

A HYBRID DEEP LEARNING FRAMEWORK FOR ADVANCED DETECTION OF
DOMAIN GENERATION ALGORITHMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SİNAN DÜZTAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF CYBER SECURITY

JANUARY 2026

**A HYBRID DEEP LEARNING FRAMEWORK FOR ADVANCED DETECTION
OF DOMAIN GENERATION ALGORITHMS**

Submitted by Sinan DÜZTAŞ in partial fulfillment of the requirements for the degree of
Master of Science in Cyber Security Department, Middle East Technical University by,

Date: 16.01.2026



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Sinan DÜZTAŞ

Signature : _____

ABSTRACT

A HYBRID DEEP LEARNING FRAMEWORK FOR ADVANCED DETECTION OF DOMAIN GENERATION ALGORITHMS

DÜZTAŞ, Sinan

MSc., Department of Cyber Security

Supervisor: Assoc. Prof. Dr. Cihangir TEZCAN

January 2026, 110 pages

Cyber threat actors (CTAs) rely on botnets to carry out a wide range of malicious operations. These botnets are controlled by command-and-control (C2) servers. If these servers are exposed, CTAs can be detected and their activities can be disrupted. In order to disguise their C2 servers, CTAs employ domain generation algorithms (DGAs). These algorithms produce large number of random domain names. CTAs use DGAs to bypass traditional detection mechanisms like blacklisting or static signature-based solutions. Since blacklisting all DGA-generated domains is not possible, the automatic detection of DGAs remains a critical necessity.

The problem of distinguishing legitimate domains from DGA generated domains is an ongoing challenge. Despite years of research, modern malware continues to evolve, and this makes detection increasingly difficult. Traditional machine learning models for DGA detection heavily rely on verbal analysis of domain names through manual feature engineering. However, these models are time-consuming to construct and often produce high false positive rates. This thesis addresses these limitations by employing a hybrid framework that integrates CNN based n-gram learning and LSTM for automatic feature extraction and Logistic Regression (LR) for final classification. The model input is further enriched by Top-Level Domain (TLD) information for improved accuracy. The model is trained with a publicly available dataset compiled from ICANN, Alexa and DGArchive. Its performance is compared against other deep learning models like CNN, LSTM, GRU and Transformers. This research aims to contribute to the cybersecurity domain by offering a comprehensive solution for detecting DGAs through a novel hybrid framework that enhances overall detection performance.

Keywords: Domain Generation Algorithm (DGA), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) networks, Logistic Regression (LR), Top-Level Domain (TLD).



ÖZ

KÖTÜCÜL ALAN ADI ÜRETME ALGORİTMALARININ HİBRİT DERİN ÖĞRENME MODELİ İLE GELİŞMİŞ TESPİTİ

DÜZTAŞ, Sinan

Yüksek Lisans, Siber Güvenlik Bölümü

Tez Yöneticisi: Doç. Dr. Cihangir TEZCAN

Ocak 2026, 110 sayfa

Siber tehdit aktörleri, kötü amaçlı faaliyetlerini yürütmek için zombi ağlara başvurmakta ve bu ağları komuta-kontrol (C2) sunucuları aracılığıyla yönetmektedir. Ancak, bu sunucuların açığa çıkması durumunda, saldırganlar tespit edilebilecek ve faaliyetleri sekteye uğratılabilecektir. Bu nedenle siber tehdit aktörleri, oluşturdukları zombi ağları yönetebilmek ve C2 sunucularını gizleyebilmek maksadıyla alan adı üretim algoritmalarını (DGA) kullanmaktadır. Bu yöntem, kara listeleme veya statik imza tabanlı güvenlik çözümleri gibi geleneksel tespit mekanizmalarının atlatılabilmesine imkân sağlamaktadır. DGA'lar tarafından üretilmiş bütün alan adlarının kara listeye alınması mümkün olmadığından, DGA'ların otomatik olarak tespit edilmesi kritik bir gereklilik haline gelmiştir.

Meşru alan adlarını DGA ile üretilen alan adlarından ayırt etmek, hâlâ süregelen bir zorluktur. Modern zararlı yazılımlar gelişmeye devam etmekte ve bu durum DGA ile üretilen alan adlarının tespitini daha da zorlaştırmaktadır. Geleneksel makine öğrenmesi modelleri, DGA tespiti için büyük ölçüde alan adlarının sözel analizine ve özellik çıkarımının manuel olarak yapılmasına dayanmaktadır. Ancak bu özelliklerin oluşturulması zaman alıcıdır ve genellikle yüksek yanlış pozitif oranlarına yol açmaktadır. Bu tez, söz konusu sınırlamaları aşmak maksadıyla özellik çıkarımı için CNN tabanlı n-gram öğrenme ve LSTM ağlarını ve nihai sınıflandırma için Lojistik Regresyon yöntemini entegre eden hibrit bir çerçeve sunmaktadır. Model girdisi, doğruluğu artırmak için Üst Düzey Alan Adı (TLD) bilgileriyle zenginleştirilmiştir. Model, ICANN, Alexa ve DGArchieve veri setlerinden derlenerek oluşturulan bir veri setiyle eğitilmiş ve performansı CNN, LSTM, GRU ve Transformers gibi derin öğrenme modelleriyle karşılaştırılmıştır. Bu çalışma, manuel özellik mühendisliği ihtiyacını ortadan kaldıran ve

genel tespit başarısını artıran özgün bir hibrit çerçeve ile DGA'ların tespiti için kapsamlı bir çözüm sunmak suretiyle siber güvenlik alanına katkıda bulunmayı hedeflemektedir.

Anahtar Sözcükler: Alan Adı Üretim Algoritmaları (DGA), Evrimsel Sinir Ağları (CNN), Uzun Kısa Süreli Bellek (LSTM) Ağları, Lojistik Regresyon (LR), Üst Düzey Alan Adı (TLD).





To My Family.

ACKNOWLEDGMENTS

I am deeply thankful to my supervisor, Assoc. Prof. Dr. Cihangir TEZCAN, for his invaluable guidance and support.



TABLE OF CONTENTS

ABSTRACT	iv
ÖZ.....	vi
DEDICATION	viii
ACKNOWLEDGMENTS.....	ix
TABLE OF CONTENTS	x
LIST OF TABLES	xiii
LIST OF FIGURES.....	xiv
LIST OF ABBREVIATIONS	xvii
CHAPTERS	
1.INTRODUCTION.....	1
1.1. Understanding DGAs	3
1.2. How DGAs Work?	3
1.3. Taxonomy of DGAs	5
1.4. DGA Detection	6
1.5. The Structure of the Thesis.....	8
1.6. Aim and Scope.....	8
1.7. Research Questions.....	9
1.8. Contributions of the Study.....	9
2.LITERATURE REVIEW.....	11
2.1. Statistical Models	11
2.2. DNS-Based Models	12
2.3. ML Models	12
2.4. DL Models.....	14
2.5. Hybrid Models.....	15
2.6. Summary.....	17
3.METHODOLOGY.....	21
3.1. Data Preparation	21
3.1.1. What is a Top-Level Domain (TLD)?	24
3.2. Model Training.....	25

3.2.1. Standalone DL Model Trainings	25
3.2.1.1. CNN	25
3.2.1.2. LSTM	27
3.2.1.3. GRU	28
3.2.1.4. Transformer Networks	30
3.2.2. Hybrid Model Training	31
3.2.2.1. CNN Based n-gram Model Training	32
3.2.2.2. LSTM Model Training	34
3.2.2.3. TLD Feature Fusion	35
3.2.2.4. Final Classification with LR	36
3.3. Model Evaluation	36
3.3.1. Accuracy	36
3.3.2. Precision	37
3.3.3. Recall	37
3.3.4. F1-Score	37
3.3.5. Receiver Operating Characteristic – Area Under the Curve (ROC-AUC)	38
3.3.6. Confusion Matrix	39
3.4. Model Validation Method	40
3.5. Experimental Environment	40
4. EXPERIMENTS	41
4.1. Data Preparation	41
4.2. Model Training	43
4.2.1. CNN Model Training	43
4.2.2. LSTM Model Training	44
4.2.3. GRU Model Training	45
4.2.4. Transformer Model Training	46
4.3. Hybrid Model Training	47
4.3.1. CNN Based n-gram Encoding	48
4.3.2. CNN-LSTM Hybrid Pipeline	48
4.3.3. TLD Feature Fusion	49
4.3.4. Final Classification with LR	51

5.RESULTS AND DISCUSSION	53
5.1. Standalone DL Model Results.....	53
5.1.1. CNN.....	53
5.1.2. LSTM.....	56
5.1.3. GRU.....	58
5.1.4. Transformer Networks.....	61
5.2. Hybrid Model	63
5.3. Hybrid Model Validation.....	66
5.4. Comparison of Models	67
5.5. Comparison with Previous Studies on DGA Detection.....	68
5.6. Comparison of 5 models' Qualitative Aspects.....	70
5.7. Advantages of DL Use for DGA Detection.....	72
5.8. Summary.....	73
6.CONCLUSION AND FUTURE WORK.....	75
6.1. Conclusion.....	75
6.2. Future Work.....	77
REFERENCES.....	79
APPENDICES.....	83
APPENDIX A – STANDALONE CNN MODEL TRAINING PYTHON CODE	83
APPENDIX B – STANDALONE LSTM MODEL TRAINING PYTHON CODE	89
APPENDIX C – STANDALONE GRU MODEL TRAINING PYTHON CODE	91
APPENDIX D – TRANSFORMER MODEL TRAINING PYTHON CODE	97
APPENDIX E – HYBRID MODEL TRAINING PYTHON CODE.....	103

LIST OF TABLES

Table 1: Sample domain names generated by DGAs with corresponding malware and used seed type.....	4
Table 2: A sample of prominent lexical features applied in ML models.....	14
Table 3: Experiment results of DGA detection studies in literature, illustrated in a chronological order, showing the models employed along with the evaluation scores...18	
Table 4: Top 10 TLDs used by DGA-generated and benign domains in our dataset.	23
Table 5: URL components of a sample domain.....	25
Table 6: The layers of standalone CNN model are explained along with their primary functions.....	26
Table 7: The layers of standalone LSTM model are explained along with their primary functions.....	28
Table 8: The layers of standalone GRU model are explained along with their primary functions.....	29
Table 9: The layers of Transformer model are explained along with their primary functions.....	31
Table 10: A step-by-step example of how LSTM processes domain characters.	34
Table 11: Key differences between CNN n-gram and LSTM encoding techniques.....	35
Table 12: Key evaluation metrics that will be used in order to compare model performances.	38
Table 13: Evaluation of AUC values.	38
Table 14: Confusion matrix structure.	39
Table 15: Top 10 TLDs exhibiting the strongest association with DGAs.	51
Table 16: The training, validation and test results of standalone CNN model.	54
Table 17: The training, validation and test results of standalone LSTM model.	57
Table 18: The training, validation and test results of standalone GRU model.	60
Table 19: The training, validation and test results of Transformer model.....	61
Table 20: Performance metrics obtained through 5-fold cross-validation of the proposed hybrid model.	66
Table 21: Comparison of 5 models' experiment results. The highest scores are shown in green colors and the lowest scores are shown in red color, illustrating that the hybrid model achieved the top scores in ROC-AUC and recall, and performed second in other metrics just with minimal differences.....	67
Table 22: Performance comparison of DL-based DGA detection models across different studies, with author's experiment outcomes illustrated in red color.....	69
Table 23: Comparison of 5 models' qualitative aspects.....	71

LIST OF FIGURES

Figure 1: Cyber-attack types that can be carried out through DGA implementation.	1
Figure 2: Typical architecture and components of a botnet.	2
Figure 3: General workflow of a DGA, illustrating the process from seed generation to the final domain query phase.	4
Figure 4: C2 communication mechanism established via DGA-generated domains. Once a registered domain resolves to the attacker's C2 server, a communication channel is established.	5
Figure 5: Hierarchical relationship between AI, ML, DL, and Generative AI, illustrating how each domain represents a subset of the broader AI field.	7
Figure 6: Bar chart showing the distribution of domain classes, with nearly equal representation of legitimate and DGA-generated (malicious) domains.	22
Figure 7: Pie chart illustrating the equal distribution of 25 distinct malware families, each comprising 4.0% of the DGA dataset, highlighting the balanced representation used for classification analysis.	22
Figure 8: Architecture of the proposed hybrid model. The pipeline processes domain names through n-gram CNN and LSTM to extract features, while TLD names are encoded via one-hot encoding. The resulting vectors are merged and passed to LR for final prediction.	32
Figure 9: Sample illustration of 4-gram encoding applied to the domain name "google.com", demonstrating the generation of overlapping character sequences for use in sequential feature extraction.	33
Figure 10: Output of the initial sample check function used to verify correct dataset loading and identify early anomalies. The figure presents a subset of domain entries with associated labels and origins.	42
Figure 11: Example of domain name decomposition into constituent components. Each domain is split into its subparts to facilitate structured feature extraction for classification tasks.	42
Figure 12: Final dataset obtained after extracting, standardizing and merging domain components across varying lengths.	43
Figure 13: Training log of the LSTM model over 10 epochs, showing the progression of training and validation loss and accuracy.	45
Figure 14: Training log of the GRU-based neural network over 10 epochs, showing the evolution of accuracy, AUC and loss.	46
Figure 15: Training log of the Transformer network over 10 epochs, showing the evolution of AUC and loss.	47
Figure 16: Training output of the hybrid model over 10 epochs, including accuracy, AUC and loss metrics.	49

Figure 17: List of the top 250 most frequent TLDs identified from the dataset. These TLDs were extracted during preprocessing and used to construct the feature space for domain classification.....	50
Figure 18: Evaluation metrics of the CNN model over 10 epochs, showing training and validation loss and accuracy curves.	54
Figure 19: ROC curve of the CNN model applied for DGA detection. The curve demonstrates high classification performance with an AUC score of 0.9857, indicating the model's strong ability to distinguish between legitimate and algorithmically generated domain names.....	55
Figure 20: Confusion matrix illustrating the CNN model's classification performance on test data.....	56
Figure 21: Graphic of the LSTM model over 10 epochs, showing training and validation accuracy curves.	56
Figure 22: Graphic of the LSTM model over 10 epochs, showing training and validation loss curves.	57
Figure 23: Confusion matrix illustrating the LSTM model's classification performance on test data.....	58
Figure 24: Graphic of the GRU model over 10 epochs, showing training and validation accuracy curves.	59
Figure 25: Graphic of the GRU model over 10 epochs, showing training and validation loss curves.	59
Figure 26: Graphic of the GRU model over 10 epochs, showing training and validation AUC curves.....	59
Figure 27: Confusion matrix illustrating the GRU model's classification performance on test data.....	60
Figure 28: ROC curve of the Transformer model applied to DGA detection. The curve demonstrates high classification performance with an AUC score of 0.9898, indicating the model's strong ability to distinguish between legitimate and algorithmically generated domain names.....	62
Figure 29: Confusion matrix illustrating the Transformer model's classification performance on the independent 20% test set.....	62
Figure 30: Confusion matrix illustrating the hybrid model's classification performance on training data.....	63
Figure 31: Confusion matrix illustrating the hybrid model's classification performance on the independent 20% test set.	64
Figure 32: ROC curve of the hybrid model applied to DGA detection. The curve demonstrates high classification performance with an AUC score of 0.9917, indicating the model's strong ability to distinguish between legitimate and algorithmically generated domain names.....	65

Figure 33: Bar chart comparing the hybrid model’s performance metrics between training and test datasets. The results demonstrate strong generalization capability, with minimal performance drop across evaluation phases.	65
Figure 34: ROC curves generated from 5-fold cross-validation, illustrating the model’s classification performance across multiple validation splits. The mean AUC of 0.9941 confirm the model’s robustness and generalization capability in detecting DGA-generated domains.	66
Figure 35: Workflow illustrating the automated response to DGA-based threats. The process begins with DGA detection, followed by SIEM correlation, SOAR-triggered actions, firewall intervention, and concludes with alerting and investigation in the SOC.	76

LIST OF ABBREVIATIONS

AGD	Algorithmically Generated Domains
AI	Artificial Intelligence
APT	Advanced Persistent Threat
BERT	Bidirectional Encoder Representations from Transformers
C2	Command and control
CNN	Convolutional Neural Network
Conv1D	One-Dimensional Convolutional Layer
CTA	Cyber Threat Actors
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDoS	Distributed Denial-of-Service
DGA	Domain Generation Algorithm
DL	Deep Learning
DNS	Domain Name System
DT	Decision Tree
FFN	Feed-Forward Network
FN	False Negative
FP	False Positive
FQDN	Fully Qualified Domain Name
GAN	Generative Adversarial Network
GBM	Gradient Boosting Machines
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
ICANN	Internet Corporation for Assigned Names and Numbers
ICT	Information and Communication Systems
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IoC	Indicator of Compromise
IP	Internet Protocol
IPS	Intrusion Prevention System
KNN	k-Nearest Neighbors
LR	Logistic Regression
LSTM	Long Short-Term Memory

MHSA	Multi-Head Self-Attention
ML	Machine Learning
NB	Naive Bayes
NXDomain	Non-existent Domain
RF	Random Forest
RNN	Recurrent Neural Networks
RDNS	Reverse Domain Name System
ROC-AUC	Receiver Operating Characteristic – Area Under the Curve
SIEM	Security Information and Event Management
SOAR	Security Orchestration, Automation and Response
SOC	Security Operations Center
SVM	Support Vector Machines
TLD	Top Level Domain
TN	True Negative
TP	True Positive
URL	Uniform Resource Locator
XGB	XGBoost

CHAPTER 1

INTRODUCTION

Domain Generation Algorithms (DGAs) are powerful tools in the arsenals of Cyber Threat Actors (CTAs). CTAs use DGAs to hide their command-and-control (C2) servers by directing network traffic to rapidly changing domain names. DGAs produce large numbers of domain names which are difficult to predict and block in advance. By this way, DGAs provide a resilient attack infrastructure for CTAs. In other words, DGAs can serve as an attack enabler for any cyber-attack type that can be carried out through botnets as illustrated in Figure 1.

