



Intelligent System for Classification of DGA Botnets Using Machine Learning Algorithms

Ruth Ofoh¹, Ugwu C.², Eke B. O³

¹Department of Computer science, Faculty of Science, University of Port Harcourt. E-mail: ruth.ofoh@gmail.com

² Department of Computer science, Faculty of Science, University of Port Harcourt. Email: Chidiebere.ugwu@uniport.edu.ng

³Department of Computer science, Faculty of Science, University of Port Harcourt. Email batholoyel@gmail.com

ABSTRACT

In this work, the aim is to classify wordlist DGA and distinguish it from other DGA types. Different features were extracted from domain names to represent them for model training. Domains were cleaned and tokenized at character level. Word2Vec embeddings were trained on the domain characters to capture patterns in character sequences. Auxiliary features were also calculated, such as domain length, entropy, digit ratio, vowel ratio, and special character ratio. These features together created a feature space that included both semantic information from Word2Vec and structural information from auxiliary metrics. This helped models to see differences between normal domains and generated domains. Machine learning models were trained on these features. Machine learning models included KNN, SVM, Random Forest, Decision Tree, Gradient Boosting, and Logistic Regression. The models used the Word2Vec embeddings plus auxiliary features. An autoencoder was also trained to learn the latent structure of domains and compute reconstruction error for anomaly detection. This helped in identifying domains with unusual patterns. Evaluation of models showed that the AUC, representing the model's ability to distinguish between legitimate and DGA domains across thresholds, is highest at 0.88 for Random Forest and lowest at 0.82 for both SVM and Decision Tree. This shows the model can learn complex patterns in the domain sequences and generalize well. The system was implemented in a Gradio interface to allow interactive domain testing. The interface allows a user to enter a domain and get the predicted class, auxiliary feature values, probability for each class, wordcloud visualization, and autoencoder loss curve,. The autoencoder threshold provided insight into abnormal domain structure and added explainability. Overall, the combination of Word2Vec embeddings, auxiliary features, and the Machine learning model provided a robust and explainable system for classifying domains.

Keywords- Autoencoder botnet, word2vec, botmaster, machine learning

1.0 INTRODUCTION

A botnet is a network of computers that have been linked together by malware, which are called bots, controlled by an unkind and cruel controller called the botmaster Li et al (2017). Botnet is a network of bots. A bot is a special kind of malware running on an Internet-connected device that can be a computer, a smartphone or an IoT device Alieyan et al (2017). Bots are often created and maintained by hacking groups, called botmasters. When a bot is infected and running on a device, it allows the botmaster to remotely control the device. A botmaster usually uses a control system, called the Command and Control (C&C or CnC) server to control and maintain a botnet. On one side, the botmaster sends commands and code updates to his botnet's C&C server. On the other side, bots in his botnet are equipped with the capability of using communication channels to connect to and receive commands and code updates from the C&C server. Bots can also send their working status to their C&C server.

Bots in a botnet periodically send DNS queries containing the botnet's C&C server name to the local DNS service to find the server's IP address in order to connect to the C&C server. To avoid the C&C server from being blacklisted and then blocked if using a static name and IP address, the botmaster usually uses special techniques such as Domain Generation Algorithm (DGA) to dynamically generate and register domain names for his botnet's C&C server. Bots in the botnet are also equipped with the capability to dynamically generate server's names using the same DGA technique. Therefore, bots can still find the IP address of the C&C server to connect to by sending queries of server names to the local DNS service. Attackers developed DGAs so that malware can quickly generate a list of domains that it can use for the sites that give it instructions and receive information from the malware Since DGA technique has been very popular and then botnets that utilize DGA technique are called DGA botnets.

Despite collaborative efforts by law enforcement agencies and governments, as well as botnet detection approaches widely discussed and developed by numerous researchers, botnets continue to grab top news headlines all over the world. Numerous issues have been brought to light, but they have not been fully addressed, while new challenges keep emerging, stemming from new botnet attacks. Today, the most significant threat on the Internet is botnet (Shinan et al., 2021).

Botnets have been constantly evolving on the global Internet in both scale and sophistication of control techniques (Hoang et al., 2018).

Each botnet member is called a bot. A bot is a malware created by a hacking group (called botmaster) that allows them to control infected computer systems remotely. Bots are different from other forms of malware in that they are highly autonomous and are equipped with the ability to use communication channels to receive commands and code updates from their control system. They can also notify their working status to their control system periodically. The botnet control system, or Command & Control (C & C) servers, is the means by which the botmaster sends commands and code updates to bots.

Today, botnets are considered a major threat facing anyone using the web (Silva et al., 2013). Through using botnets, attackers can gain control of large-scale networks through infecting them. That is done in the aim of achieving specific malicious goals (Bilge et al., 2014).

Botnets have been causing significant losses that have negative impacts on the global economy.

Domain names generated by DGA botnets using mathematical algorithms usually have no meaning and are generally different from legitimate domain names in the order of characters. Approaches based on statistics and machine learning can be used to classify DGA and legitimate domain names. Recent machine learning-based proposals for DGA botnet detection, which achieved relatively high detection accuracy and fairly low false alarm rate. However, these proposals only work well on the domain names generated by character-based DGA botnets. They are unable to detect new families of DGA botnets that are named word-based or dictionary-based DGA botnets. The word-based DGA botnets usually use predefined lists of English words to generate meaningful domain names for their C&C servers, which are highly similar to legitimate domain names.

This research proposes a model that is based on machine learning techniques to effectively detect word-based DGA botnets.

2.0 LITERATURE REVIEW

2.1. Botnet

A botnet is a collection of computers connected to the Internet which have been compromised and are being controlled remotely by an intruder via malicious software called bots (Banerjee & Samantaray 2019). A collection of bots working together for the same bot-master constitutes a botnet. They are network of compromised host devices that are used to carry out malicious activities (Shinan et al., 2021). A botnet is the melding of many threats into one (Schiller et al., 2011). A botnet is a network of computers that have been linked together by malware, which are called bots, controlled by an unkind and cruel controller called the botmaster Li et al. (2017).

Generally, a botnet is a network of Internet-connected devices that have been infected with a special type of malware, called bot (Alieyan et al., 2017; Li et al., 2017). Bots are usually created by hacking groups, called botmasters. A bot running on an Internet connected device allows the botmaster to control the device remotely. The botmaster first sets up and configures the botnet control system, or the Command & Control (C&C) server and then he sends commands and code updates to bots through the C&C server (Alieyan et al., 2017; Li et al., 2017).

A central command and control (C&C) server operated remotely over a network of infected machines (Zeng et al., 2012). Peer-to-peer connectivity and encryption are two further advanced tactics used by cybercriminals to evade cybersecurity professionals' attempts to track down and take down these botnets

2.2 Botnet Life Cycle.

Botnet often follows five stages to accomplish or execute instruction ordered by the botmaster through the C&C channel Umar et al 2021. The stages are shown in figure .1: The Conception Stage, Recruitment Stage, Interaction Stage, Execution Stage, Upgrade and Maintenance Stage

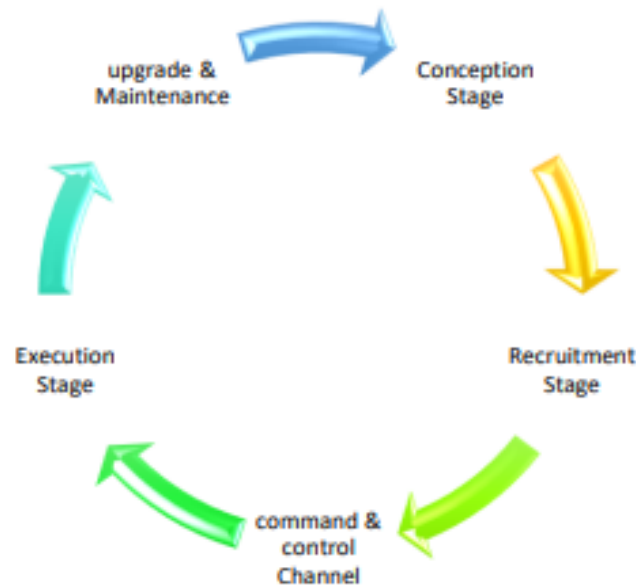


Figure 1: Life Cycle of a typical botnet. (Source: Umar et al 2021)

2.3 Word2vec

Word2vec was developed by Tomáš Mikolov, Kai Chen, Greg Corrado, Ilya Sutskever and Jeff Dean at Google, and published in 2013. July 29, 2013 (Mikolov et al., 2013). They are used to create and train semantic vector spaces, often consisting of several hundred dimensions, based on a corpus of text (Mikolov et al., 2013). In this vector space, each word from the corpus is represented as a vector. Words that share a context are geographically close to each other in this space. The word2vec architecture can be based on two approaches: a continuous bag-of-words, or a continuous skip-gram model (CBOW). The former uses the context to predict the current word, whereas the latter uses the current word to predict surrounding words (Mikolov et al., 2013). Both models have a low computational complexity, so they can easily handle a corpus with a size ranging in the billions of words in a matter of hours.

2.4 Autoencoders (AE)

An autoencoder is a specific type of neural network where the input layer has the same dimension as the output layer. An autoencoder compresses the input into a lower-dimensional representation at the middle hidden layer and then it reconstructs the output from the representation. The middle hidden layer of an autoencoder is also known as the bottleneck layer and its dimension is lower than the input/output layer Catillo et al (2022). Deep learning can be applied to autoencoders. In particular, multiple hidden layers can be used to provide depth: the resulting network is known as deep or stacked autoencoder. Figure 2. shows the representation of an autoencoder with three hidden layers.

