content.

The proposed analysis process (see Figure 1) consists of five major components: a *passive* MS Office file parser that retrieves a set of procedures, a simple feature extractor that extracts 15 state-of-the-art features when macros are detected in one of the aforementioned files, a general machine learning model that provides initial feedback on the examined file, and an additional LLM-based feature extractor that is activated if the previous model has made a low-confidence prediction, enabling a cascading approach to be created, which is widely used in other malware analysis sectors such as mobile malware detection [15] and Windows detection [16] or in other security fields, such as intrusion detection systems [17, 18] or in spam detection [19]. This supplementary module extracts an additional set of five features and integrates a LLM. Subsequently, a second machine learning model verifies whether the phenomenon is due to the presence or absence of appropriately obfuscated code using the entire set of features. The flexibility of the approach allows for the use of different ML models; those we used for experimental validation will be presented in Section 3.

Initially, the content of an MS Office file $f$ is read without executing it, thus avoiding the execution of any potentially malicious code. The first module accepts the file as input and extracts the part related to macros dividing the VBA code into procedures and exports both routines and functions to prevent the loss of information from auxiliary functions that could be exploited to manipulate the code.

| Feature | Description |
|---|---|
| F1 | max{length($l_i$)}, $\forall i \in 1 \leq i \leq n$ |
| F2 | max{count_concatenation($l_i$)}, $\forall i \in 1 \leq i \leq n$ |
| F3 | max{count_arithmetic($l_i$)}, $\forall i \in 1 \leq i \leq n$ |
| F4 | max{count_parentheses($l_i$)}, $\forall i \in 1 \leq i \leq n$ |
| F5 | max{count_strings($l_i$)}, $\forall i \in 1 \leq i \leq n$ |
| F6 | max{count_concatenation($p_i$)}, $\forall i \in 1 \leq i \leq m$ |
| F7 | max{count_arithmetic($p_i$)}, $\forall i \in 1 \leq i \leq m$ |
| F8 | max{count_parentheses($p_i$)}, $\forall i \in 1 \leq i \leq m$ |
| F9 | max{count_strings($p_i$)}, $\forall i \in 1 \leq i \leq m$ |
| F10 | max{count_assignment($p_i$)}, $\forall i \in 1 \leq i \leq m$ |
| F11 | length(P) |
| F12 | length(L) |
| F13 | the frequency of function "`CallByName`" |
| F14 | the frequency of each **conversion**, **math** and **string** function |
| F15 | the frequency of each **suspicious** keyword |

**Table 1**
Features extracted by the first feature extractor.

Once the procedures to be analyzed have been obtained, they are fed into a feature extractor, which extracts statistics from both entire procedures and individual lines of code. Specifically, the features that were extracted are those illustrated in [10], as outlined in Table 1. As stated in the reference document, the extracted features are inadequate for conducting a comprehensive analysis of potential inputs obscured by an attacker. However, these features already provide an initial assessment of the possibility of obfuscation. In fact, features F13 and F14 verify the frequency with which some of the statistically most frequent functions appear when it comes to obfuscation. Feature F15, on the other hand, reports the frequency of certain keywords often used in malicious macros, such as the `ShellExecute` function, which are not typically used by legitimate users.

Following the extraction of features, the subsequent step involves the classification of these features by a machine learning model that has been appropriately trained on the same features. As previously mentioned, the problem we are addressing is a binary classification problem but, in this case, it is also important to evaluate the confidence $\varphi$ of the model.

As previously emphasized, the features input to the classifier are not useful for understanding whether there is obfuscation. A particularly careful attacker could appropriately manipulate the text even using only a few of the statistically used functions, as well as producing code through textual attacks [20, 21] in the context of Adversarial Machine Learning (AML) attacks, as discussed in Section 1. This could include inserting non-human-readable code as well as possible comments containing text strings that are not understandable to humans in order to deceive a less sophisticated classifier and that does not belong to any language. For this reason, in [10], comments are removed before extracting features.

In practice, if there is appropriate obfuscation within the strings or variables used, a model that does not focus on obfuscation or the prevention of AML attacks could easily be misled. Indeed, for this reason, the prediction module provides both a feedback on the nature (malicious or not) of the file $f$ under review and the corresponding confidence $\varphi$ level. If the confidence of the model is lower than a threshold $\theta$, the second part of the architecture is activated. This focuses on detecting obfuscated VBA code or code resulting from adversarial machine learning operations by extracting five additional features (F16-F20), which are listed in Table 2.