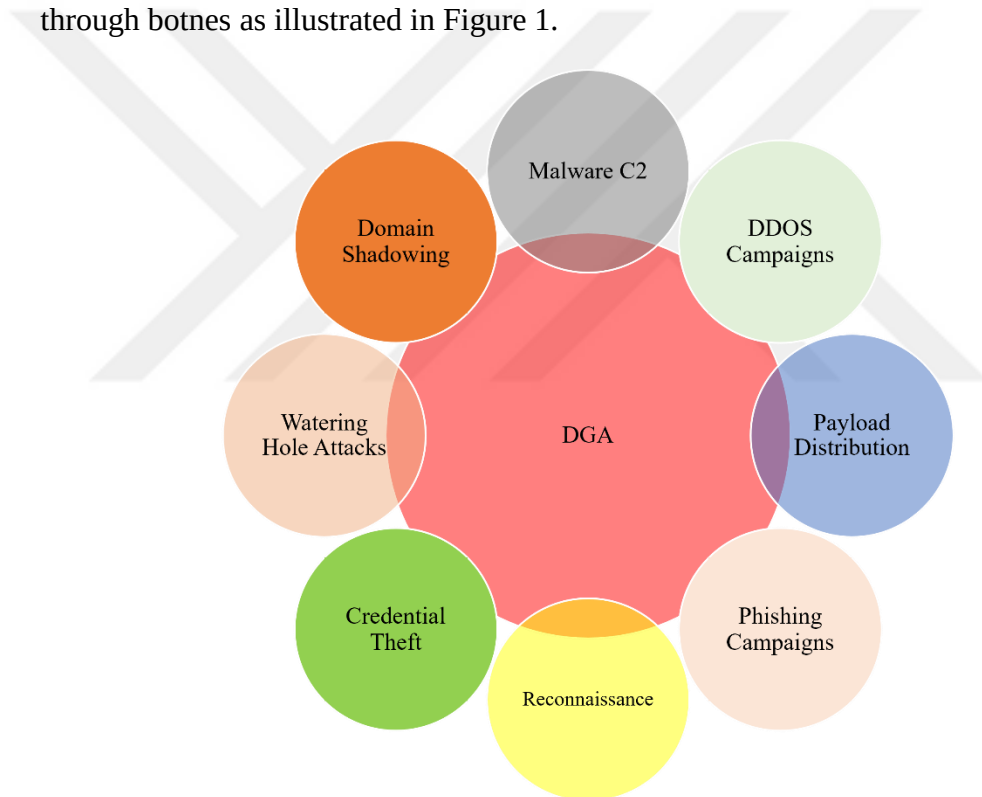


Figure 1: Cyber-attack types that can be carried out through DGA implementation.

DGAs are not attacks vectors, but they are cyber-attack enablers. From attackers' perspective, DGAs provide survivability, evasion and operational flexibility. By producing large number of random domain names, DGAs can evade static blacklists and signature-based detection and prevention systems. By this way, DGAs provide anonymity and they form an additional defensive layer for the attackers.

CTAs rely on botnets to carry out their malicious operations. A botnet is a network of compromised computers or connected devices. These devices are known as bots or zombies, and they are remotely controlled by CTAs without the owners' knowledge. These hosts are infected with malware that allows the attacker to issue commands.

Botnets are hidden for most of their lifecycle. They become active, when they receive instructions from the attacker through a C2 server. A botnet consists of a BotMaster, a C2 server and a network of bots or zombies as illustrated in Figure 2.

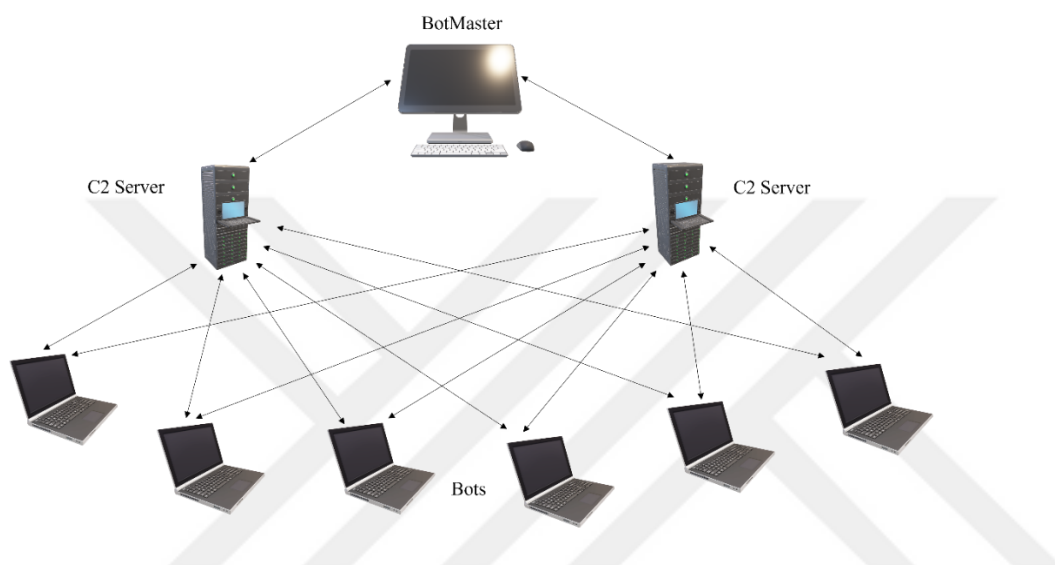


Figure 2: Typical architecture and components of a botnet.

Botnets are controlled with C2 servers. These servers act as a bridge between the BotMaster and the bots. They transmit instructions from the botmaster, spread malicious code and initiate attacks to maintain control over infected hosts.

Throughout the botnet lifecycle, communication with the C2 server is crucial. For this connection, bots must know the C2 server's IP address. However, using a fixed IP address for the C2 server has a high risk of being detected and blocked by security mechanisms like firewalls and intrusion detection or prevention systems (IDS/IPS). If defenders take down the C2 servers, the whole botnet collapse.

To solve this problem, CTAs employ DGAs. DGAs construct an operational infrastructure that directs network traffic between the attacker and botnet. In practice, traffic intended for the botnet is routed through DGA-generated domains, which resolve to the attacker's C2 IP addresses. That mechanism conceals the true location of the C2 servers. It provides resiliency, even if individual domains are taken down. The result is a covert communication layer that masks malicious activity (Tong *et al.*, 2022).

In this context, DGAs have emerged as a persistent and ever-evolving threat. As modern malware continues to evolve, DGAs continue to evolve as well. This ongoing

evolution poses a serious challenge to network security operations, and it makes DGAs a moving target for defenders. Therefore, in order to design effective detection strategies, it is essential to thoroughly analyze the structure and behavioral patterns of DGAs.

1.1. Understanding DGAs

A DGA is a program or script embedded in a malware that generates large numbers of domain names. CTAs use DGAs to create an asymmetry between attackers and defenders. While an attacker only registers a few numbers of algorithmically generated domains to the Domain Name System (DNS), defenders struggle to block every possible malicious domain pre-emptively (Patsakis *et al.*, 2021).

Security solutions such as anti-malware software, DNS sinkholes, blacklists and firewall rules have traditionally been used to prevent unauthorized traffic. However, many of these solutions rely on static signatures, hashing schemes or network communication whitelisting, which make them ineffective against adaptive threats like DGAs (Li *et. al.*, 2019).

In this context, in order to ensure network security, cybersecurity domain study how DGAs operate. By analyzing patterns in DGA-generated domains, defenders can predict and block suspicious domains before they are used. This proactive approach goes beyond reacting to known threats. Instead, it focuses on stopping attacks before they happen. By this way, overall network security can be enhanced.

1.2. How DGAs Work?

DGAs produce domain names by using predefined algorithms. These pre-arranged algorithms take current date, time, exchange rates or content fetched from popular websites as a seed. Since the botmaster and the bots share the same algorithm and seed, only the botmaster can predict which domains will be active at a given time.

The DGA generation process starts with seed generation. Subsequently, the algorithm generates a domain string according to its rules. Then, a top-level domain (TLD) is chosen and appended to form a fully qualified domain name. Afterwards, the attacker registers the selected domain in DNS. Ultimately, infected hosts and the botmaster query the same DGA generated names until a registered domain resolves to the attacker's C2 IP address. This arrangement provides a covert and resilient rendezvous mechanism between bots and their C2 servers. Figure 3 represents the general workflow of a DGA generated domain from seed generation to domain query stage.

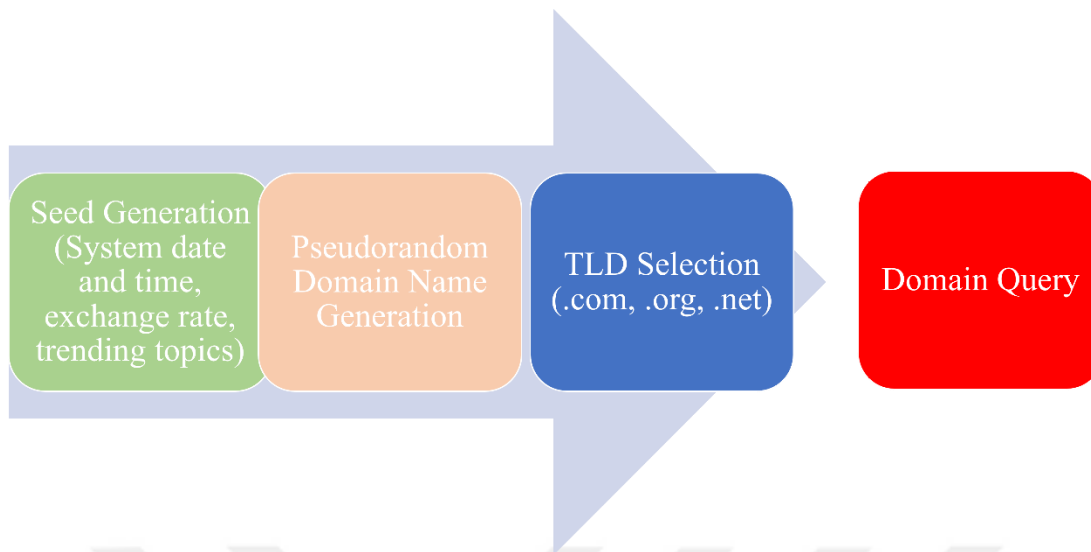


Figure 3: General workflow of a DGA, illustrating the process from seed generation to the final domain query phase.

The generation of domains in a DGA-based botnet is a coordinated process between the malware and the attacker. The malware uses DGAs to create a list of possible domain names. These DGA mechanisms often use predefined seeds with algorithmic rules. DGAs produce a large pool of domains that change frequently. Modern DGAs can produce 50,000 algorithmically generated domain names in a single day. The number of domains that some modern malware can generate in a single day, along with the seed type used are presented in Table 1.

Table 1: Sample domain names generated by DGAs with corresponding malware and used seed type.

Malware Family	Seed Type	Domains Per Day	Example
Conficker	Time-based and constant key	50,000	klvupgfm.biz
Cryptolocker	Time-based	1,000	jaorw.com
Zeus Gameover	Pseudo-random number generator	1,000	qwer12abcd.com
Matsnu	Wordlist-based	1,000	softblue-cloud.net

DGAs are embedded into malicious code to ensure the malware can keep contacting its operator. The attacker registers only a few of these domains to DNS. When the malware tries to reach its C2 server, it cycles through the generated domains until it finds one that has been registered by the attacker. If defenders block a domain, the malware simply moves on to the next and maintains uninterrupted communication. Figure 4 shows how DGAs work.

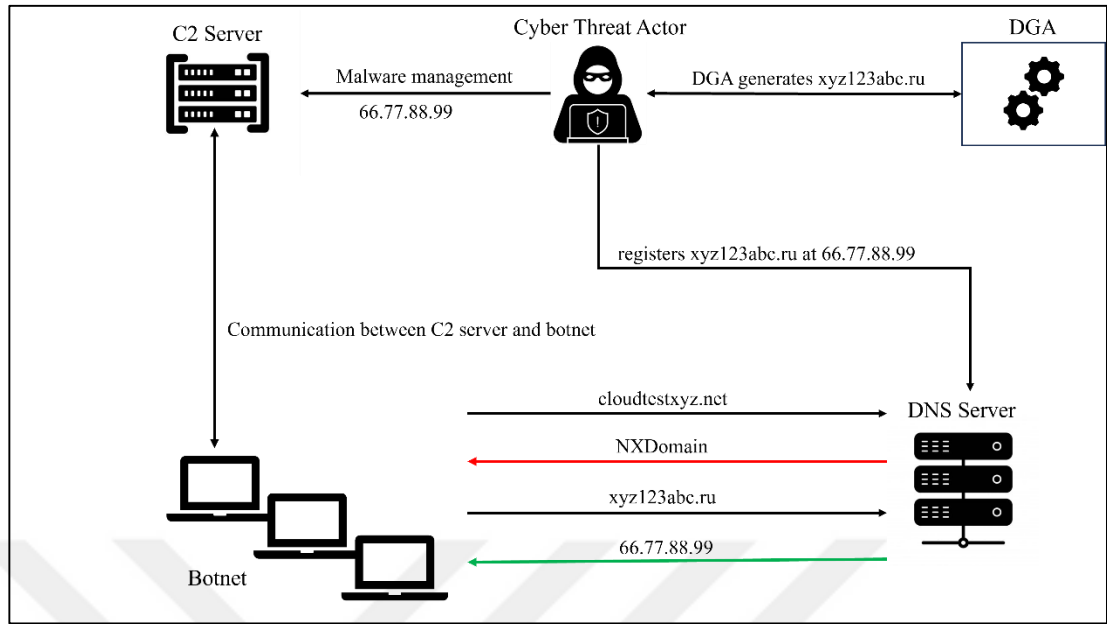


Figure 4: C2 communication mechanism established via DGA-generated domains. Once a registered domain resolves to the attacker’s C2 server, a communication channel is established.

Botmasters leverage DGAs to expose their C2 servers for the shortest possible timeframe. Domains are registered to DNS just minutes before bots attempt to query them. The C2 servers are quickly shut down afterwards. This rapid domain turnover makes static domain blacklists ineffective. Furthermore, from the cybersecurity perspective, it requires a proactive approach for detection. This approach goes beyond reacting to known threats and instead focuses on detecting malicious domains before attacks take place (Grill *et al.*, 2015).

1.3. Taxonomy of DGAs

DGAs are classified into two main categories based on their infection methods. Basically, we can classify them as binary-based DGAs and script-based DGAs.

- Binary-based DGAs are embedded as a binary code in the malware. When the malware is installed on a victim’s system, the DGA module starts running. This module creates a list of random-looking domain names by using a fixed seed value. These domains are used to connect to the attacker’s C2 server. Examples of malware using this method include GameOver Zeus, Cryptolocker, PushDo and Ramdo.
- Script-based DGAs are hidden in a JavaScript code in the browser. When a user visits a malicious web page, the script runs and generates random domain names inside HTTP URLs. First, the victim is tricked into visiting a malicious site, usually by clicking a fake link in a phishing email. Once the user clicks the link, the browser is redirected to the DGA generated domains.

As a part of efforts to classify DGAs, Plohmann *et al.* introduced the DGArchive. DGArchive is a collection of Algorithmically Generated Domains (AGD) with 92 botnet families. According to them, DGAs can be grouped into the following five categories based on employed generation techniques (Plohmann *et al.*, 2016).

- Arithmetic DGAs constitute 74 out of 92 families. They generate domain names by using simple arithmetic operations based on a random seed. The seed is typically derived from a timestamp.
- Hash DGAs constitute 11 out of 92 families. They generate domains by using a random seed. The output is further processed through a hashing function such as MD5.
- Wordlist or Dictionary DGAs constitute 4 out of 92 families. They combine common words and phrases. So, this type of DGAs resemble very much to legitimate domain names. As a result, they are the most challenging DGAs in terms of differentiating them from benign domains.
- Permutation DGAs constitute 1 out of 92 families. It generates domains by permuting a small set of seed domains.
- Adversarial DGAs constitute 2 out of 92 families. They generate domains by leveraging knowledge about non-DGA samples, decision models or previously detected families.

1.4. DGA Detection

When malware using a DGA is discovered, it must be reverse-engineered to understand the domain generation pattern. This process requires advanced reverse engineering skills and it can be time consuming. In addition, modern DGA families generate domains that resemble very much to natural language to evade detection. To address these limitations and to counteract DGA-based threats, modern detection methods focus on machine learning (ML) and deep learning (DL) models.

Traditional ML approaches to DGA detection primarily rely on linguistic analysis of domain names. ML models depend on manually engineered features. These features can be time consuming to construct. Furthermore, they often lack adaptability across diverse DGA families. ML methods involve extracting features such as domain length, character distribution, entropy or n-gram frequencies. Extracted features are then fed into ML classifiers such as Random Forests, Decision Trees or Gradient Boosted Trees (Wang *et al.*, 2018).

DL has enhanced the capability to detect DGAs, by offering considerable advantages over traditional ML approaches. DL models, especially architectures like LSTM, CNN, Gated Recurrent Units (GRU) and Transformer-based networks, can automatically extract relevant patterns from raw domain names without the need for

manual feature engineering. This allows them to capture complex and subtle differences between benign and AGDs by improving performance and reducing false positives (FP).

