Expert Systems With Applications 293 (2025) 128629

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

RAMPAGE: a software framework to ensure reproducibility in algorithmically generated domains detection

Tomás Pelayo-Benedet [a], Ricardo J. Rodríguez [a,*], Carlos H. Gañán [b]

[a] Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain
[b] Delft University of Technology, the Netherlands

ARTICLE INFO

Keywords:
Malware
Machine learning models
Neural network models
Algorithmically generated domains detection
Evaluation

ABSTRACT

As part of its life cycle, malware can establish communication with its command and control server. To bypass static protection techniques, such as blocking certain IPs in firewalls or DNS server deny lists, malware can use *algorithmically generated domains* (AGD). Many different solutions based on deep learning have been proposed during the last years to detect this type of domains. However, there is a lack of ability to compare the proposed models because there is no common framework that allows experiments to be replicated under the same conditions. Each previous work shows its evaluation results, but under different experimentation conditions and even with different datasets. In this paper, we address this gap by proposing a software framework, dubbed RAMPAGE (*fRAMework to comPAre aGd dEtectors*), focused on training and comparing machine learning models for AGD detection. Furthermore, we propose a new model that uses logistic regression and, using RAMPAGE to obtain a fair comparison with different state-of-the-art models, achieves slightly better results than those obtained so far. In addition, the dataset built from real-world samples for evaluation, as well as the source code of RAMPAGE, are also publicly released to facilitate its use and promote experimental reproducibility in this research field.

1. Introduction

In recent years, the number of cyberattacks has increased because cybercrime has become a profitable business that moves large amounts of money, even more profitable than the global illegal drug trade combined (Cybersecurityventures, 2023). As a result, techniques for developing malicious code (*malware*) are becoming more sophisticated, requiring continuous updating of prevention, detection, and response techniques.

Lockheed Martin's Cyber Kill Chain (Martin, 2023) describes the seven phases an attack must complete to be successful. Once the first five phases are completed (*reconnaissance*, *armament*, *delivery*, *exploitation*, and *installation*), the attacker has full control over the victim's machine. The attacker then establishes a command and control channel to communicate with the compromised system, allows them to issue commands and control the system remotely, often to move laterally within the network or exfiltrate data. This phase is known as *Command & Control* (C&C).

There are several techniques to establish such a communication channel with the C&C server. Many malware uses *constant* data to obtain the C&C address, such as an IP address string or a domain name in its binary data (Palo Alto, 2023). However, this data is very easy to extract (Yong Wong et al., 2021) and therefore the communication channel can be stopped very quickly. For instance, adding specific rules to firewalls or intrusion detection systems (Bejtlich, 2013).

To overcome these detections and continue establishing communication, malware began to incorporate techniques such as *Domain Generation Algorithms* (DGAs). Since their appearance in the `Conflicker` malware (Porras, Saïdi, & Yegneswaran, 2009), DGAs have been a very effective strategy for establishing communication between attackers and compromised systems.

A DGA pseudo-randomly generates many domain names known as *Algorithmically Generated Domains* (AGDs). For this technique to work, both the malware and the attacker must know the specific DGA and the initial seed used for the pseudo-random number generator. The attacker first registers one of the possible AGDs to deploy their C&C server. Meanwhile, the malware in the compromised system periodically generates new AGDs and tries to communicate with them. It will eventually connect to the registered AGD and thus establish the connection. When the C&C server is taken down, the attacker simply registers a new AGD and deploys again their C&C server infrastructure, and waits again for the compromised system to connect again.

Detection of malicious AGDs has been an important research topic in the last 15 years. To detect them, different solutions have been proposed,

* Corresponding author.
E-mail addresses: tpelayo@unizar.es (T. Pelayo-Benedet), rjrodriguez@unizar.es (R.J. Rodríguez), C.HernandezGanan@tudelft.nl (C.H. Gañán).

https://doi.org/10.1016/j.eswa.2025.128629
Received 25 October 2024; Received in revised form 29 May 2025; Accepted 13 June 2025
Available online 17 June 2025

mainly based on deep learning approaches (Drichel, von Querfurth, & Meyer, 2024; Woodbridge, Anderson, Ahuja, & Grant, 2016). When a new solution appears, they always claim to provide significant benefits compared to previous solutions, providing numerical experiments that support this claim. However, different pitfalls have been encountered in the machine learning research in cybersecurity, such as inappropriate baselines (Arp et al., 2023) or the use of non-reproducible methodology to define dataset and design experiments (Botacin, Ceschin, Sun, Oliveira, & Grégio, 2021). In this sense, comparison of new proposals becomes difficult since many works do not freely share their implementations or the datasets used for evaluation. This work aims to help solve this problem.

While existing DGA detection techniques have shown promising results, they face several key challenges that hinders progress in DGA detection research. First, the lack of standardized evaluation frameworks makes it difficult to fairly compare different approaches, as researchers often use varying datasets, metrics, and experimental setups. Second, many of the proposed solutions are not publicly available or lack sufficient documentation, impeding reproducibility and verification of results claimed in the literature.

Our software, dubbed RAMPAGE (*fRAMework to comPAre aGd dEtectors*), directly addresses these limitations by providing a standardized environment for training and evaluating DGA detection models. Furthermore, RAMPAGE's modular architecture allows researchers to easily compare different detection techniques, thereby accelerating innovation in this field while ensuring reproducibility and fair comparison of results.

To demonstrate the impact of RAMPAGE, we replicate a set of machine learning-based models proposed in the literature and use our tool to compare them. Furthermore, we then create a new meta-model that combines several previously deep learning networks, together with a logistic regression, to determine whether a given domain name is an AGD or not. Additionally, we employ SHAP analysis and ablation studies to gain insights into the internal dynamics of our meta-model and evaluate the contribution of each individual classifier to the overall performance.

Our experiments show that this new meta-model, while conceptually simple, offers several practical advantages: it achieves comparable and even slightly better results than current state-of-the-art models (Drichel et al., 2024; Woodbridge et al., 2016), while providing better interpretability and easier deployment in production environments. As a dataset for experiments, we use the Tranco (2023) list and (Tuan, Anh, Luong, & Long, 2023) for training and testing, as well as a dataset we constructed from one year of DNS requests from the University of Zaragoza to perform experiments with real-world domains. For the sake of open science and reproducibility, the source code of the framework is public and fully operational for use, along with the models and datasets used in the experiments performed in this paper.

In summary, the contribution of this paper is three-fold:

- We propose a software framework to train and compare approaches based on deep learning for DGA detection. This software, dubbed RAMPAGE, aims to improve and accelerate the development of new approaches for the detection of AGDs, providing a reproducible methodology and an adequate baseline for evaluation and comparison of models. The source code of RAMPAGE has been released under the GNU/GPLv3 license in a GitHub repository[1] for the scientific community to use and improve.
- We demonstrate a practical approach to combining existing neural networks using logistic regression. To analyze the internal dynamics and evaluate the contribution of each classifier within the meta-model, we perform SHAP analysis and ablation studies. We also analyze the time complexity to evaluate the computational overhead introduced by our approach. This results in a model that maintains competitive performance while reducing architectural complexity. By integrating established techniques, we show how existing methods can be effectively combined to create robust solutions suitable for real-world deployment.
- We developed a dataset of real-world domains. Specifically, we built a new dataset containing the domain names requested to the University of Zaragoza's DNS server between June 2023 and May 2024. This dataset allows us to perform more precise experiments with real-world domains and has been released to facilitate the evaluation and comparison of machine learning-based proposals. By making it publicly and freely available, we aim to support open science and promote reproducibility in research.

This paper is organized as follows. Section 2 presents previous concepts necessary to understand our work. Section 3 discusses related work. Section 4 describes the proposed software framework and new model. Section 5 presents the dataset constructed for the experiments, as well as the experiments carried out in this work. Discussion of results and limitations is provided in Section 6. Finally, Section 7 concludes the paper and sets out future work.

2. Background

This section aims to provide a comprehensive overview of two key areas for our work: DGAs and artificial neural networks.

2.1. Domain generation algorithms

A *Domain Generation Algorithm* (DGA) is an algorithm that generates domain names following behavior similar to pseudo-random number generators. Since these algorithms follow pseudo-random bases, they make the generated domains appear random. These algorithms need a seed, which is an initial value used by the algorithm to generate the sequence of pseudo-random domain names, ensuring that the same seed will always produce the same sequence of domains, allowing both the malware and its C&C servers to stay synchronized. Domains generated using DGAs are known as *Algorithmically Generated Domains* (AGDs) (MITRE, 2023).

Since their first appearance in malware (Porras et al., 2009), DGAs have evolved. Plohmann, Yakdan, Klatt, Bader, and Gerhards-Padilla (2016) characterize DGAs based on the seed source and based on the pseudo-random number generation algorithm (PRNGA). Most relevant for this work are the types of PRNGAs. Specifically, the PRNGA of a DGA can be Plohmann et al. (2016):

1. *arithmetic*, which calculates a sequence of values that are then transformed into ASCII characters to construct a domain;
2. *hash-based*, which uses a cryptographic hash function on certain values (such as the date) to construct a domain;
3. *dictionary-based*, which uses dictionaries to concatenate words and thus construct a domain; and
4. *permutation-based*, which permutes single characters or substrings from an original domain to construct a new domain.

The AGDs resulting from each type of DGA are very different. For example, a dictionary-based DGA generates domains that are readable (so they may appear to be legitimate domains), while domains from a hash- or arithmetic-based DGA are simply a sequence of meaningless characters, which look like random or rare words to the human eye.

2.2. Artificial neural networks

Artificial Neural Networks or simply *Neural Networks* (NNs) are computational models inspired by the structure and function of biological neural networks in the brain (Wu & Feng, 2018). An ANN is made up of artificial neurons interconnected in layers that process information in a way that mimics how human neurons communicate.

[1] https://github.com/reverseame/RAMPAGE

A NN consists of multiple layers: *input layer*, which receives the raw data and passes it to the next layer; a set of *hidden layers*, which process the data and extract relevant features using non-linear transformations; and an *output layer*, which produces the final result of the prediction or classification. In this regard, the term *deep learning* refers to neural networks with multiple hidden layers, allowing them to model more complex relationships in the data.

Forward propagation refers to the process of passing input data through the network, layer by layer, to produce an output (Svozil, Kvasnicka, & Pospichal, 1997). This is done by calculating weighted sums and applying *activation functions* on each neuron until the output is calculated. These activation functions introduce non-linearity into the network, allowing it to solve complex tasks beyond simple linear classification. Common activation functions include, but are not limited to, sigmoid, ReLU, and softmax (Ding, Qian, & Zhou, 2018).

NNs are trained to minimize a loss function, which quantifies the difference between the predicted output and the actual target value. Popular loss functions include mean squared error (for regression tasks) and cross-entropy (for classification problems). The process of minimizing the loss function is performed using optimization algorithms such as Gradient Descent (Amari, 1993), although some variants such as Adam (Zhang, 2018) and RMSprop (Huk, 2020) are commonly used to speed up convergence and improve performance.

The algorithm used to calculate the gradients of the loss function with respect to the network weights is called *backpropagation*. This algorithm allows the network to adjust the weights to minimize the loss and involves calculating the gradients from the output layer back to the input layer.

Like any other machine learning model, NN are evaluated using datasets that are typically split into three parts: *training set*, which is used to train the model; *validation set*, which is used to fine-tune the model parameters and prevent overfitting. Overfitting occurs when a NN learns to fit the training data too well, capturing noise and irrelevant details, which harms its ability to generalize to new data. To prevent this, regularization techniques such as dropout and L2 regularization, to name a few, can be used; and *test set*, which is used to evaluate the model's performance on unseen data.

NNs are widely used in various fields such as computer vision (Guo et al., 2022), natural language processing (Chowdhary, 2020), speech recognition (Malik, Malik, Mehmood, & Makhdoom, 2021) and time series forecasting (Benidis et al., 2022). NNs have also emerged as a useful technique for AGD detection due to their ability to process raw data with minimal preprocessing. This characteristic is particularly advantageous compared to traditional machine learning methods, which often require extensive feature engineering and domain-specific knowledge to accurately identify AGDs.

3. Related work

Many researchers have focused their efforts on AGD detection since its first appearance in 2008 (Porras et al., 2009). As a result of these efforts, various approaches to detect and mitigate the threats posed by DGAs have been proposed, with a notable emphasis on approaches leveraging neural networks. In this section, we provide a brief review of the state-of-the-art techniques for AGD detection approaches (particularly those based on neural networks), highlighting key contributions in this rapidly evolving field.

Table 1 summarizes the current state of the art in AGD detection using neural networks, considering the characteristics of interest for this work (model and dataset used). Below, we describe only the most relevant works that use NNs to identify AGDs. These works have been selected because they either introduce novel models that have not been used before or because they provide a new approach for applying NNs to detect AGDs.

Woodbridge et al. (2016) introduced the first AGD detection model using neural networks with a *Long Short-Term Memory* (LSTM) comprising 128 units. The proposed model has two different variants: a binary classification model (to differentiate between AGD and no AGD), and a multiclass classification model intended to identify specific DGA families. The study highlights the adaptability of LSTM networks in handling the variable lengths and complex structures of domain names generated by different DGA families. Another significant contribution to the field is the multi-class classifier based also on LSTM was given in Tran et al. (2018). This model achieves multiclass invariance by integrating a binary classifier with a multiclass classifier, offering a robust framework for distinguishing between various DGA families.