Specifically, the feature F16 represents the characteristic of a procedure $p_i$ within the entire set of procedures $P$, while F17 represents the characteristic of the entire set. This approach allows for the identification of a procedure that is more anomalous than the others and provides a global representation, as previously implemented in the other procedures presented. The underlying idea behind these two features is to use Large Language Models (LLMs) to give the classifier extra and valuable information about the possible presence of obfuscation, manipulation, or potentially malicious text, such as those typical of adversarial machine learning techniques applied to macros.

A particularly noteworthy property of F16 and F17 is their independence from the language model

| Feature | Description |
|---|---|
| F16 | naturalRatio($P$) |
| F17 | globalNaturalRatio |
| F18 | the frequency of procedures containing comments |
| F19 | max{comment_ratio($p_i$)}, $\forall i \in 1 \leq i \leq m$ |
| F20 | the frequency of characters in comments in all procedures |

**Table 2**
Features extracted by the LLM-based feature extractor.

used to generate preliminary textual representations. In this work, these representations are obtained using the GPT-2 [22] model, which provides latent structures useful for the subsequent extraction phase. To be more specific, GPT-2 is trained on an extensive corpus of approximately 8 million web pages from various sources and in multiple languages, as outlined in [22]. This enables the model to provide high robustness and effective generalization, as the dataset also encompasses a wealth of content related to the realm of programming. However, the GPT-2 model does not directly perform the feature calculation but is used as a provider of information signals. Although it is not the latest release, the choice of GPT-2 is motivated by practical and methodological considerations. First, GPT-2 is publicly available, well-documented, and easily accessible through widely adopted development tools, which ensures reproducibility of the experiments and facilitates comparison with future work. Moreover, its relatively moderate computational requirements make it suitable for integration into scalable malware analysis pipelines, where efficiency and deployment constraints play a critical role. Finally, since the model is used solely to generate intermediate linguistic representations rather than to perform end-to-end reasoning or classification, the benefits of significantly larger and more recent models would likely not justify the additional computational overhead in this specific setting. This approach provides the system with significant methodological flexibility, allowing any model capable of producing structured linguistic representations to be substituted for GPT-2 without altering the logic behind feature derivation. Lighter solutions, such as models based on lexical representations or static structures trained on narrow domains, can be used. If there is a need to capture more complex linguistic phenomena or counter new obfuscation techniques, newer and more powerful models can be used.

In the case of the GPT-2 transformer, the approach adopted to identify possible code manipulations is based on analyzing the perplexity returned by the model. It takes individual lines of code as input, dividing them into tokens, and calculates the relative perplexity value for each. Perplexity is a common metric for evaluating language models that quantifies how plausible an LLM finds a given sequence of tokens. The model's probability assigned to text sequences is derived from the training data. A well-trained model typically assigns higher probabilities to sequences that align with the training data, and lower probabilities to sequences that are anomalous, unusual, or linguistically unnatural. Perplexity is defined as the exponential of the mean cross-entropy between the distribution estimated by the model and the observed token sequence. In practice, given a set of tokens $t_1, t_2, \ldots, t_n$, the model estimates the probability of each token, conditioned on the previous ones, $P(t_i|t_1, \ldots, t_{i-1})$. As outlined in [23], the perplexity is determined as follows:

$$PPL = \exp\Big( -\frac{1}{n}\sum_{i=1}^{n} logP(t_i|t_{j<i})\Big), \tag{1}$$

This formulation indicates that a low perplexity value suggests that the sequence is *expected* or, in any case, highly compatible with the model's knowledge. Conversely, a high value indicates that the model struggles to predict these tokens, interpreting the sequence as anomalous or unusual. When applied to the context of VBA code, perplexity can serve as an indicator of its linguistic consistency. Code that has been manipulated, obfuscated, or artificially generated tends to exhibit less predictable patterns and, as a result, higher perplexity. Conversely, code produced by human developers who adhere to standard conventions is more likely to be recognized by the model.