As malware techniques grew more advanced, botnets started to employ complex and adaptive DGAs. As a response, detection methods evolved to incorporate ML models and DL architectures for enhanced DGA detection and classification accuracy. Figure 5 shows the relationship between Artificial Intelligence (AI), ML, DL and Generative AI models with regards to DGA detection.

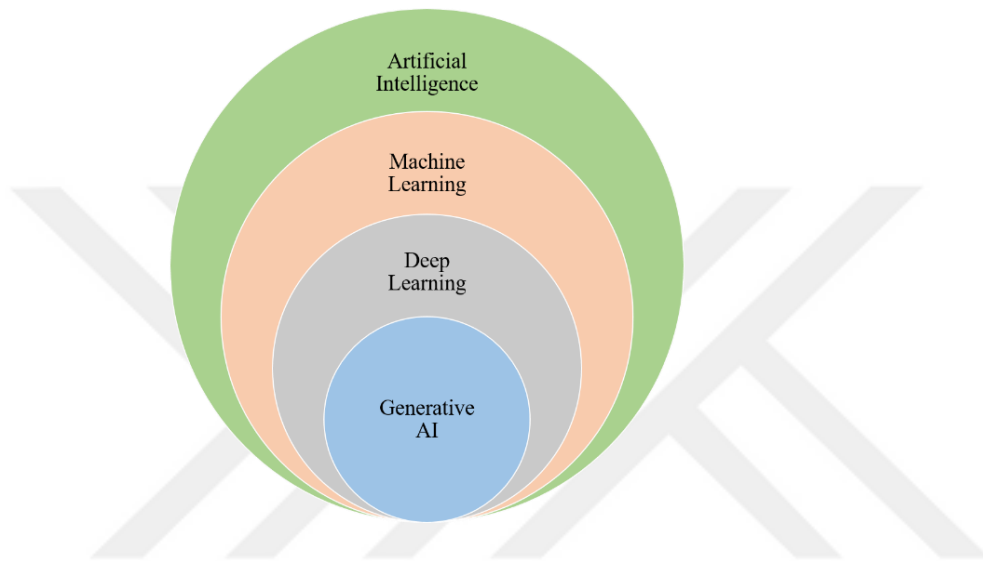


Figure 5: Hierarchical relationship between AI, ML, DL, and Generative AI, illustrating how each domain represents a subset of the broader AI field.

As can be understood from Figure 5, AI is the broadest concept. It refers to systems or machines that can perform tasks that normally require human intelligence, such as reasoning, learning, problem-solving and decision-making.

ML is a subset of AI focused on algorithms that learn from data rather than being explicitly programmed. ML requires manual feature engineering.

On the other hand, DL is a subset of ML that uses neural networks with multiple layers. It enables automatic feature extraction with hidden layers. And it handles large, complex, high-dimensional data better than traditional ML models.

Finally, Generative AI is a specialized branch of AI that creates new data such as text, images, audio, video or code, based on learned patterns from existing data. Instead of just predicting or classifying, it generates novel content. An AI tool, writing a story can be an example of Generative AI use.

In this context, DL models have been preferred in this thesis due to their superior performance compared to traditional ML approaches. Furthermore, CTAs can analyze manually engineered features which are extracted by ML models, and then they can

design their DGAs adaptively to evade detection mechanisms. To address this challenge, we propose a hybrid model that leverages the strengths of different DL models. Therefore, in our thesis, Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks will be merged into a single pipeline for automatic feature extraction. Moreover, TLD features will be leveraged for increased DGA detection performance and final classification will be carried out by a Logistic Regression (LR).

1.5. The Structure of the Thesis

The structure of the thesis is as follows.

- Chapter I provides an introduction to DGAs and the fundamental concepts of DGA detection.
- Chapter II presents a literature review of DGA detection methods.
- Chapter III introduces the main DGA detection methodology employed in this study.
- Chapter IV describes the experiments of both standalone and proposed hybrid DL models for identifying DGA-generated domains.
- Chapter V evaluates the experimental results of proposed framework and compares the results with standalone DL methods.
- Chapter VI concludes the thesis and discusses future research directions and potential improvements in DGA detection.

1.6. Aim and Scope

The primary aim of this thesis is to develop a novel and effective approach for detecting domain names generated by DGAs by the use of a hybrid DL framework. The proposed method seeks to integrate CNN based n-gram model and LSTM for feature extraction and LR for final classification. Additionally, the model input is enriched by TLD side information to increase detection performance. This framework aims to automatically extract meaningful representations from raw domain names and accurately distinguish between benign and malicious domain names.

The scope of this study includes the collection and preprocessing of real-world DGA and non-DGA domain datasets, the training of DL-based classification models and the evaluation of the proposed hybrid model by using various performance metrics, including accuracy, precision, recall, F1-score and AUC-ROC.

In addition, experiments will be conducted on standalone DL models, namely CNN, LSTM, GRU and Transformer networks for DGA detection, in order to compare their performance with the proposed hybrid approach.

1.7. Research Questions

Our research questions in this thesis are as follows.

- What is the performance of standalone CNN, LSTM, GRU and Transformer models in terms of DGA detection?
- What are the advantages of combining CNN and LSTM for feature extraction?
- How can TLD side information enhance detection performance and which TLDs are more associated with DGAs?
- How effectively can a hybrid model, combining CNN based n-gram model, LSTM networks, TLD side information and LR, detect domain names generated by DGAs compared to individual DL models?

1.8. Contributions of the Study

This thesis contributes to the field of cybersecurity by proposing a novel hybrid DL framework that combines CNN based n-gram, LSTM and LR models into a unified pipeline by also using TLD as side information for enhanced detection performance. In this model, the CNN and LSTM components complement each other for automatic feature extraction. CNN efficiently extracts local n-gram features from domain strings and the LSTM captures long-term sequential dependencies. This design leverages the individual strengths of each component and results in improved capability against various DGA families.

The study also compares the proposed hybrid model against traditional DL baselines by using a real-world dataset compiled from ICANN, Alexa and DGArchive.

And finally, the model architectures and implementation codes developed in this thesis have been made publicly available to support reproducibility and facilitate future research.



CHAPTER 2

LITERATURE REVIEW

The problem of distinguishing legitimate domain names from those generated by DGAs is an ongoing challenge. Despite years of research, modern malware continues to evolve, and this makes detection increasingly difficult. Various methods and approaches have been proposed to detect DGAs. Related works are as follows.

2.1. Statistical Models

Yadav *et al.* developed a methodology to detect botnets using DGAs, based on their statistical anomalies. The researchers analyzed DNS data to understand how domain names are formed. They compared legitimate human-created domains with algorithmically generated ones. Their main idea was malicious domains look different from normal ones, since their letters and combinations appear more random. So, they calculated statistical differences between them by using several mathematical “distance” measures. Their method used three different metrics which are Kullback-Leibler (KL) distance, Edit Distance (ED) and Jaccard Index (JI). The Jaccard Index gave the best results. Their method successfully detected known botnets like Conficker. It even discovered a previously unknown botnet, which they named Mjuyh. Their model successfully showed that DNS analysis can effectively reveal hidden botnet activity (Yadav *et al.*, 2012).

Fu *et al.* created two new types of DGAs that can trick the detection systems proposed by Yadav *et al.* They compared their new DGAs with 5 real botnet DGAs (Conficker, Zeus, Kraken, Srizbi, Torpig) and 1 research DGA (Kwyjibosing) by using the same detection methods proposed by Yadav *et al.* which are KL, ED and JI. They also used game theory to model the struggle between botmasters and security analysts, in order to find the best strategies for each side. As a result, two novel DGAs, Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs), created domains that looked much more like real domain names. Their study showed that the old detection methods (KL, ED, JI) failed to recognize them. Their true detection rate was even below 30% in tests. Their study proved that current DGA-detection methods were no longer enough, and new techniques were needed to catch these next-generation botnets (Fu *et al.*, 2017).

2.2. DNS-Based Models

Zhou *et al.* introduced a novel method to detect DGAs by leveraging DNS traffic. Instead of looking at the structure or letters of the domain names, the researchers focused on how these domains behave in DNS traffic. They observed that domains generated by DGAs usually share similar lifetime and query patterns. They realized that the legitimate domains were accessed randomly. In contrast, different DGA generated domains were queried periodically and then abandoned. Their method builds upon the observation that botnets using DGAs generate numerous non-existent domains (NXDomain) responses. The method was tested on real DNS traffic and was able to accurately identify DGA activity and even found previously unseen botnets (Zhou *et al.*, 2013).

Grill *et al.* proposed a DGA detection model that analyses network traffic by calculating the ratio of DNS requests and visited IPs for every host in the local network. Their study is driven by the fact that; a normal user should have a number of DNS requests proportional to number of contacted IPs. However, DGA-infected hosts exhibit a high ratio of DNS requests with minimal IP resolutions. The researchers created a lightweight system to detect DGAs by using network statistics. By checking if a host made too many DNS requests without connecting to new IPs, they could identify DGAs very accurately. The model achieved very high detection accuracy, with AUC score of 0.97. Their results proved that even with lightweight systems, DGAs can be detected efficiently (Grill *et al.*, 2015).

Patsakis *et al.* proposed a lightweight detection framework for identifying wordlist-based DGA domains that mimic human language. This kind of DGAs is much harder to detect. Their method builds upon the fact that, adversaries try to exploit the knowledge about defensive DGA detection algorithms. And they try to create adaptive AGDs in order to evade these detection methodologies. To tackle this problem, the authors proposed a three-layer parallel filtering system that processes NXDomain requests captured from network traffic. Their method is fundamentally based on exploiting the repeated use of words from a DGA's typically limited dictionary. The frequency of each word was tracked and an alert was raised when a word's repetition counter exceeded a fixed threshold. Their experiments were carried out on a large and diverse dataset of over 2.1 million domains. The ML-based classification filter achieved high detection rates, ranging from 85.70% to 99.74% across the various DGA families (Patsakis *et al.*, 2021).

2.3. ML Models

Traditional ML approaches for detecting DGAs focus on linguistic and statistical characteristics of domain names. These techniques extract features like domain length, character frequency patterns, entropy and n-gram distributions. These features are used as inputs to ML classifiers. Related works that used ML models are as follows.

Wang *et al.* introduced a new method to detect DGAs by using a Support Vector Machine (SVM) classifier. They extracted five statistical features from domain names which are length, entropy, vowel ratio, consecutive consonant ratio and digit proportion. Real-world datasets were implemented to differentiate between normal and DGA generated domains. Experimental results showed that the SVM-based model achieved high detection performance with accuracy over 91%, precision above 88% and recall greater than 87% and outperformed decision-tree methods. Their study highlighted the importance of lightweight, string-based detection methods (Wang *et al.*, 2018).

Soleymani *et al.* proposed an ML based framework for detecting DGAs by analyzing DNS query data. They studied four supervised learning algorithms which are Random Forest, Decision Tree, SVM and LR. Features like Shannon entropy, domain length, digit, consonant, vowel ratios and semantic similarity were engineered to distinguish legitimate and malicious domain names. The model was trained and tested on a dataset of 100,000 domain names. Random Forest outperformed other models by reaching 99% accuracy and precision rates (Soleymani *et al.*, 2021).

Prominent lexical features used in ML models are as follows.

- **Domain length:** DGA generated domains are longer than legitimate domains.
- **Character entropy:** DGA generated domains have higher randomness and entropy rate.
- **Digit-to-letter ratio:** DGAs insert numbers into domain names.
- **N-grams distribution:** Legitimate domains contain meaningful strings like dictionary words. However, DGA-generated domains look random and their n-grams are unusual compared to normal domains.
- **WHOIS registration age:** DGAs are newly registered to DNS.

Table 2 shows prominent lexical features used in ML models with both benign (0) and DGA generated (1) domain name examples.

Table 2: A sample of prominent lexical features applied in ML models.

Domain	Label	Length	Digit Ratio	Entropy	Vowel Ratio
facebook.com	0	12	0.00	3.02	0.42
google.com	0	10	0.00	2.65	0.40
outlook.live.com	0	16	0.00	2.81	0.50
xj32k9zpa.biz	1	13	0.23	3.55	0.15
abcde12345.info	1	15	0.33	3.91	0.27
xyz987vvv.net	1	13	0.23	3.33	0.08

2.4. DL Models

DL has significantly advanced DGA detection by providing notable improvements over traditional ML methods. Conventional ML models depend on manually crafted features, such as domain length, character distribution and n-gram frequencies. This feature engineering process is labor intensive. In addition, it may struggle to generalize well across different DGA families. In contrast, DL models are capable of learning meaningful patterns and semantic structures automatically from raw domain names. This enables them to identify subtle and complex distinctions between legitimate domains and AGDs. This results in higher detection accuracy and lower FP rates. Related works that used DL models are as follows.

Woodbridge *et al.* presented a DGA classifier that leverages LSTM networks for DGA detection. Unlike traditional ML-based methods, LSTMs learn patterns from the sequence of characters rather than relying on predefined features. LSTMs work as a black box. This capability makes this model very difficult for adversaries to reverse engineer. Their study showed that LSTMs outperform earlier ML models in detecting DGA generated domains with higher accuracy of 0.99 and lower FP rates of 0.001. It also showed that LSTM networks can adapt to the dynamic and evolving nature of DGAs (Woodbridge *et al.*, 2016).

In 2017, Yu *et al.* proposed and explored a novel approach of implementing supervised learning methods for DGA detection. They trained convolutional and recurrent neural networks (RNN) which bypasses the need for manual feature extraction. Their study builds upon the fact that, all previous approaches for creating DGA datasets were based on synthetic data rather than real traffic. Therefore, the performance metrics such as accuracy, FP and false negative (FN) rates, did not provide insight on real

world performances. Their work represents a significant step toward adaptive DGA detection by achieving extremely low FP rates as low as 0.01%. Their resulting classification models provide real time predictive capability on a per-domain basis. This capability is essential for actively preventing C2 communication before a connection is established. Their study shift DGA detection from reactive to proactive prevention (Yu *et al.*, 2017).

In 2019, Yu *et al.* proposed a novel approach to DGA detection by using deep neural networks (DNN) trained on DNS traffic data. They developed labelling rules based on NXDomain responses. They explored several neural architectures such as LSTM, CNN and CNN+LSTM pipeline. The hybrid model achieved a ROC-AUC score above 0.98 and outperformed traditional ML models. Their study demonstrated that deep models can learn effective features directly from domain name strings without manual engineering (Yu *et al.*, 2019).

Chethana *et al.* proposed a hybrid DL framework for detecting DGA-generated malicious domains. The authors integrated various DL architectures such as RNN, LSTM, GRU and CNN for automated feature extraction and classification of network threats. Their study evaluated performance of multiple models and showed superior results for GRU and CNN+LSTM. Both models achieved over 99% precision (Chethana *et al.*, 2022).

2.5. Hybrid Models

As botnets utilize increasingly complex and adaptive DGAs, malware techniques have become more sophisticated as well. As a response, detection strategies have also evolved accordingly. In hybrid frameworks, strong features of ML and DL models were integrated in order to improve the accuracy of DGA detection and classification. Related works that used hybrid models are as follows.

In 2019, Li *et al.* presented a comprehensive framework to detect and classify DGA-based malware by using a combination of ML and DL techniques. The proposed approach consisted of four main components. These components are a dynamic blacklist filter, a feature extractor, a two-level model for classification and a clustering prediction model based on Hidden Markov Models (HMMs). J48 and DBSCAN algorithms were used with a feature extractor which is built upon 6 linguistic and 27 DNS-based features. Additionally, the authors developed a DNN model to handle large-scale domain data. Results of DNN model was compared against traditional models and an LSTM-based approach. Their extensive evaluation on real-world datasets showed 95.89% accuracy with J48, 95.21% with HMM prediction and 97.79% with the DNN. Their study emphasized the importance of combining traditional ML with DL models in DGA detection (Li *et al.*, 2019).

In 2019, Curtin *et al.* introduced a ML framework for detecting DGA-generated domains, especially those resemble English-language domain names. They proposed a novel metric called the "smashword score" to quantify the difficulty of detecting

DGA families. Their detection system combined character-level RNN which use a generalized likelihood ratio test (GLRT). Side information from WHOIS records and Top-Level Domain (TLD) were also leveraged with GLRT. Subsequently, these features were integrated into a final LR model. This hybrid approach significantly outperformed traditional models particularly in detection of DGA families like “matsnu” and “suppobox” (Curtin *et al.*, 2019).

De Luca explored the detection of DGAs by using both traditional ML and DL methods, in his master's thesis "DGA Detection with Big Data Approaches". He implemented a mechanism consisting of two phases. First, he used feature-based supervised learning models such as Random Forest and XGBoost. Then he applied DL models like LSTM and Bidirectional LSTM directly to domain strings. He created his own dataset by getting benign domains from “Alexa” and malicious DGA domains from “Netlab 360”. According to his experiments; while DL models perform well, the XGBoost classifier showed the best performance. XGBoost achieved around 92% accuracy with a 4% FP rate (De Luca, 2023).

Liew *et al.* explored DGA detection by particularly focusing on word looking DGAs. They proposed using subword (SW) tokenization technique, which is common in natural language processing. This method proved to better capture linguistic patterns in domain names that resemble human words. They introduced two models, SW-CNN and SW-LSTM, which integrate subword-level and character-level features. While ML models like Random Forest or XGBoost were applied for random-looking domains detection, DL models were applied for word-like domains detection. Experiments on the UMUDGA dataset demonstrated that subword tokenization significantly improved classification performance for word-looking DGAs (Liew *et al.*, 2023).

In 2025, Biros *et al.* presented a two phased ML based framework for detecting DGAs. They used a large dataset comprising of 900,000 domain names from 56 malware families to train and evaluate the models. For character-based DGAs, they evaluated seven classical ML models such as Random Forest, SVM and KNN by using 35 domain name features. Random Forest emerged as the top performer by achieving 97.03% accuracy. For word based DGAs, they experimented LSTM and CNN models. The CNN model performed best, with 98.30% accuracy and strong detection performance across multiple DGA families (Biros *et al.*, 2025).