Yu et al. (2017) conducted a comparative study of DGA detection models, comparing the effectiveness of an LSTM as proposed by Woodbridge et al. (2016) with that of a one-dimensional convolutional layer. The authors proposed a novel model for DGA detection based on a convolutional neural network (CNN), demonstrating the versatility and potential of convolutional layers in this context. The comparative study provides valuable insights into the strengths and limitations of different neural network models for DGA detection. Specifically, the convolutional layers can capture local patterns within domain names, which are crucial for identifying characteristic substrings that are often indicative of domains generated by a DGA. Yang et al. (2018) proposed a more complex CNN model, which expands the hidden layers of the network and compares its results with models that only use LSTM layers (as in Woodbridge et al. (2016)).

Yu et al. (2018) presented an empirical comparison of various machine learning models for AGD detection. Using a dataset comprising 2 million domains, they demonstrated that simpler models often achieve superior results in the validation phase, emphasizing the importance of model simplicity and efficiency and the value of simplicity in machine learning model design. Similarly, Sivaguru et al. (2018) evaluated different classifiers, although the model details are not as detailed as those provided by Yu et al. (2018). Both works highlighted the need for a balanced approach to model complexity, where simplicity should not be sacrificed for the sake of incorporating advanced neural network architectures.

Vinayakumar et al. (2019) introduced a new model named Deep Bot Detect (DBD). This model is characterized by its simplicity and fast training process, which fortunately does not compromise its classification performance. The authors effectively demonstrated that an optimized model can achieve competitive results, contributing to the current discourse on how to balance model complexity and performance. The DBD model's ability to achieve high accuracy with reduced computational overhead presents important practical advantages, particularly in real-world deployment scenarios where real-time monitoring is a critical consideration.

Berman (2019) proposed a new model called 1D Capsule Network, which is compared with other models suggested in previous works. This new type of neural network emerged as a response to the shortcomings of convolutional networks (Sabour, Frosst, & Hinton, 2017). Other studies proposed new models also based on convolutional layers, such as Aloysius and Geetha (2017); Zhou et al. (2019). However, these works did not present significant advancements or novel model configurations. They also lack specificity when defining the values used in the models, which made them not reproducible. The lack of detailed configuration parameters in these studies highlights the importance of transparency and reproducibility in machine learning research (Arp et al., 2023; Botacin et al., 2021). Similarly, Curtin et al. (2019) explored the detection of AGDs using recurrent networks and additional DNS query information. However, despite using additional information beyond the domain name itself, their approach did not produce significant improvements in detection results. This finding suggests that incorporating auxiliary data sources does not necessarily translate into better model performance, underscoring the complexity of the AGD detection problem and the need for continued innovation in feature engineering and model design.

Table 1
Summary of previous studies.

Paper	Models used	Datasets used	Framework	Public
(Woodbridge et al., 2016)	LSTM	b) (2025) & c) (2025)	Keras	✓
(Lison & Mavroeidis, 2017)	RNN	Alexa (2025), DGArchive (2023) & Bambenek (2025)	Keras	
(Saxe & Berlin, 2017)	CNN	N/A	Keras	
(Yu, Gray, Pan, Cock, & Nascimento, 2017)	LSTM & CNN	N/A	Keras	✓
(Catania, Garcia, & Torres, 2018)	CNN	Alexa (2025) & Bambenek (2025)	N/A	
(Shi, Chen, & Li, 2018)	ELM	N/A	N/A	
(Tran, Mac, Tong, Tran, & Nguyen, 2018)	LSTM	Alexa (2025) & Bambenek (2025)	Keras	
(Yang et al., 2018)	LSTM & CNN	Umbrella (2025) & Netlab-360 (2025a)	Keras	
(Yu, Pan, Hu, Nascimento, & De Cock, 2018)	CNN, CMU, MIT, Baseline & MLP	Alexa (2025) & Bambenek (2025)	Keras	✓
(Berman, 2019)	LSTM, CNN & CapsNet	Alexa (2025) & Bambenek (2025)	Keras	✓
(Catania, García, & Torres, 2019)	CNN	Alexa (2025) & Bambenek (2025)	Keras	
(Choudhary et al., 2019)	LSTM, CNN, CMU, MIT & NYU	public DGAs, Alexa (2025), (Netlab-360, 2025a), DGArchive (2023), openDNS (2025) & Bambenek (2025)	N/A	
(Curtin, Gardner, Grzonkowski, Kleymenov, & Mosquera, 2019)	RNN	openDNS (2025) & N/A	Keras	
(Qiao, Zhang, Zhang, Sangaiah, & Wu, 2019)	LSTM	Alexa (2025) & Bambenek (2025)	N/A	
(Vinayakumar, Soman, Poornachandran, Alazab, & Jolfaei, 2019)	DBD	Alexa (2025), openDNS (2025), DGArchive (2023) & Bambenek (2025)	Keras	✓
(Xu, Shen, & Du, 2019)	CNN	Alexa (2025) & DGArchive (2023)	Keras	
(Zhou et al., 2019)	CNN	Alexa (2025) & DGArchive (2023)	Keras	
(Liu, Zhang, Chen, Fan, & Dong, 2020)	RCNN	Alexa (2025), Netlab-360 (2025a) & Bambenek (2025)	Keras	
(Sivaguru, Peck, Olumofin, Nascimento, & De Cock, 2020)	LSTM	Alexa (2025) & DGArchive (2023)	Keras	✓
(Yang, Liu, Dai, Wang, & Zhai, 2020)	HDNN	Umbrella (2025) & Netlab-360 (2025a)	N/A	
(Namgung, Son, & Moon, 2021)	Bi-LSTM	Alexa (2025) & Bambenek (2025)	Keras	✓
(Selvi, Rodríguez, & Soria-Olivas, 2021)	LSTM	Alexa (2025) & public DGAs	Keras	✓
(Shahzad, Sattar, & Skandaraniyam, 2021)	LSTM & Bi-LSTM	Alexa (2025), Bambenek (2025), Netlab-360 (2025a) & Umbrella (2025)	Pytorch	
(Huang, Zong, Shi, Wang, & Liu, 2022)	DPCNN	Alexa (2025), Bambenek (2025), DGArchive (2023) & Netlab-360 (2025a)	Keras	
(Liang, Chen, Wei, Zhao, & Zhao, 2022)	CNN	Alexa (2025), Majestic (2025), DGArchive (2023) & Netlab-360 (2025a)	Pytorch	
(Morbidoni, Spalazzi, Teti, & Cucchiarelli, 2022)	LSTM	Alexa (2025) & public DGAs	N/A	
(Suryotrisongko & Musashi, 2022)	PQCs	Alexa (2025) & public DGAs	Keras	
(Tuan, Long, & Taniar, 2022)	LSTM	Alexa (2025), Bambenek (2025), Netlab-360 (2025a) & public DGAs	Keras	
(Vranken & Alizadeh, 2022)	LSTM & MLP	Tranco (2023) & DGArchive (2023)	Keras	
(Maia et al., 2024)	LSTM, CNN & MLP	Alexa (2025) & DGArchive (2023)	N/A	
(Drichel et al., 2024)	ResNeXt, ConvNeXt, Transformer & RWKV	DGArchive (2023) & N/A	Keras	✓
(Cebere, Flueren, Sebastián, Plohmann, & Rossow, 2024)	MLP, RNN, GRU, LSTM, CNN, Transformer & ResNset	Tranco (2023), DGArchive (2023) & public DGAs	Pytorch	✓

N/A: Not available.

In recent years, several works have introduced more models for AGD detection using neural networks, such as (Selvi et al., 2021; Sivaguru et al., 2020; Yang et al., 2020). Namely, a model using a heterogeneous deep neural network is proposed in (Yang et al., 2020), while Sivaguru et al. (2020) conducted a study on the importance of each feature of a domain name for AGD detection. Likewise, Selvi et al. (2021) improve the configuration of an LSTM network by preprocessing the input data with the aim of achieving better results. Despite the innovative approaches presented, the first two studies are not very specific regarding the network layer configurations, making it difficult to reproduce the models. The third study, unlike the others, shares the developed configuration. The lack of detailed methodological documentation poses a significant barrier to replication and validation, highlighting again the need for rigorous standards when reporting machine learning research (Arp et al., 2023; Botacin et al., 2021). Nevertheless, these works represent important contributions to the continued evolution of AGD detection methodologies, demonstrating the potential of heterogeneous and feature-centric approaches to advance the state of the art. Drichel et al. (2024) propose a novel method based on transfer learning and fine-tunning. This new approach allows learning nuances of specific AGDs, which improves classification.

Finally, Cebere et al. (2024) reviews detection techniques from a meta perspective. This work highlights certain assumptions made, some of which, as explained, are erroneous. In addition, current problems identified in AGD detection are discussed.

As evidenced by the extensive literature reviewed, deep learning approaches have demonstrated significant strengths in detecting AGDs. The main advantage lies in their ability to automatically extract meaningful features from raw domain names, eliminating the need for manual feature engineering. Deep learning architectures, particularly LSTMs and CNNs, have consistently achieved high classification accuracy on various datasets and DGA families. The evolution from basic LSTM architectures (Woodbridge et al., 2016) to sophisticated transfer learning approaches (Drichel et al., 2024) highlights the continuous improvement in detection capabilities. Both LSTMs and CNNs have proven effective at capturing sequential patterns and structural regularities in domain names.

Despite these advances, significant challenges remain. Recent meta-analyses (Cebere et al., 2024) have exposed fundamental flaws in common assumptions about AGDs and legitimate domains, while the lack of detailed documentation in many studies impedes reproducibility (Arp et al., 2023; Botacin et al., 2021). The observation that simpler models

often match or outperform complex architectures (Vinayakumar et al., 2019; Yu et al., 2018), coupled with the limited scope of analyzing only the domain name structure (Curtin et al., 2019), suggests that the field would benefit from a shift toward more comprehensive and reproducible approaches rather than an over-reliance on increasingly complex architectures. To address these challenges, particularly the issue of reproducibility, we propose RAMPAGE as a framework that provides a standardized environment for implementing and comparing AGD detection approaches.

4. The RAMPAGE software framework

In this section, we describe our software framework, RAMPAGE (*fRAMework to comPAre aGd dEtectors*), which allows the comparison of different neural network models. After an exhaustive study of the programming languages and frameworks used in the literature for AGD detection (see Table 1), we found that the most used language by far is `Python` with its `Keras` library, as shown in Fig. 1. Therefore, our tool is implemented in `Python` and `Keras` to facilitate its adoption by the scientific community.

Fig. 2 shows the proposed structure for our software. As can be seen, the framework is made up of two main modules: the `Core` and the `Dataset Manager`, which are explained in more detail below.

The `Core` module works as the central component that interconnects all other modules and handles the execution logic. It manages the integration of the datasets that the classifiers will use for training or comparison, as well as the classifiers to be evaluated, acting as the foundation of the framework's architecture. In contrast, the `Dataset Manager` module functions exclusively as the dedicated handler for all dataset-related operations during the framework's runtime. It has several customization parameters to split the data into training, validation, and testing subsets. To do this, it uses `Data Element`, which represents an element of information from which the classifiers are trained. To simplify the process, `Dataset Manager` allows the user to read datasets from a file, requiring them to only develop the function that parses each line read from the file. This parsing flexibility allows users to tailor the framework to their specific needs, accommodating various data formats and structures without modifying the core functionality.

`Classifier` defines the base structure that all models must follow for training and comparison. Researchers using the framework must implement their models by extending this module, ensuring a standardized interface for all classifiers. Classifiers of interest for the evaluation and comparison must be added to the `Core` module in order for them to be run.

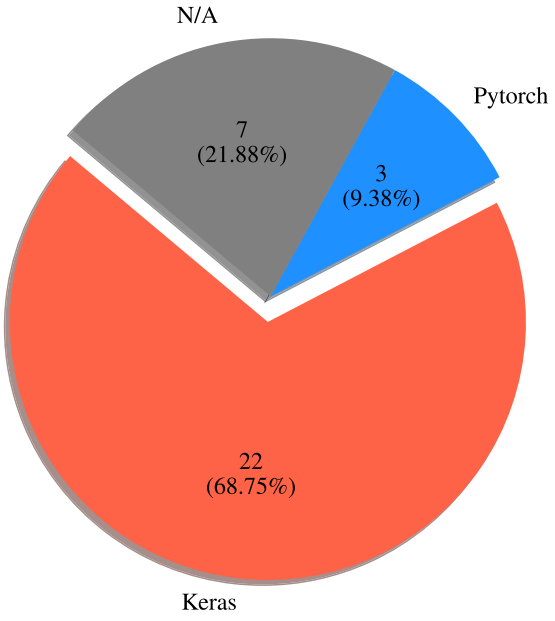

Fig. 1. Python frameworks used in the 32 reviewed state-of-the-art papers.

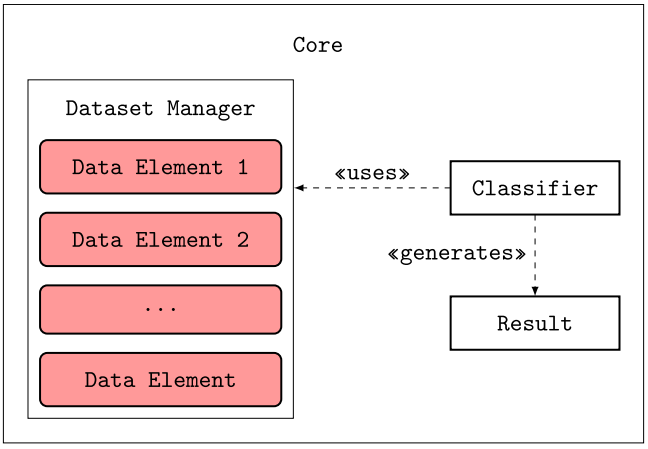

Fig. 2. Architectural diagram of RAMPAGE.