Conversely, F18, F19 and F20 are designed to address the presence of comments. This is particularly

relevant since many adversarial techniques aim to introduce large amounts of text into comments to artificially alter the statistics used by the classifier. By inserting non-operational text portions, the attacker can modify the distribution of words or the overall length of the document [24]. This is done in an attempt to shift the model's prediction towards erroneous classes. Specifically, F18 represents the percentage of procedures containing comments, F19 represents the maximum ratio among all procedures between commented characters and the characters of the entire procedure, and F20 is the percentage of comments among all procedures. In order to obtain the most accurate proportion possible, keywords were not removed from the analysed code when calculating these three features. The features in question therefore allow this imbalance to be measured quantitatively and anomalies that are difficult to attribute to legitimate behavior to be detected.

As previously stated, this second module is activated depending on the confidence value $\varphi$ value. This choice is justified by the higher computational cost introduced by the extraction of features F16-F17, which suggests to use them only if necessary.

The whole set of features is then used by a binary classifier that has the same characteristics and hyperparameters as the first and will classify the code under analysis as malicious or not.

## 3. Experimental Evaluation

All experiments are conducted using an Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz in an Ubuntu VM with 16 CPU cores and 24GB of RAM and using the [25] dataset, which is the same one used in [10]. The dataset was originally comprised of $2,968$ benign Office documents and $15,571$ malicious documents, which were collected with the aim of studying the behaviour and characteristics of files containing potentially dangerous VBA macros.

Oletools, an open-source toolkit written in Python and widely used in the field of cybersecurity for the analysis of OLE files and Office formats, was used to extract macros from Office documents. Oletools includes several specialised modules; among these, olevba is dedicated to extracting and analysing VBA macros contained in Microsoft Office documents (such as .doc, .docm, .xls, .xlsm, .pptm files, and other OLE- or OpenXML-based formats). This module enables the automatic detection of VBA macros within files, along with the extraction of their source code. It also provides integrated heuristics for the analysis of suspicious patterns, such as potentially dangerous functions, obfuscated strings, and calls to system components. Furthermore, it can detect common obfuscation techniques and malicious behaviour. The use of Oletools has proven advantageous because the toolkit is specifically designed to leverage the internal structures of Office formats and provides a reliable and automatable interface for parsing large volumes of documents.

However, during the extraction phase, the tool was unable to recover macros from $168$ malicious documents in the dataset. There are a number of potential reasons for this issue, which could include corrupted files, non-supported formats, particularly complex obfuscations, or macros integrated using non-standard structures. Furthermore, it was determined that certain documents within the dataset were missing macros and thus were not suitable for analysis and were subsequently removed.

Following the cleaning and filtering operations, the final dataset used in the experiments consists of $1,877$ benign documents and $14,754$ malicious documents, i.e. only those from which it was possible to successfully extract at least one VBA macro using Oletools.

### 3.1. Evaluation Strategy

In order to provide a baseline for the proposed extraction method, a lightweight algorithm was designed to extract features F16 and F17 relying solely on statistical measures, without therefore employing GPT-2. This procedure is described in Algorithm 1, which in mainly aimed at estimating the proportion of natural text in the VBA code of a document.

This is achieved by analyzing each procedure extracted from the parser. In particular, for each of them, the keywords specific to the VBA language are eliminated, as they do not represent natural text

**Algorithm 1:** Calculation of natural text percentage.

**Input :** $P$: set of procedures detected by the VBA parser;
$K$: set of VBA keywords;
$N$: set of languages;
$\eta$ , $\lambda$ , $\upsilon$ : hyperparameters.
**Output:** NaturalRatio: dictionary containing the percentage of natural text for each procedure;
TotalNaturalRatio: overall natural text percentage.

$\Gamma \leftarrow 0$ ; $\Xi \leftarrow 0$
**for** $\forall\ P_i \in P$ **do**
  $\mathrm{T}_N \leftarrow []$
  **if** $P_i\ \cap\ K \neq \varnothing$ **then**
    $P_i := P_i \setminus K$
  **end**
  S, I ← extract(line) ▷ Extracts strings and identifiers from a given line of code
  T ← tokenize(S ∪ I)
  **for** $\forall\ t \in T$ **do**
    **if** $\exists N_j \in N \mid t \in N_j$ **then**
      $\mathrm{T}_N$.append($t$)
    **end**
    **else**
      Calculate $H(t)$, $L(t)$, $V(t)$
      **if** $H(t) \leq \eta$ and $L(t) \geq \lambda$ and $V(t) \geq \upsilon$ **then**
        $\mathrm{T}_N$.append($t$)
      **end**
    **end**
  **end**
  NaturalRatio[$P_i$] ← count_char($\mathrm{T}_N$) / count_char(T)
  $\Gamma \leftarrow \Gamma +$ count_char($\mathrm{T}_N$)
  $\Xi \leftarrow \Xi +$ count_char(T)
**end**
TotalNaturalRatio ← $\Gamma$ / $\Xi$

but structural elements of the code, and a set of tokens is obtained by extracting strings, comments and identifiers from each line of the procedure.