In 2025, Kusumi *et al.* presented an ML approach to detect domains generated by DGAs by using a combination of domain string characteristics, WHOIS based public information and DNS query-response logs. Their study utilized a publicly available dataset with 90,000 domain names which comprises malicious and benign domain name samples. A wide array of features was extracted including lexical properties, registration metadata and DNS resolution behavior. Decision Tree, Random Forest and XGBoost models were evaluated. The best performance was achieved by XGBoost, which obtained an F1-score of 98.4% and recall of 98.3% (Kusumi *et al.*, 2025).

2.6. Summary

As seen in the related work and literature review, DGA detection has undergone through significant evolution. The detection methods have been transitioning from statistical analysis techniques to more sophisticated ML and DL models. As botnets employ increasingly complex DGAs, detection methods also evolved to incorporate hybrid architectures. Overtime, these models have significantly enhanced the accuracy of DGA detection and classification.

Table 3 summarizes the experimental results of previous studies on DGA detection in a chronological order, carried out by several authors by employing different ML and DL models.



Table 3: Experiment results of DGA detection studies in literature, illustrated in a chronological order, showing the models employed along with the evaluation scores.

Study	Year	Dataset Size	DGA	Benign	Model	Accuracy	Precision	Recall	F1_Score	ROC-AUC
Woodbridge et al.	2016	1.750.000	750.000	1.000.000	LSTM	-	0.9942	0.9937	0.9906	0.9993
Wang et al.	2018	10.000	6.000	4.000	SVM	0.91	0.88	0.87	-	-
He et al.	2019	22.656	11.328	11.328	RF	0.941	0.938	0.943	-	-
					DT	0.907	0.909	0.905	-	-
					KNN	0.915	0.924	0.905	-	-
Li et al.	2019	1.942.000	942.000	1.000.000	LSTM	0.9822	0.9826	0.9819	0.9822	0.9970
					RF (unigram)	0.9629	0.9637	0.9624	0.9628	0.9919
					RF (bigram)	0.9728	0.9737	0.9772	0.9727	0.9965
					RF (trigram)	0.9438	0.9482	0.9424	0.9434	0.9928
					XGB (unigram)	0.9665	0.9669	0.9661	0.9664	0.9934
					XGB (bigram)	0.9852	0.9854	0.9850	0.9852	0.9986
					XGB (trigram)	0.9614	0.9618	0.9611	0.9614	0.9914
					Ensemble model	0.9982	0.9982	0.9982	0.9982	0.9999
Soleymani et al.	2021	100.000	40.000	60.000	DT	0.98	0.98	-	-	-
					RF	0.99	0.99	-	-	-
					SVM	0.96	0.96	-	-	-
					LR	0.93	0.93	-	-	-
Hu et al.	2022	4.800	2.400	2.400	Fanci SVM	0.8080	0.7488	0.9271	0.8284	-
					Fanci RF	0.8506	0.8331	0.8769	0.8544	-
					LSTM.mi	0.9452	0.9378	0.9538	0.9457	-
Gogoi et al.	2023	60.000	30.000	30.000	Transformer	0.9900	0.9900	0.9900	0.9900	0.9800
Rao et al.	2024	160.000	Not reported		BERT+RNN	0.9800	-	0.9750	-	-
					BERT+LSTM	0.9900	-	0.9900	-	-
Kusumi et al.	2025	90.000	45.000	45.000	DT	-	0.988	0.980	0.984	-
					RF	-	0.981	0.986	0.982	-
					XGB	-	0.986	0.983	0.984	-
Pan et al.	2025	500.000	250.000	250.000	AN+LSTM	-	0.9550	0.9428	0.9495	-
					CNN+BiGRU	-	0.9438	0.9325	0.9385	-
					LSTM+Attention	-	0.9502	0.9491	0.9504	-
					CNN-BiLSTM-M	-	0.9772	0.9868	0.9820	-
Lu et al.	2025	Not reported			CNN	0.9500	0.9350	0.9130	0.9240	-
					LSTM	0.9490	0.9280	0.9170	0.9220	-
					GRU	0.9510	0.9330	0.9020	0.9170	-
					Transformer	0.9580	0.9410	0.9130	0.9260	-

Note: “-” indicates that the corresponding metric was not reported in the original study.

As can be understood from Table 3, previous DGA detection studies leveraged several ML and DL methods. Research on DGA detection has evolved significantly from 2016 to 2025. Early work primarily relied on traditional ML techniques such as SVM, RF, XGB and DT. These models achieved accuracy scores between 0.90 and 0.95 on varying sized datasets.

From 2019 onward, DL models such as RNNs, LSTM, GRUs, CNNs and Transformer models were dominantly experimented for DGA detection. These models demonstrated superior performance and reported precision and recall values above 0.98. The ensemble and hybrid architectures even reported almost perfect results by achieving ROC-AUC and F1-scores above 0.99.

However, the datasets used in these studies varied in size from small datasets to very large ones. Still, in order to prevent any bias, the authors of former studies aimed to compile balanced datasets by maintaining equal proportions of DGA generated and benign domain samples. In addition to datasets, the model evaluation metrics also varied with some models only focusing on accuracy and precision and not calculating recall, F1-score and ROC-AUC. As a result, the lack of consistent evaluation metrics makes sound comparison of these models extremely difficult.

The results of previous studies summarized in Table 3, will be further utilised in Chapter 5, when comparing the results of our proposed hybrid method.



CHAPTER 3

METHODOLOGY

This chapter presents the methodology implemented for detecting domains generated by DGAs. The approach begins with training standalone DL models to establish baseline performances. Namely, CNN, LSTM, GRU and Transformer models will be trained separately for DGA detection. Subsequently, a hybrid framework will be developed. CNN based n-gram model will be combined with LSTM networks for automatic feature extraction and these features will be enriched by TLD side information. Then, combined features will be fed into an LR classifier for final classification. Finally, results and performance metrics obtained from hybrid model will be compared against standalone models. The methodology that will be implemented in this thesis is structured into following phases.

- Data Preparation,
- Model Training,
 - CNN Model Training,
 - LSTM Model Training,
 - GRU Model Training,
 - Transformer Model Training,
 - Hybrid Model Training,
- Model Evaluation.

3.1. Data Preparation

The data preparation phase is a fundamental step in both ML and DL workflows. Primary goal of this phase is to transform raw and inconsistent data into a clean and structured format suitable for model training and evaluation. This process begins with data collection. Then it is followed by cleaning procedures such as handling missing values, removing duplicates and correcting inconsistent entries.

The first step involves constructing a compiled dataset from publicly available ICANN, Alexa and DGArchive datasets. Our compiled dataset includes 674.898 domain names in total, consisting of 337.500 DGA generated domains and 337.398

benign domains. Figure 6 shows the percentage of benign vs malicious domain counts. As can be understood from the figure, we will use a balanced dataset in our thesis.

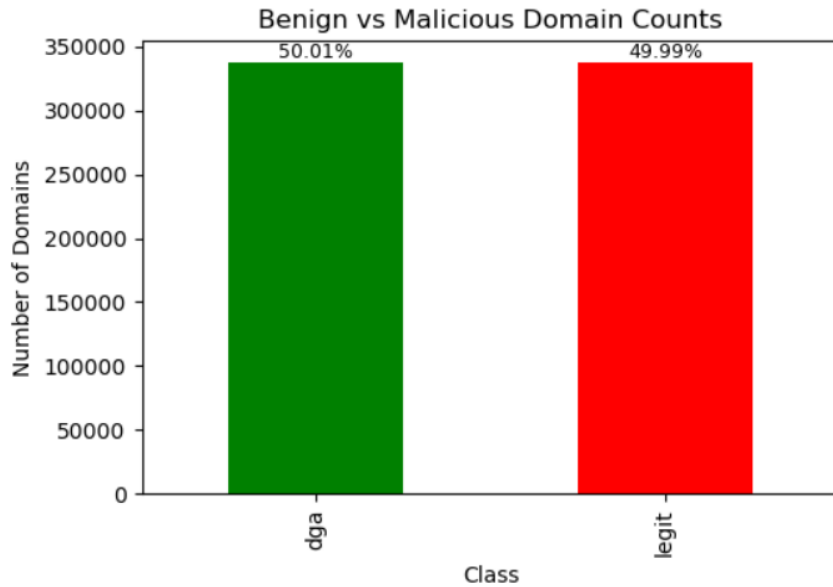


Figure 6: Bar chart showing the distribution of domain classes, with nearly equal representation of legitimate and DGA-generated (malicious) domains.

We will use DGAs from 25 different malware families with 13,500 malicious domain names selected from each family to prevent any bias. Figure 7 shows the equal malware family distribution in our dataset.

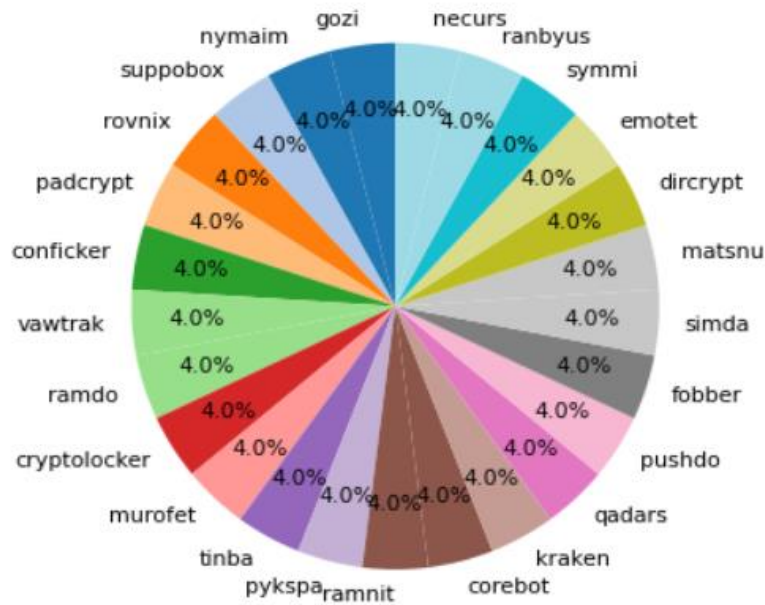


Figure 7: Pie chart illustrating the equal distribution of 25 distinct malware families, each comprising 4.0% of the DGA dataset, highlighting the balanced representation used for classification analysis.

In real-world network traffic, benign domain queries overwhelmingly outnumber DGA-generated domains. However, DGA detection studies commonly rely on balanced datasets, where benign and malicious domains are represented in roughly equal proportions. This practice is intentionally adopted to prevent classifier bias toward the majority class and to ensure that the model effectively learns discriminative patterns associated with DGA behavior.

By using balanced datasets, models are compelled to learn discriminative patterns between benign and DGA-generated domains rather than exploiting class prevalence. This experimental setup ensures that reported performance reflects the model’s actual detection capability, particularly its sensitivity to malicious domains, rather than the dominance of benign traffic. Consequently, balanced datasets are not intended to mirror real-world traffic distributions, but to provide a fair and controlled evaluation environment in which different detection approaches can be meaningfully compared.

Top-level domain (TLD) information is another important feature in our study, as it will be used to enrich the feature set in model extensions. Therefore, we analyze the distribution of TLDs and compute the top 10 TLDs for both DGA-generated and benign domains. Table 4 presents the top 10 TLDs used by DGA-generated and benign domains in our dataset, along with their corresponding frequencies.

Table 4: Top 10 TLDs used by DGA-generated and benign domains in our dataset.

Number	DGA		Benign	
	TLD	Count	TLD	Count
1	.com	112931	.com	171217
2	.net	60023	.net	16173
3	.org	28554	.org	15550
4	.ru	21482	.ru	13485
5	.eu	16683	.de	11065
6	.info	15689	.uk	6580
7	.biz	12306	.br	5774
8	.cc	7099	.jp	5696
9	.top	4478	.it	4691
10	.pw	3296	.ir	4807

The raw frequency counts show that TLDs such as ‘.com’, ‘.net’ and ‘.org’ are the most frequently used TLDs in both DGA-generated and benign domains, which is expected given their dominant presence in the global DNS ecosystem. Therefore, high usage alone does not indicate malicious intent. This confirms that frequency-based metrics can be misleading when used in isolation, as popular TLDs naturally appear in large volumes across all domain classes.

So, instead of relying solely on raw frequency counts, we will represent TLD information using one-hot encoding to capture class-specific associations. This allows the model to learn which TLDs are more strongly associated with DGA-generated domains.

Subsequently, we will perform the data cleaning phase. Data cleaning process includes following steps.

- The DGA dataset file will be read into a Pandas DataFrame.
- A header will be added.
- All values will be converted to lowercase to ensure consistency.
- Primary domain names will be separated from their TLDs.
- Missing or empty labels will be removed from the dataset.

For all experiments, the domain names in the datasets will be broken down into its components by splitting on dots. By this way, a column named “domain_list” will be created that holds these parts as lists. The primary domain and TLD components will be kept in this column.

For example, the domain "google.com" will be split into 'google' as the primary domain and 'com' as the TLD. This separation will form a new DataFrame. The same approach will be applied to domains with three, four and five components. For instance, the domain "site.co.uk" will be split into 'site', 'co', and 'uk'. In such cases, the last two components ('co.uk') represent a compound TLD. This concept is commonly seen in certain country code domain structures (co.uk, gov.tr). Once the components are extracted and standardized, they will be merged into a single clean dataset.

3.1.1. What is a Top-Level Domain (TLD)?

A TLD is the part of a domain that comes after the dot, for example, “com”, “org” or “net”. Let us take “https://www.metu.edu.tr/” as an example in order to understand URL components that will be used during the feature fusion process, as illustrated in Table 5.

Table 5: URL components of a sample domain.

Complete URL	https://www.metu.edu.tr/
Fully Qualified Domain Name (FQDN)	www.metu.edu.tr/
Subdomain	www
Primary Domain	metu
Top-Level Domain (TLD)	edu.tr

3.2. Model Training

In the model training phase, our initial focus will be on experimenting standalone DL architectures to establish baseline performances for DGA detection. Based on insights from the literature review; four models, CNN, LSTM, GRU and Transformer networks will be trained independently. These architectures were selected due to their proven success in previous DGA detection studies, as explained in Chapter 2.

Following the baseline experiments, a hybrid model will be designed to integrate the feature extraction strengths of CNN and LSTM. The input data will be further enriched by TLD side information and LR will be used for final classification. All models will be trained using the same dataset to ensure a consistent comparison.

3.2.1. Standalone DL Model Trainings

CNN, LSTM, GRU and Transformer architectures will be trained independently. These models will serve as benchmarks and will provide a basis for a comprehensive comparison with our proposed hybrid model.

3.2.1.1. CNN

CNN is a class of DL models primarily designed to extract three dimensional and local features from structured data. CNNs are highly effective for computer vision tasks like image classification, face recognition and action recognition in videos. CNNs have also demonstrated strong performance in sequence classification problems. Unlike recurrent models that process sequences step by step, CNNs apply convolutional filters across the input to detect local patterns. In the context of our thesis, CNNs are employed to capture localized character level dependencies within domain names.

DGA generated domain names often contain repetitive characters or uncommon character distributions. These indicators will help us to distinguish them from legitimate domains. CNNs are very good at detecting these local irregularities through

convolutional kernels. By employing multiple convolutional layers, the model can extract subtle patterns from raw input data without manual feature engineering. This ability to automatically learn discriminative local features makes CNNs a powerful tool for DGA detection. The CNN model takes domain names as input and converts them into numerical sequences of characters. Then, it automatically learns patterns of characters that appear in malicious domains.

Our standalone CNN model will be consisting of 9 layers. Embedding layer forms the first layer of our CNN model. It converts each character into a small dense numeric vector. Then, the first Conv1D will be implemented to detect small suspicious character patterns by scanning 3 char long characters in domain names. Following layer, BatchNormalization stabilizes learning and reduces variability. Subsequently, the second Conv1D will be implemented to detect longer irregular sequences by scanning 5 char long characters. Afterwards, a GlobalMaxPooling1D will be applied to keep only the strongest detected pattern from each filter. Next, the first dropout layer randomly turns off 50% of the neurons after the convolution and pooling layers. This prevents the model from depending too heavily on specific features and helps to avoid overfitting. The dense layer combines learned patterns into a final representation. The second dropout layer turns off 30% of the neurons and regularizes the final decision-making part of the network. Finally, the dense layer will output a probability score of a domain being DGA. The standalone CNN model architecture which will be implemented in this thesis is summarized in Table 6.

Table 6: The layers of standalone CNN model are explained along with their primary functions.

Layer	Type	Function
1	Embedding	Converts characters into dense numeric vectors.
2	Conv1D (3-char filters)	Detects small suspicious character patterns.
3	BatchNormalization	Stabilizes learning, reduces variability.
4	Conv1D (5-char filters)	Detects longer irregular sequences.
5	GlobalMaxPooling1D	Keeps only the strongest signals per pattern.
6	Dropout	Prevents overfitting.
7	Dense	Combines learned patterns into a final representation.
8	Dropout	Additional regularization.
9	Dense (1, sigmoid)	Outputs probability of being DGA.

For CNN model training, our dataset will be split into 80% training and 20% test sets. 15% of the training set will be further allocated for validation. The model will be trained for 10 epochs with a batch size of 128. Adam optimizer and binary cross entropy loss will be employed in the architecture. Model performance will be monitored using accuracy, precision, F1-score, recall and ROC-AUC metrics on both training, test and validation sets. It is expected that the model's accuracy and AUC values will improve throughout the training process. The best performance metrics will be saved for subsequent evaluation. Implementation code for our standalone CNN model has been shared in Appendix A.