During the model comparison, the classifiers generate `Results`, which contain all the statistics obtained during the execution with the test dataset. As before, this must also be defined by the researcher.

To compare two or more models, it is mandatory that the new `DataElement` and `Result` can only extend or maintain the same structure as with the previous models. Otherwise, an update to the class fields from a `DataElement` or `Result` already processed or calculated can result can cause a failed execution and therefore the models cannot be compared.

For execution, it is first necessary to define the required `Classifiers` and add them to the `Core`. Next, the `Dataset Manager` must be defined and added to the `Core`, also incorporating the dataset of interest to be evaluated. With this, the framework is correctly initialized and ready to run. Once the run is complete, the models are trained, and the `Results` are generated. The framework can be configured to run only training or only testing. A full description of the framework, along with usage examples, is available in our GitHub repository.[2]

5. Evaluation

In this section, we evaluate RAMPAGE by answering the following research questions:

RQ1.- (RQ1.1) What specific challenges arise when comparing different models for AGD detection and (RQ1.2) how can a standardized framework address these issues effectively? (See Section 5.1).

RQ2.- What are the benefits and potential drawbacks of using metamodel and mixed models compared to single unmixed models in AGD detection? (See Section 5.2).

RQ3.- To what extent do academic models work effectively in detecting AGD in real-world scenarios? (See Section 5.3).

To answer these questions, we first present the datasets constructed for the experiments, as well as the experimental setup and metrics used. We then answer each question.

Construction of Datasets. Five different datasets were constructed for this work. The first three, D_1, D_2, and D_3, each contain 250,000 DGA-generated domains and 250,000 non-malicious domains, for a total of

[2] See https://github.com/reverseame/RAMPAGE

500,000 domains per dataset. These datasets are disjoint from each other. The fourth dataset, D_4, includes 7.5 million domains extracted from the University of Zaragoza DNS server logs, from June 2023 to May 2024 (347 days). Finally, D_5 consists of 2.9 million real AGDs from DGArchive (Plohmann et al., 2016).

D_1 is used to train and compare all the models evaluated in this work to perform an initial classification and determine which have achieved the best results, helping to address RQ1. D_2 and D_3 are used to address RQ2. Finally, D_4 and D_5 are used to evaluate RQ3.

The malicious AGDs in datasets D_1, D_2 and D_3 come from the `UTL_DGA22 dataset` (Tuan et al., 2023). This dataset comprises 76 different families of DGA domains and aims to provide a broad sample of potential DGA-generated domains without biases towards any specific family. To have a similar sample size per malware family, we randomly select $\lfloor 250,000/76 \rfloor$ AGDs from each family. The non-malicious domains have been obtained from the Tranco (2023) list.

D_4 was created from the domains recorded in the logs of the University of Zaragoza. Since DNS servers receive a high volume of domain resolution requests, a filtering process is necessary to ensure all domains meet the conditions of a well-formed domain. To ensure the validity of the analyzed domains, we implement several validation filters. The first filter checks the technical characteristics of the domain name: the total length must not exceed 253 characters; it must start with a letter and continue with a string containing only letters, numbers, and hyphens; this string cannot start or end with hyphens; and must be between 1 and 63 characters in length. The second filter focuses on domain classification: we use the Tranco list Tranco (2023) as a reference to identify non-malicious domains. Specifically, if both the SLD and TLD of a domain appear on the Tranco list, we classify it as non-malicious. The remaining domains are classified as unknown. It should be noted that while we use the Tranco list as a reliable source, we recognize the possibility that this list may contain some malicious domains, as pointed out by (Pochat, Goethem, & Joosen, 2019). Furthermore, we acknowledge that there exist more benign domains than those included in the Tranco list. Accordingly, this filtering approach is subject to possible improvements and may introduce classification errors. The remaining domains are categorized as unknown. Note that unknown domains may be either malicious or non-malicious, while non-malicious are considered benign because they are sub-domains of domains that belong to Tranco list.

Let us clarify that the goal of the D_4 dataset is not to capture the complete diversity of real-world AGDs. Rather, it aims to provide a representative sample of domains observed on a production DNS server, specifically in the context of a university environment such as the University of Zaragoza. This allows us to better understand the types of domains, both legitimate and potentially malicious, that security analysts are likely to encounter on similar networks. While we acknowledge that this dataset may not cover all DGA families or emerging threats, it does provide valuable insight into the domain resolution patterns typical of academic DNS infrastructures, offering a realistic basis for testing and evaluating AGD detection techniques.

D_5 comprises 2.9 million real-world AGDs from DGArchive (Plohmann et al., 2016), representing 58 different malware families. To construct this dataset, we first selected malware families with at least 50,000 domains. We then randomly sampled 50,000 AGDs from each of these families to ensure a diverse and representative sample of common malware families in the real world.

Experimental Setup. The experiments were performed on a ThinkPad E14 Gen 2 machine equipped with an Intel Core i7-1165G7 processor (8 cores, up to 4.7GHz), 32 GiB of RAM, and running Arch Linux 2024.06.01. All experiments were run in identical software environments, using Python 3.12.3, Keras 3.5.0, and TensorFlow 2.17.0. All additional dependencies and their exact versions are listed in the `requirements.txt` file, available in our repository. To facilitate complete reproducibility, complete source code and experimental configurations are publicly available in our GitHub repository (see footnote 1) under the GNU/GPLv3 license. The specific version of the codebase used in this work is v1.1.0. The repository includes detailed installation instructions, usage examples, and experimental procedures to facilitate accurate replication of our results.

Deep Learning Models. A total of 17 models have been implemented on RAMPAGE. Specifically, models containing a LSTM layer (Berman, 2019; Selvi et al., 2021; Woodbridge et al., 2016; Yang et al., 2018; Yu et al., 2017), convolutional networks (Yang et al., 2018; Yu et al., 2017, 2018), a combined LSTM and convolutional network (Berman, 2019), two versions of Tweet2Vec (CMU and MIT) (Dhingra, Zhou, Fitzpatrick, Muehl, & Cohen, 2016; Yu et al., 2018), an Parallel CNN network (Yu et al., 2018), a Baseline network (Yu et al., 2018), an MLP network (Yu et al., 2018), a convolutional network with max pooling (Berman, 2019), a network with a bidirectional LSTM layer (Berman, 2019) and a DBD network (Vinayakumar et al., 2019). To ensure consistency and allow for fair comparison, all models were trained using the hyperparameter settings described in Appendix B.

Due to the complexity of 1D Capsule Network (Berman, 2019) model, it has not been possible to implement and compare it with the other models in this work. In a similar way, due to technical issues with (Drichel et al., 2024) implementation, it has not been possible to reproduce their models despite them being available to use.

Meta-model. The proposed meta-model implements an ensemble approach that combines predictions from multiple deep learning models via logistic regression. Fig. 3 illustrates the architectural design of this ensemble system. The architecture can be mathematically expressed as follows:

Let $x = [x_1, \ldots, x_n]^T$ be an input domain name represented as a sequence of characters, where each x_i represents a character. The meta-model $\mathcal{M}$ processes this input through 7 independent deep learning models, where each model m_j generates a probability $p_j = m_j(x), \quad j \in \{1, \ldots, 7\}$.

These probabilities form a feature vector $p = [p_1, p_2, p_3, p_4, p_5, p_6, p_7]^T$, which serves as input to a logistic regression layer. The logistic regression then models the probability as (Yu, Huang, & Lin, 2011) $P_{LR}(p) = \sigma(w^T p)$, where $w \in \mathbb{R}^7$ is the weight vector, and σ is the sigmoid function defined as $\sigma(z) = \frac{1}{1+e^{-z}}$.

This architecture allows the meta-model to learn optimal weights w that combine the predictions of the individual models in a way that maximizes the overall performance. The logistic regression layer ensures that the output $P_{LR}(p)$ is bounded within the interval $[0, 1]$, providing a valid probability score, while simultaneously learning an optimal combination of the individual deep learning models predictions.

Metrics. The metrics used are accuracy (*Acc*), precision (*Prec*), recall (*Rec*), F1-score (*F1*), false positive rate (*FPR*), true positive rate (*TPR*), Matthew's correlation coefficient (*MCC*), and Cohen's Kappa coefficient (κ), where the formulas for each metric are defined as follows:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Prec = \frac{TP}{TP + FP} \tag{2}$$

$$Rec = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = 2 \cdot \frac{Prec \cdot Rec}{Prec + Rec} \tag{4}$$

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

$$TPR = \frac{TP}{TP + FN} \tag{6}$$

$$MCC = \frac{TN \cdot TP - FN \cdot FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{7}$$

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \text{where} \tag{8}$$

$$p_o = \frac{(TN + TP)}{(TN + FP + FN + TP)} \tag{9}$$

$$p_e = \frac{(TN + TP)(TN + FN)}{(TN + FP + FN + TP)^2} +$$

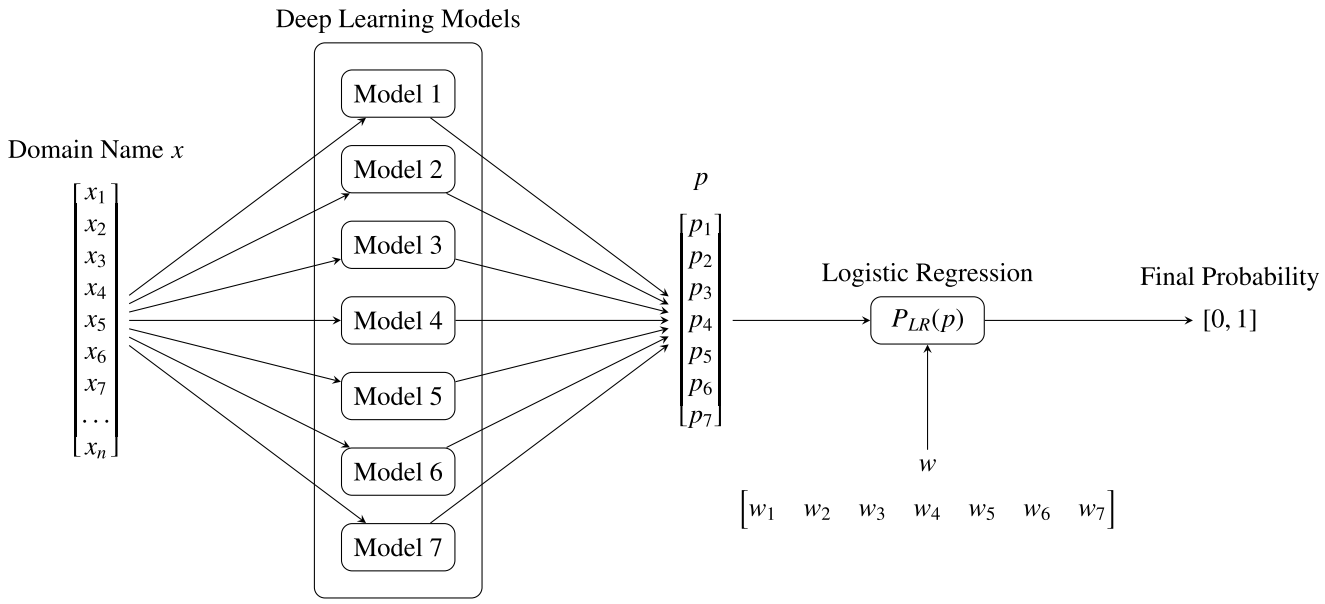

Fig. 3. Architecture of the proposed meta-model.

$$\frac{(FN+TP)(FP+TP)}{(TN+FP+FN+TP)^2} \tag{10}$$

True Negative (TN) and True Positive (TP) are, respectively, the number of legitimate domains correctly classified as non-AGD and the number of AGDs correctly classified as AGDs. In contrast, False Negative (FN) and False Positive (FP) are, respectively, the number of AGDs incorrectly classified as legitimate and the number of legitimate domains incorrectly classified as AGDs.

Accuracy, precision, recall, and F1-score measure the classification efficiency of the models. FPR and TPR measure the rate of false positives and true positives among the classified domains. MCC is used as an indicator of quality of classification, and κ indicates the effect of randomness on the proportion of agreement observed (the closer κ is to 1, the greater the degree of agreement; and vice versa).

5.1. Q1: challenges in comparing AGD detection models and the role of standardized frameworks

To answer Q1, we need to gather and analyze specific information related to the challenges in comparing different deep learning models for AGD detection. To do so, we first implement and evaluate all the 17 models using the same dataset. We use the same dataset for training, validation, and subsequently testing across all models (specifically, D_1). This avoids any variability in dataset composition, size, and diversity that could lead to inconsistent performance between models, making it difficult to draw definitive conclusions.

Table 2 shows the results of this experiment. From these results we can draw several conclusions, discussed below, thus answering Q1.

Inconsistency in Performance Metrics. Comparison of different models revealed variability in performance metrics such as accuracy, precision, false positive rate, and other key indicators. For instance, the MIT model (Yu et al., 2018) achieves the highest accuracy (95.48 %) and F1-score (96.59 %), but the CMU model (Yu et al., 2018) has the highest precision (97.46 %) and the lowest FPR (4.92 %). This variability complicates direct comparisons and the selection of the "best" model, as different metrics may prioritize different aspects of performance.