Tokens are then examined to determine whether they can be considered natural, i.e., if they belong to one of the languages contained within set $N$; otherwise, three metrics are evaluated:

1. **Shannon's entropy [26]** $H(X) = -\sum_i \omega_i \log_2 \omega_i$, where $\omega_i$ is the frequency of each character in the text;
2. **Letter's frequency** $L = \frac{\text{Total number of letters}}{\text{Text length}}$;
3. **Vowel's frequency** $V = \frac{\text{Total number of vowels}}{\text{Text length}}$.

If all three metrics meet the conditions imposed by the hyperparameters $\eta$, $\lambda$, and $\upsilon$, the token is considered consistent with natural text. Once all the natural tokens in the procedure have been identified, the ratio between the number of characters belonging to natural tokens and the total number of characters in the analysed tokens is calculated. Also, the algorithm calculates the overall percentage of natural text in the document is obtained as the ratio between the total natural characters and the total characters examined.

As a result, a local measure for each procedure and an aggregate index that describes the overall level of linguistic naturalness of the VBA content in the file is obtained. In particular, feature F16 represents the minimum natural text frequency value among all procedures, while F17 represents the average natural text frequency within the entire set of procedures.

GPT-2 was used in its pre-trained configuration, both for the tokenization phase, which uses Byte-Pair Encoding [27], and for the inference phase, avoiding further fine-tuning procedures that could have modified the original linguistic distribution of the model and $5\%$ of the total $16,631$ available documents were used as a benchmark to calculate the perplexity value on these files. For the remaining $95\%$ of the files, the perplexity, once obtained, was normalized using the z-score, obtained from the mean and standard deviation of this $5\%$ of documents.

In this particular instance, perplexity was calculated on individual lines, as the GPT-2 model utilizes a maximum context of 1024 tokens for all its variants. This indicates that the model can process up to

| Model | Hyperparameters | Accuracy | F1 | FAR |
|---|---|---|---|---|
| **Random Forest** | {n_estimators=100} | .960 | .978 | .237 |
| **KNN** | {n_neighbors=5} | .925 | .959 | .476 |
| **MLP** | {hidden_dim=100, lr=$10^{-5}$, max_iter=$10^5$} | .892 | .942 | .895 |