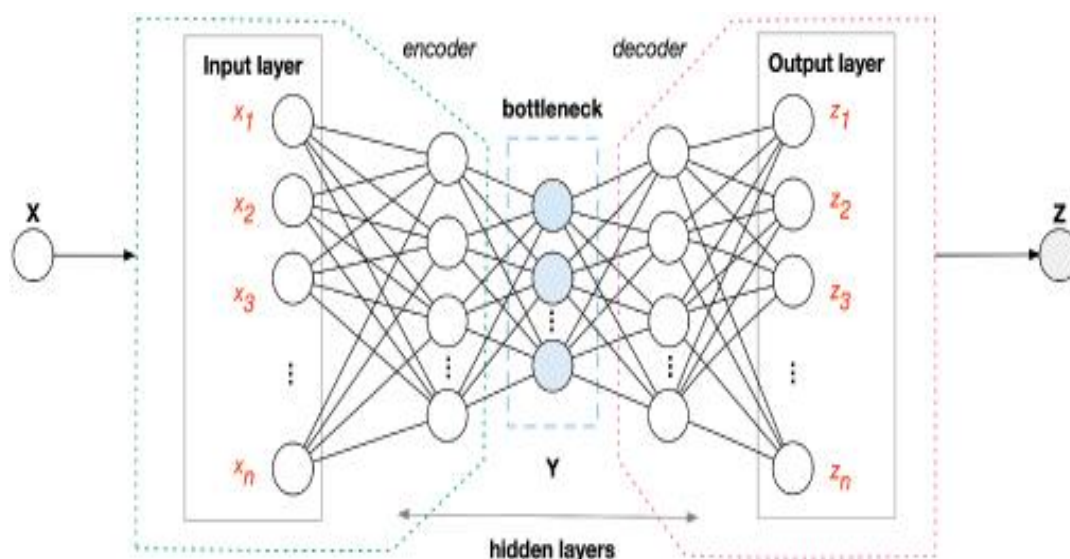


Figure 2. Representation of an autoencoder. Source: Catillo et al (2022)

2.5 Reviews on Botnet Detection

Alauthman et al., (2020), Alauthman et al. proposed a mechanism to help reduce complex traffic, integrated with a reinforcement learning technique. The reduced network traffic helps to reduce the computational weight of the proposed algorithm. The test results show that the new method has a correct detection rate of 98.3%, a low false-positive rate of 0.012%.

Charan et al., (2020), proposes a new method to detect word-list based DGA domain names using ensemble learning algorithms. They propose to use 15 lexical and network-level domain name features to construct and validate the detection method. Several supervised machine learning techniques, including C4.5, C5.0, CART decision tree and random forest are used in ensemble models to enhance the detection performance. Experimental results confirm that the C5.0 decision tree is the best algorithm with the prediction accuracy of 95.03%. The only issue with the proposed approach is it does not give the details on how are the ensemble models work to generate the results from individual models

Huong et al., (2021), proposed an edge-cloud architecture for attack detection. To minimize the complexity of the detection model, PCA was employed for feature dimensionality reduction. ANN model, which has a single hidden layer with 6-46 hidden units, was used for multi-class classification. The method was evaluated with the 11-class Bot-IoT dataset with a class imbalance ratio of 1:172162. The results showed that the F1 score of the FL model in the minority classes (D-H, DD-H, and KL) was less than 80%. Also, the FL model could not detect any of the DE samples in the testing set due to class imbalance problem in the training set.

Liang et al., (2020), proposed a novel system called MalPortrait, which combines individual features and association information of domains to detect malicious domains. In MalPortrait, They showed the association information among domains by a domain association graph where vertices represent domains and edges connect domains resolved to the same IP. Based on the graph, They combined individual features (e.g., string-based, network-based) of each domain and its association information to generate new features. They evaluated MalPortrait on the passive DNS traffic collected from real-world large ISP networks. Their experimental results showed that MalPortrait can accurately identify malicious domain names with a precision of 96.8% and a recall of 95.5%.

Vranken et al., (2022), in their paper, addressed how DGA-generated domain names can be detected by means of machine learning and deep learning. They presented an extensive literature review on recent prior work in which machine learning and deep learning have been applied for detecting DGA-generated domain names. They proposed the use of TF-IDF to measure frequencies of the most relevant n-grams in domain names, and use these as features in learning algorithms. They performed experiments with various machine-learning and deep-learning models using TF-IDF features, of which a deep MLP model yields the best results. For comparison, they also apply an LSTM model with embedding layer to convert domain names from a sequence of characters into a vector representation. The performance of the LSTM and MLP models is similar, achieving 0.994 and 0.995 AUC, and average F1-scores of 0.907 and 0.891 respectively.

Xu et al., (2019), in their paper, combined n-gram and a deep convolutional neural network and then proposed a novel n-gram combined character based domain classification (n-CBDC) model. The n-CBDC model runs in an end-to-end way that doesn't require hand-extracted features or domain name system (DNS) contextual information. It only needs to input the domain name itself and can automatically estimate the probability that the domain name was generated by DGAs. Experiments on real-world data show that the proposed method can effectively detect domain names generated by DGAs with 98.69% average detection rate and 0.9829 average F-measure, and significantly outperformed the state-of-art methods in detecting pronounceable and wordlist-based DGA domain names with more than 93.89% detection rate.

3. METHODOLOGY

The system architecture of the proposed work in figure 3.1 shows how the whole system will move from raw domain names to the final classification output. It explains every part of the system and what each part will do. The architecture is designed in a way that the domain names pass through different stages, and each stage adds something important that helps the final model understand the patterns in the domains. The aim is to make the system able to detect legitimate domains, wordlist based DGA domains and other DGA domains in a clear and organised way.

This section talks about each component one after the other so that the reader can understand how the system works as a whole. The work here is simple and direct. Every component has a role and a tool that will be used to carry out the role. By explaining the system in this form, it becomes easier to understand how the system was built and how it supports the main goal of detecting wordlist based DGA domains

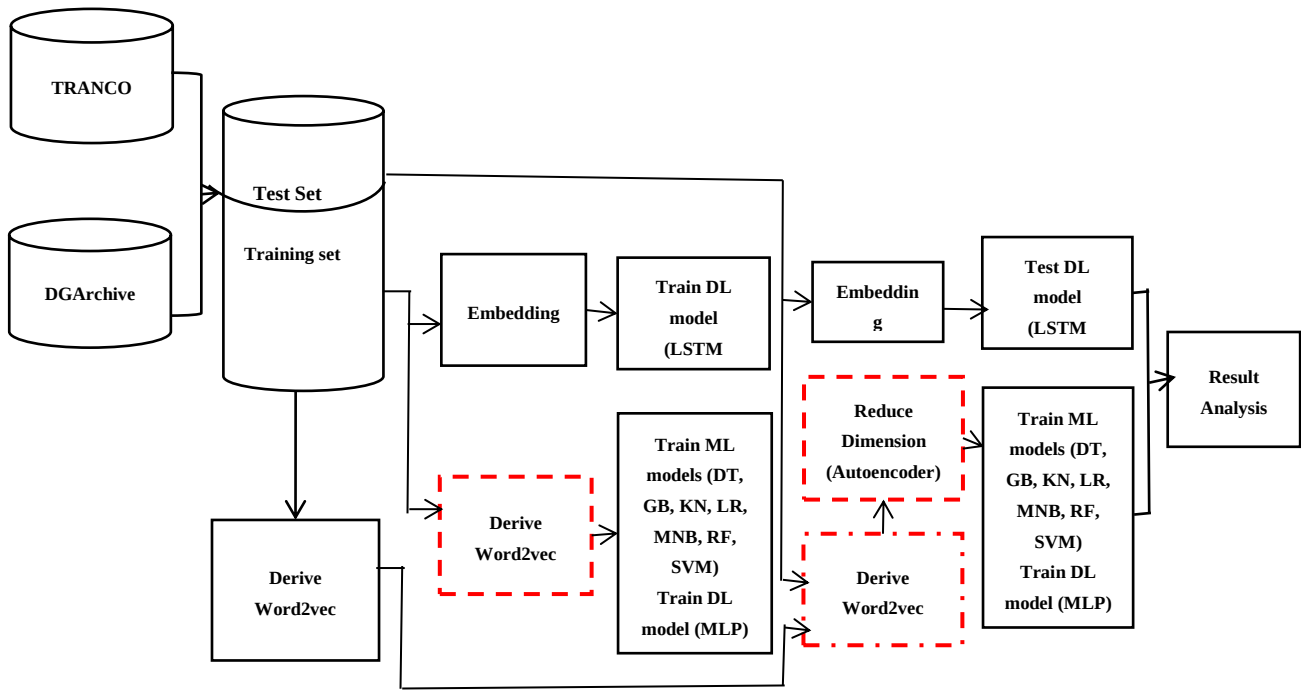


Fig 3.1: Architecture of the Proposed System

The use case diagram in Figure 3.2 illustrates a collaborative process between the user and the system to detect and classify DGA botnet attacks. The user initiates the process by providing the necessary data and viewing the outcomes, while the system handles the technical aspects of feature extraction, model training, and attack detection/classification using the models. This interaction ensures that the user can effectively leverage the system's capabilities to identify and mitigate potential botnet threats.