In this sense, a standardized framework can define a set of core evaluation metrics that all models must report, such as accuracy, precision, recall, F1 score, FPR, TPR, MCC, and κ, among others. This eliminates discrepancies in how performance is reported, ensuring that model comparisons are performed consistently. Similarly, by defining and enforcing the same procedures for calculating metrics, the framework helps avoid variations due to differences in computation or interpretation of metrics. Additionally, a standardized framework can also provide a benchmark for comparison, where all models are assessed against the same baseline or reference, thus ensuring fair and reliable evaluation.

Overfitting in Complex Models. Complex models tend to overfit when dealing with many different AGD families. This overfitting occurs because complex models need to adjust more weights, leading to poor generalization to new, unseen data. In this sense, the results indicated that simpler models, such as the LSTM models in Woodbridge et al. (2016); Yu et al. (2017), generally perform well across multiple metrics, while some more complex models, like Berman (2019), exhibit poorer performance (e.g., accuracy of 83.88 % and F1-score of 86.99 %). This presents a challenge in model selection, where simpler models can outperform complex ones under certain conditions. Let us recall that while simpler models tend to generalize better due to their limited ability to memorize the training data, they may not capture complex patterns that are essential for distinguishing subtle differences between various AGD families. This can result in underfitting, where the model performs poorly because it is too simplistic.

In this regard, using a standardized framework can enforce consistent regularization techniques and hyperparameter tuning procedures across models, thereby reducing the risk of overfitting by maintaining uniform practices for managing model complexity. Likewise, using the same cross-validation methods ensures that all models are evaluated in a way that accurately reflects their generalizability, helping to mitigate the impact of overfitting. Finally, it can make it easier to track performance metrics across different datasets and conditions, providing insights into whether complex models are truly overfitting or if the problem lies elsewhere.

Balancing Complexity and Generalization. Complex models, such as the MIT (Yu et al., 2018) and DBD (Vinayakumar et al., 2019) networks, show strong performance. These results indicate that, when well regularized and properly tuned, complex models can achieve high accuracy and generalize well. However, this requires careful management of model complexity using techniques such as regularization, cross-validation, and hyperparameter tuning. In contrast, as discussed above, the results also indicate that simpler models tend to adapt better to

Table 2
Results obtained using D_1 and the 17 implemented models.

Model (Reference)	Acc	Prec	Rec	F1	FPR	TPR	MCC	κ
LSTM (Woodbridge et al., 2016)	95.42	97.39	95.69	96.53	5.12	95.69	89.82	0.8045
LSTM (Yu et al., 2017)	95.44	97.25	95.87	96.55	5.40	95.87	89.84	0.8059
CNN (Yu et al., 2017)	94.96	97.39	94.98	96.17	5.07	94.98	88.86	0.7849
LSTM (Yang et al., 2018)	95.02	96.82	95.67	96.24	6.27	95.67	88.88	0.7896
CNN (Yang et al., 2018)	92.94	96.29	92.99	94.61	7.16	92.99	84.49	0.7056
CMU (Yu et al., 2018)	94.87	**97.46**	94.77	96.10	**4.92**	94.77	88.69	0.7810
MIT Yu et al. (2018)	**95.48**	96.96	**96.23**	**96.59**	6.03	**96.23**	**89.87**	**0.8083**
Parallel CNN (Yu et al., 2018)	93.48	96.64	93.48	95.03	6.49	93.48	85.68	0.7265
Baseline (Yu et al., 2018)	86.51	93.36	85.87	89.46	12.19	85.87	71.31	0.4745
MLP (Yu et al., 2018)	92.59	96.41	92.32	94.32	6.86	92.32	83.84	0.6907
CNN (Berman, 2019)	95.28	97.08	95.81	96.44	5.76	95.81	89.48	0.7998
Max Pooling (Berman, 2019)	90.48	95.62	89.84	92.64	8.21	89.84	79.53	0.6107
LSTM (Berman, 2019)	92.40	96.98	91.44	94.13	5.68	91.44	83.67	0.6804
LSTM + CNN (Berman, 2019)	83.88	94.12	80.87	86.99	10.09	80.87	67.44	0.3796
Bidireccional (Berman, 2019)	93.40	95.92	94.10	95.00	8	94.10	85.33	0.7261
DBD (Vinayakumar et al., 2019)	94.19	96.92	94.28	95.58	5.98	94.28	87.18	0.7545
LSTM (Selvi et al., 2021)	88.09	86.59	90.13	88.33	13.95	90.13	76.24	0.6247

Acc: Accuracy; **Prec**: Precision; **Rec**: Recall; **F1**: F1-score; **FPR**: False Positive Rate; **TPR**: True Positive Rate; **MCC**: Matthews's Correlation Coefficient; κ: Cohen's Kappa Score.

more variations and perform consistently across different metrics. In conclusion, balancing model complexity and generalization remains a challenge, as simpler models might miss intricate patterns that complex models can capture.

Regarding this challenge, the use of a standardized framework allows systematic experimentation with different levels of model complexity under controlled conditions. This includes using techniques such as regularization and tuning to ensure that both simple and complex models are evaluated equally. Similarly, the framework can also provide guidelines for hyperparameter optimization, helping to find the best settings for simple and complex models, thus improving their balance between complexity and generalization. The framework also provides a standardized approach to evaluating simple and complex models, helping to understand how well each model generalizes to unseen data and captures underlying complex patterns.

Consistency in Evaluation and Reporting. Challenges can arise from differences in how models are applied and interpreted. Variations in aspects such as preprocessing techniques, specific hyperparameter settings, and handling different data splits can impact performance metrics, even within a standardized framework like the one we use here. Ensuring that all models are evaluated under identical conditions and with the same rigor is essential to avoid misleading comparisons. Any subtle differences in implementation or dataset handling could still lead to variations in the reported results, which may influence the overall interpretation of model performance.

In this case, using a standardized framework ensures that preprocessing steps, data splits, and other handling methods are applied consistently across all models. This eliminates inconsistencies that arise from variations in data preparation. As for reporting, it can provide a structured way to document and report evaluation procedures and results, helping to understand the conditions under which each model was tested and reducing the impact of any subtle differences in implementation. Additionally, automated tools within the framework can generate consistent reports and visualizations, ensuring that performance comparisons are based on the same evaluation criteria and reducing the likelihood of human error.

5.2. Q2: meta-model vs. single models for AGD detection

Here, we evaluate the benefits and drawbacks of using meta-model and mixed models instead of single, unmixed models in AGD detection. To do this, we define a logistic regression model that integrates the results of our top-seven-performing neural network models shown above. This logistic regression takes the probabilities generated by each individual model –which indicates the likelihood of a domain being AGD-generated– as input features. By aggregating these probabilities, the logistic regression model is trained to produce a single probability score for each domain. This approach allows us to leverage the strengths and mitigate the weaknesses of individual models, providing a comprehensive analysis of whether meta-models offer significant performance improvement over stand-alone models in AGD detection scenarios. All models are available as examples in our GitHub repository, with their respective hyperparameters detailed in Appendix A. For consistency, we retain the original hyperparameters from previous works without making any additional adjustments.

Table 3
Results of the ablation study, illustrating the performance impact of removing each classifier from the meta-model.

Classifier removed	Δ accuracy	Δ F1 score	Δ MCC	Δ κ
LSTM (Woodbridge et al., 2016)	0.0010	0.0010	0.0020	0.0019
LSTM (Yu et al., 2017)	0.0012	0.0012	0.0024	0.0022
MCU (Yu et al., 2018)	0.0004	0.0004	0.0008	0.0007
MIT (Yu et al., 2018)	0.0015	0.0016	0.0031	0.0029
CNN (Berman, 2019)	0.0015	0.0015	0.0030	0.0028
DBD (Vinayakumar et al., 2019)	0.0001	0.0001	0.0002	0.0002
Parallel CNN (Yu et al., 2018)	0.0005	0.0005	0.0010	0.0009

In response to Q2, we conducted an extensive set of studies to evaluate the effectiveness and performance improvement achieved through our meta-model approach, as described below.

Understanding the Internal Dynamics of the Meta-model. To better understand the inner workings of our meta-model, we performed a SHapley Additive exPlanations (SHAP) (Lundberg & Lee, 2017) analysis and an ablation study (Sheikholeslami, 2019). SHAP results explain the impact of each feature (in this case, each model's prediction) on the overall result. This facilitates the interpretation of the model's decision-making process, making it more transparent and understandable. On the other hand, the ablation study helps assess the importance of each classifier within the meta-model by systematically removing one classifier at a time and observing the resulting performance change. This allows us to determine each model's contribution to overall performance and identify whether any classifiers can be removed without significantly affecting the meta-model's predictive performance.

Fig. 4 illustrates the contribution of each individual model to the final prediction, as determined by the SHAP analysis, while Table 3 shows the performance impact when each classifier is removed from the meta-model, based on the ablation study.

Table 4
Evaluation of D_3 in the 7 best models (individual) and logistic regression (combines the 7 models).

Model	Acc	Prec	Rec	F1	FPR	TPR	MCC	κ
LSTM (Woodbridge et al., 2016)	95.41	95.81	94.97	95.39	4.14	94.97	90.83	0.8307
LSTM (Yu et al., 2017)	95.65	95.35	95.98	95.66	4.67	95.98	91.30	0.8408
MCU (Yu et al., 2018)	95.64	96.07	95.17	95.62	3.88	95.17	91.29	0.8385
MIT (Yu et al., 2018)	95.02	94.81	95.26	95.03	5.20	95.26	90.05	0.8197
DBD (Vinayakumar et al., 2019)	94.55	94.35	94.78	94.57	5.66	94.78	89.11	0.8044
CNN (Berman, 2019)	95.71	96.31	95.06	95.68	3.36	95.06	91.43	0.8403
Parallel CNN (Yu et al., 2018)	91.87	91.56	92.25	91.90	8.49	92.25	83.76	0.7220
Our proposed meta-model	**96.68**	**97.04**	**96.29**	**96.66**	**2.93**	**96.29**	**93.36**	**0.8746**

Acc: Accuracy; **Prec**: Precision; **Rec**: Recall; **F1**: F1-score; **FPR**: False Positive Rate; **TPR**: True Positive Rate;**MCC**: Matthews's Correlation Coefficient; κ: Cohen's Kappa Score.

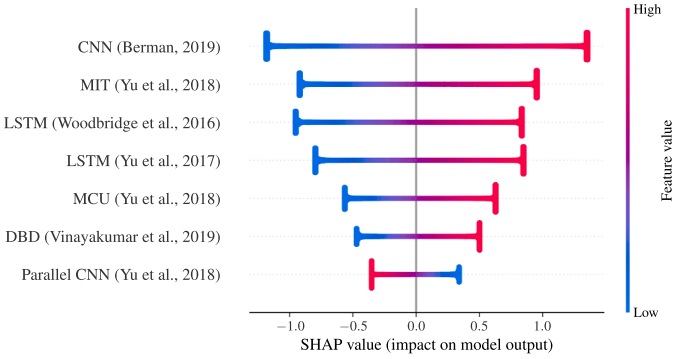

Fig. 4. Impact of individual models on meta-model prediction (SHAP analysis results).

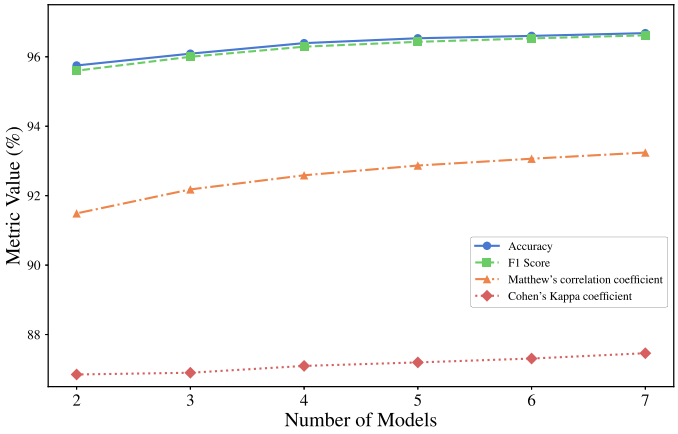

Fig. 5. Meta-model performance metrics as a function of the number of constituent models.

The results reveal a clear pattern of hierarchical influence, with the CNN (Berman, 2019) model exhibiting the highest impact range (−1.0 to +1.5), followed by the MIT (Yu et al., 2018) and LSTM (Woodbridge et al., 2016) models, which exhibit moderate influence ranges (−0.75 to +1.0). SHAP analysis shows that most models contribute positively to the final correlation patterns, meaning higher probability outputs tend to contribute positively to the final meta-model decision, with higher likelihood scores being positively correlated. However, an exception is observed with the Parallel CNN (Yu et al., 2018) model, which exhibits an inverse correlation pattern, as further elaborated below.

The effectiveness of our meta-model approach is further enhanced by the complementary nature of the individual models' contributions. While some models, such as CNN (Berman, 2019) and MIT (Yu et al., 2018), exert a broader impact and significantly influence the final prediction, others, such as DBD (Vinayakumar et al., 2019) and MCU (Yu et al., 2018), provide more specialized contributions within specific probability ranges. This complementary interaction allows the meta-model to leverage broad and focused detection capabilities, improving overall robustness.

Our ablation study further validates the contributions of individual models to the overall meta-model performance. Table 3 shows the impact of removing each classifier from the meta-model on each metric. The Δ represents the change in the corresponding performance metric when the specific classifier is removed from the meta-model. The results are consistent with our SHAP analysis, which reveals that the MIT (Yu et al., 2018) and CNN (Berman, 2019) models have the most significant influence, with a 0.15 % decrease in accuracy and F1-score when removing either one. These findings suggest that our meta-model does not rely heavily on a single classifier, demonstrating a well-distributed contribution across models.