**Table 3**
Classifiers used to evaluate the architecture and results obtained using only the features F1-F15.

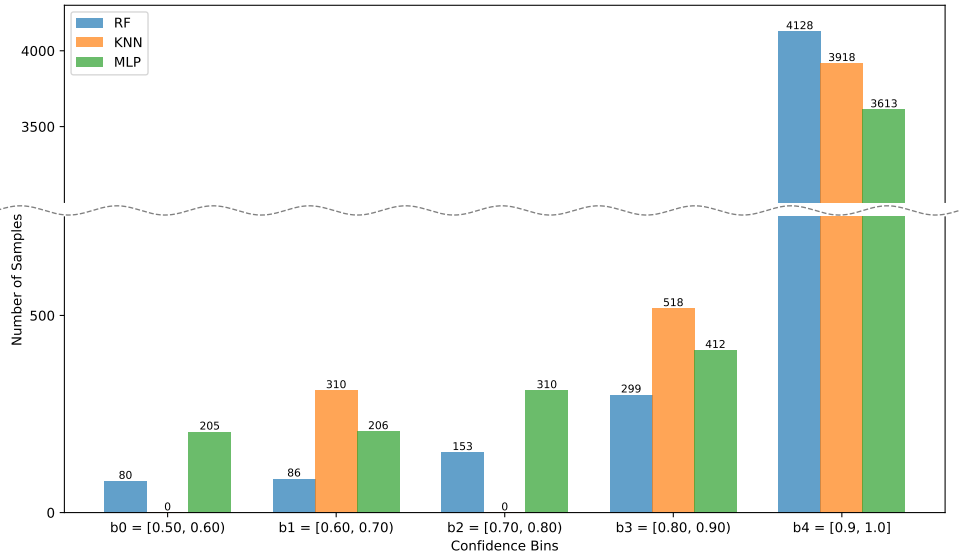


**Figure 2:** Classification confidence considering the features F1-F15.

1024 tokens during inference or perplexity calculation, which may be insufficient for procedures with extensive coding. Therefore, when referring to $\texttt{perplexity}(P)$, we mean the average value among all the perplexity values of the individual lines of code given as input to the model. On the other hand, $\texttt{perplexity}(M)$ refers to the average value among all the $P$ procedures that make up an entire macro file $M$.

## 3.2. Results

In this section the results of the tests conducted on the remaining 95% samples of the dataset after the extraction of the perplexity to obtain z-scores (see Section 3.1) are shown. These samples were split into training and testing sets, with a 70-30 proportion.

The models used for the experimental evaluation, and the corresponding results in terms of accuracy, F1, and FAR achieved using only the features F1-F15 are reported in Table 3. In order to better assess the necessity of the thresholding mechanism, these results were further examined by analyzing the level of confidence with which the different classifiers scored the input samples. Figure 2 shows the bins defined by the different threshold intervals considered and the corresponding number of samples falling within each of them. As can be seen, several samples are classified with inadequate confidence levels, and it is on these that the analysis of the second classification stage will focus. Moreover, the results in terms of accuracy, F1 and FAR of each bin are detailed in Table 4.

In general, the proposed approach ensures that only documents classified with a confidence lower than $\theta$ undergo extraction of additional features and second stage classification. Following the results obtained so far, we investigated the extraction of the more computationally expensive features considering three different thresholds, namely $\theta = \{0.6, 0.7, 0.8\}$. Moreover, four different combinations of features (the sets F1-F17, F1-F20, F16-F17, and F16-F20) were extracted using both the statistical approach described in Algorithm 1 and GPT-2, and tested separately.

Performance of the second classification stage with different value of $\theta$ are shown in Table 5. It is worth

| Classifier | Features | Bin | Accuracy | F1 | FAR |
|---|---|---|---|---|---|
| **RF** | F1-F15 | $b0$ | .487 | .495 | .48 |
| | | $b1$ | .593 | .589 | .547 |
| | | $b2$ | .712 | .589 | .456 |
| | | $b3$ | .873 | .869 | .273 |
| | | $b4$ | .992 | .992 | .062 |
| **KNN** | F1-F15 | $b1$ | .593 | .579 | .635 |
| | | $b3$ | .759 | .738 | .589 |
| | | $b4$ | .973 | .971 | .329 |
| **MLP** | F1-F15 | $b0$ | .590 | .587 | .537 |
| | | $b1$ | .646 | .617 | .671 |
| | | $b2$ | .858 | .856 | .255 |
| | | $b3$ | .888 | .886 | .326 |
| | | $b4$ | .977 | .976 | .408 |

**Table 4**
Performance of each probability bin.

highlighting that the results reported for the different threshold values are obtained by aggregating samples from multiple bins. Specifically, for $\theta = 0.6$, only samples belonging to bin $b0$ were considered. For $\theta = 0.7$, the analysis included samples from both bins $b0$ and $b1$. Similarly, for $\theta = 0.8$, all samples assigned to bins $b0$, $b1$, and $b2$ were taken into account.

Results are summarized in Table 5. Starting from Random Forest, when $\theta$ is set to $0.6$, we can see that utilizing solely the features extracted in the second step, i.e., F16-F20, leads to worse performance than including also those extracted early. The same applies to the exclusive use of features F16 and F17, i.e., those extracted from linguistic models. It is interesting to observe that performance decrease somewhat overall when considering $\theta = 0.7$, although in this case samples with slightly higher confidence values (i.e., bin $b1$) are included. This behavior can be explained by looking at the results for $\theta = 0.8$, where despite an overall improvement in performance, the metrics are not high enough to suggest that features F16-F20 are useful when considered alone. However, when these features are combined with those extracted in the first phase, we can see a marked improvement in performance across all three theta values. As expected from the proposed configuration, the best performance is obtained when all the new extracted features are added to the initial set F1-F15, and the results obtained can justify the cost of extracting the new features.