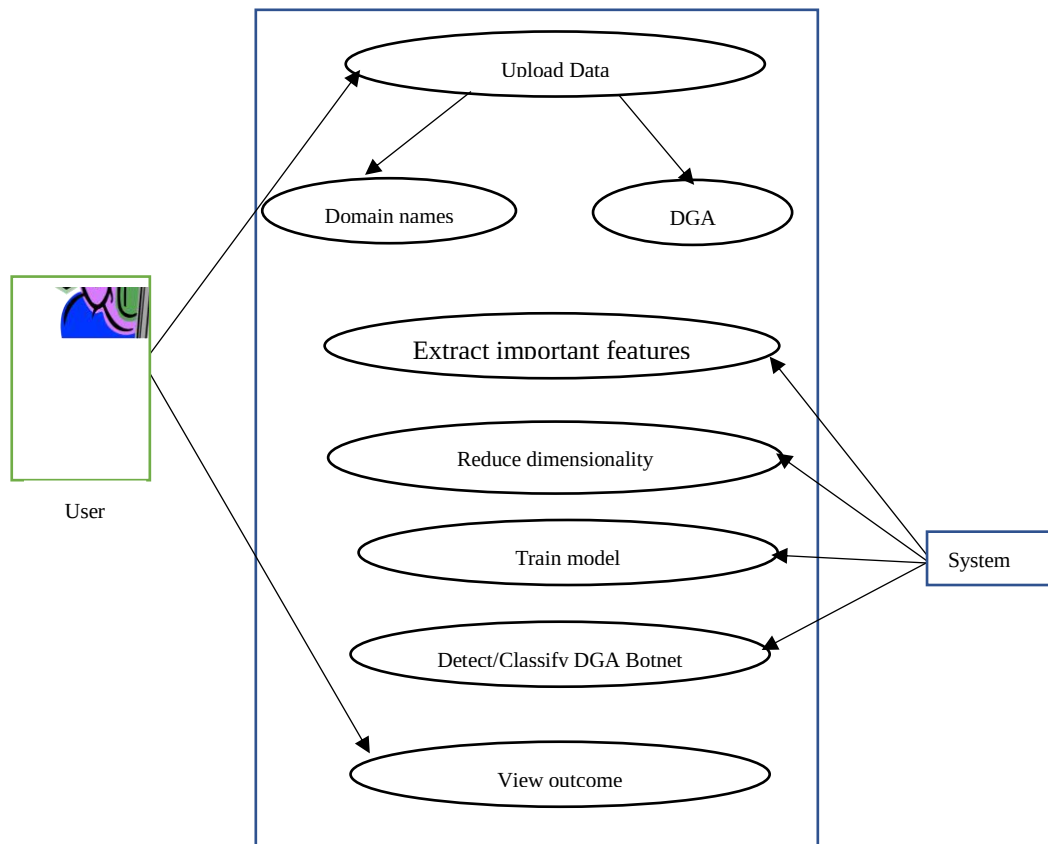


Fig. 3.2: Use Case Diagram of the proposed system

3.1 System Components Explanation

A. User

The user in this use case diagram is responsible for several key activities that initiate and interact with the system's functionalities. The user is responsible for:

i. Upload Data: The user uploads the data into the system. This data includes legitimate domain names and DGA(Domain Generated Algorithm)

ii. View Outcome: After the system processes the data and performs its analysis, the user views the outcomes. This could include the results of the detection and classification processes, such as whether a given domain is associated with a botnet and any relevant details about the detection.

B. System

The system is responsible for handling the core technical processes required to analyze the data and detect DGA botnet activities. The system's responsibilities include:

i. Extract Important Features: The system processes the uploaded data to extract significant features. This involves analyzing the domain names, to identify patterns and characteristics that are indicative of DGA botnet activity.

ii. Model Training: The system trains the models using the extracted features.

iii. Detect/Classify DGA Botnet: The system uses models to detect and classify DGA botnet attacks.

c. Interaction between User and System

The use case diagram depicts the interaction between the user and the system in a sequential manner:

i. Upload Data: The user begins by uploading the necessary data into the system. This could include various forms of data such as domain lists, and suspected DGA domains.

ii. Extract Important Features: Upon receiving the data, the system extracts important features. This step is crucial as it involves identifying key characteristics and patterns in the data that will be used for model training and subsequent detection.

iii. Model Training: The system trains its models, using the extracted features. This process involves learning from past data to develop a robust model that can detect and classify new instances of DGA botnet activity.

iv. Detect/Classify DGA Botnet: Once trained, the system applies the MLP model to the new data provided by the user. It detects and classifies any DGA botnet attacks based on the learned patterns and features.

v. View Outcome: Finally, the system presents the results of its analysis to the user. The user can view whether the provided domains are associated with DGA botnet activity and gain insights from the system's classification.

The activity diagram in Figure 3.3 depicts a detailed flow of interactions between the user and the system components. The user initiates the process by launching the application and opening a browser. The system then performs data preparation, including data preprocessing, word embedding, dimension reduction, and feature extraction. The data is fed into the models for training and evaluation. Finally, the system outputs the results, showing the detection and classification of DGA botnet activities, which the user can view to gain insights and take action. This structured sequence ensures a comprehensive analysis, leveraging machine learning and deep learning for effective botnet detection and classification. The activity diagram is in the figure below

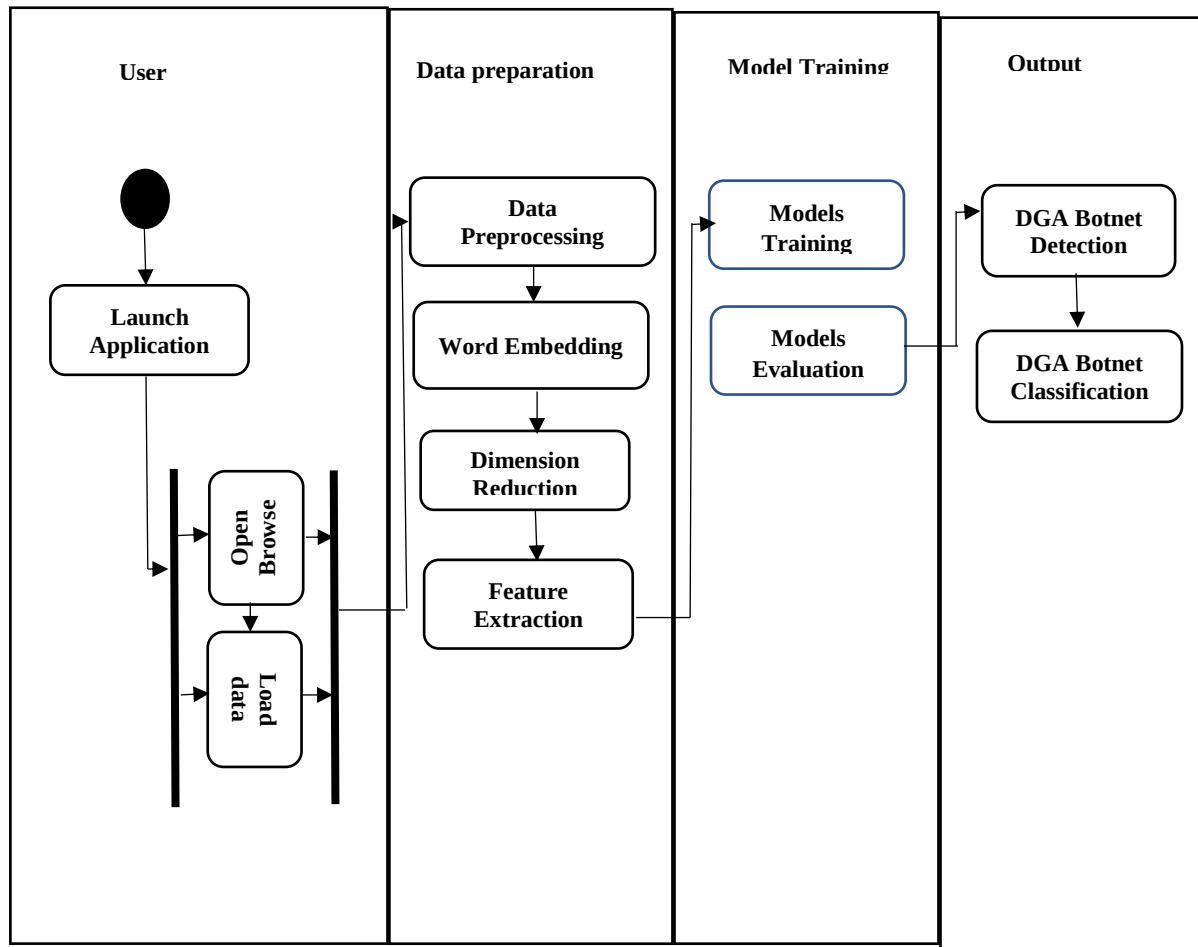


Figure 3.3: Activity Diagram of the Proposed System

3.2 Data Preparation

The following steps were involved in the preparation of the data.

a. Data Preprocessing

The system begins data preprocessing with data cleaning. Data cleaning involves removing any noise, errors, or irrelevant information from the dataset to ensure that the data is accurate and consistent, and also the tokenization of each domain name into sequence of character of N-gram to capture the character pattern for each of the domain names.

b. Word Embedding

The next step in data preparation is word embedding using word2vec for text representation.

c. Dimensionality reduction

At this point, Autoencoder is trained on CBOW(continuous bag of words) to compress the domain name embedding into a latent space with a lower dimension

d. Feature Extraction

The final step in data preprocessing is feature extraction. Feature extraction involves identifying and selecting the most relevant features from the data that will be used for model training and evaluation. These features are critical for accurate detection and classification.

3.3 Model Training and Model Evaluation

a. Model Training

The system proceeds to model training. The model learns from the preprocessed data to minimize error and improve prediction accuracy.

b. Model Evaluation

After training, the system evaluates the model. Model evaluation assesses the performance of the trained model using a separate validation dataset. This step measures how well the model can generalize to new, unseen data, and identifies any potential issues.