Extending the single-model ablation study, we performed a comprehensive study in which we systematically varied the number of constituent models in the meta-model. Specifically, we evaluated all possible model combinations for each group size, from 2 to 7, and calculated average performance metrics for these combinations, grouped by the number of models included. Fig. 5 presents the resulting performance trends across all evaluation metrics as the meta-model size increases.

As shown in Fig. 5, all evaluation metrics show a steady upward trend with the inclusion of additional models in the meta-model. While the performance improvements are incremental, the aggregate improvements underscore the effectiveness of the ensemble strategy. These results indicate that the meta-model leverages the complementary strengths of the individual classifiers, with each additional model contributing distinct detection capabilities that improve the overall system performance.

A notable observation is the distinctive behavior of the Parallel CNN (Yu et al., 2018) model, which exhibits an inverse correlation pattern compared to other models. While most models contribute positively when their output probabilities are high, the Parallel CNN demonstrates a stronger influence on lower probability outputs. This unique behavior, combined with the systematic distribution of influence across multiple models, ensures that the final prediction is not overly dependent on any single model, reducing susceptibility to bias or failure of individual models and providing a more reliable detection mechanism.

Meta-model Performance Evaluation. We train the top seven models along with the logistic regression meta-model using D_2 for training and evaluate their performance on D_3. The results of this experiment, presented in Table 4, show a significant advantage of employing a meta-model approach for AGD detection.

Our proposed meta-model, which integrates the likelihood results of the seven individual models via logistic regression, consistently achieves the highest scores across all performance metrics. These results highlight the effectiveness of the meta-model in aggregating the strengths of the individual models, leading to improved performance and robustness compared to any individual model.

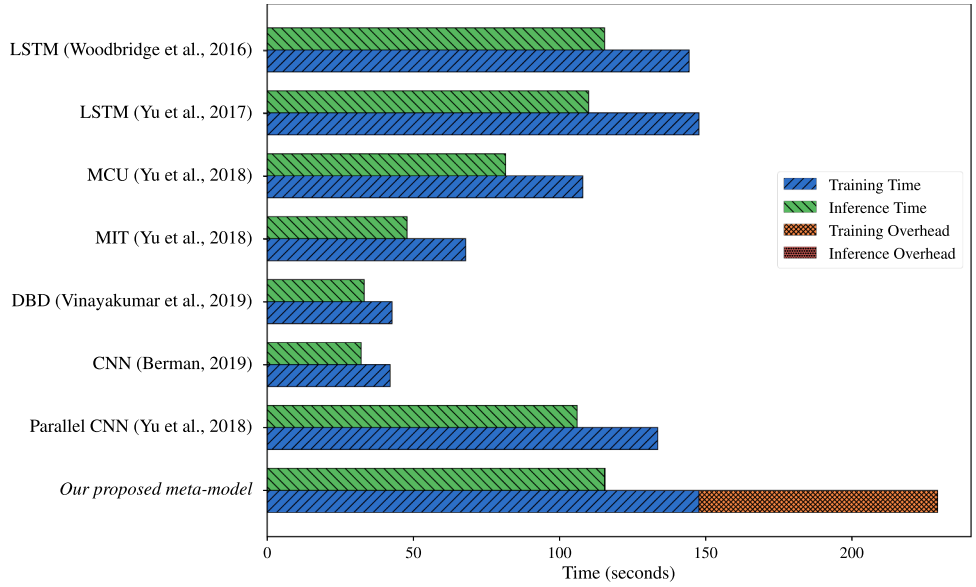

Fig. 6. Comparison of training times (on D_2) and inference times (on D_3) for individual models and the meta-model.

Furthermore, the consistency and improvements observed across several evaluation metrics reinforce the benefits of this ensemble-based approach. These findings validate our initial hypothesis that metamodels can more effectively manage the complexities and variations inherent in AGD data, ultimately improving overall detection capabilities and ensuring more reliable and adaptive threat detection.

Time Complexity Analysis of the Meta-model vs. Individual Models. A critical factor in evaluating the viability of our meta-model approach is its computational efficiency compared to standalone models. While performance metrics indicate a slight advantage in detection accuracy for the meta-model, a holistic evaluation must also consider the time spent in the training and inference phases. To do so, we measure the training time required to fine-tune each individual model, as well as the meta-model, along with the corresponding inference time for domain classification.

Fig. 6 presents a comparative overview of the training and inference times of the seven individual models and the proposed meta-model. A key advantage of our meta-model architecture is that the component models can be run in parallel, substantially reducing the computational overhead that would otherwise occur with sequential execution. During training, we observe an additional overhead of approximately 80 seconds for the meta-model, compared to training the individual models. However, this cost is incurred only once and does not affect runtime performance after deployment.

From a deployment perspective, inference time is the most critical metric. In our experiments, the meta-model demonstrated minimal inference overhead. Since inference is performed in parallel across all component models, the total inference time is largely bounded by the slowest model, followed by a lightweight logistic regression computation for the final prediction. The negligible additional latency introduced by this last step confirms that the meta-model achieves its performance improvements without imposing significant execution costs. These results confirm the viability of our meta-model in real-world AGD detection systems, where operational efficiency is a determining factor.

Resource Usage Analysis. To evaluate the practical viability of our meta-model approach in hardware-constrained operating environments, we measured the computational resource consumption during the training and inference phases. In our implementation, each individual model runs on a dedicated CPU core at 100 % utilization. As a result, the meta-model, composed of seven parallel classifiers, operates simultaneously on seven CPU cores at full capacity.

Table 5
Maximum RAM usage during training (in D_2) and inference (in D_3).

Model	Training (GiB)	Inference (GiB)
LSTM (Woodbridge et al., 2016)	1.34	1.15
LSTM (Yu et al., 2017)	1.60	1.30
MCU (Yu et al., 2018)	1.72	1.38
MIT (Yu et al., 2018)	2.13	1.62
DBD (Vinayakumar et al., 2019)	2.18	1.65
CNN (Berman, 2019)	2.02	1.58
Parallel CNN (Yu et al., 2018)	2.26	1.72
Our proposed meta-model	5.63	4.28

Table 5 summarizes the peak RAM usage for the training and inference phases for the seven highest-performing individual models, as well as our proposed meta-model.

As shown, our meta-model exhibits higher peak RAM consumption in both phases, reflecting the cumulative memory footprint of the concurrently executed base models. Despite this increased usage, resource demands remain within the capabilities of contemporary multicore systems. RAM consumption is highly dependent on the size of the underlying dataset and the architectural complexity of each component model. Overall, these measurements confirm that our meta-model is suitable for deployment in production environments with standard hardware configurations, supporting its operational feasibility under practical conditions.

Statistical Validation of Meta-model Performance. To rigorously validate the performance of the models, we perform a Scott-Knott ESD (Effect Size Difference) analysis (Tantithamthavorn, McIntosh, Hassan, & Matsumoto, 2019). This statistical method employs hierarchical clustering to divide treatment means into statistically distinct groups based on statistical significance and effect size, providing insight into significant differences between and within groups. Fig. 7 illustrates the results of the Scott-Knott ESD analysis on four key performance metrics, while Table 6 presents the corresponding detailed statistical measures.

The results of the analysis reveal a consistent clustering pattern across all performance metrics, with the models forming four statistically distinct groups (G1-G4). The meta-model achieves the highest performance across all metrics and is consistently placed in its own group (G1), confirming its superiority. Next, a group of five models (specifically, CNN (Berman, 2019), LSTM (Yu et al., 2017), MIT (Yu et al., 2018), LSTM (Woodbridge et al., 2016), and MCU

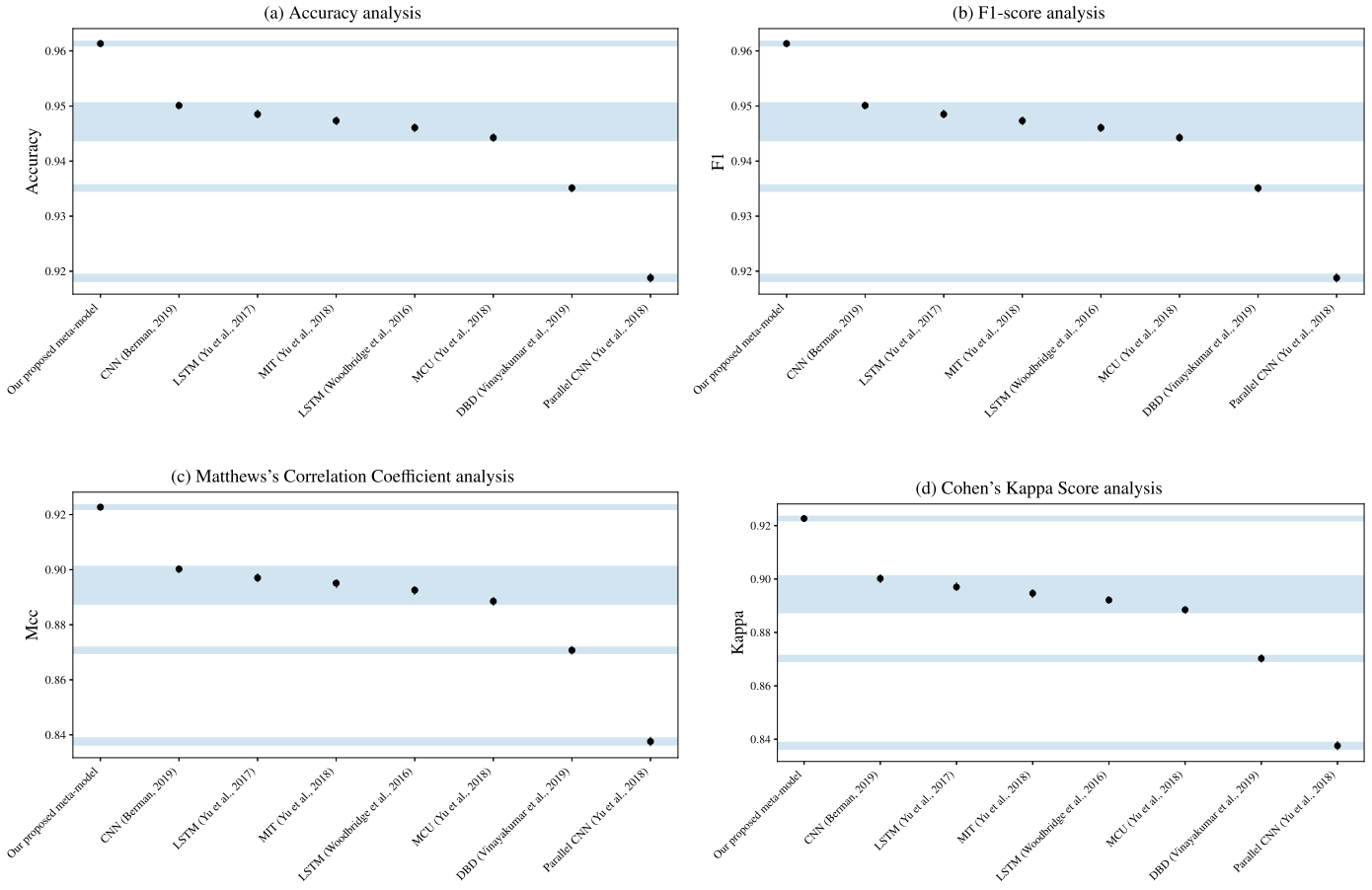

Fig. 7. Scott-Knott ESD analysis for different performance metrics.

(Yu et al., 2018)) form the second group (G2), with the CNN model leading this group. The DBD (Vinayakumar et al., 2019) model constitutes a separate group (G3), while the Parallel CNN (Yu et al., 2018) consistently ranks lowest across all metrics, forming the final group (G4).

The statistical validation provided by the Scott-Knott ESD analysis offers compelling evidence of the superior performance of the meta-model. As illustrated in both Fig. 7 and Table 6, the meta-model not only outperforms all individual models in terms of absolute performance but also shows statistically significant improvements, justifying its distinctive classification in the G1 group. Furthermore, narrow confidence intervals (CI) and low standard deviations (σ) across all metrics indicate that the superior performance of the meta-model is consistent and reliable. These findings strongly support the efficacy of our approach to aggregating individual models into a meta-model, as it consistently outperforms even the best-performing stand-alone models by a statistically significant margin. This validation further reinforces the robustness and reliability of our proposed ensemble approach for AGD detection.

Calibration Analysis of the Meta-model. In addition to performance metrics, we evaluate the quality of our meta-model's calibration (Trucano, Swiler, Igusa, Oberkampf, & Pilch, 2006) to assess the adequacy of its probability estimates to the actual class probabilities. Fig. 8 shows the distribution of predicted probabilities for the benign and AGD domains in D_3.

The clear separation between classes highlights the meta-model's confidence in its classifications. As shown in Table 7, approximately 92% of all instances fall into the extreme probability ranges (0.0-0.1 and 0.9-1.0), further confirming the model's decisive classification behavior.

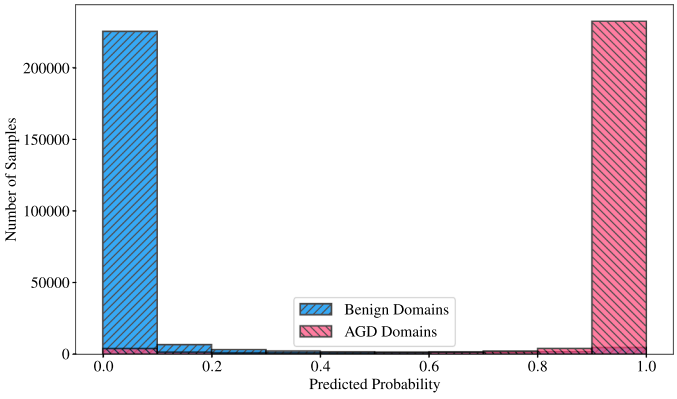

Fig. 8. Distribution of predicted probabilities for the benign and AGD domains in D_3.