3.2.1.2. LSTM

LSTM networks are a specialized form of RNNs which are designed to capture long range dependencies in sequential data. Sequential data refers to data where the order of elements matters. Each item is dependent on or related to the previous items in some meaningful way.

DGA generated domain names often have structural patterns that differentiate them from legitimate domain names. These patterns may not be easily detectable by using traditional techniques. However, LSTM excels at learning such sequential dependencies from raw input data without the need for manual feature engineering. The ability of LSTM to remember both short- and long-term dependencies allows it to capture structural characteristic of malicious domain names.

LSTM model reads domain names and learns the character patterns that distinguish benign domains from DGA generated ones. It analyses how characters follow each other. Because DGAs usually create random, meaningless sequences of characters.

Our standalone LSTM model consists of 6 layers. First layer is the Embedding layer. It converts each character in a domain name into a numeric vector. It gives the model a better understanding of which characters commonly appear together in normal domains versus DGAs. Second layer is the SpatialDropout1D layer. It randomly drops 20% of entire feature maps during training in order to prevent overfitting. Bidirectional LSTM layer with 128 units is the third layer. Bidirectional means it reads the sequence both forward and backward. So, the model understands context from both directions. Forth layer is the Dense Layer. This layer helps the model generalize what a normal or suspicious domain looks like based on the patterns found by the LSTM. Dropout layer is the fifth layer and it randomly turns off 50% of the neurons during training. This prevents the model from becoming too confident in certain neurons. Finally, the Output layer is the final decision stage. It will produce a probability between 0 and 1. The standalone LSTM model architecture which will be implemented in this thesis is explained in Table 7.

Table 7: The layers of standalone LSTM model are explained along with their primary functions.

Layer	Type	Function
1	Embedding	Converts each character into a numeric vector.
2	SpatialDropout1D	Randomly drops entire character feature maps.
3	Bidirectional LSTM (128 units)	Reads sequences forward and backward; remembers long-term patterns.
4	Dense	Combines features from the LSTM into higher-level patterns.
5	Dropout	Randomly turns off half of the neurons during training.
6	Dense (1, Sigmoid)	Outputs a single probability (0–1).

For LSTM model training, our dataset will be split into 80% training and 20% test sets. 15% of the training set will be further allocated for validation. Subsequently, the model will be trained for 10 epochs with a batch size of 128. Model performance will be monitored using accuracy, precision, F1-score, recall and ROC-AUC metrics on both training, test and validation sets. It is expected that the model’s accuracy and AUC values will improve throughout the training process. The best performance metrics will be saved for subsequent evaluation. Implementation code for our standalone LSTM model has been shared in Appendix B.

3.2.1.3. GRU

GRUs are a variant of RNNs which are designed to model long-range dependencies in sequential data. When compared to LSTM, GRUs maintain computational efficiency. Similar to LSTMs, GRUs excel at handling sequence-based tasks such as time series analysis, natural language processing, and, in the context of this study, identifying domain names generated by DGAs.

Due to their simplified gating mechanism compared to LSTMs, GRUs require fewer parameters. So, they can be trained efficiently while still maintaining the ability to capture both short- and long-term dependencies.

In this thesis, a GRU model will be utilized to learn sequential dependencies within primary domain names, since DGA generated domains often display distinctive structural patterns.

Our GRU model will compose of 7 layers. First layer will be the Embedding layer. This layer will turn each character into a numerical vector. Following layer will be the SpatialDropout1D and it will randomly drop 20% of character features during training in order to prevent overfitting. Next layer will be the Bidirectional GRU. This layer will read each domain name forward and backward and will learn how characters appear together. Subsequent layer will be the GlobalMaxPooling1D. This layer will select the most important features from the GRU output. Then, dense layer will take all the information learned by the earlier layers and will combine it. Sixth layer will be the Dropout and it will randomly turn off 40% of neurons in the dense layer during training in order to prevent overfitting. Final layer will be the dense layer which will produce a probability between 0 and 1. The standalone GRU model architecture which will be implemented in this thesis is explained in Table 8.

Table 8: The layers of standalone GRU model are explained along with their primary functions.

Layer	Type	Function
1	Embedding	Turns each character into a numerical vector that the model can process.
2	SpatialDropout1D	Randomly drops some character features during training.
3	Bidirectional GRU	Reads each domain name forward and backward, remembering character sequences using memory cells.
4	GlobalMaxPooling1D	Selects the most important features (strongest activations) from the GRU output.
5	Dense	Fully connected layer that combines extracted features into higher-level patterns.
6	Dropout	Randomly turns off 40% of neurons in the dense layer during training.
7	Dense (1, Sigmoid)	Produces a probability between 0 and 1 (DGA vs. benign).

For GRU model training, our dataset will be split into 80% training and 20% test sets. 15% of the training set will be further allocated for validation. Subsequently, the model will be trained for 10 epochs with a batch size of 128. Model performance will be monitored using accuracy, precision, F1-score, recall and ROC-AUC metrics on both

training, test and validation sets. It is expected that the model's accuracy and AUC values will improve throughout the training process. The best performance metrics will be saved for subsequent evaluation. Implementation code for our standalone GRU model has been shared in Appendix C.

3.2.1.4. Transformer Networks

Transformer networks are a family of DL models which understand relationships between different parts of a sequence without relying on recurrence or convolution. Instead, they use a mechanism called self-attention. Instead of processing sequences element-by-element like RNNs do, Transformers process the entire sequence in parallel. This feature makes transformer networks faster when compared to RNNs.

Transformer networks can be applied to almost any task involving sequential or contextual data. Natural language processing, machine translation, text classification, question answering, text summarization, image classification, object detection, anomaly detection and phishing detection are among the tasks that can be carried out by transformers. Since transformers automatically learn discriminative patterns in domain name structures, they can also be leveraged for DGA detection.

The transformer model uses a character-level encoder consisting of multi-head self-attention and feed-forward layers to extract global contextual features from domain names. Positional encoding preserves sequential information, while mean pooling aggregates attention-enhanced representations into a fixed-size vector for binary classification.

Our Transformer model implemented in this thesis is composed of 6 layers. Embedding layer forms the first layer of our Transformer model. It converts each character into a dense vector of dimension. This layer transforms domain names from integer sequences to meaningful numerical representations. Positional encoding is the second layer of our model. Positional encoding tells the model which position each character belongs to. It also captures long-range structural patterns. Transformer encoder layer is the core of the architecture. Each transformer encoder layer contains two sub-layers, which are Multi-Head Self-Attention (MHSA) and Feed-Forward Network (FFN). Following layer is Mean Pooling Layer. It converts the sequence into a single vector. Then dropout layer is added in order to prevent overfitting. Subsequently, final classification layer produces the probability of a domain being DGA. The transformer model architecture which will be implemented in this thesis is summarized in Table 9.

Table 9: The layers of Transformer model are explained along with their primary functions.

Layer	Type	Function
1	Embedding	Converts each character into a dense vector of dimension.
2	Positional Encoding	Determines the position each character belongs to.
3	Transformer Encoder	Captures global dependencies.
4	Mean Pooling	Converts the sequence into a single vector.
5	Dropout	Prevents overfitting.
6	Final Classification	Produces a probability between 0 and 1 (DGA vs. benign).

The model learns multiple types of relationships such as character distribution and entropy-like randomness simultaneously. These abilities make transformers ideal for DGA detection tasks.

For transformer model training, our dataset will be split into 80% training and 20% test sets. 15% of the training set will be further allocated for validation. Subsequently, the model will be trained for 10 epochs with a batch size of 128. Model performance will be monitored using accuracy, precision, F1-score, recall and ROC-AUC metrics on both training, test and validation sets. It is expected that the model's accuracy and AUC values will improve throughout the training process. The best performance metrics will be saved for subsequent evaluation. Implementation code for our standalone Transformer model has been shared in Appendix D.

3.2.2. Hybrid Model Training

The proposed hybrid model constitutes the main focus of our thesis. The ultimate aim is to explore the interoperability of different DL and ML models in context of DGA detection and evaluate the performance of proposed architecture against standalone methods.

In order to achieve this goal, a CNN based n-gram model will be built and merged with an LSTM network. This hybrid pipeline will act as a feature extractor. The CNN component will be utilized to capture local character level patterns within domain names. LSTM network will be applied to model long-range sequential dependencies. The outputs of both components will be unified and forwarded to fully connected layers, where feature integration will be performed. These features will be enriched by TLD side information and final classification will be carried out by LR.

This hybrid approach will provide a more robust representation of domain name characteristics. The hybrid model architecture that will be implemented in the experiments is illustrated in Figure 8.

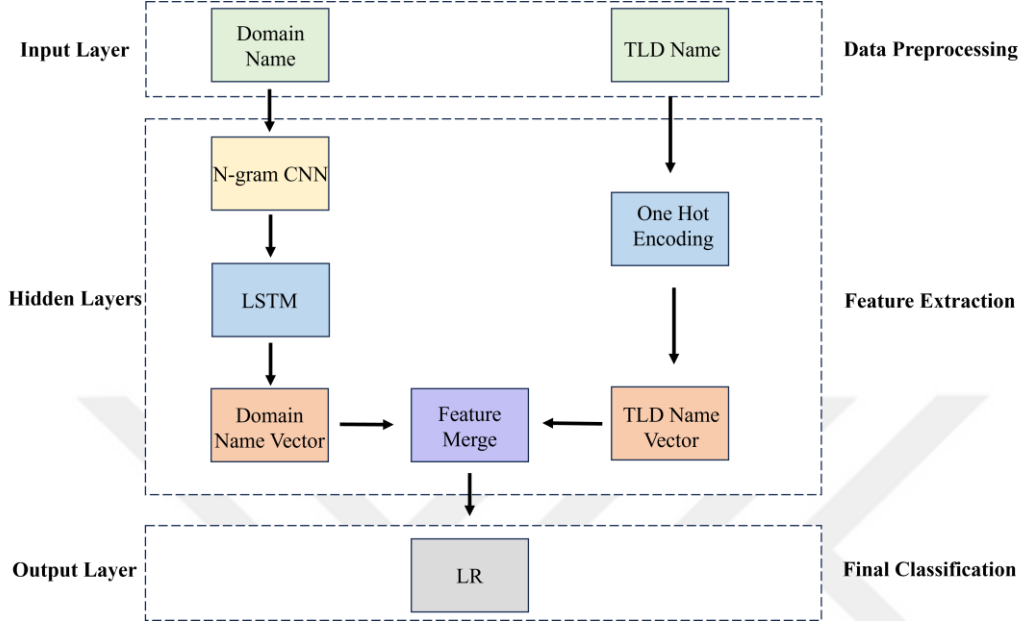


Figure 8: Architecture of the proposed hybrid model. The pipeline processes domain names through n-gram CNN and LSTM to extract features, while TLD names are encoded via one-hot encoding. The resulting vectors are merged and passed to LR for final prediction.

3.2.2.1. CNN Based n-gram Model Training

Raw domain names which are conveyed through the input layer will be fed into n-gram CNN architecture as the first step. CNNs are widely used in text-based tasks due to their ability to extract local patterns effectively. In the context of DGA detection, domain names are treated as character sequences rather than words. Therefore, it is essential to capture local dependencies such as uncommon character distributions which can be indicative of AGDs.

Traditional n-gram models explicitly generate fixed-length sequences of characters. CNN based n-gram modeling learns these local patterns automatically through trainable filters. This capability provides an efficient approach without requiring manual feature engineering.

To enhance feature representation, our proposed model utilizes 2-gram, 3-gram, 4-gram and n-gram patterns extracted from domain names. Bigrams (2-grams) are sequences of two consecutive characters from a domain name. For example, the domain “google.com” produces bigrams such as “go”, “oo”, “og”, “gl”, “le”. Similarly, trigrams (3-grams) consist of three consecutive characters. For the same domain, the resulting trigrams would be “goo”, “oog”, “ogl”, “gle”. Extending further,

4-grams are sequences of four characters, producing patterns like “goog”, “oogl”, “ogle” from “google.com”. A sample for 4-gram encoding is illustrated in Figure 9.

d	o	m	a	i	n	n	a	m	e
d	o	m	a	i	n	n	a	m	e
d	o	m	a	i	n	n	a	m	e
d	o	m	a	i	n	n	a	m	e
d	o	m	a	i	n	n	a	m	e
d	o	m	a	i	n	n	a	m	e
d	o	m	a	i	n	n	a	m	e
d	o	m	a	i	n	n	a	m	e

Figure 9: Sample illustration of 4-gram encoding applied to the domain name "google.com", demonstrating the generation of overlapping character sequences for use in sequential feature extraction.

By extracting these n-gram features, the model can capture local character patterns that differ significantly between legitimate domains and AGDs. This ability makes them highly effective for text classification tasks.

In our proposed model, the input domain names are first transformed into sequences of character embeddings. By this way, a dense representation of the input space will be created. Convolutional layers with multiple kernel sizes will then be applied to these embeddings. Each kernel size corresponds to an n-gram length. These filters will slide across the domain names and will detect local patterns that frequently appear in malicious or benign domains. The output of each convolutional filter will undergo a global max pooling operation. This operation selects the most outstanding feature across the entire sequence for each filter. It will also ensure that the most significant n-gram pattern is preserved regardless of its position within the domain name. This approach enables our CNN model to learn a combination of bigram, trigram and higher order patterns effectively.

By leveraging CNN based n-gram feature extraction, the model achieves several advantages over traditional n-gram approaches. First, it avoids the high-dimensionality related challenges with enumerating all possible n-grams, since the filters are learned dynamically during training rather than predefined. Second, CNNs are robust to variations in domain length and character distribution. This enables the model to generalize well across different DGA families and benign domains. Finally, integrating multiple kernel sizes allows the model to capture patterns at different irregularity. By this way, short repetitive sequences and longer irregular structures commonly found in AGDs can be effectively detected. These properties make CNN based n-gram modeling a powerful component for feature extraction in DGA detection tasks.

Followingly, the extracted features from the convolutional layer will be passed into LSTM layer to form a hybrid pipeline.

3.2.2.2. LSTM Model Training

In our proposed hybrid method, features extracted by the n-gram CNN model will be fed into LSTM architecture as the second step. LSTM model will be employed to capture sequential patterns within the primary domain names and will capture long-term dependencies.

In our hybrid model, the LSTM also acts as a feature extractor from domain name strings in addition to the CNN based n-gram model. Each domain name is treated as a sequence of characters. The characters are embedded into vector representations. Then, these vectors are passed through the LSTM network. At each character position, the LSTM outputs a hidden state, which is a vector that captures what the LSTM has learned up to that point in the sequence.

The sequence of hidden states is transformed into a single fixed-length feature vector that represents the entire domain name. The pooling operation over all hidden states is used to represent the domain in a feature space. The resulting vector from pooling is a numerical representation of the domain which is not human readable. However, it encodes structural, statistical and sequential characteristics learned by the LSTM model. A step-by-step example of how our LSTM model processes “google” at the character level is illustrated in Table 10.

Table 10: A step-by-step example of how LSTM processes domain characters.

Time Step	Character	Hidden State
1	g	[g]
2	o	[g → o]
3	o	[g → o → o]
4	g	[g → o → o → g]
5	l	[g → o → o → g → l]
6	e	[g → o → o → g → l → e]

Each hidden state captures both local transitions (like “oo”) and long-range dependencies (like “g...e”). Table 11 shows the key differences between CNN based n-gram and LSTM encoding techniques.

Table 11: Key differences between CNN n-gram and LSTM encoding techniques.

Feature	CNN based n-gram	LSTM
Focus	Local patterns	Entire sequence with long range dependencies
Example	"oo" in "google"	"g...e" across "google"
Strength	Detect repeated and unusual character patterns fast	Understands sequential structure and context
Limitation	Misses global order	Slower and more computationally expensive

3.2.2.3. TLD Feature Fusion

In this thesis, we will incorporate TLD information as side features to enhance the performance of our hybrid detection model. Specifically, we will extract the top 250 most frequent TLDs from the dataset. Due to the extremely short length of TLD strings, which are typically two or three characters, it would be inefficient to train an RNN or CNN solely on TLD sequences. Instead, we will employ one-hot encoding to represent the TLDs. This approach will enable the model to effectively capture the TLDs without introducing unnecessary complexity.

One-hot encoding transforms each TLD into a binary vector. In this vector, only the position corresponding to the observed TLD is set to one. All other positions remain zero. The actual learning occurs in the classifier, which identifies statistical associations between specific TLDs and the target labels. For example, the model may learn that rare TLDs such as “.xyz”, “.top” or “.info” frequently reoccur with DGA generated domains. On the other hand, more common TLDs such as “.com”, “.org” and “.net” are more often associated with legitimate domains. By capturing these distributional patterns, the model leverages TLD information as an additional discriminative signal alongside lexical features extracted from the domain strings.

The one-hot encoded TLD vectors will then be fused with the DL features extracted by the CNN-LSTM component of the model. This will create an enriched feature representation that combines both character level domain structure and TLD side information. By merging these complementary feature sets, we expect our hybrid model to learn which TLDs are strongly associated with malicious or benign domains. By this way, we plan to improve classification accuracy compared to models that only rely on raw domain name features.

Finally, this combined feature vector will be fed into an LR classifier for final prediction.

3.2.2.4. Final Classification with LR

For the final classification stage, we will employ LR as a supervised learning algorithm to distinguish between benign and DGA generated domains. In our approach, we choose LR because it provides strong performance on binary classification tasks while maintaining computational efficiency. LR will take as input the enriched feature set formed by the DL features extracted from the CNN-LSTM network with the one-hot encoded TLD features. The output will classify each domain as either 0 for legitimate (benign) domains, or 1 for DGA generated (malicious) domains.

Implementation code for our proposed hybrid model has been shared in Appendix E.