Moving to KNN, it is important to note that this model cannot be evaluated at $\theta = 0.6$ since, as illustrated in Figure 2, there are no predicted samples in bin $b0$. As observed for RF, the performance of KNN also improves when initial features are augmented with F16 and F17, yielding results comparable to the configuration that incorporates F18-F20. This set still surpasses the base KNN model on bin $b1$, reinforcing the efficacy of incorporating additional features to enhance model performance. It is evident that the performance of the model for $\theta = 0.7$ is equivalent to that for $\theta = 0.8$, as there are no samples in bin $b2$ (see Figure 2).

Similar considerations can be made on MLP network; in this case, the use of features F16 and F17 alone, or even features F18-F20, is not sufficient to enhance the model's performance on critical bins, as evidenced by a decline in performance for all three theta values examined. As was the case for the first two models, however, the best performance is obtained when features F16 and F17 are combined with the features extracted in the first phase, and even better performance is obtained when features F18-F20 are also included.

As indicated by the bold values in Table 5 , which refer to the performance comparison between the GPT-2 model and the statistical algorithm for each feature set examined, it is noteworthy that the GPT-2 model consistently outperformed the statistical algorithm in terms of accuracy, F1 score, and FAR, highlighting the efficacy of use more complex approaches for the detection of anomalous patterns or natural text. This enhancement in performance comes at the expense of increased computational cost and longer extraction times. In our configuration, feature extraction with Algorithm 1 took an average of $1.215$ seconds, while extraction with GPT-2 took $9.647$ seconds.

| | | | $\theta = 0.6$ | | | $\theta = 0.7$ | | | $\theta = 0.8$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | Features | Approach | Acc | F1 | FAR | Acc | F1 | FAR | Acc | F1 | FAR |
| RF | F1-F20 | Statistical | .625 | .571 | .400 | .566 | .561 | .534 | .627 | .661 | .509 |
| | | GPT-2 | **.712** | **.657** | **.300** | **.596** | **.594** | **.515** | **.640** | **.674** | **.503** |
| | F1-F17 | Statistical | .588 | .560 | .480 | .596 | .568 | .466 | .643 | .669 | .474 |
| | | GPT-2 | **.638** | **.623** | **.460** | **.645** | **.624** | **.437** | **.671** | **.696** | **.450** |
| | F16-F20 | Statistical | .400 | .547 | .940 | .392 | .543 | .951 | .473 | .632 | .959 |
| | | GPT-2 | **.412** | **.561** | **.940** | **.404** | **.556** | **.951** | **.480** | **.638** | **.959** |
| | F16-F17 | Statistical | .400 | .547 | .940 | .392 | .543 | .951 | .473 | .632 | .959 |
| | | GPT-2 | **.412** | **.561** | .940 | **.404** | **.556** | .951 | **.483** | **.639** | **.953** |
| KNN | F1-F20 | Statistical | *n.a.* | | | .629 | **.743** | .770 | .629 | **.743** | .770 |
| | | GPT-2 | | | | **.632** | .709 | **.548** | **.632** | .709 | **.548** |
| | F1-F17 | Statistical | *n.a.* | | | .629 | **.743** | .770 | .629 | **.743** | .770 |
| | | GPT-2 | | | | **.635** | .710 | **.532** | **.635** | .710 | **.532** |
| | F16-F20 | Statistical | *n.a.* | | | .584 | .737 | 1.000 | .584 | .737 | 1.000 |
| | | GPT-2 | | | | **.594** | **.745** | **1.000** | **.594** | **.745** | **1.000** |
| | F16-F17 | Statistical | *n.a.* | | | .587 | .739 | .992 | .587 | .739 | .992 |
| | | GPT-2 | | | | **.594** | **.745** | **1.000** | **.594** | **.745** | **1.000** |
| MLP | F1-F20 | Statistical | .715 | .465 | .168 | .724 | .622 | .207 | .756 | .755 | .253 |
| | | GPT-2 | **.775** | **.559** | **.111** | **.770** | **.673** | **.149** | **.788** | **.780** | **.192** |
| | F1-F17 | Statistical | .715 | .525 | .216 | .699 | .562 | **.191** | .735 | .722 | **.233** |
| | | GPT-2 | **.727** | **.425** | **.121** | **.706** | **.632** | .274 | **.744** | **.754** | .311 |
| | F16-F20 | Statistical | **.294** | **.450** | **.992** | **.378** | **.546** | **.993** | **.493** | **.659** | **.994** |
| | | GPT-2 | .290 | .448 | .997 | .376 | .545 | .998 | .488 | .656 | .998 |
| | F16-F17 | Statistical | .288 | .448 | 1.000 | .374 | .544 | 1.000 | .490 | .657 | 1.000 |
| | | GPT-2 | **.290** | **.448** | **.997** | **.376** | **.545** | **.998** | **.491** | **.658** | **.998** |