To visualize the calibration precision, we generate a reliability diagram (see Fig. 9) comparing predicted probabilities with observed frequencies across the 10 equal-width probability bins detailed in Table 7.

The calibration analysis in Table 7 reveals several key findings: (i) our meta-model demonstrates strong overall calibration, with a Brier score of 0.0307 and an Expected Calibration Error (ECE) of 0.0098; (ii) the model shows good calibration precision at the probability extremes, with minimal calibration errors of 0.0011 in the 0.0 to 0.1 interval and 0.0081 in the 0.9 to 1.0 interval; and (iii) calibration errors increase in the middle probability ranges, with a Maximum Calibration Error (MCE)

Table 6
Detailed performance metrics for all models with Scott-Knott ESD grouping.

Model	Accuracy				F1-score				MCC				Kappa			
	μ	σ	CI	G	μ	σ	CI	G	μ	σ	CI	G	μ	σ	CI	G
Our proposed meta-model	0.9613	0.0003	[0.9608, 0.9619]	1	0.9613	0.0003	[0.9608, 0.9619]	1	0.9227	0.0005	[0.9216, 0.9238]	1	0.9227	0.0005	[0.9216, 0.9237]	1
CNN (Berman, 2019)	0.9501	0.0003	[0.9495, 0.9507]	2	0.9501	0.0003	[0.9495, 0.9507]	2	0.9002	0.0006	[0.8990, 0.9014]	2	0.9002	0.0006	[0.8990, 0.9014]	2
LSTM (Yu et al., 2017)	0.9485	0.0003	[0.9479, 0.9491]	2	0.9485	0.0003	[0.9479, 0.9491]	2	0.8970	0.0006	[0.8958, 0.8983]	2	0.8970	0.0006	[0.8958, 0.8983]	2
MIT (Yu et al., 2018)	0.9473	0.0003	[0.9467, 0.9480]	2	0.9473	0.0003	[0.9467, 0.9480]	2	0.8951	0.0007	[0.8938, 0.8964]	2	0.8946	0.0007	[0.8933, 0.8960]	2
LSTM (Woodbridge et al., 2016)	0.9461	0.0003	[0.9455, 0.9467]	2	0.9460	0.0003	[0.9454, 0.9467]	2	0.8926	0.0006	[0.8914, 0.8938]	2	0.8921	0.0006	[0.8909, 0.8933]	2
MCU (Yu et al., 2018)	0.9442	0.0003	[0.9436, 0.9449]	2	0.9442	0.0003	[0.9436, 0.9449]	2	0.8885	0.0007	[0.8872, 0.8898]	2	0.8885	0.0007	[0.8872, 0.8898]	2
DBD (Vinayakumar et al., 2019)	0.9351	0.0003	[0.9344, 0.9358]	3	0.9351	0.0003	[0.9344, 0.9358]	3	0.8707	0.0007	[0.8694, 0.8721]	3	0.8702	0.0007	[0.8689, 0.8716]	3
Parallel CNN (Yu et al., 2018)	0.9188	0.0004	[0.9180, 0.9195]	4	0.9188	0.0004	[0.9180, 0.9195]	4	0.8376	0.0008	[0.8361, 0.8391]	4	0.8376	0.0008	[0.8361, 0.8391]	4

μ = Mean; σ = Standard Deviation; CI = 95 % Confidence Interval; G = Scott-Knott ESD Group.

Table 7
Probability bin calibration analysis for the meta-model.

Bin range	Size	Mean confidence	Calibration error
0.00-0.10	229,331	0.0162	0.0011
0.10-0.20	8192	0.1426	0.0370
0.20-0.30	4326	0.2471	0.0085
0.30-0.40	3261	0.3479	0.0345
0.40-0.50	2727	0.4492	0.0696
0.50-0.60	2648	0.5503	0.1059
0.60-0.70	2975	0.6511	0.1237
0.70-0.80	3541	0.7521	0.1495
0.80-0.90	5823	0.8560	0.1613
0.90-1.00	237,176	0.9883	0.0081

Fig. 9. Reliability diagram comparing predicted probabilities with observed frequencies.

Table 8
Evaluation of D_4 in the 7 best individual models individual and logistic regression.

Model	Benign		Unknown	
	AGD	Non-AGD	AGD	Non-AGD
LSTM (Woodbridge et al., 2016)	81.67 %	18.33 %	19.66 %	80.34 %
LSTM (Yu et al., 2017)	83.17 %	16.83 %	21.97 %	78.03 %
MCU (Yu et al., 2018)	80.44 %	19.56 %	30.93 %	69.07 %
MIT (Yu et al., 2018)	83.78 %	16.22 %	22.01 %	77.99 %
DBD (Vinayakumar et al., 2019)	84.31 %	15.69 %	20.13 %	79.87 %
CNN (Berman, 2019)	81.37 %	18.63 %	17.14 %	82.86 %
Parallel CNN (Yu et al., 2018)	82.17 %	17.83 %	22.84 %	77.16 %
Our proposed meta-model	84.40 %	15.60 %	18.63 %	81.37 %

of 0.1613 occurring in the 0.8-0.9 bin, indicating a tendency toward overconfidence in this range.

Notably, the bins with the largest calibration errors contain relatively few instances: the six middle intervals from 0.4 to 0.9 combined represent only 17,714 instances (approximately 3.5 % of the dataset). This pattern is advantageous for AGD detection systems, where domains are often clearly benign or clearly malicious. Robust calibration of the meta-model at probability extremes ensures that confidence indices reliably reflect actual probabilities.

5.3. Q3: real-world effectiveness of academic AGD detection models

An effective AGD detection model is necessary in the context of today's threat landscape. In this section, we address RQ3 by evaluating the performance of the classifiers described in Section 5.2 in real-life situations. To do so, we conduct experiments on the D_4 and D_5 datasets to test the classifiers. This approach allows us to estimate their effectiveness under real-life conditions. The results of this experiment are presented in Table 8. We then draw several conclusions based on these results, which answer RQ3.

Table 9
Evaluation results on the D_5 dataset.

Model	Acc	F1
LSTM (Woodbridge et al., 2016)	89.23	94.31
LSTM (Yu et al., 2017)	88.07	93.66
MCU (Yu et al., 2018)	89.11	94.24
MIT (Yu et al., 2018)	88.92	94.14
DBD (Vinayakumar et al., 2019)	88.03	93.64
CNN (Berman, 2019)	87.97	93.60
Parallel CNN (Yu et al., 2018)	87.93	93.58
Our proposed meta-model	**89.46**	**94.44**

Misclassification of Benign Domains. Current classifiers are not suitable for real-world environments. The results in Table 8 show that more than 80 % of benign domains are incorrectly classified by all classifiers, which is inaccurate if we consider these domains as benign because their SLD and TLD are included in the Tranco (2023) list. Regarding the domains classified as unknown in D_4, approximately only 20 % of these domains have been positively identified as AGDs.

Inadequacy of Training Datasets to Represent Real-World Domains. If we analyze the benign domains in D_4, we can see that most of them are domains with hierarchical subdomains. In contrast, datasets used to train AGD detection models only consist of second- and top-level domains. Therefore, these datasets cannot be considered representative of domains in real-world environments, as they do not consider that domains can have multiple subdomains. Furthermore, these subdomains can be generated dynamically and randomly, but not used for malicious purposes. This happens with some well-known Internet services that use such subdomains, such as `googlesyndication.com`, `dropbox.com`, or `cloudfront.net`. This lack of representation of real-world domains in training datasets is the main cause of the misclassification problem discussed above.

DGArchive's Real-World AGDs Evaluation. To further validate our findings, we evaluate all models on the D_5 dataset, which contains 2.9 million real-world AGDs from DGArchive (Plohmann et al., 2016). The results are presented in Table 9. All models achieved accuracy values between 87.93 % and 89.46 %, with our proposed meta-model performing the best at 89.46 %. The perfect precision, reflected in the F1-scores, is expected since D_5 contains exclusively AGDs. However, the accuracy values indicate that all models fail to detect approximately 10 to 12 % of the real-world AGDs, highlighting performance shortcomings compared to controlled environments in previous experiments.

Notably, there is a significant gap between model performance on the academic AGD datasets (D_1, D_2, and D_3) and those on real-world AGDs in D_5. This performance degradation underscores the challenges of transferring academic models to operational environments, where DGAs may be more diverse, sophisticated, or previously unknown to detection models. These findings emphasize the need for continuous model updating and more representative training datasets that better reflect the evolving landscape of real-world malicious domains.

Common Classification Failures. We also analyze common classification failures observed in both individual models and the meta-model, focusing on the patterns of misclassification of benign domains as DGAs and vice versa. Our analysis reveals that benign domains misclassified as AGDs in D_4 often share characteristics with malicious DGAs, despite serving legitimate purposes. For example, domains such as `dd3187aea93c4f86.safeframe.googlesyndication.com`, `i2-qvprxfpecstaegbeldyyno.init.cedexis-radar.net`, and `ucea5465a78eff0d990.dl.dropboxusercontent.com` contain random-looking alphanumeric strings as subdomains of legitimate parent domains. Similarly, domains like `zzu4e.tdum.alibaba.com` and `np.dl.playstation.com` use abbreviations or shortcodes that trigger false positives due to their entropy patterns, even though they belong to well-established organizations.

These misclassifications are primarily due to current models being trained on datasets composed mainly of second-level domains, which lack adequate representation of legitimate domains with complex hierarchical structures. Models tend to focus on character-level features that appear algorithmically generated, without regard for the legitimacy of the original domain.

Regarding the AGDs in the D_5 dataset that were misclassified as benign, no clear patterns were found to distinguish correctly classified from incorrectly classified samples. This suggests that the models either fail to detect subtle variations in DGA algorithms or encounter entirely new generation patterns not present in their training data.

These findings indicate that current academic models require substantial adaptation to address the diversity and complexity of domains present in real-world settings. Future research should focus on developing more representative training datasets that include hierarchical domains, legitimate randomization patterns, and continuously updated AGD examples to improve detection accuracy in operational environments.

6. Threat model and limitations

In this section, we first describe the threat model of our proposed model and then outline the limitations of our work.

6.1. Threat model

The meta-model proposed in Section 5.2 consists of a logistic regression model that integrates the outputs of multiple neural network models for AGD detection. In this regard, if any of the neural network models feeding the logistic regression model are compromised (e.g., through adversarial attacks or data poisoning), their output probabilities can be skewed, affecting the performance of the entire meta-model. This can lead to incorrect AGD classifications due to poisoned or manipulated input probabilities. Similarly, logistic regression models are also vulnerable to adversarial attacks, where carefully crafted inputs can manipulate the decision boundary to cause misclassification. To address these issues, we can implement adversarial defense mechanisms as well as validate the input data for each neural network model to ensure it has not been tampered with.

An attacker can also attempt a model inversion attack on the logistic regression model to infer details about the underlying NN models or the training data. If successful, this can lead to privacy violations, revealing details about how AGD domains are detected or even leaking information about the system's detection capabilities. To prevent this, we can limit the exposure of model's internals and use differential privacy techniques.

Our proposed model may be overfitted to particular patterns and outputs from the underlying models. If one or more of these NN models are biased or flawed, the logistic regression may inherit these biases, resulting in reduced generalizability to new, unseen domains. To mitigate this, we can use techniques such as cross-validation and regularization mechanisms to reduce overfitting and facilitate generalization.

The effectiveness of our meta-model is highly dependent on the quality and consistency of the input data feeding each neural network. In real-world scenarios, where domain names often exhibit complex hierarchical patterns and dynamically generated characteristics, we may encounter noisy, incomplete, or inconsistent data that could impact the reliability of our detection system. Data quality issues such as missing features, inconsistent formats, or variations in subdomain patterns could propagate through individual neural networks and ultimately impact the meta-model's decision-making process. These challenges are particularly evident when dealing with legitimate services that employ dynamic subdomain generation for non-malicious purposes.

To address these limitations, we propose to implement robust preprocessing pipelines, including data validation checks to identify and

handle missing or corrupted values, standardization procedures to ensure consistent representation of features across models, and noise reduction techniques to improve the signal-to-noise ratio in input data. These steps are designed to preserve the legitimate complexity of domain structures while reducing the impact of noisy or incomplete data, thereby improving the overall performance and reliability of the model.

Similarly, an attacker can attempt to perform a denial-of-service attack in our meta-model. Since the system depends on multiple NN models followed by a logistic regression layer, the attacker can create inputs that cause excessive resource usage, slowing down the detection process. Resource monitoring and graceful degradation strategies to manage resource-intensive operations and avoid system overload can help mitigate this issue.

Our proposed model aggregates the results of multiple NN models. This process, while improving performance, introduces additional complexity that can make it vulnerable to exploitation. For instance, an adversary can compromise one or more models by manipulating the neural networks to generate biased probabilities, which could disproportionately influence the final output of the logistic regression classifier. This can lead to incorrect classifications, especially if the logistic regression gives significant weight to the outputs of the compromised models.

Another concern arises from the sequential nature of the aggregation process, which introduces temporal dependencies that can also be exploited. Delays or manipulations in the timing of the outputs of individual models can create race conditions or synchronization issues, ultimately affecting the reliability of the final classification decision.