3.3. Model Evaluation

In the model evaluation phase, the performance of both standalone and hybrid models will be systematically assessed by using a consistent set of metrics. The evaluation will involve a held-out test set to provide an unbiased estimate of model performances.

Comparative analysis will be conducted between the standalone CNN, LSTM, GRU, Transformer and the proposed hybrid model. Key evaluation metrics include following parameters.

- Accuracy,
- Precision,
- Recall,
- F1-Score,
- ROC-AUC.

3.3.1. Accuracy

Accuracy is a metric that measures the overall correctness of a classification model. It represents the proportion of total predictions that the model got right, whether they were positive or negative. It gives a general sense of how often the model makes correct predictions. Accuracy is calculated by following formula.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

True Positive (TP): The number of correct DGA detections.

True Negative (TN): The number of correct benign domains.

False Positive (FP): The number of false DGA detections, where benign domains were incorrectly classified as DGA.

False Negative (FN): The number of missed DGA detections where DGAs were incorrectly classified as benign.

In a DGA detection system, accuracy indicates the percentage of both malicious and benign domains that were correctly classified by the model.

3.3.2. Precision

Precision is a metric that indicates how many of the samples predicted as positive by the model are actually correct. In other words, it measures the accuracy of the model's positive predictions. It is especially important in cases where FPs are costly or undesirable. It is calculated by following formula.

$$Precision = \frac{TP}{TP + FP}$$

In a DGA detection system, high precision means that when the model says "this is a DGA," it's very likely to be correct.

3.3.3. Recall

Recall measures the percentage of the real positive instances the model was able to detect. It is calculated by following formula.

$$Recall = \frac{TP}{TP + FN}$$

In a DGA detection scenario, a high recall means the model is good at catching most malicious domains, so it does not miss many threats. However, a high recall may come at the cost of also labeling some normal domains as malicious and consequently increasing FPs. So, it should be considered together with precision for a more balanced assessment.

3.3.4. F1-Score

F1-score is the harmonic mean of precision and recall. It provides a single score that balances both the accuracy of positive predictions and the ability to detect all actual positives. The F1-score is especially useful when the dataset is imbalanced. This parameter is obtained by following formula.

$$F1 - Score = 2 \times \frac{Recall \times Precision}{Recall + Precision}$$

This metric ranges from 0 to 1. 1 means perfect precision and recall. 0 means the model is completely failing on either or both. In a DGA detection task, a high F1-score means the model is effectively identifying malicious domains, which is critical for reducing false alarms. Table 12 shows a simple comparison of key evaluation metrics.

Table 12: Key evaluation metrics that will be used in order to compare model performances.

Metric	Formula	What it measures
Accuracy	$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$	Overall correctness, how often the model is right.
Precision	$Precision = \frac{TP}{TP + FP}$	Among predicted positives, how many are actually correct.
Recall	$Recall = \frac{TP}{TP + FN}$	Among actual positives, how many the model correctly detected (sensitivity).
F1-Score	$F1 - Score = 2 \times \frac{Recall \times Precision}{Recall + Precision}$	Balance between precision and recall, useful for imbalanced datasets.

3.3.5. Receiver Operating Characteristic – Area Under the Curve (ROC-AUC)

This parameter measures a model's ability to distinguish between the positive and negative classes across all possible classification thresholds. The ROC curve plots the True Positive Rate (TPR) or sensitivity against the False Positive Rate (FPR) at various threshold levels. The AUC summarizes this plot into a single value between 0 and 1. 1 indicates perfect classification and 0.5 represents random guessing. ROC-AUC is especially useful when dealing with imbalanced datasets, since it evaluates the model's discriminatory power. Table 13 shows AUC evaluation values.

Table 13: Evaluation of AUC values.

AUC close to 1.0	Excellent classifier
AUC around 0.5	Not better than random guessing
AUC below 0.5	Worse than random and there is an issue with the model

3.3.6. Confusion Matrix

The confusion matrix is a 2x2 table that summarizes the model's classification results. It is useful for calculating metrics like precision, recall and accuracy as shown in Table 14. It shows how many test samples were correctly or incorrectly classified by the model.

Table 14: Confusion matrix structure.

	Predicted Non-DGA (0)	Predicted DGA (1)
Actual Non-DGA (0)	True Negative (TN)	False Positive (FP)
Actual DGA (1)	False Negative (FN)	True Positive (TP)

Following the discussion of all evaluation metrics, the subsequent step is to determine which metric should be regarded as the most significant indicator of model performance in terms of DGA detection.

Accuracy, while commonly used, can be misleading in the context of DGA detection. For instance, in a dataset containing 95 benign and only 5 malicious domains, a model that classifies all domains as benign would still achieve 95% accuracy. However, it completely fails to identify any DGA generated domains. Precision, which emphasizes the correctness of malicious detections, is critical when the objective is to minimize the misclassification of benign domains as malicious. Recall, in contrast, is essential when the priority is to detect as many DGAs as possible. The F1-score forms a balance between precision and recall. Finally, AUC-ROC provides an assessment of the model's discriminative ability by illustrating how effectively it distinguishes between benign and DGA generated domains. For these reasons, AUC-ROC is widely regarded as the most critical metric in DGA detection studies.

In summary, while metrics such as accuracy, precision, recall and F1-score provide a comprehensive view of performance, ROC-AUC generally stands out as the most important metric. This is primarily due to the nature of the problem. DGA studies typically involves highly imbalanced datasets where the number of benign domains exceeds the number of malicious (DGA generated) ones. In such cases, other metrics can be misleading because a model might achieve high performance scores simply by predicting the majority class. Thus, the model may fail to effectively discriminate benign and malicious domains.

ROC-AUC offers a more nuanced assessment by measuring the model's ability to differentiate between benign and malicious domains. Therefore, ROC-AUC stands out as a more robust metric that reflects how well the model can distinguish between classes, regardless of class imbalance.

3.4. Model Validation Method

In order to evaluate the performance of our hybrid model for DGA detection, we will employ Stratified K-Fold Cross-Validation method. In this approach, the dataset is divided into K equally sized folds. Each fold keeps the same proportion of benign and malicious domains. The model is trained K times. Each time, one-fold is used for testing, and the remaining K-1 folds are used for training. After K runs, the performance scores are averaged. This method will provide an unbiased estimate of our hybrid model performance. Compared to a simple train-test split, Stratified K-Fold Cross-Validation reduces variance in evaluation metrics. Furthermore, it also helps to mitigate overfitting risks, especially when working with datasets containing diverse DGA families. In this study, we will set K to 5 as a trade off between computational efficiency and evaluation accuracy.

3.5. Experimental Environment

All models have been designed and trained in the same virtual environment by using the same dataset in order to ensure consistency. Python has been preferred as the coding language and Visual Studio Code has been used as the Integrated Development Environment (IDE).

AI assisted code optimization tools were employed to improve model training efficiency. All experimental designs and evaluations were independently developed and verified by the author. The other details regarding our experimental environment are stated below.

- **Environment:** Visual Studio Code as IDE,
- **Programming Language:** Python 3.10,
- **Hardware:** 11th Gen Intel(R) Core (TM) i5-1155G7 @ 2.50GHz,
- **Operating System:** Windows 11.

CHAPTER 4

EXPERIMENTS

This section provides a detailed overview of the experiments conducted in order to accurately detect DGA generated domains. To achieve this, standalone CNN, LSTM, GRU and Transformer models were trained and evaluated separately to serve as reference models. The rationale behind this approach is to enable a comprehensive performance comparison between these models and the proposed hybrid model. The outcomes of these experiments will be thoroughly analyzed in the next chapter, Results and Discussion. The experimental process is divided into following main stages which have already been explained in the previous chapter.

- Data Preparation,
- Model Training,
 - CNN Model Training,
 - LSTM Model Training,
 - GRU Model Training,
 - Transformer Model Training,
 - Hybrid Model Training,
- Model Evaluation.

4.1. Data Preparation

Our compiled dataset was constructed from publicly available ICANN, Alexa and DGArchive datasets. The dataset includes 674,898 domain names in total, consisting of 337,500 DGA generated domains and 337,398 benign domains. DGA generated domains equal to 50.01% and benign domains equal 49.99% of all samples. 25 different malware families with 13,500 malicious domain names were selected from each family to prevent any bias.

Subsequently, the DGA dataset was prepared for training through a series of cleaning steps.

First, the necessary libraries were imported, and the dataset was loaded into a pandas DataFrame. Pandas is a popular Python library used for data manipulation and analysis. It is used especially when working with structured data like tables, namely DataFrames in our thesis. NumPy library was also imported. NumPy is used for numerical computations.

Then, in order to verify that the dataset was correctly loaded and to identify any potential early anomalies, an initial sample check function was carried out. The results of this initial sample check are presented in Figure 10.

	label	origin	domain
0	dga	gozi	mortiscontrastatim.com
1	dga	corebot	cvyh1po636avysxebwbkn7.ddns.net
2	legit	alexa	plasticbags.sa.com
3	legit	alexa	mzltrack.com
4	legit	alexa	miss-slim.ru
5	dga	ranbyus	txumyqrubwutbb.cc
6	legit	alexa	myhostingpack.com
7	dga	symmi	ixekrihagimau.ddns.net
8	dga	emotet	rjyuosmhfaedlyg.eu
9	dga	dircrypt	djqrmauttllloabj.com

Figure 10: Output of the initial sample check function used to verify correct dataset loading and identify early anomalies. The figure presents a subset of domain entries with associated labels and origins.

Next, all domain values were standardized by converting them to lowercase. Then, domain names were broken down into their components by splitting on dots. For example, the domain "google.com" was split into 'google' as the primary domain and 'com' as the TLD. This partition forms a new DataFrame. A "domain_list" column was created which holds these parts as lists. The primary domain and TLD components were kept in "domain_list" column, as illustrated in Figure 11.

	label	origin	domain	domain_list
0	dga	gozi	mortiscontrastatim.com	['mortiscontrastatim', 'com']
1	dga	corebot	cvyh1po636avysxebwbkn7.ddns.net	['cvyh1po636avysxebwbkn7', 'ddns', 'net']
2	legit	alexa	plasticbags.sa.com	['plasticbags', 'sa', 'com']
3	legit	alexa	mzltrack.com	['mzltrack', 'com']
4	legit	alexa	miss-slim.ru	['miss-slim', 'ru']

Figure 11: Example of domain name decomposition into constituent components. Each domain is split into its subparts to facilitate structured feature extraction for classification tasks.

Once the components were extracted and standardized across all domain lengths, they were merged into a single clean dataset which is shown in Figure 12.

	primary1	primary2	tld1	tld2	label
0	mortiscontrastatim		com		dga
1	mzltrack		com		legit
2	miss-slim		ru		legit
3	bxumyqrbwutbb		cc		dga
4	myhostingpack		com		legit

Figure 12: Final dataset obtained after extracting, standardizing and merging domain components across varying lengths.

Finally, the label column is created in order to label domain names into two classes, which are 'dga' for malicious and 'legit' for benign domains. This structured preprocessing approach ensures that our models receive consistent and well-labeled input.

4.2. Model Training

4.2.1. CNN Model Training

To build and train our CNN model for DGA detection, all necessary libraries were imported and the dataset was preprocessed.

As stated in previous chapter, our standalone CNN model consisted of 9 layers. Embedding layer converted each character into a small dense numeric vector. Then, the first Conv1D was implemented to detect small suspicious character patterns by scanning 3 char long characters in domain names. Subsequently, the second Conv1D was implemented to detect longer irregular sequences by scanning 5 char long characters. Afterwards, a GlobalMaxPooling1D was applied to keep only the strongest detected pattern from each filter. Next, the first dropout layer randomly turned off 50% of the neurons after the convolution and pooling layers. This prevented the model from overfitting. The dense layer combined learned patterns into a final representation. The second dropout layer turned off 30% of the neurons. Finally, the dense layer produced a probability score of a domain being generated by DGAs.

The training process of the CNN model was initiated by first validating the input data. The feature matrix and the corresponding label vector were printed to ensure that the data had been prepared correctly. Once the data integrity was confirmed, the dataset was split into training and test subsets using an 80/20 ratio. The model was then trained with a batch size of 128. During training, both training and validation metrics were monitored.

The CNN model was trained for 10 epochs. After each epoch, loss, accuracy and the ROC-AUC scores were calculated to assess model performance. During training, the best-performing model's details were saved.

During the training phase, a portion of the dataset was reserved for validation. After the initial train–test split, in which 80% of the data was allocated for training and 20% was set aside as a held-out test set, the training subset was further divided by applying a hold-out validation strategy. Specifically, 15% of the training data, which is equivalent to 12% of the entire dataset, was used for validation.

4.2.2. LSTM Model Training

The necessary packages were imported, including Pandas and NumPy for data manipulation. Matplotlib was imported for visualization, and TensorFlow/Keras modules were imported for building and training the LSTM model. The working directory was set to the specified path where the dataset was stored to ensure correct file access.

The dataset was loaded by using Pandas, and the columns were assigned the names 'label', 'origin', and 'domain'. The total number of rows was printed, and the distribution of the labels was displayed to provide an overview of the dataset's class balance. Additionally, modules from scikit-learn were imported to enable train-test splitting.

The dataset was first cleaned by removing rows with missing values in the 'domain' or 'label' columns, and a copy of the resulting DataFrame was created. The domain names were converted to string type, and domain names were transformed to lowercase to ensure uniformity. The labels were then processed for binary classification. Non-binary labels (e.g., 'dga', 'benign') were mapped to 0 or 1 respectively.

A character-level tokenizer was applied on the domain names. Each domain was converted into a sequence of character indices. The dataset was split into training and test sets by using an 80/20 ratio.

An LSTM model was constructed using the Keras Sequential API. The model began with an embedding layer to convert character indices into dense vectors. This layer was followed by a SpatialDropout1D layer to randomly drop embedding dimensions. It randomly dropped 20% of entire feature maps during training in order to prevent overfitting.

Next, two bidirectional LSTM layers were added sequentially to capture both forward and backward dependencies in the domain sequences, so that the model can understand context from both directions.

The output of the LSTM layers was passed through Dense Layer. This layer helps the model to learn what a normal or suspicious domain looks like based on the patterns found by the LSTM. Subsequently, Dropout layer randomly turned off 50% of the neurons during training. This prevents the model from becoming too confident in certain neurons and overfitting to the training data.

These layers were followed by a final Output layer for binary classification. Performance evaluation metrics used in our model included accuracy, precision, recall, F1-score and ROC-AUC.

The LSTM model was trained for 10 epochs. After each epoch, loss, accuracy and the ROC-AUC scores were calculated to assess model performance. During training, the best-performing model's details were saved.

During the training phase, a portion of the dataset was reserved for validation. After the initial train-test split, in which 80% of the data was allocated for training and 20% was set aside as a held-out test set, the training subset was further divided by applying a hold-out validation strategy. Specifically, 15% of the training data, which is equivalent to 12% of the entire dataset, was used for validation.

The LSTM model was trained for 10 epochs as illustrated in Figure 13.

```
Epoch 1/10
4219/4219 [=====] - 8306s 2s/step - loss: 0.3110 - accuracy: 0.8649 - val_loss: 0.1941 - val_accuracy: 0.9176
Epoch 2/10
4219/4219 [=====] - 8952s 2s/step - loss: 0.1827 - accuracy: 0.9260 - val_loss: 0.1318 - val_accuracy: 0.9471
Epoch 3/10
4219/4219 [=====] - 8847s 2s/step - loss: 0.1448 - accuracy: 0.9438 - val_loss: 0.1148 - val_accuracy: 0.9554
Epoch 4/10
4219/4219 [=====] - 8644s 2s/step - loss: 0.1275 - accuracy: 0.9518 - val_loss: 0.1004 - val_accuracy: 0.9623
Epoch 5/10
4219/4219 [=====] - 8612s 2s/step - loss: 0.1188 - accuracy: 0.9554 - val_loss: 0.0956 - val_accuracy: 0.9637
Epoch 6/10
4219/4219 [=====] - 8691s 2s/step - loss: 0.1118 - accuracy: 0.9586 - val_loss: 0.0924 - val_accuracy: 0.9656
Epoch 7/10
4219/4219 [=====] - 9192s 2s/step - loss: 0.1070 - accuracy: 0.9603 - val_loss: 0.0900 - val_accuracy: 0.9665
Epoch 8/10
4219/4219 [=====] - 9253s 2s/step - loss: 0.1035 - accuracy: 0.9619 - val_loss: 0.0848 - val_accuracy: 0.9684
Epoch 9/10
4219/4219 [=====] - 9243s 2s/step - loss: 0.1004 - accuracy: 0.9632 - val_loss: 0.0843 - val_accuracy: 0.9693
Epoch 10/10
4219/4219 [=====] - 9241s 2s/step - loss: 0.0982 - accuracy: 0.9642 - val_loss: 0.0859 - val_accuracy: 0.9688
```

Figure 13: Training log of the LSTM model over 10 epochs, showing the progression of training and validation loss and accuracy.

4.2.3. GRU Model Training

The GRU network was employed to evaluate its effectiveness in detecting domains generated by DGAs. GRUs are recurrent neural networks specifically designed to capture long-term dependencies. However, they maintain a relatively simple architecture compared to LSTM networks. In our thesis, a bidirectional GRU (BiGRU) architecture was implemented to model sequential dependencies in domain names at the character level.

The input domains were first converted into lowercase and tokenized at the character level. Each domain string was mapped to an indexed sequence of characters.