**Table 5**
Evaluation of each low confidence set for all three different $\theta$ values.

## 4. Conclusions and Future work

In this paper, we have presented a framework for detecting malicious MS Office macros that have been obfuscated by an attacker. The framework's objective is to extract statistical features from the file under analysis. If the initial analysis on a reduced set of features does not yield a prediction with high confidence, further feature extraction is performed using a linguistic approach based on an LLM. This approach allows for a higher-level, more detailed investigation of the sample under analysis, in order to make a more precise prediction. As outlined in Section 3, this approach offers a significant advantage in terms of scalability. This benefit is derived from the modular design of the proposed architecture. Depending on the intended use case, such as fast, large-scale classification of executable files [28], or on the available computational resources, the LLM used to compute two of the five additional features for low-confidence samples can be replaced with a less computationally intensive linguistic model or algorithm. This design choice makes the framework suitable for deployment in high-throughput malware analysis pipelines. Experiments conducted on a public dataset confirmed the effectiveness of the proposal, highlighting how the model's performance varies depending on the extraction algorithm used.

Future work will consider adversarial machine learning scenarios [29], where attackers deliberately craft samples to evade detection. Moreover, the evaluation of the framework could be extended considering additional MS Office file datasets collected at different time intervals to assess whether, in the face of more sophisticated threats or new obfuscation techniques, changing the linguistic model responsible for feature extraction may impact on the performances. In this context, evaluating datasets gathered over time will also allow us to explicitly analyze the presence of concept drift [30] and its impact on the model's performance.

## Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

## References


[1] ESET, Eset threat report (h1 2024), https://www.welivesecurity.com/en/eset-research/eset-threat-report-h1-2024/, 2024.

[2] R. Future, Malware and vulnerability trends report (h1 2024), https://www.recordedfuture.com/research/h1-2024-malware-and-vulnerability-trends-report, 2024.

[3] V. Koutsokostas, N. Lykousas, T. Apostolopoulos, G. Orazi, A. Ghosal, F. Casino, M. Conti, C. Patsakis, Invoice# 31415 attached: Automated analysis of malicious microsoft office documents, Computers & Security 114 (2022) 102582.

[4] M. Gutfleisch, M. Peiffer, S. Erk, M. A. Sasse, Microsoft office macro warnings: a design comedy of errors with tragic security consequences, in: Proceedings of the 2021 European Symposium on Usable Security, 2021, pp. 9–22.

[5] A. Augello, A. De Paola, G. Lo Re, Hybrid multilevel detection of mobile devices malware under concept drift, Journal of Network and Systems Management 33 (2025). doi:10.1007/s10922-025-09906-3.

[6] T. Schreck, S. Berger, J. Göbel, Bissam: Automatic vulnerability identification of office documents, in: International conference on detection of intrusions and malware, and vulnerability assessment, Springer, 2012, pp. 204–213.

[7] A. Sanna, F. Cara, D. Maiorca, G. Giacinto, Oblivion: an open-source system for large-scale analysis of macro-based office malware, Journal of Computer Virology and Hacking Techniques 20 (2024) 783–802.

[8] N. Nissim, A. Cohen, Y. Elovici, Aldocx: detection of unknown malicious microsoft office documents using designated active learning methods based on new structural feature extraction methodology, IEEE Transactions on Information Forensics and Security 12 (2016) 631–646.

[9] S. Kim, S. Hong, J. Oh, H. Lee, Obfuscated vba macro detection using machine learning, in: 2018 48th annual ieee/ifip international conference on dependable systems and networks (dsn), IEEE, 2018, pp. 490–501.