To mitigate these vulnerabilities, several defensive strategies can be employed. First, weighted ensembling techniques can reduce the impact of any model on the final decision, making the system more robust to attacks targeting specific models. Additionally, anomaly detection can be incorporated to monitor and flag unusual results from any individual model, reducing the chances that an adversary can manipulate the overall result without being detected. To address timing concerns, we recommend implementing synchronization mechanisms, such as timeout protocols, that would ensure that all models produce results within a consistent time frame. These measures can help prevent timing inconsistencies, race conditions, and unresponsive models from affecting the aggregation process.

Similarly, if individual NN models are poorly calibrated (i.e., predicted probabilities do not reflect actual likelihoods), inaccurate input features for the logistic regression model may arise. The meta-model can therefore incorrectly classify AGD domains, as the input probabilities of the base models are not reliable indicators of AGD likelihood. Correct fine-tuning of the underlying NNs can help mitigate this problem.

The output of the logistic regression meta-model is considered the final predicted value. In this sense, this model acts as a single point of failure. If the logistic regression algorithm is compromised (e.g., through model inversion attacks), the entire AGD detection system can fail, regardless of the performance of the individual NNs. We can monitor the integrity of the logistic regression model and consider redundancy or ensembling even at the meta-model level to mitigate this problem.

6.2. Limitations

RAMPAGE may face challenges in scaling to accommodate a larger number of models or more complex models. This is particularly problematic when integrating more advanced neural network architectures or a larger ensemble of models, which may require significant engineering efforts.

Furthermore, the lack of reproducible research and publicly available data in the field poses another substantial limitation. As noted in (Arp et al., 2023; Botacin et al., 2021), many state-of-the-art datasets are not shared openly or the hyperparameters settings of the models are not consistently defined, hampering the ability to validate and compare results between studies. This scarcity of accessible data can prevent comparison with previous models, as well as the development and subsequent benchmarking of new models.

Like any other DGA detection model based on DNS requests, our logistic regression assumes that DNS requests are available. In real-world networks, transit data may be incomplete, encrypted, or obfuscated, complicating the detection process. Furthermore, the performance of our model may vary under different operational environments. The prevalence of certain DGA families may influence the effectiveness of our model.

On top of that, interpretability remains a major challenge in meta-model approaches. Neural networks, particularly when combined in a meta-model approach, often act as black boxes, making it difficult to understand the network's decision-making process. To address this challenge, we integrate explainability techniques such as SHAP values, which provide transparency by highlighting feature importance and clarifying local decision boundaries. Additionally, we maintain detailed logs of the intermediate results of each neural network component, allowing for better traceability and transparency in the decision-making process along the meta-model pipeline.

Performance is also another issue. Running multiple neural network models, followed by a logistic regression model, demands substantial computational resources. To mitigate this, resource-constrained organizations can benefit from employing model compression techniques or leveraging distributed computing approaches. From a scalability perspective, our logistic regression architecture is designed to handle increasing data volumes through horizontal scaling of computing resources and modular component design. Although this approach may not be feasible in all environments, particularly for organizations with limited access to high-performance computing infrastructure or real-time networks, these strategies offer potential workarounds that could make deployment more accessible.

Finally, although our proposed meta-model performs well on the existing dataset, its ability to generalize to new unseen AGDs remains uncertain. To address concerns about overfitting, we implement cross-validation and include dropout layers within the training. The model's ability to detect emerging threats depends on the diversity and representativeness of the training data. As with all deep learning models, our system requires regular updates and retraining to adapt to changing conditions. In this regard, periodic retraining programs using newly collected data can help. In our case, we have developed an automated pipeline to support continuous data collection and model updates. This automated process, coupled with continuous monitoring of model performance metrics, ensures that the system remains responsive to real-world dynamics.

7. Conclusions and future work

Despite numerous proposed machine learning models for AGD detection, their true efficacy remains uncertain due to the lack of standardized evaluation methods and reproducible datasets. In this paper, we addressed these challenges by presenting RAMPAGE, a novel Python3 software framework designed to standardize and facilitate the development, evaluation, and comparison of machine learning models for AGD detection. Our framework provides a reproducible approach to overcome common issues in cybersecurity and machine learning research, such as inconsistent baselines and non-reproducible methodologies. To demonstrate the utility of RAMPAGE, we implemented several existing models from the literature and evaluated them under the same conditions. Furthermore, we built a new meta-model that combines distinct deep learning models and a logistic regression, which we evaluated on a developed dataset containing real-world DNS requests. The results show that our meta-model achieves slightly better performance than the state-of-the-art models, highlighting the effectiveness of our approach and the applicability of our framework. In the spirit of open science, our software

framework and proposed model, as well as the dataset used for experimentation, are freely and publicly available under the GNU/GPLv3 license.

As future work, we aim to create a repository dedicated to sharing and downloading pre-trained models for detecting AGDs. This repository would allow researchers to easily share and compare their results, fostering collaboration and improving reproducibility in this field. We are also working on the explainability of our model to elucidate how it reaches its conclusions and identify the most influential features in its predictions. We also aim to evaluate the computational efficiency, resource consumption, and scalability of the meta-model in large-scale deployments. Additionally, we aim to explore the potential of incorporating lexical and metadata features, such as character n-gram distributions, entropy scores, and WHOIS/registration-based data, as well as hierarchical parsing strategies that treat each subdomain label individually to expand validation efforts in AGD detection. These techniques could improve the ability to distinguish between benign dynamic subdomains and true AGDs in more heterogeneous or large-scale environments, and we consider them promising for future research. We also aim to expand validation by incorporating DNS data from diverse organizational or geographic sources, and by cross-validating with public threat intelligence sources to improve the generalizability of our approach.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used ChatGPT-4 (OpenAI) to improve readability and language. After using this tool/service, the authors reviewed and edited the content as needed and assume full responsibility for the content of the publication.

CRediT authorship contribution statement

Tomás Pelayo-Benedet: Conceptualization, Methodology, Software, Validation, Formal analysis, Data curation, Investigation, Writing - original draft, Writing - review & editing; **Ricardo J. Rodríguez:** Conceptualization, Methodology, Funding acquisition, Project administration, Resources, Validation, Supervision, Writing - original draft, Writing - review & editing; **Carlos H. Gañán:** Formal analysis, Methodology, Visualization, Writing - review & editing.

Data availability

Data will be made available on request.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to sincerely thank the reviewers for their constructive comments and insightful suggestions, which have significantly contributed to improving the quality and clarity of this work. This research was supported in part by grant PID2023-151467OA-I00 (CRAPER), funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU, by grant TED2021-131115A-I00 (MIMFA), funded by MICIU/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR, by grant *Ayudas para la recualificación del sistema universitario español 2021–2023*, funded by the European Union NextGenerationEU/PRTR, the Spanish Ministry of Universities, and the University of Zaragoza, by grant *Proyecto Estratégico Ciberseguridad EINA UNIZAR*, funded by the Spanish National Cybersecurity Institute (INCIBE) and the European Union NextGenerationEU/PRTR, by grant *Programa de Proyectos Estratégicos de Grupos de Investigación* (DisCo research group, ref. T21-23R), funded by the University, Industry and Innovation Department of the Aragonese Government, and by the RAPID project (grant no CS.007) financed by the Dutch Research Council (NWO).

Appendix A. Model hyperparameter configurations

This appendix describes the detailed hyperparameter settings for all models evaluated in this work. For each model, we adopted the settings specified in their respective original publications, applying minimal additional tuning. This decision facilitates a fair comparison with previous work and ensures that any observed performance differences primarily reflect the influence of our experimental framework, rather than exhaustive hyperparameter optimization.

Tables A.10 to A.16 provide comprehensive information on the model architectures, hyperparameter values, optimization strategies, and loss functions employed in our implementation. These details are intended to improve reproducibility. All models described in this appendix are publicly available in our GitHub repository.[3]

[3] https://github.com/reverseame/RAMPAGE

Table A.10
Hyperparameter configuration for the LSTM model (Woodbridge et al., 2016).

Parameter	Value
Architecture	Sequential
Embedding Layer	
Input Dimension	256
Output Dimension	128
Input Length	128
LSTM Layer	
Units	128
Unroll	True
Dropout Layer	
Rate	0.5
Dense Layer	
Units	1
Activation Function	sigmoid
Optimizer	Adam
Learning Rate	0.001
Beta 1	0.9
Beta 2	0.999
Epsilon	1e-08
Weight Decay	0.001
Loss Function	binary_crossentropy

Table A.11
Hyperparameter configuration for the LSTM model (Yu et al., 2017).

Parameter	Value
Architecture	Sequential
Embedding Layer	
Input Dimension	256
Output Dimension	128
Input Length	128
LSTM Layer	
Units	128
Unroll	True
Dropout Layer	
Rate	0.5
Dense Layers	
First Layer Units	100
Second Layer Units	1
Activation Function	sigmoid
Optimizer	Adam
Learning Rate	0.001
Beta 1	0.9
Beta 2	0.999
Epsilon	1e-08
Weight Decay	0.001
Loss Function	binary_crossentropy

Table A.12
Hyperparameter configuration for the MCU model (Yu et al., 2018).

Parameter	Value
Architecture	Sequential
Embedding Layer	
Input Dimension	128
Output Dimension	128
Input Length	128
Bidirectional LSTM Layer	
Units	64
Return Sequences	False
Unroll	True
Merge Mode	concat
Dense Layer	
Units	1
Activation Function	sigmoid
Optimizer	Adam (default)
Loss Function	binary_crossentropy

Table A.13
Hyperparameter configuration for the MIT model (Yu et al., 2018).

Parameter	Value
Architecture	Sequential
Embedding Layer	
Input Dimension	128
Output Dimension	128
Input Length	128
Conv1D Layer	
Filters	128
Kernel Size	3
Padding	same
Activation	relu
Strides	1
MaxPooling1D Layer	
Pool Size	2
Padding	same
LSTM Layer	
Units	64
Return Sequences	False
Unroll	True
Output Dense Layer	
Units	1
Activation Function	sigmoid
Optimizer	Adam (default)
Loss Function	binary_crossentropy

Table A.14
Hyperparameter configuration for the CNN model (Berman, 2019).

Parameter	Value
Architecture	Sequential
Embedding Layer	
Input Dimension	256
Output Dimension	50
Input Length	128
Dropout Layer	
Rate	0.25
First Conv1D Layer	
Filters	250
Kernel Size	4
Padding	same
MaxPooling1D Layer	
Pool Size	3
Second Conv1D Layer	
Filters	300
Kernel Size	3
Padding	same
Flatten Layer	
BatchNormalization Layer	
Dense Layer	
Units	300
Dropout Layer	
Rate	0.2
BatchNormalization Layer	
Output Dense Layer	
Units	1
Activation Function	sigmoid
Optimizer	Adam (default)
Loss Function	binary_crossentropy

Table A.15
Hyperparameter configuration for the DBD model (Vinayakumar et al., 2019).

Parameter	Value
Architecture	Sequential
Embedding Layer	
Input Dimension	256
Output Dimension	128
Input Length	128
Conv1D Layer	
Filters	64
Kernel Size	5
MaxPooling1D Layer	
Pool Size	4
LSTM Layer	
Units	70
Unroll	True
Output Dense Layer	
Units	1
Activation Function	sigmoid
Optimizer	Adam (default)
Loss Function	binary_crossentropy

Table A.16
Hyperparameter configuration for the Parallel CNN model (Yu et al., 2018).

Parameter	Value
Architecture	Sequential
Embedding Layer	
Input Dimension	128
Output Dimension	128
Input Length	128
Parallel Conv1D Layers	
Kernel Sizes	2, 3, 4, 5
Filters	256 (each)
Padding	same
Activation	relu
Strides	1
Reduction	sum (axis = 1)
Dropout Layer (after each conv)	
Rate	0.5
Dense Layers	
First Layer Units	1024
First Layer Activation	relu
Second Layer Units	1024
Second Layer Activation	relu
Output Layer Units	1
Dropout Layers (between dense)	
Rate	0.5
Output Activation Function	sigmoid
Optimizer	Adam (default)
Loss Function	binary_crossentropy

Appendix B. Hyperparameter optimization methodology

This appendix provides detailed documentation of the hyperparameter optimization methodology applied to the 17 models evaluated in this work. The goal is to improve transparency and facilitate the reproducibility of our experimental procedures.

B.1. Optimization strategy

Rather than performing exhaustive hyperparameter searches across all possible parameter configurations, we adopted a principled strategy designed to balance reproducibility and computational efficiency. Our approach was based on the following key decisions:

Baseline Configuration. We initialized each model with the hyperparameter settings reported in its original publication. This ensures that our comparisons accurately reflect the configurations intended by the authors and provide a fair and standardized basis for evaluation.

Standardized Training Parameters. To ensure consistent and comparable training conditions across all models, we apply a unified training configuration, regardless of model architecture. This standardization mitigates confounding factors that could arise from heterogeneous training procedures.

B.2. Training setup details

The following training parameters were consistently applied to all 17 models:

- **Epochs:** 500
- **Batch size:** 50
- **Optimizer:** Adam optimizer with:
  - Learning rate: 0.001
  - Beta 1: 0.9
  - Beta 2: 0.999
  - Epsilon: 10^{-8}
  - Weight decay: 0.001
- **Learning function loss:** Binary Cross-Entropy

B.3. Model selection and validation strategy

Early Stopping. All models incorporated early stopping based on validation accuracy to prevent overfitting. Training was stopped if no improvement was observed over a predefined time interval.

Model Checkpointing. Model checkpointing was used to retain the state of the best-performing model based on validation accuracy, thereby ensuring that the final model reflects the optimal configuration observed during training.