3.4 Output (DGA Botnet Detection and Classification)

a. DGA Botnet Detection

The system performs DGA botnet detection. Using the trained and evaluated model, the system analyzes new data to detect the presence of DGA botnet activities. This involves identifying patterns that match known botnet behaviors.

b. DGA Botnet Classification

The final step is DGA botnet classification. After detecting the presence of a botnet, the system classifies it into specific categories or types.

3.5 Algorithm of the Proposed System

This algorithm shows how domain names move from raw data to final 3-class prediction. The classes are: legitimate_domain, wordlist_based_DGA_domain, other_DGA_domain. The algorithm has two main tracks. One track trains many machine learning models on Word2Vec + auxiliary features. The other track trains an LSTM on sequences. At the end models are compared and the best model is chosen.

A. Algorithmic Steps of the Proposed System

Step 0 — Setup

- i. Create project folders: /data, /models, /reports.
- ii. Install needed libs: Google Colab Environment stup, pandas, numpy, gensim, scikit-learn, tensorflow/keras, imbalanced-learn, shap.
- iii: Set random seed for reproducibility.

Step 1 — Data load

- i. Read raw domain lists from sources into pandas.
- ii. Keep two fields: domain_name, class.
- iii. Drop rows with missing domain_name.

Step 2 — Basic cleaning and normalization

- i. Lowercase domain strings.
- ii. Strip URL prefixes (http://, https://, www.).
- iii. Remove leading/trailing spaces.
- iv. Normalize unicode (NFKC).
- v. Remove records that are too short (< 3 chars) or empty after cleaning.

Step 3 — TLD removal and subdomain handling

- i. Use TLD extractor to split domain into SLD (main name) and TLD.
- ii. Keep SLD for analysis. Optionally save TLD as feature.

Step 4 — Tokenization / word segmentation

- i. For each SLD, run word segmentation to get tokens (dictionary-based + statistical).
- ii. If no tokens found, fallback to character n-grams or single-character tokens.
- iii. Save token list for each domain.

Step 5 — Auxiliary feature extraction

For each domain we compute and store:

- i. length (chars)
- ii. num_tokens
- iii. avg_token_len
- iv. digit_count and digit_ratio
- v. vowel_consonant_ratio
- vi. shannon_entropy of characters
- vii. char_ngram_counts (optional)

We normalize numeric features later in the code for the model.

Step 6 — Build Word2Vec vocabulary & train embeddings

- i. Build corpus: list of token lists from step 4.
- ii. Train Word2Vec (gensim) with chosen params:
 - a). vector_size = 100 (example)
 - b). window = 3
 - c). min_count = 1 or 2
 - d). sg = 1
 - e). epochs = 10
- iii. Save model.

Step 7 — Construct feature vectors for ML models

- i. For each domain map tokens -> vectors using Word2Vec.
- ii. Aggregate token vectors to single vector per domain: e.g., avg_vec = mean(token_vecs)
- iii. Optionally compute max_vec and std_vec and concat: [mean|max|std].
- iv. Concatenate normalized auxiliary features to the embedding vector.
- v. Final X_ml is a numeric array. Final y are encoded classes.

Step 8 — Prepare sequences for LSTM model

- i. Build tokenizer (word-level or char-level) from token lists or characters.
- ii. Convert each domain to sequence of token ids.
- iii. Pad sequences to max_len (e.g., 12 tokens).
- iv. Optionally prepare embedding matrix from Word2Vec to initialize embedding layer.

Step 9 — Train / split data (80/20)

- i. Shuffle dataset.
- ii. Stratified split into training (80%) and testing (20%).
- iv. Fit scaler (StandardScaler) on training X_ml and transform train and test.
- v. For LSTM, keep X_seq_train, X_seq_test and y_train, y_test.

Step 10 — Handle class imbalance (during training)

- i. The dataset is structured to ensure classes have enough samples to result in good performance scores
- ii. Do not oversample test set.

Step 11 — Train multiple ML models

For each model in [SVM, KNN, RandomForest, DecisionTree, GradientBoosting, LogisticRegression, MLP]:

- i. Define hyperparameter grid / distribution.
- ii. Run RandomizedSearchCV or GridSearchCV with stratified CV on training set.
- iii. Fit best estimator on full training set.
- iv. Save model and cross-val results.

Step 12 — Train LSTM model

Build model: Embedding (LSTM embeds tokens/vectors implicitly)

Step 13 — Evaluate all models on test set

- i. For each ML model, predict y_{pred} on X_{ml_test} .
- ii. For LSTM, predict y_{pred} on X_{seq_test} .
- iii. Compute metrics: accuracy, precision, recall, F1 per class, confusion matrix, ROC-AUC (one-vs-rest), PR-AUC for each class.

Step 14 — Statistical comparison and selection

- i. Compare class-wise F1 for wordlist_based_DGA_domain across models.
- ii. Choose final model that best balances recall and precision for wordlist class and is stable.

Step 15 — Explainability and error analysis

- i. Run SHAP for top ML model(s) to get feature importance and per-sample explanations.
- ii. Inspect false positives and false negatives for wordlist class.
- iii. Tune tokenization or features if many errors are found.

Step 16 — Finalize and save

- i. Save chosen model, scaler, tokenizer, and Word2Vec model.
- ii. Store test results and confusion matrix figures.
- iii. Document hyperparameters and seed values.

Step 17 — Deployment

Build inference API: input domain -> preprocess -> tokens -> features -> model predict -> return class + score -> explainability using Gradio environment on Colab

3.6 Dataset for the Proposed System

The dataset used in this study came from different open sources that provide domain names for security research. The dataset holds only two fields, which are the domain name itself and the class label for that domain. The class label shows if the domain is a **legitimate domain**, a **wordlist based DGA domain**, or an **other_DGA domain**. This makes the dataset suitable for the three-class learning task in this work. The dataset size is also large enough to allow the models learn different patterns that appear in normal and DGA domains.

The dataset went through cleaning before use. All repeated domains were removed, and domains that had strange symbols or invalid structures were also taken out. The dataset shows a clear class imbalance, because legitimate domains appear much more than the two DGA groups. The counts are: 627123 legitimate, 179922 wordlist based DGA, and 192955 other DGA domains. This imbalance was noted because it affects how the models learn. Simple handling steps were applied later in the work so that the models will not favor the legitimate class too strongly.

The dataset gives enough spread across the three classes, and it supports the main aim of this research, which is to detect wordlist-based DGA domains while still recognizing normal domains and other types of DGA domains. The dataset structure is simple, but it carries the needed information to build, train, and test the proposed models.

To ensure consistency and reduce noise in the dataset. All domain names were first converted to lowercase so that variations caused by letter casing did not affect the learning process. This step was important because domain names are case-insensitive in practice, but machine learning models treat uppercase and lowercase characters as different tokens if they are not normalized.

Next, unnecessary components of the domain strings were removed. These included internet protocols such as *http* and *https*, the *www* prefix, and common top-level domains (TLDs) like *.com*, *.net*, *.org*, *.ng*, and others. Removing these elements helped the system focus only on the core structure of the domain name, which is more relevant for identifying algorithmically generated patterns. This step reduced redundancy and ensured that domains with the same base name but different TLDs were treated consistently.

All remaining special characters were removed, leaving only lowercase alphabets and digits. This produced clean and uniform domain strings suitable for tokenization, Word2Vec embedding, and auxiliary feature extraction such as length, entropy, and character ratios. The result of this preprocessing stage was a new feature, *clean_domain*, which served as the standardized input for all subsequent machine learning and deep learning models used in this research.

3.7 Model Evaluation

The following were used to evaluate the model

A. Accuracy

Accuracy shows how many domain names the model predicts correctly out of all predictions. It is one of the basic ways to check model performance.

B. Precision, Recall, and F1-Score

These metrics are very important for this study because the goal is not just general correctness. The model must identify word-based DGA domains, which often look like normal domains. If precision is low for this class, the model will wrongly label many legitimate domains as DGA. If recall is low, the system will miss many real word-based DGA domains. So F1-Score becomes a strong measure to check how well the system balances both sides.

C. Confusion Matrix

A confusion matrix shows how the model gets each class right or wrong. It is a table with actual classes on one side and predicted classes on the other. For a three-class system, it forms a 3×3 grid. No formula is needed, but the idea is:

diagonal cells = correct predictions

off-diagonal cells = wrong predictions

The confusion matrix helps this work because it shows exactly where the model makes mistakes. It reveals if the system is confusing **word-based DGA domains with legitimate domains**, which is a common problem in older studies. So it becomes a very clear way to judge if the new approach truly improves detection on the hardest class.

4. RESULTS AND DISCUSSION

The dataset used in this research contains three main classes of domain names. These classes are legitimate domains, other DGA domains, and wordlist-based DGA domains. The legitimate domains are regular domains that do not exhibit characteristics of botnets. The other DGA domains are domains generated by various DGA techniques except for the wordlist-based type. The wordlist-based DGA domains are specially generated using wordlists and are the main focus of this study because they are harder to detect compared to other types of domains

Table 4.1: Dataset information

Class Name	Count	Percentage (%)	Description
Legitimate Domain	627,123	62.71	Regular domains with no botnet behavior.
Other DGA Domain	192,955	19.3	Domains generated by DGAs except wordlist-based types.
Wordlist-based DGA Domain	179,922	17.99	Domains generated using wordlists, harder to detect, main focus of research.

Domain tokenization was carried out after cleaning the domain names to break each domain into smaller, meaningful units. In this work, tokenization was done at the character level, where each cleaned domain string was split into individual characters. For example, the domain *youtube* was converted

into the tokens [y,o,u,t,u,b, e], while *facebook* became [f, a, c, e, b, o, o, k]. This approach was chosen because DGA domains are often generated as random character sequences rather than real words, and character-level tokens make it easier to capture such patterns.