[10] X. Chen, W. Wang, W. Han, Malicious office macro detection: Combined features with obfuscation and suspicious keywords, Applied Sciences 13 (2023) 12101.

[11] M. Mimura, T. Ohminami, Towards efficient detection of malicious vba macros with lsi, in: International Workshop on Security, Springer, 2019, pp. 168–185.

[12] M. Mimura, Using sparse composite document vectors to classify vba macros, in: International Conference on Network and System Security, Springer, 2019, pp. 714–720.

[13] J. Yan, M. Wan, X. Jia, L. Ying, P. Su, Z. Wang, Ditdetector: Bimodal learning based on deceptive image and text for macro malware detection, in: Proceedings of the 38th Annual Computer Security Applications Conference, 2022, pp. 227–239.

[14] S. Gaglio, A. Giammanco, G. Lo Re, M. Morana, Adversarial machine learning in e-health: Attacking a smart prescription system, in: International Conference of the Italian Association for Artificial Intelligence, Springer, 2021, pp. 490–502.

[15] A. Augello, A. De Paola, M. Jestin, G. Lo Re, A stackelberg approach to federated learning for malware detection, in: CEUR Workshop Proceedings, volume 3962, 2025.

[16] O. Sharma, A. Sharma, A. Kalia, Windows and iot malware visualization and classification with deep cnn and xception cnn using markov images, Journal of Intelligent Information Systems 60 (2023) 349–375.

[17] V. Agate, D. Felice Maria, A. De Paola, P. Ferraro, G. L. Re, M. Morana, A behavior-based intrusion detection system using ensemble learning techniques., in: ITASEC, 2022, pp. 207–218.

[18] M. K. U. Ahamed, A. Karim, Cascaded intrusion detection system using machine learning, Systems and Soft Computing 7 (2025) 200182.

[19] F. Batool, G. Lo Re, M. Morana, G. Rizzo, Blocking fake news propagation exploiting osns users interests and connections, in: 2025 5th Intelligent Cybersecurity Conference (ICSC), 2025, pp. 49–54. doi:10.1109/ICSC65596.2025.11139926.

[20] H. Zhang, S. Lu, Z. Li, Z. Jin, L. Ma, Y. Liu, G. Li, Codebert-attack: Adversarial attack against source

code deep learning models via pre-trained model, Journal of Software: Evolution and Process 36 (2024) e2571.
[21] F. Batool, F. Canino, F. Concone, G. Lo Re, Morana, A black-box adversarial attack on fake news detection systems, in: CEUR Workshop Proceedings, volume 3731, IT, 2024.
[22] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI blog 1 (2019) 9.
[23] Y. Wu, Y. Bizzoni, P. Moreira, K. Nielbo, Perplexing canon: A study on gpt-based perplexity of canonical and non-canonical literary works, in: Proceedings of the 8th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL 2024), 2024, pp. 172–184.
[24] F. Concone, S. Gaglio, A. Giammanco, G. L. Re, M. Morana, Adverspam: Adversarial spam account manipulation in online social networks, ACM Trans. Priv. Secur. 27 (2024). URL: https://doi.org/10.1145/3643563. doi:10.1145/3643563.
[25] V. Koutsokostas, N. Lykousas, G. Orazi, T. Apostolopoulos, A. Ghosal, F. Casino, M. Conti, C. Patsakis, Malicious ms office documents dataset, Zenodo (2021).
[26] C. E. Shannon, A mathematical theory of communication, The Bell System Technical Journal 27 (1948) 379–423. doi:10.1002/j.1538-7305.1948.tb01338.x.
[27] P. Gage, A new algorithm for data compression, The C Users Journal 12 (1994) 23–38.
[28] A. De Paola, S. Gaglio, G. L. Re, M. Morana, A hybrid system for malware detection on big data, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2018, pp. 45–50. doi:10.1109/INFCOMW.2018.8406963.
[29] B. Fehrman, F. Akowuah, S. Xu, Adversarial attacks in problem space for vba code samples, in: International Symposium on Intelligent Computing and Networking, Springer, 2024, pp. 267–281.
[30] V. Agate, A. De Paola, S. Drago, P. Ferraro, G. Lo Re, Enhancing iot network security with concept drift-aware unsupervised threat detection, in: 2024 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2024.