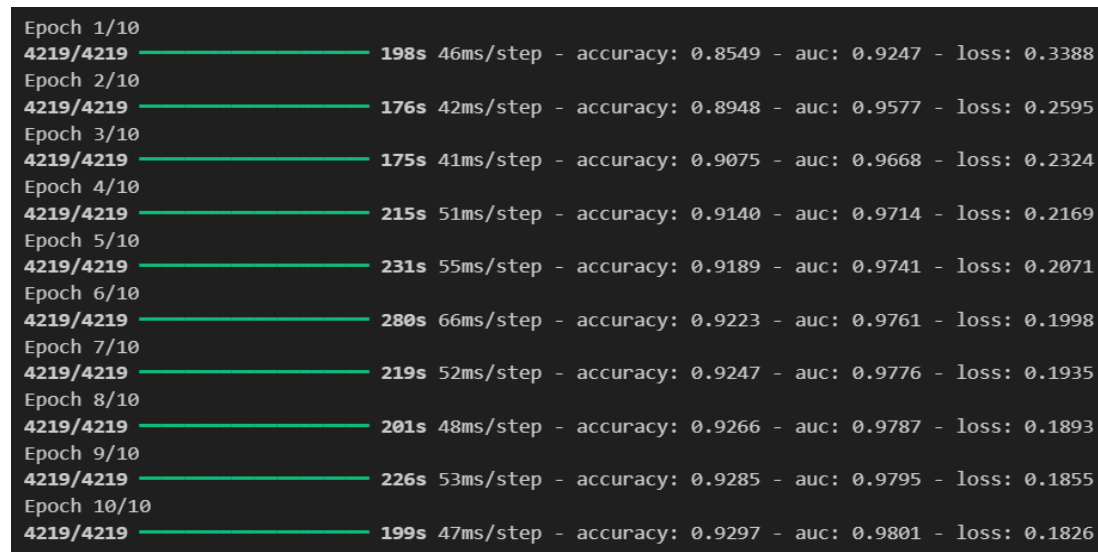
The GRU model consisted of an embedding layer, a spatial dropout layer, a bidirectional GRU layer with recurrent dropout, a global max pooling layer, fully connected dense layers and a dropout layer. This architecture was specifically chosen

to mitigate overfitting. The Adam optimizer was used with an initial learning rate of 0.001.

The GRU model was trained for 10 epochs. After each epoch, loss, accuracy and the ROC-AUC scores were calculated to assess model performance. During training, the best-performing model's details were saved.

During the training phase, a portion of the dataset was reserved for validation. After the initial train-test split, in which 80% of the data was allocated for training and 20% was set aside as a held-out test set, the training subset was further divided by applying a hold-out validation strategy. Specifically, 15% of the training data, which is equivalent to 12% of the entire dataset, was used for validation.

The model was trained for 10 epochs as illustrated in Figure 14.



Epoch	Time	Step	Accuracy	AUC	Loss
Epoch 1/10	4219s	4219	0.8549	0.9247	0.3388
Epoch 2/10	176s	4219	0.8948	0.9577	0.2595
Epoch 3/10	175s	4219	0.9075	0.9668	0.2324
Epoch 4/10	215s	4219	0.9140	0.9714	0.2169
Epoch 5/10	231s	4219	0.9189	0.9741	0.2071
Epoch 6/10	280s	4219	0.9223	0.9761	0.1998
Epoch 7/10	219s	4219	0.9247	0.9776	0.1935
Epoch 8/10	201s	4219	0.9266	0.9787	0.1893
Epoch 9/10	226s	4219	0.9285	0.9795	0.1855
Epoch 10/10	199s	4219	0.9297	0.9801	0.1826

Figure 14: Training log of the GRU-based neural network over 10 epochs, showing the evolution of accuracy, AUC and loss.

4.2.4. Transformer Model Training

To build and train a Transformer network for DGA classification, all necessary libraries were imported and the dataset was preprocessed.

The transformer model consisted of Embedding, Positional encoding, Transformer encoder, Mean Pooling, Dropout and Final classification layers.

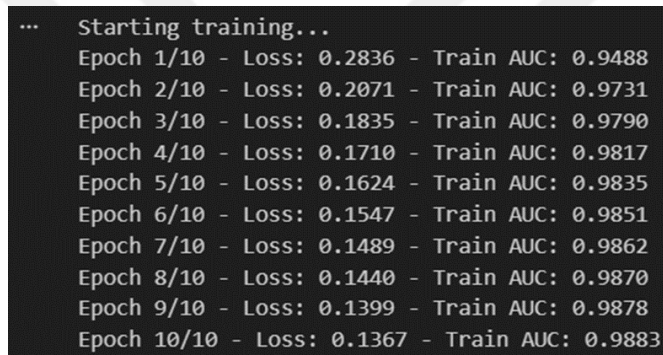
Embedding layer converted each character into a dense vector of dimension. This layer transformed domain names from integer sequences to meaningful numerical representations. Positional encoding layer captured long-range structural patterns. Subsequently, transformer encoder layer formed the core of model training. Next,

mean pooling layer converted the sequence into a single vector. Then dropout layer was added in order to prevent overfitting. Ultimately, final classification layer produced the probability of a domain being generated by DGAs.

The Transformer model was trained for 10 epochs. After each epoch, loss, accuracy and the ROC-AUC scores were calculated to assess model performance. During training, the best-performing model's details were saved.

During the training phase, a portion of the dataset was reserved for validation. After the initial train-test split, in which 80% of the data was allocated for training and 20% was set aside as a held-out test set, the training subset was further divided by applying a hold-out validation strategy. Specifically, 15% of the training data, which is equivalent to 12% of the entire dataset, was used for validation.

The model was trained for 10 epochs as illustrated in Figure 15.

A screenshot of a terminal window showing the training progress of a Transformer network. The text is white on a black background. It starts with '... Starting training...' followed by 10 lines of epoch data. Each line shows the epoch number, the loss, and the training AUC score. The loss decreases from 0.2836 to 0.1367, and the AUC increases from 0.9488 to 0.9883.

```
... Starting training...
Epoch 1/10 - Loss: 0.2836 - Train AUC: 0.9488
Epoch 2/10 - Loss: 0.2071 - Train AUC: 0.9731
Epoch 3/10 - Loss: 0.1835 - Train AUC: 0.9790
Epoch 4/10 - Loss: 0.1710 - Train AUC: 0.9817
Epoch 5/10 - Loss: 0.1624 - Train AUC: 0.9835
Epoch 6/10 - Loss: 0.1547 - Train AUC: 0.9851
Epoch 7/10 - Loss: 0.1489 - Train AUC: 0.9862
Epoch 8/10 - Loss: 0.1440 - Train AUC: 0.9870
Epoch 9/10 - Loss: 0.1399 - Train AUC: 0.9878
Epoch 10/10 - Loss: 0.1367 - Train AUC: 0.9883
```

Figure 15: Training log of the Transformer network over 10 epochs, showing the evolution of AUC and loss.

4.3. Hybrid Model Training

The hybrid model integrated a CNN-based n-gram encoder with an LSTM network for automated feature extraction. The CNN component was used for learning local character dependencies from n-gram representations of domain names. These features were then processed by the LSTM, which captured sequential dependencies to enhance discrimination between classes. No manual feature engineering was required, and the input data was further enriched by TLD feature fusion. Ultimately, final classification was performed by the LR to determine whether a domain was generated by a DGA. This hybrid DL architecture was designed to outperform standalone models by effectively combining character-level patterns with TLD side information.

4.3.1. CNN Based n-gram Encoding

The first step of our proposed hybrid model is to design and train a CNN based n-gram encoding architecture.

In this phase, the necessary libraries were imported to configure the modeling environment. From TensorFlow's Keras module, key components such as one-dimensional convolutional layer (Conv1D), GlobalMaxPooling1D, Embedding, Dropout and Dense layers were utilized to construct the CNN based n-gram model. The Sequential class was used to build the model in a linear, stackable fashion.

During preprocessing, class labels were extracted from the label column and domain names from the primary1 column. A character-level tokenization process was applied. By this process, each unique character was assigned to a unique integer index and a vocabulary mapping was created. Domain names were then converted into sequences of these integers. The resulting sequences were fed into an Embedding layer, which learned dense 128-dimensional vector representations for each character.

Following the embedding, CNN layer extracted local patterns from the embedded sequences. A GlobalMaxPooling1D layer was implemented to reduce the feature dimensionality by retaining only the most significant activation from each feature map. Then, a dropout layer was included to mitigate overfitting. Finally, the output was passed through a fully connected Dense layer to LSTM model of the pipeline.

4.3.2. CNN-LSTM Hybrid Pipeline

In this phase, necessary libraries required to construct and train a LSTM model for sequential data were imported.

A CNN–LSTM pipeline was built using TensorFlow's Keras API. The sequential class was employed to stack layers in a linear order. An embedding layer was first added and this layer was used to transform integer-encoded tokens into 128-dimensional vectors. A ReLU activation was then added to extract local n-gram features from the embedded sequences. The output from the convolutional layer was passed to an LSTM layer with 128 units, which was designed to capture sequential patterns in the extracted features. A Dropout layer with a rate of 0.5 was applied afterward to reduce overfitting by randomly setting half of the input units to zero during training. The model was compiled using the binary cross-entropy loss function and the Adam optimizer.

Subsequently, the CNN–LSTM pipeline was trained and evaluated for binary classification by using domain name data. Initially, labels and raw domain names were extracted from the input DataFrame. A vocabulary of unique characters was created by mapping each character to a unique integer. Labels were encoded into binary format, with 'legit domains' mapped to 0 and DGA generated domains mapped to 1. The dataset was split into training and testing subsets. 20% of the dataset was reserved for testing. A further 15% of the training data was separated as a holdout validation

set to monitor performance. The CNN-LSTM pipeline was trained as shown in Figure 16.

```
Epoch 1: auc = 0.981427 (best=0.970022)
4219/4219 ————— 75s 18ms/step
confusion matrix on test data:
[[63985  3481]
 [ 5818 61696]]
4008/4008 ————— 1886s 471ms/step - accuracy: 0.9330 - loss: 0.1684
844/844 ————— 10s 12ms/step
Epoch 2: auc = 0.985849 (best=0.981427)
4219/4219 ————— 58s 14ms/step
confusion matrix on test data:
[[63715  3751]
 [ 4320 63194]]
4008/4008 ————— 17353s 4s/step - accuracy: 0.9436 - loss: 0.1439
844/844 ————— 17s 20ms/step
Epoch 3: auc = 0.988672 (best=0.985849)
4219/4219 ————— 107s 25ms/step
...
[[64475  2991]
 [ 3193 64321]]
4008/4008 ————— 619s 155ms/step - accuracy: 0.9645 - loss: 0.0923
844/844 ————— 20s 24ms/step
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 16: Training output of the hybrid model over 10 epochs, including accuracy, AUC and loss metrics.

After each epoch, the ROC-AUC score and confusion matrixes were calculated to assess model performance. Early stopping was applied by terminating the training if the AUC did not improve for more than five consecutive epochs. Throughout training, the best-performing model's details were saved.

4.3.3. TLD Feature Fusion

We incorporated TLD information as side features to enhance the performance of our hybrid detection model. Specifically, we extracted the top 250 most frequent TLDs from the dataset. We employed one-hot encoding to represent the TLDs. The one-hot encoded TLD vectors were then fused with the DL features extracted by the CNN-LSTM component of the model. In the next step, we combined this CNN-LSTM feature vector with the one-hot encoded TLD vector. This merge created an enriched feature representation that combined both character-level domain structure and TLD side information. Finally, this unified feature vector was fed into an LR classifier for final prediction.

There is no universal or generic list of the top 250 TLDs, as TLD frequency varies across datasets. Therefore, our ranking comes from our own dataset. Otherwise, relying on a public ranking may not align with our thesis. The top 250 TLDs identified from our dataset are illustrated in Figure 17.