By representing domains as sequences of characters, the system was able to learn how characters commonly appear together in legitimate domains compared to DGA-generated ones. Domains like *microsoft* and *apple* show clear, readable character patterns, while algorithm-generated domains usually lack this structure. These tokenized sequences were later used as input for the Word2Vec model, allowing the system to learn meaningful vector representations of domains based on character relationships. This process helped improve classification accuracy by preserving the natural structure of domain names.

Auxiliary features were extracted to capture simple but useful characteristics of each domain name that may not be fully represented by learned embeddings. These features focus on the structure and composition of the domain strings. The extracted attributes include domain length, character entropy, digit ratio, vowel ratio, and special character ratio. Each feature gives a different view of how a domain is formed, making it easier to separate normal human-readable domains from those generated by algorithms.

The output of these auxiliary features showed clear patterns across the different domain classes. Legitimate domains generally had moderate lengths, higher vowel ratios, and lower entropy values, which reflect readable and meaningful words. In contrast, DGA domains, especially those generated randomly, tended to have longer or inconsistent lengths, low vowel ratios, and higher entropy values. Domains with many digits or unusual character combinations also stood out clearly through the digit ratio and entropy measurements.

These auxiliary features played an important role in strengthening the overall classification performance of the system. While embedding-based features captured contextual character relationships, the auxiliary features provided direct statistical signals that helped the model make better decisions. For example, even when two domains had similar character embeddings, differences in entropy or digit ratio often guided the model toward the correct class.

The inclusion of auxiliary features improved the robustness and interpretability of the system. They helped explain why a domain was classified in a certain way by showing measurable properties of the input domain. This made the results easier to understand and justify, especially when presenting the system outputs to users or during result analysis in this study.

Table 4.2 Auxiliary Features Computation

	Domain_name	class	clean_domain	tokens	length	entropy	digit_ratio	vowel_ratio	special_ratio
0	google.com	legitimate_domain	google	[g, o, o, g, l, e]	6	1.918296	0.0	0.500000	0.0
1	mail.ru	legitimate_domain	mailru	[m, a, i, l, r, u]	6	2.584963	0.0	0.500000	0.0
2	microsoft.com	legitimate_domain	microsoft	[m, i, c, r, o, s, o, f, t]	9	2.947703	0.0	0.333333	0.0
3	facebook.com	legitimate_domain	facebook	[f, a, c, e, b, o, o, k]	8	2.750000	0.0	0.500000	0.0
4	apple.com	legitimate_domain	apple	[a, p, p, l, e]	5	1.921928	0.0	0.400000	0.0
5	amazonaws.com	legitimate_domain	amazonaws	[a, m, a, z, o, n, a, w, s]	9	2.641604	0.0	0.444444	0.0
6	dzen.ru	legitimate_domain	dzenru	[d, z, e, n, r, u]	6	2.584963	0.0	0.333333	0.0
7	youtube.com	legitimate_domain	youtube	[y, o, u, t, u, b, e]	7	2.521641	0.0	0.571429	0.0
8	googleapis.com	legitimate_domain	googleapis	[g, o, o, g, l, e, a, p, i, s]	10	2.921928	0.0	0.500000	0.0
9	cloudflare.com	legitimate_domain	cloudflare	[c, l, o, u, d, f, l, a, r, e]	10	3.121928	0.0	0.400000	0.0
10	akamai.net	wordlist_DGA_domain	akamai	[a, k, a, m, a, i]	6	1.792481	0.0	0.666667	0.0

The first ML model used is Regression Analysis. Table 4.5 shows the Logistic Regression Report

Table 4.3: Logistic Regression Report

Class	Precision	Recall	F1-score	Support
Legitimate Domain	0.84	0.98	0.9	1537
Other DGA Domain	1	0.57	0.73	256
Wordlist DGA Domain	0.78	0.53	0.63	407
Accuracy			0.85	2200
Macro Average	0.87	0.69	0.75	2200
Weighted Average	0.85	0.85	0.83	2200

This logistic regression model was trained using the combined feature space from Word2Vec embeddings, auxiliary statistical features, and AutoEncoder latent representations. From the final evaluation, the model achieved an accuracy of 0.8455, which means about 84.55% of all tested domain names were classified correctly. The precision value of 0.8740 shows that when the model makes a prediction, it is usually correct, especially important for reducing false alarms in botnet detection. The recall of 0.6913 is lower than precision, which means the model is more conservative and does not always capture all malicious domains. The F1-score of 0.7534 balances this behavior and shows that, overall, the model performs reasonably well given the multi-class nature of the problem and the difficulty of distinguishing wordlist-based DGA domains from legitimate ones.

Looking at the class-level results, the legitimate_domain class has a precision of 0.84, recall of 0.98, and F1-score of 0.90 with a support of 1537 samples. This means the model is very strong at recognizing legitimate domains, correctly identifying almost all of them, which is important to avoid false positives in real deployments. For the other_DGA_domain class, the precision is 1.00, recall is 0.57, and F1-score is 0.73, with 256 samples. This shows that whenever the model predicts this class, it is almost always correct, but it misses a notable number of actual other-DGA domains. This suggests that these domains are harder to capture consistently, possibly due to overlapping patterns with wordlist-based DGAs. The wordlist_DGA_domain class has a precision of 0.78, recall of 0.53, and F1-score of 0.63, across 407 samples, showing that this class is the most challenging.

Table 4.6: KNN Classification Performance on Wordlist DGA Domain

Class	Precision	Recall	F1-Score	Support
Legitimate Domain	0.84	0.95	0.89	13,665
Other DGA Domain	0.89	0.59	0.71	2,872
Wordlist DGA Domain	0.71	0.54	0.61	3,554
Overall Accuracy			0.82	20,091
Macro Average	0.81	0.69	0.74	20,091
Weighted Average	0.82	0.82	0.81	20,091

This experiment was carried out on a dataset containing 100,455 domain samples, each labeled into one of the target classes. After combining Word2Vec embeddings (100 features) with five auxiliary statistical features, the resulting feature matrix had 105 input features per domain. These features were then passed through an AutoEncoder to reduce dimensionality and remove noise. During training, the AutoEncoder showed a steady and smooth reduction in reconstruction loss, starting from a high loss of 0.7593 at the first epoch and gradually decreasing to 0.0024 by the 25th epoch, with a final validation loss of 0.0027. This consistent decrease in both training and validation loss indicates that the AutoEncoder learned a stable and compact representation of domain patterns without overfitting. The compressed latent space produced 32-dimensional feature vectors for each domain, which were used as input to the KNN classifier.

After feature compression, the dataset was split into 80,364 training samples and 20,091 testing samples, ensuring a sufficient amount of data for both learning and evaluation. The KNN model was then trained on the 32-dimensional latent features. On the test set, the model achieved an accuracy of 0.8242, meaning that about 82.4% of the domains were correctly classified. The precision of 0.8106 shows that when the model predicts a class, it is usually correct, while the recall of 0.6916 indicates that the model is able to identify roughly 69% of all true malicious and legitimate domains across classes. The F1-score of 0.7360 reflects a reasonable balance between precision and recall, which is important for DGA detection where both false alarms and missed attacks matter.

These results show that KNN can perform fairly well when supported by Word2Vec features, auxiliary domain statistics, and AutoEncoder-based latent representations. The model benefits from the compact feature space learned by the AutoEncoder, which makes distance-based learning more effective and less sensitive to noise.

The KNN classification results show how well the model was able to separate legitimate domains from DGA-generated domains, with special attention to wordlist-based DGA botnet domains, which are the main focus of this work. For legitimate domains, the model achieved a high precision of 0.84 and a very strong recall of 0.95, resulting in an F1-score of 0.89. This means that most legitimate domains were correctly identified, and very few were wrongly flagged as malicious. This behavior is important because it reduces false alarms in real-world deployment. For other DGA domains, the precision is 0.89, showing that when the model predicts this class, it is usually correct, but the recall drops to 0.59, indicating that a noticeable number of other DGA domains were missed or confused with other classes.

For the wordlist-based DGA botnet domains, which are the hardest to detect, the model recorded a precision of 0.71 and a recall of 0.54, with an F1-score of 0.61. This result shows that while the model can correctly identify some wordlist DGA domains, a significant portion is still misclassified, often because these domains look more like legitimate words.