Cross-Validation. All experiments employed consistent training-validation-test splits of 70 %-15 %-15 %. This fixed split was applied uniformly to each model to eliminate dataset variability as a confounder in performance comparisons.

References

Netlab-360 (2025a). Netlab-360 feed. [Online]; https://data.netlab.360.com/dga/.

Alexa (2025b). Alexa Top 1 Million Domains feed. [Online]; http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

Aloysius, N., & Geetha, M. (2017). A review on deep convolutional neural networks. In *2017 international conference on communication and signal processing (ICCSP)* (pp. 0588–0592). IEEE.

Amari, S.-i. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, *5*(4), 185–196.

Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., & Rieck, K. (2023). Lessons learned on machine learning for computer security. *IEEE Security & Privacy*, *21*(5), 72–77.

Bambenek (2025c). Bambenek Consulting - master feeds. [Online]; https://osint.bambenekconsulting.com/feeds/.

Bejtlich, R. (2013). The practice of network security monitoring. San Francisco, CA: No Starch Press.

Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, Y., Maddix, D., Turkmen, C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., Aubet, F.-X., Callot, L., & Januschowski, T. (2022). Deep learning for time series forecasting: Tutorial and literature survey. *ACM Computing Surveys*, *55*(6).

Berman, D. S. (2019). DGA Capsnet: 1D application of capsule networks to DGA detection. *Information*, *10*(5).

Botacin, M., Ceschin, F., Sun, R., Oliveira, D., & Grégio, A. (2021). Challenges and pitfalls in malware research. *Computers & Security*, *106*, 102287.

Catania, C., Garcia, S., & Torres, P. (2018). An analysis of convolutional neural networks for detecting DGA. In *Xxiv congreso argentino de ciencias de la computación (la plata, 2018)*. (pp. 1060–1069).

Catania, C., García, S., & Torres, P. (2019). Deep convolutional neural networks for DGA detection. In P. Pesado, & C. Aciti (Eds.), *Computer science – CACIC 2018* (pp. 327–340). Cham: Springer International Publishing.

Cebere, B. C., Flueren, J. L. B., Sebastián, S., Plohmann, D., & Rossow, C. (2024). Down to earth! guidelines for DGA-based malware detection. In *Proceedings of the 27th international symposium on research in attacks, intrusions and defenses* RAID '24 (pp. 147–165). New York, NY, USA: Association for Computing Machinery.

Choudhary, C., Sivaguru, R., Pereira, M., Yu, B., Nascimento, A. C., & De Cock, M. (2019). Algorithmically generated domain detection and malware family classification. In S. M. Thampi, S. Madria, G. Wang, D. B. Rawat, & J. M. Alcaraz Calero (Eds.), *Security in computing and communications* (pp. 640–655). Singapore: Springer Singapore.

Chowdhary, K. R. (2020). Natural Language Processing. In *Fundamentals of Artificial Intelligence*, pp. 603–649). New Delhi: Springer India.

Curtin, R. R., Gardner, A. B., Grzonkowski, S., Kleymenov, A., & Mosquera, A. (2019). Detecting DGA domains with recurrent neural networks and side information. In *Proceedings of the 14th international conference on availability, reliability and security* ARES '19 (p. 10). Association for Computing Machinery.

Cybersecurityventures(2023). Cybercrime To Cost The World 10.5 Trillion Annually By 2025. Online Accessed on 15 August, 2023; https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/.

DGArchive (2023). DGArchive feed. Online Accessed on 1 August, 2023; https://dgarchive.caad.fkie.fraunhofer.de/.

Dhingra, B., Zhou, Z., Fitzpatrick, D. J., Muehl, M., & Cohen, W. W. (2016). Tweet2Vec: Character-based distributed representations for social media. *CoRR*, *abs/1605.03481*.

Ding, B., Qian, H., & Zhou, J. (2018). Activation functions and their characteristics in deep neural networks. In *2018 Chinese control and decision conference (CCDC)* (pp. 1836–1841).

Drichel, A., von Querfurth, B., & Meyer, U. (2024). Extended abstract: A transfer learning-Based training approach for DGA classification. In F. Maggi, M. Egele, M. Payer, & M. Carminati (Eds.), *Detection of intrusions and malware, and vulnerability assessment* (pp. 381–391). Cham: Springer Nature Switzerland.

Guo, M.-H., Xu, T.-X., Liu, J.-J., Liu, Z.-N., Jiang, P.-T., Mu, T.-J., Zhang, S.-H., Martin, R. R., Cheng, M.-M., & Hu, S.-M. (2022). Attention mechanisms in computer vision: A survey. *Computational Visual Media*, *8*(3), 331–368.

Huang, W., Zong, Y., Shi, Z., Wang, L., & Liu, P. (2022). PEPC: A deep parallel convolutional neural network model with pre-trained embeddings for DGA detection. In *2022 international joint conference on neural networks (IJCNN)* (pp. 1–8).

Huk, M. (2020). Stochastic optimization of contextual neural networks with RMSprop. In N. T. Nguyen, K. Jearanaitanakij, A. Selamat, B. Trawiński, & S. Chittayasothorn (Eds.), *Intelligent information and database systems* (pp. 343–352). Cham: Springer International Publishing.

Liang, J., Chen, S., Wei, Z., Zhao, S., & Zhao, W. (2022). HAGDetector: Heterogeneous DGA domain name detection model. *Computers & Security*, *120*, 102803.

Lison, P., & Mavroeidis, V. (2017). Automatic detection of malware-Generated domains with recurrent neural models. *CoRR*, *abs/1709.07102*.

Liu, Z., Zhang, Y., Chen, Y., Fan, X., & Dong, C. (2020). Detection of algorithmically generated domain names using the recurrent convolutional neural network with spatial pyramid pooling. *Entropy*, *22*(9).

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems* NIPS'17 (pp. 4768–4777). Red Hook, NY, USA: Curran Associates Inc.

Maia, R. J. M., Ray, D., Pentyala, S., Dowsley, R., De Cock, M., Nascimento, A. C. A., & Jacobi, R. (2024). An end-to-end framework for private DGA detection as a service. *PloS one*, *19*(8), e0304476.

Majestic (2025). The majetic million feed. Online https://majestic.com/reports/majestic-million.

Malik, M., Malik, M. K., Mehmood, K., & Makhdoom, I. (2021). Automatic speech recognition: A survey. *Multimedia Tools and Applications*, *80*(6), 9411–9457.

Martin, L.(2023). The Cyber Kill Chain. Online; Accessed on 22 August, 2023 https://lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html Accessed on 22 August, 2023.

MITRE(2023). Dynamic Resolution: Domain Generation Algorithms. https://attack.mitre.org/techniques/T1568/002/. Online; Accessed on 23 August, 2023.

Morbidoni, C., Spalazzi, L., Teti, A., & Cucchiarelli, A. (2022). Leveraging n-gram neural embeddings to improve deep learning DGA detection. In *Proceedings of the 37th ACM/SIGAPP symposium on applied computing* SAC '22 (pp. 995–1004). New York, NY, USA: Association for Computing Machinery.

Namgung, J., Son, S., & Moon, Y.-S. (2021). Efficient deep learning models for DGA domain detection. *Security and Communication Networks*, *2021*(1), 8887881.

openDNS (2025). openDNS feed. [Online; https://umbrella.cisco.com/blog.

Palo Alto(2023). Threat Brief: Understanding Domain Generation Algorithms (DGA). https://unit42.paloaltonetworks.com/threat-brief-understanding-domain-generation-algorithms-dga/ [Online Accessed on 23 August, 2023.]

Plohmann, D., Yakdan, K., Klatt, M., Bader, J., & Gerhards-Padilla, E. (2016). A comprehensive measurement study of domain generating malware. In *25th USENIX security symposium (USENIX security 16)* (pp. 263–278). Austin, TX: USENIX Association.

Pochat, V. L., Goethem, T. V., & Joosen, W. (2019). Evaluating the long-term effects of parameters on the characteristics of the tranco top sites ranking. In *12th USENIX workshop on cyber security experimentation and test (CSET 19)* (p. 10). Santa Clara, CA: USENIX Association.

Porras, P. A., Saïdi, H., & Yegneswaran, V. (2009). A foray into Conficker's logic and rendezvous points. *LEET*, *9*, 7.

Qiao, Y., Zhang, B., Zhang, W., Sangaiah, A. K., & Wu, H. (2019). DGA Domain name classification method based on long short-Term memory with attention mechanism. *Applied Sciences*, *9*(20).

Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. In *Proceedings of the 31st international conference on neural information processing systems* NIPS'17 (pp. 3859–3869). Red Hook, NY, USA: Curran Associates Inc.

Saxe, J., & Berlin, K. (2017). Expose: A character-Level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *CoRR*, *abs/1702.08568*.

Selvi, J., Rodríguez, R. J., & Soria-Olivas, E. (2021). Towards optimal LSTM neural networks for detecting algorithmically generated domain names. *IEEE Access*, *9*, 126446–126456.

Shahzad, H., Sattar, A. R., & Skandaraniyam, J. (2021). DGA Domain detection using deep learning. In *2021 IEEE 5th international conference on cryptography, security and privacy (CSP)* (pp. 139–143).

Sheikholeslami, S. (2019). Ablation programming for machine learning.

Shi, Y., Chen, G., & Li, J. (2018). Malicious domain name detection based on extreme machine learning. *Neural Processing Letters*, *48*(3), 1347–1357.

Sivaguru, R., Choudhary, C., Yu, B., Tymchenko, V., Nascimento, A., & Cock, M. D. (2018). An evaluation of DGA classifiers. In *2018 IEEE international conference on big data (big data)* (pp. 5058–5067). IEEE.

Sivaguru, R., Peck, J., Olumofin, F., Nascimento, A., & De Cock, M. (2020). Inline detection of DGA domains using side information. *IEEE Access*, *8*, 141910–141922.

Suryotrisongko, H., & Musashi, Y. (2022). Evaluating hybrid quantum-classical deep learning for cybersecurity botnet DGA detection. *Procedia Computer Science*, *197*, 223–229. Sixth Information Systems International Conference (ISICO 2021).

Svozil, D., Kvasnicka, V., & Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, *39*(1), 43–62.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2019). The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, *45*(7), 683–711.

Tran, D., Mac, H., Tong, V., Tran, H. A., & Nguyen, L. G. (2018). A LSTM based framework for handling multiclass imbalance in DGA botnet detection. *Neurocomputing*, *275*, 2401–2413.

Tranco (2023). Tranco List. https://tranco-list.eu/ [Online; Accessed on 1 August, 2023].

Trucano, T. G., Swiler, L. P., Igusa, T., Oberkampf, W. L., & Pilch, M. (2006). Calibration, validation, and sensitivity analysis: What's what. *Reliability Engineering & System Safety*, *91*(10), 1331–1357. The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004).

Tuan, T. A., Anh, N. V., Luong, T. T., & Long, H. V. (2023). UTL_DGA22 a dataset for DGA botnet detection and classification. *Computer Networks*, *221*, 109508.

Tuan, T. A., Long, H. V., & Taniar, D. (2022). On detecting and classifying DGA botnets and their families. *Computers & Security*, *113*, 102549.

Umbrella (2025). Cisco Umbrella feed. Online; http://s3-us-west-1.amazonws.com/umbrellastatic/index.html.

Vinayakumar, R., Soman, K. P., Poornachandran, P., Alazab, M., & Jolfaei, A. (2019). DBD: Deep learning DGA-based botnet detection. In M. Alazab, & M. Tang (Eds.), *Deep learning applications for cyber security*, pp. 127–149). Springer International Publishing.

Vranken, H., & Alizadeh, H. (2022). Detection of DGA-generated domain names with TF-IDF. *Electronics*, *11*(3).

Woodbridge, J., Anderson, H. S., Ahuja, A., & Grant, D. (2016). Predicting domain generation algorithms with long short-Term memory networks. *CoRR*, *1611.00791*.

Wu, Y.-c., & Feng, J.-w. (2018). Development and application of artificial neural network. *Wireless Personal Communications*, *102*(2), 1645–1656.

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.

Yang, L., Liu, G., Dai, Y., Wang, J., & Zhai, J. (2020). Detecting stealthy domain generation algorithms using heterogeneous deep neural network framework. *IEEE Access*, *8*, 82876–82889.

Yang, L., Liu, G., Zhai, J., Dai, Y., Yan, Z., Zou, Y., & Huang, W. (2018). A novel detection method for word-based DGA. In X. Sun, Z. Pan, & E. Bertino (Eds.), *Cloud computing and security* (pp. 472–483). Springer International Publishing.

Yong Wong, M., Landen, M., Antonakakis, M., Blough, D. M., Redmiles, E. M., & Ahamad, M. (2021). An inside look into the practice of malware analysis. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security* CCS '21 (pp. 3053–3069). New York, NY, USA: Association for Computing Machinery.

Yu, B., Gray, D. L., Pan, J., Cock, M. D., & Nascimento, A. C. A. (2017). Inline DGA detection with deep networks. In *2017 IEEE international conference on data mining workshops (ICDMW)* (pp. 683–692). IEEE.

Yu, B., Pan, J., Hu, J., Nascimento, A., & De Cock, M. (2018). Character level based detection of DGA domain names. In *2018 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Yu, H.-F., Huang, F.-L., & Lin, C.-J. (2011). Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, *85*(1), 41–75.

Zhang, Z. (2018). Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQos)* (pp. 1–2).

Zhou, S., Lin, L., Yuan, J., Wang, F., Ling, Z., & Cui, J. (2019). CNN-Based DGA detection with high coverage. In *2019 IEEE international conference on intelligence and security informatics (ISI)* (pp. 62–67). IEEE.