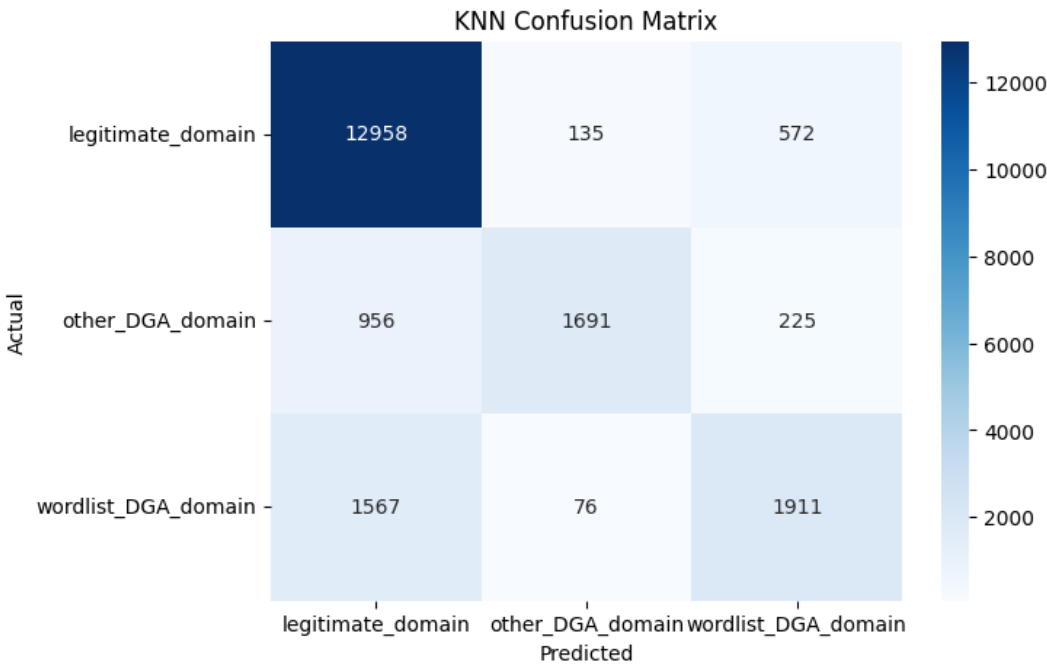
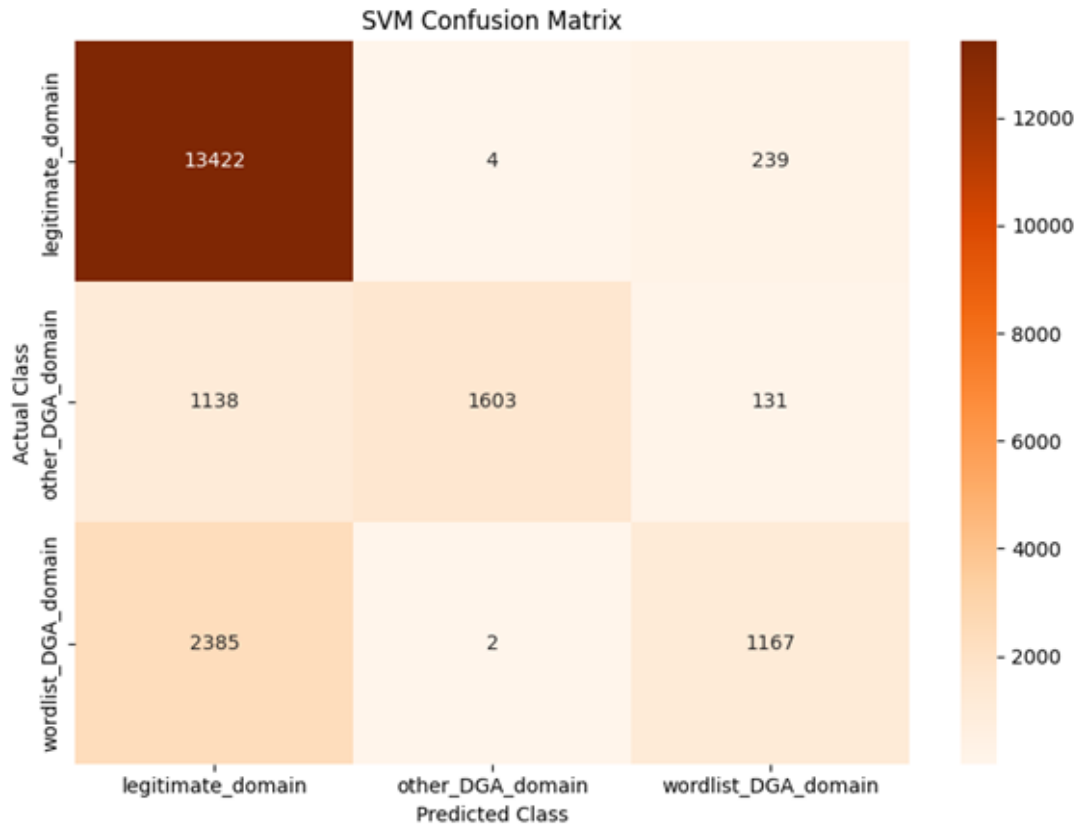


Fig. 4.1: KNN Confusion Matrix

Table 4. 7: SVM Classification Report

Class	Precision	Recall	F1-score	Support
legitimate_domain	0.79	0.98	0.88	13665
other_DGA_domain	1	0.56	0.72	2872
wordlist_DGA_domain	0.76	0.33	0.46	3554
Accuracy			0.81	20091
Macro Avg	0.85	0.62	0.68	20091
Weighted Avg	0.82	0.81	0.78	20091

The output reflects the training and evaluation of a LinearSVC model on the domain dataset, which contains three classes: 'legitimate_domain', 'wordlist_DGA_domain', and 'other_DGA_domain'. Before training the SVM, the dataset was preprocessed through a combination of Word2Vec embeddings, auxiliary features, and an AutoEncoder. Each domain name was tokenized and converted into a 100-dimensional Word2Vec embedding to capture semantic and lexical patterns, while auxiliary features such as domain length, character entropy, digit ratio, vowel ratio, and special character ratio were added to provide additional distinguishing signals. These features were concatenated and passed through an AutoEncoder, which compressed the combined feature set into 32-dimensional latent features, preserving essential patterns while reducing noise and redundancy. The AutoEncoder training over 20 epochs shows a steady decrease in both training loss (from 0.7875 to 0.0040) and validation loss (from 0.2760 to 0.0038), indicating that the latent representations effectively captured the key characteristics of each domain class.

**Fig. 4.2:** SVM Confusion Matrix**Table 4.8:** Random Forest Classification Report

Class	Precision	Recall	F1-score	Support
legitimate_domain	0.83	0.98	0.9	13,665
other_DGA_domain	0.98	0.57	0.72	2,872
wordlist_DGA_domain	0.81	0.51	0.62	3,554
Accuracy			0.84	20,091
Macro Avg	0.87	0.68	0.75	20,091
Weighted Avg	0.85	0.84	0.82	20,091

The output shows the preprocessing and feature extraction process for a Random Forest classification pipeline on a domain dataset of 100,455 samples divided into three classes: 'legitimate_domain', 'wordlist_DGA_domain', and 'other_DGA_domain'. Each domain was tokenized and transformed into Word2Vec embeddings to capture semantic patterns, while auxiliary features including domain length, character entropy, digit ratio, vowel ratio, and special character ratio were calculated to provide additional discriminatory signals. These features were concatenated and then passed through an AutoEncoder, which learned a compressed 32-dimensional latent representation of the data. The AutoEncoder training over 20 epochs shows a steady decline in both training loss (from 0.7881 to 0.0035) and validation loss (from 0.2823 to 0.0037), indicating that the latent features effectively capture the important structural and lexical patterns in both legitimate and DGA domains while reducing noise.

After feature extraction, the dataset was split into 80% training (80,364 samples) and 20% testing (20,091 samples). The AutoEncoder latent features, which encode both the Word2Vec embeddings and auxiliary features, were used as input to the Random Forest classifier. The training and testing set shapes confirm that each sample is represented in a 32-dimensional latent space, providing a compact but informative feature set for the classifier. This preprocessing and feature learning pipeline ensures that the Random Forest model has access to both semantic and handcrafted information, which is critical for distinguishing between legitimate domains and the more complex, wordlist-based DGA domains.

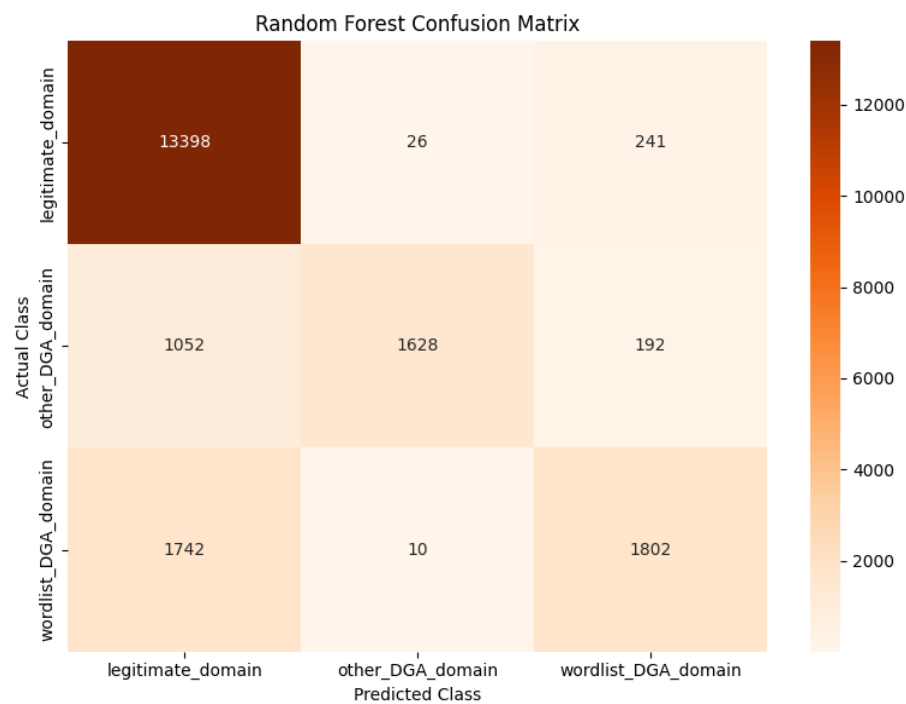


Fig. 4.3: Random Forest Confusion Matrix

Table 4.9: Decision Tree

Class	Precision	Recall	F1-score	Support
legitimate_domain	0.82	0.94	0.88	13,665
other_DGA_domain	0.92	0.56	0.69	2,872
wordlist_DGA_domain	0.64	0.47	0.54	3,554
Accuracy	0.8			20,091
Macro Avg	0.79	0.65	0.7	20,091
Weighted Avg	0.8	0.8	0.79	20,091

The Decision Tree classifier demonstrates strong performance on legitimate domains, achieving a precision of 0.82, recall of 0.94, and F1-score of 0.88. This means that most domains predicted as legitimate were correct (precision), and the model successfully identified the majority of actual legitimate domains (recall). For other DGA domains, the classifier reached a high precision of 0.92, indicating that predictions labeled as other DGA domains were highly reliable, but a lower recall of 0.56 suggests that nearly half of the true other DGA domains were missed. The wordlist DGA domains proved the most challenging, with precision 0.64, recall 0.47, and F1-score 0.54, reflecting the inherent difficulty of detecting domains that mimic legitimate lexical patterns.

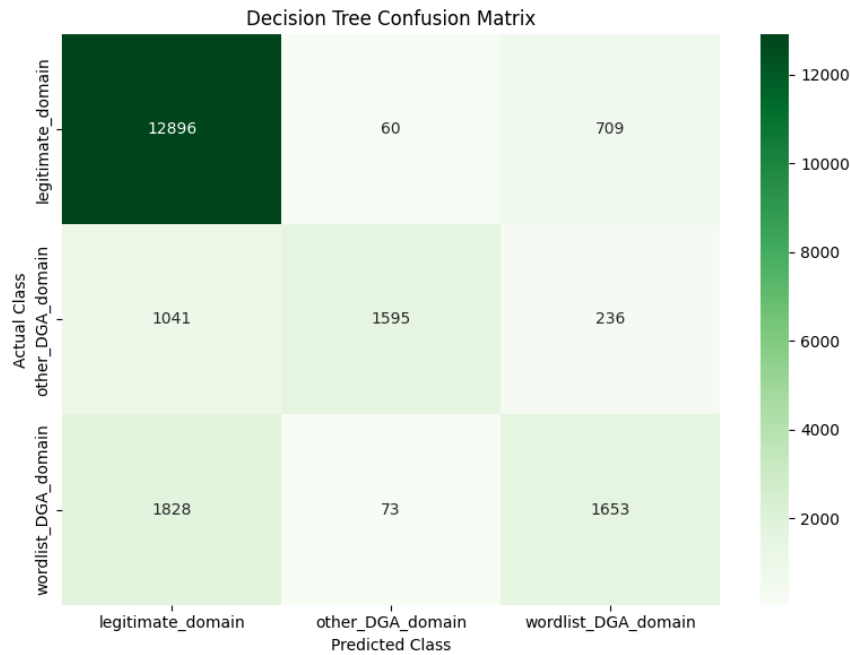


Fig. 4.4. Decision Tree Confusion Matrix

Table 4.10: Gradient Boosting Report

Class	Precision	Recall	F1-score	Support
legitimate_domain	0.83	0.97	0.9	13,665
other_DGA_domain	0.99	0.56	0.71	2,872
wordlist_DGA_domain	0.77	0.55	0.64	3,554
Accuracy			0.84	20,091
Macro Average	0.86	0.69	0.75	20,091
Weighted Average	0.84	0.84	0.83	20,091

The Gradient Boosting classification report shows that the model is performing well overall on distinguishing between legitimate and DGA domains, with some variation across classes. The legitimate_domain class has a precision of 0.83, meaning that 83% of the domains predicted as legitimate were correct, and a recall of 0.97, indicating that 97% of all actual legitimate domains were correctly identified. This results in a high F1-score of 0.90, demonstrating excellent balance between precision and recall for this class. For other_DGA_domain, the model achieves very high precision (0.99) but a lower recall (0.56), meaning it is very accurate when it predicts a domain as other DGA, but misses a significant number of actual other DGA domains. The wordlist_DGA_domain has more moderate metrics, with precision 0.77, recall 0.55, and F1-score 0.64, indicating the model is fairly good at identifying wordlist-based DGA domains but still misses almost half of them.

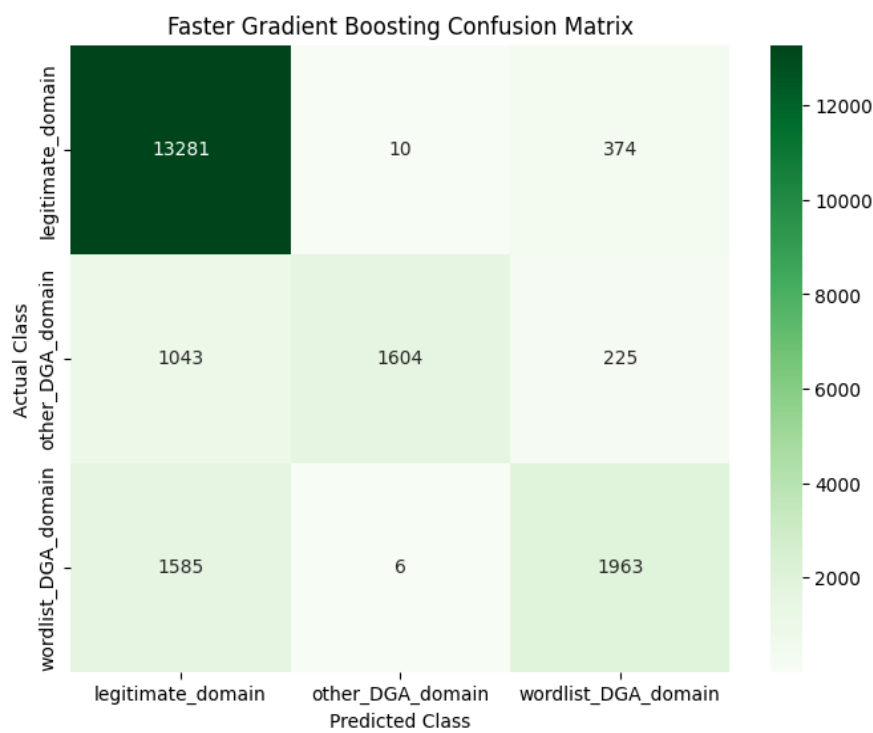


Fig. 4.5: Confusion Matrix for Gradient Boosting

Table 4.11: Comparative Analysis of ML in DGA Classification

Model	Accuracy	Precision	Recall	F1-score	AUC
KNN	0.8376	0.8707	0.6848	0.746	0.86
SVM (LinearSVC)	0.8059	0.8492	0.6229	0.6836	0.82
Random Forest	0.8376	0.8707	0.6848	0.746	0.88
Decision Tree	0.8035	0.7924	0.6547	0.7024	0.82
Gradient Boosting	0.8386	0.8637	0.6942	0.7514	0.87
Logistic Regression	0.85	0.85	0.69	0.75	0.86

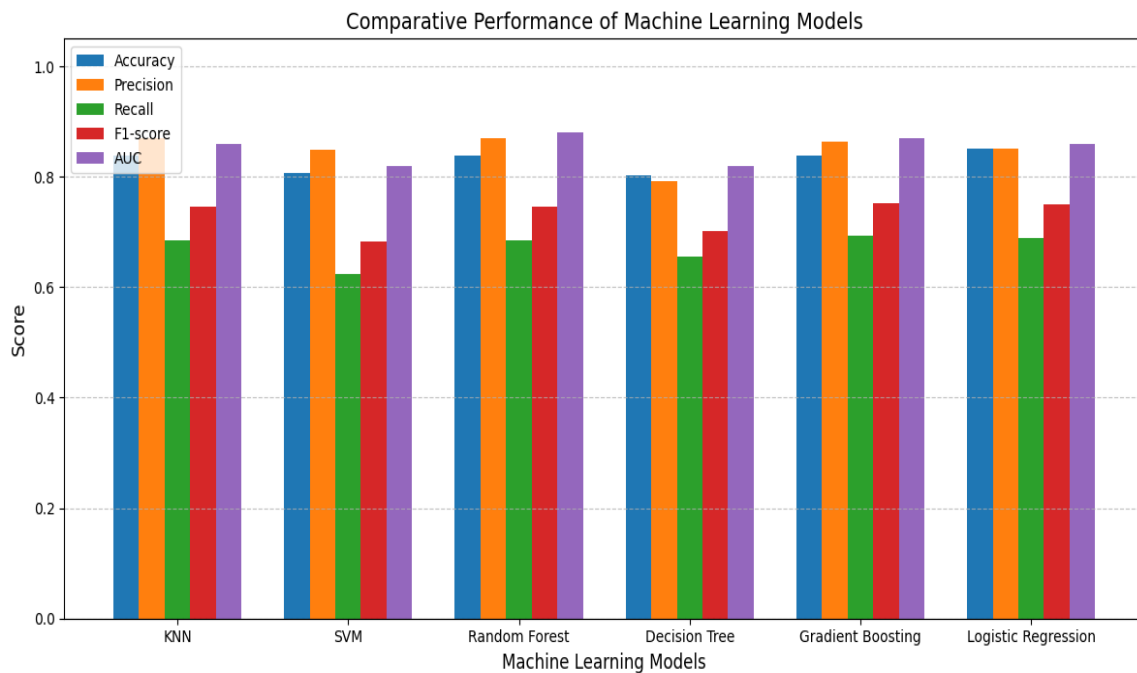


Fig. 4.6: Overall Comparative Performance Plot of ML Metrics

The grouped bar chart provides a comprehensive visual comparison of the six machine learning models, KNN, SVM, Random Forest, Decision Tree, Gradient Boosting, and Logistic Regression across five key performance metrics: Accuracy, Precision, Recall, F1-score, and AUC. Accuracy ranges from 0.8035 for Decision Tree to 0.8500 for Logistic Regression, showing overall correct predictions for each model. Precision, which measures the correctness of positive predictions, is highest at 0.8707 for KNN and Random Forest, while Decision Tree has the lowest at 0.7924. Recall, indicating the proportion of true positives detected, is strongest for Gradient Boosting at 0.6942 and lowest for SVM at 0.6229. The F1-score, balancing precision and recall, reaches a maximum of 0.7514 for Gradient Boosting and a minimum of 0.6836 for SVM. Finally, the AUC, representing the model's ability to distinguish between legitimate and DGA domains across thresholds, is highest at 0.88 for Random Forest and lowest at 0.82 for both SVM and Decision Tree. This single plot effectively highlights each model's strengths and weaknesses, providing a clear visual tool to identify which algorithms are most suitable for detecting wordlist-based and other DGA domains in alignment with the research aim.

Enter Domain

Clear Submit

Fig. 4.7: Input Terminal for Domain Name: without data

Enter a Domain name and see its classification Result

Enter Domain

brettells.com

Clear Submit

Fig. 4.8: Input Terminal for Domain name: with Data

The interface was designed to accept a domain name as input. Once entered, the system classifies the domain into one of three classes: legitimate, other DGA, or wordlist DGA. The interface also displays auxiliary features of the domain, including length, entropy, digit ratio, vowel ratio, and special character ratio, visualized as bar charts. In addition, it shows a chart of the model's predicted probabilities for each class, a word cloud of the domain

characters, and the autoencoder loss curve, providing insight into the domain's structural anomalies. All outputs are presented together, offering a comprehensive view of the domain analysis. Figure. 4.9 shows the domain analysis and class output.



Fig. 4.9: Domain Analysis and Class Output

Also in the output panel, the AutoEncoder training loss curve shows how well the model is learning to reconstruct the input features over time. On the x-axis, there are the epochs, which represent each full pass through the training dataset, and on the y-axis is the reconstruction loss, which measures the difference between the original input and what the AutoEncoder reconstructs. At the start, the loss is high because the model doesn't know how to recreate the input well, but as training progresses, the loss decreases, showing that the AutoEncoder is learning the patterns in the data. A smooth downward trend means the model is effectively learning, and the final low loss value indicates the AutoEncoder can accurately reproduce normal domain features, which is important for detecting anomalous or DGA domains that deviate from these learned patterns.

The use of confidence threshold shows how the system flags unusual domains using the AutoEncoder. The anomaly score (0.011319 in this case as seen in the domain classified above) measures how different the domain's features are from what the AutoEncoder learned as "normal" during training. The threshold (0.043624) is set at the 90th percentile of reconstruction errors from all training domains, meaning that 90% of normal domains have lower reconstruction errors. If a domain's anomaly score is above this threshold, it is considered abnormal.

For wordlist DGA domains, this is especially useful because they often have unusual structures compared to legitimate domains, like uncommon character combinations or lengths. Even if the classifier predicts a class, the anomaly score acts as an extra check: a wordlist DGA domain might have a reconstruction error higher than the threshold, signaling it as suspicious or abnormal. This strengthens the detection system by catching DGA domains that may not be obvious from the classifier alone. Figure 4.10 shows the AutoEncoder Loss Curve with the Reconstruction Loss vs Epoch with Threshold



Fig. 4.10: AutoEncoder Loss Curve: Reconstruction Loss vs Epoch with Threshold

Also, the the output panel has an auxiliary feature bar chart that displays key characteristics of the entered domain in a visual format. Each bar represents a specific feature: the domain’s length, entropy, digit ratio, vowel ratio, and special character ratio. The height of each bar indicates the magnitude of that feature for the domain, making it easy to see which characteristics are high or low. This visualization helps to understand the domain’s structure at a glance, showing patterns that might indicate whether the domain is legitimate or generated by a DGA, with particular attention to unusual distributions such as high digit or special character ratios that often occur in DGA domains. Fig. 4.11 shows the Auxiliary feature plot

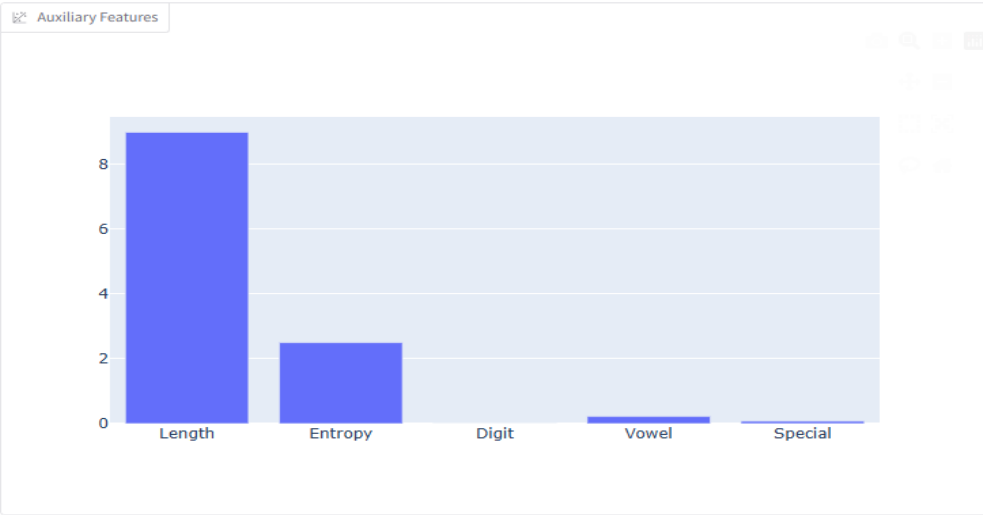


Fig. 4.11: Auxiliary Feature Plot.

4. CONCLUSION

From the experiments, it is clear that combining Word2Vec embeddings with auxiliary domain features creates a strong representation for DGA domain detection.

The autoencoder added value by highlighting domains with unusual structures using reconstruction error, which helped in understanding domain anomalies. The Gradio interface combined all results into an explainable system where users can see predicted classes, probabilities, auxiliary features, wordclouds, and loss curves. This approach allows a clear visualization of why a domain is flagged as normal or DGA. Overall, the study shows that using Word2Vec with Machine learning provides an effective, explainable, and practical solution for detecting wordlist-based DGA botnet domains.

REFERENCES

- Alieyan, K., Almomani, A., Manasrah, A., & Kadhum, M. M. (2017). A survey of botnet detection based on DNS. *Neural Computing and Applications*, 28, 1541-1558.
- Alauthman, M., Aslam, N., Al-Kasassbeh, M., Khan, S., Al-Qerem, A., & Choo, K. K. R. (2020). An efficient reinforcement learning-based Botnet detection approach. *Journal of Network and Computer Applications*, 150, 102479.
- Banerjee, M., & Samantaray, S. (2019). Network traffic analysis based iot botnet detection using honeynet data applying classification techniques. *International Journal of Computer Science and Information Security (IJCSIS)*, 17(8).
- Bilge, L., Sen, S., Balzarotti, D., Kirda, E., & Kruegel, C. (2014). Exposure: A passive dns analysis service to detect and report malicious domains. *ACM Transactions on Information and System Security (TISSEC)*, 16(4), 1-28.
- Catillo, M., Pecchia, A., & Villano, U. (2022, August). Botnet detection in the internet of things through all-in-one deep autoencoding. In *Proceedings of the 17th International Conference on Availability, Reliability and Security* (pp. 1-7).
- Charan, P. S., Shukla, S. K., & Anand, P. M. (2020). Detecting word based DGA domains using ensemble models. In *Cryptology and Network Security: 19th International Conference, CANS 2020, Vienna, Austria, December 14–16, 2020, Proceedings 19* (pp. 127-143).
- Hoang, X.D., & Nguyen, Q. C. (2018). Botnet detection based on machine learning techniques using DNS query data. *Future Internet*, 10(5), 43.
- Huong, T. T., Bac, T. P., Long, D. M., Thang, B. D., Binh, N. T., Luong, T. D., & Phuc, T. K. (2021). Lockedge: Low-complexity cyberattack detection in iot edge computing. *IEEE Access*, 9, 29696-29710.
- Li, X., Wang, J., & Zhang, X. (2017). Botnet detection technology based on DNS. *Journal Future Internet*, (2017(9), 55. <https://doi.org/10.3390/fi9040055>
- Liang, Z., Zang, T., & Zeng, Y. (2020, May). Malportrait: Sketch malicious domain portraits based on passive DNS data. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 1-8).
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. Namgung, J., Son, S., & Moon, Y. S. (2021). Efficient deep learning models for DGA domain detection. *Security and Communication Networks*, 2021(1), 8887881.
- Silva, S. S., Silva, R. M., Pinto, R. C., & Salles, R. M. (2013). Botnets: A survey. *Computer Networks*, 57(2), 378-403.
- Shinan, K., Alsubhi, K., Alzahrani, A., & Ashraf, M. U. (2021). Machine learning-based botnet detection in software-defined network: a systematic review. *Symmetry*, 13(5), 866.
- Umar, M., Abisoye, O. A., & Ganiyu, S. O. (2021). A Literature Survey on IoT Botnet Detection Techniques. 4th International Conference on Information Technology in Education and Development 2021.
- Vranken, H., & Alizadeh, H. (2022). Detection of DGA-generated domain names with TF-IDF. *Electronics*, 11(3), 414.
- Zeng, J., Tang, W., Liu, C., Hu, J., & Peng, L. (2012). Efficient detect scheme of botnet command and control communication. In *Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16, 2012. Proceedings, Part I 3* (pp. 576-581).
- Xu, C., Shen, J., & Du, X. (2019). Detection method of domain names generated by DGAs based on semantic representation and deep neural network. *Computers & Security*, 85, 77-88.