```
array(['com', 'net', 'org', 'ru', 'info', 'eu', 'co', 'biz', 'de', 'cc',
      'in', 'it', 'ir', 'top', 'cn', 'pl', 'jp', 'fr', 'me', 'pw', 'tv',
      'su', 'cz', 'es', 'tw', 'gov', 'nl', 'edu', 'ca', 'online', 'ch',
      'ac', 'xyz', 'gr', 'at', 'us', 'be', 'io', 'ro', 'mx', 'se', 'hu',
      'dk', 'sk', 'kz', 'website', 'pro', 'no', 'vn', 'cl', 'fi', 'la',
      'kr', 'sc', 'sh', 'tj', 'club', 'tmall', 'ie', 'cx', 'pt', 'nu',
      'to', 'by', 'bz', 'cm', 'az', 'ki', 'mn', 'mu', 'im', 'nf', 'id',
      'pk', 'tk', 'ms', 'lv', 'xxx', 'ga', 'ws', 'sg', 'ua', 'am', 'ps',
      'so', 'hn', 'ug', 'li', 'bit', 'win', 'ec', 'ht', 'sx', 'bg', 'gs',
      'uz', 'space', 'site', 'xn--plai', 'lt', 'ae', 'my', 'lk', 'blog',
      'pe', 'hk', 'ph', 'mobi', 'name', 're', 'mo', 'fm', 'bt', 'tf',
      'ad', 'tel', 'hr', 'wtf', 'uk', 'si', 'yt', 'tn', 'cat', 'is',
      'mv', 'md', 'tm', 'lu', 'cd', 'rs', 'ag', 'ly', 'bo', 'bid', 'tl',
      'br', 'vc', 'asia', 'as', 'ee', 'kn', 'tc', 'free', 'gd', 'dj',
      'dm', 'ml', 'gy', 'mw', 'lc', 'tech', 'ne', 'tr', 'ge', 'ma',
      'click', 'news', 'cf', 'link', 'download', 'live', 'life', 'today',
      'tokyo', 'th', 'stream', 'ai', 'al', 'ng', 'ba', 'do', 've',
      'guru', 'shop', 'gg', 'dev', 'cu', 'ar', 'mk', 'travel', 'cloud',
      'store', 'dz', 'coop', 'nz', 'qa', 'network', 'video', 'rocks',
      'media', 'eus', 'aero', 'world', 'ninja', 'men', 'press', 'work',
      'sexy', 'center', 'sa', 'ovh', 'review', 'agency', 'loan', 'mil',
      'one', 'academy', 'gq', 'host', 'date', 'vip', 'trade', 'tips',
      'fun', 'pa', 'london', 'gdn', 'cr', 'kg', 'global', 'jobs', 'city',
      'zone', 'design', 'chat', 'company', 'porn', 'nc', 'ltd', 'int',
      'moe', 'med', 'guide', 'solutions', 'berlin', 'au', 'digital',
      'faith', 'pics', 'nyc', 'lol', 'tz', 'blue', 'paris', 'ke', 'art',
      'ci', 'st', 'party', 'uy'], dtype=object)
```

Figure 17: List of the top 250 most frequent TLDs identified from the dataset. These TLDs were extracted during preprocessing and used to construct the feature space for domain classification.

To identify which TLDs are more strongly associated with DGA-generated domains, we employed one-hot encoding after extracting the top 250 TLDs from the dataset. Table 15 presents the top 10 TLDs exhibiting the strongest association with DGAs.

Table 15: Top 10 TLDs exhibiting the strongest association with DGAs.

Number	TLD	DGA Count	Benign Count	Probability
1	.top	4478	355	0.9265
2	.pw	3296	268	0.9248
3	.cc	7099	609	0.9209
4	.biz	12306	1069	0.9200
5	.su	2840	257	0.9170
6	.eu	16683	1741	0.9055
7	.info	15689	3323	0.8252
8	.net	60023	16173	0.7877
9	.me	2550	1188	0.6821
10	.org	28554	15550	0.6474

Although TLDs such as ‘.com’, ‘.org’, and ‘.net’ are frequently observed among DGA-generated domains, they are also widely used by benign domains, limiting their discriminative power. In contrast, several TLDs including ‘.top’, ‘.pw’, ‘.cc’, ‘.biz’ and ‘.su’ occur disproportionately in malicious contexts compared to benign usage, making them strong indicators of DGA activity. These TLDs exhibit very high DGA association probabilities above 90%.

TLDs such as ‘.net’ and ‘.org’ occupy an intermediate position. While they are heavily used by DGAs in high numbers, they also maintain high benign usage. This suggests that these TLDs are attractive to attackers due to their availability and perceived legitimacy, yet they cannot be considered strong standalone indicators of maliciousness without additional lexical or behavioral features.

Importantly, these association scores are dataset-dependent. Therefore, the results should be interpreted as probabilistic tendencies of utilised dataset rather than universal rules.

4.3.4. Final Classification with LR

For the final classification stage, we employed LR as a supervised learning algorithm to distinguish between benign and DGA-generated domains. The LR classifier took

the enriched feature set as input, which was formed by the DL features extracted from the CNN-LSTM pipeline together with the one-hot encoded TLD features. The output classified each domain as either 0 for benign domains or 1 for DGA generated domains.



CHAPTER 5

RESULTS AND DISCUSSION

This chapter presents and discusses the experimental results obtained from both the standalone DL models and the proposed hybrid model for DGA detection. The standalone models, CNN, LSTM, GRU and Transformer networks serve as baseline architectures. These models provide a foundation for comparative analysis of their individual strengths and limitations. Building on strengths of these baselines, the performance of proposed hybrid model is evaluated for distinguishing between benign and DGA-generated domains.

To ensure a comprehensive evaluation, performance metrics including accuracy, precision, recall, F1-score and ROC-AUC are utilised. These metrics enable a direct comparison between the baseline models and the hybrid framework. All experiments were conducted on the same dataset using a consistent preprocessing pipeline to enable fair comparison.

The following subsections detail the experimental outcomes of each model. Their respective strengths and limitations are highlighted, and their overall suitability for DGA detection are assessed according to the reported metrics.

5.1. Standalone DL Model Results

5.1.1. CNN

The CNN model demonstrated a consistent improvement in both training and validation performance during 10 epochs, as seen in the provided results. The training accuracy increased from 87.79% in the first epoch to 92.70% in the final epoch. The validation accuracy also improved, by reaching to 93.95% by the last epoch. Similarly, both training and validation losses decreased consistently, with training loss dropping from 0.3490 to 0.2307, and validation loss decreasing from 0.2453 to 0.1999. Figure 18 illustrates the graphics of CNN model's loss and accuracy scores.

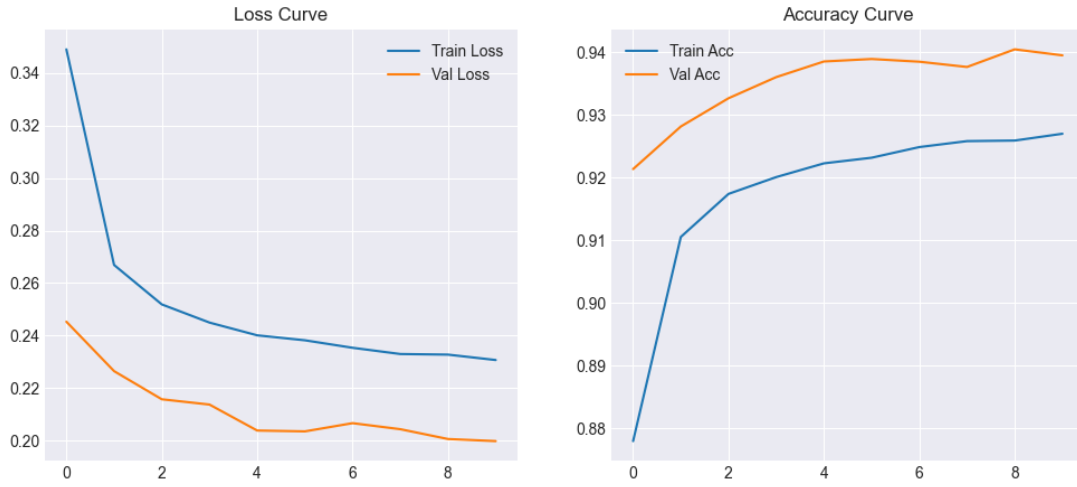


Figure 18: Evaluation metrics of the CNN model over 10 epochs, showing training and validation loss and accuracy curves.

This trend suggests that the model was learning effectively without overfitting. Because the gap between training and validation accuracy remained small during 10 epochs.

The final evaluation, conducted on the independent 20% test set, yielded a test accuracy of 94.03%, precision of 92.90%, recall of 95.35%, F1-score of 94.11%, and an AUC of 98.57. The training, validation and test results of standalone CNN model is illustrated in Table 16.

Table 16: The training, validation and test results of standalone CNN model.

	Accuracy	Precision	Recall	F1-score	ROC-AUC
Training	0.9270	0.9204	0.9423	0.9317	0.9845
Validation	0.9395	0.9310	0.9511	0.9407	0.9855
Test	0.9403	0.9290	0.9535	0.9411	0.9857

An ROC-AUC score of 0.9857 indicates that the CNN model achieved a strong performance in distinguishing between DGA-generated and benign domains. This value suggests that the model has a high TP rate while maintaining a relatively low FP rate. In the context of DGA detection, an ROC-AUC score of 0.9857, as seen in Figure 19, reflects the model's effectiveness in capturing the underlying patterns of malicious domain names. This proves CNN as a reliable classifier for DGA detection task.

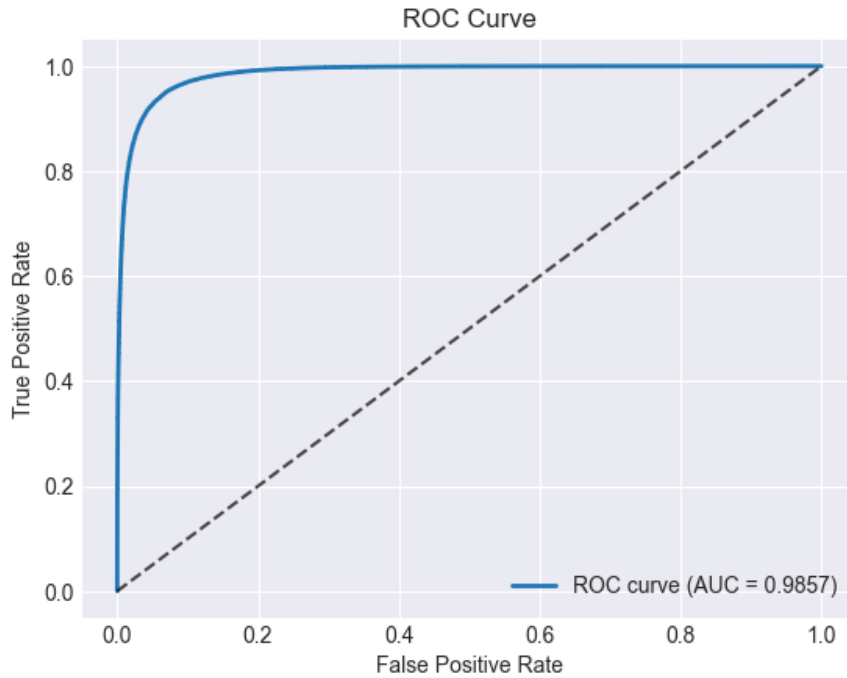


Figure 19: ROC curve of the CNN model applied for DGA detection. The curve demonstrates high classification performance with an AUC score of 0.9857, indicating the model's strong ability to distinguish between legitimate and algorithmically generated domain names.

The results indicate that CNNs are highly effective at extracting local patterns, such as n-grams or repeated substrings from domain names. Also, CNNs benefit from faster training times compared to recurrent architectures.

The CNN model confusion matrix, which is shown in Figure 20, illustrates the performance of the model in distinguishing between benign and DGA-generated domains. Out of the total benign samples, 62,580 were correctly classified, while 4,920 were misclassified as DGAs. Similarly, for the malicious DGA domains, 64,345 were accurately identified, with 3,135 instances incorrectly labeled as benign. This distribution indicates that the model demonstrates strong classification capability, with high numbers of TPs and TNs compared to the relatively lower FP and FN counts. In practical terms, this means the model is effective in both detecting malicious domains and avoiding unnecessary misclassification of benign ones. This is crucial for reducing false alarms while maintaining robust detection accuracy.

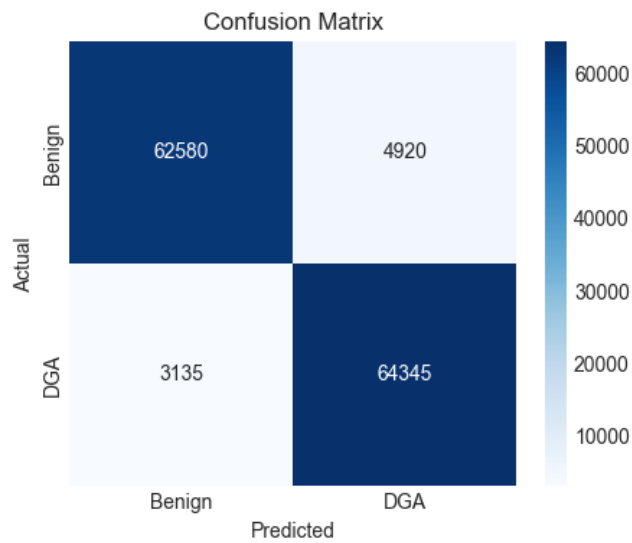


Figure 20: Confusion matrix illustrating the CNN model’s classification performance on test data.

In summary, the CNN model achieved solid performance during training. These results make CNN a reliable baseline for comparison with more complex architectures. Overall, CNNs provide a lightweight and efficient approach, making them well suited for DGA detection.

5.1.2. LSTM

The LSTM model exhibited a consistent improvement in both training and validation performance during 10 epochs. Training accuracy increased from 86.49% in the first epoch to 96.42% in the final epoch. The validation accuracy also improved from 91.76% to 96.88% as illustrated in Figure 21.

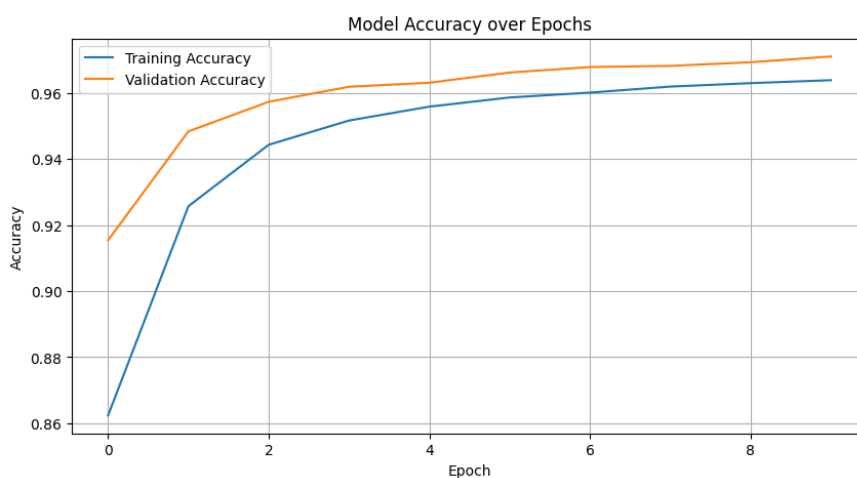


Figure 21: Graphic of the LSTM model over 10 epochs, showing training and validation accuracy curves.

Both training and validation losses followed a consistent downward trend, which decreased from 0.3110 to 0.0982 for training loss and from 0.1941 to 0.0859 for validation loss as seen in Figure 22.

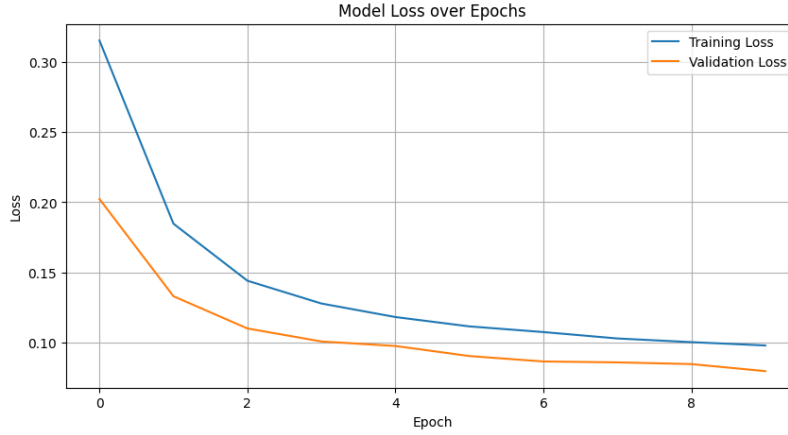


Figure 22: Graphic of the LSTM model over 10 epochs, showing training and validation loss curves.

This trend suggests that the LSTM model was learning effectively without significant overfitting, as the gap between training and validation performance remained small. The parallel improvements in validation accuracy and loss further indicate strong generalization capability to unseen data.

The test results also demonstrated strong generalization, and achieved an accuracy of 96.88%, precision of 97.18%, recall of 96.56%, F1-score of 96.87%, and an AUC of 96.88%. The training, validation and test results of standalone LSTM model is illustrated in Table 17.

Table 17: The training, validation and test results of standalone LSTM model.

	Accuracy	Precision	Recall	F1-score	ROC-AUC
Training	0.9642	0.9701	0.9642	0.9649	0.9685
Validation	0.9688	0.9711	0.9670	0.9683	0.9687
Test	0.9688	0.9718	0.9656	0.9687	0.9688

An AUC score above 0.9688 on the test set highlights the LSTM's effectiveness in distinguishing between DGA-generated and benign domains. The results demonstrate that the model maintains a high TP rate while keeping the FP rate low. In the context of DGA detection, this performance reflects the LSTM's ability to capture the sequential dependencies of AGDs. Confusion Matrix of standalone LSTM model is illustrated in Figure 23.

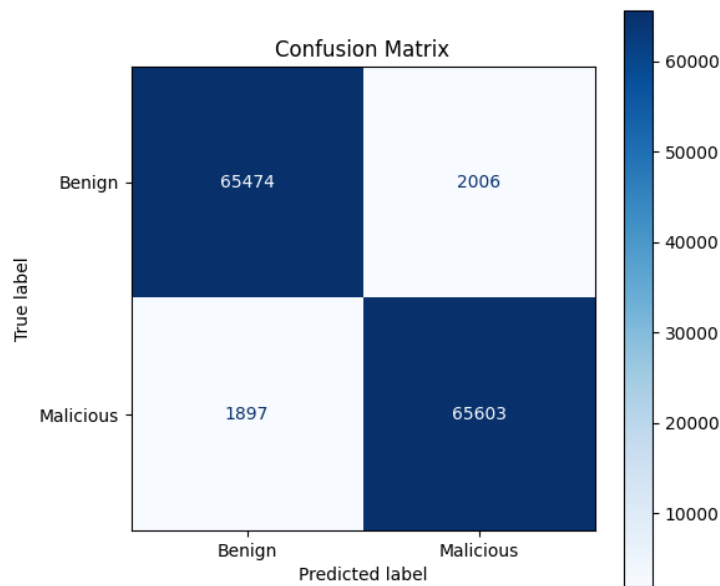


Figure 23: Confusion matrix illustrating the LSTM model’s classification performance on test data.

The results indicate that LSTMs are highly effective at modeling long-range dependencies in domain name sequences. This enables them to detect subtle sequential irregularities often embedded in DGA-generated domains. Unlike CNNs, which focus on extracting local n-gram-like patterns, LSTMs excel in capturing broader contextual relationships across the entire sequence. However, LSTMs generally require longer training times and higher computational resources compared to CNNs. Despite this limitation, their strong performance in this study underscores their suitability for DGA detection.

5.1.3. GRU

The GRU model was trained for 10 epochs with early stopping and dropout regularization to mitigate overfitting. During training, both training and validation performance improved steadily across epochs. The training accuracy increased from 85.49% in the first epoch to 92.97% in the final epoch. Similarly, training AUC improved from 0.9247 in the first epoch to 0.9801 at epoch ten. In addition, training loss consistently decreased from 0.3388 to 0.1826.

The parallel improvements in validation accuracy, AUC and loss further indicate strong generalization capability to unseen data. The gap between training and validation performance remained small during 10 epochs. Figure 24-26 illustrates the graphic depiction of GRU model’s accuracy, AUC and loss performances for both training and validation sets.

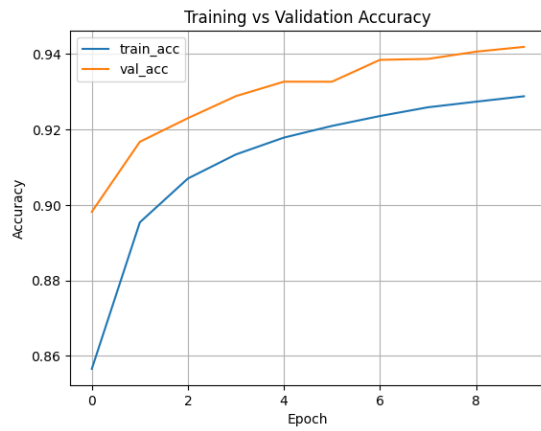


Figure 24: Graphic of the GRU model over 10 epochs, showing training and validation accuracy curves.

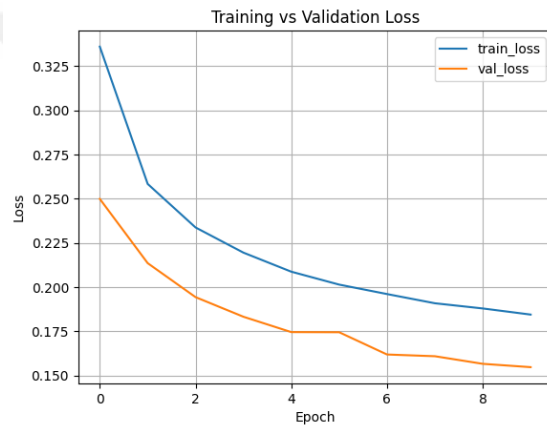


Figure 25: Graphic of the GRU model over 10 epochs, showing training and validation loss curves.

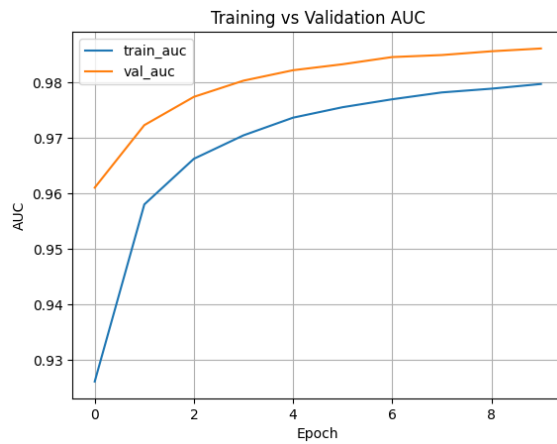


Figure 26: Graphic of the GRU model over 10 epochs, showing training and validation AUC curves.

The final evaluation on the held-out test set confirmed the effectiveness of the GRU architecture. The test achieved an accuracy of 92.72%, precision of 95.02%, recall of

90.17%, F1-score of 92.53%, and an ROC-AUC of 98.03%. The training, validation and test results of standalone GRU model is illustrated in Table 18.

Table 18: The training, validation and test results of standalone GRU model.

	Accuracy	Precision	Recall	F1-score	ROC-AUC
Training	0.9297	0.9550	0.9101	0.9312	0.9801
Validation	0.9285	0.9536	0.9048	0.9284	0.9799
Test	0.9272	0.9502	0.9017	0.9253	0.9803

The confusion matrix showed that the majority of misclassifications were FNs with 6,634 cases, where DGA domains were incorrectly labeled as benign. FPs were less frequent with 3,192 cases as shown in Figure 27.

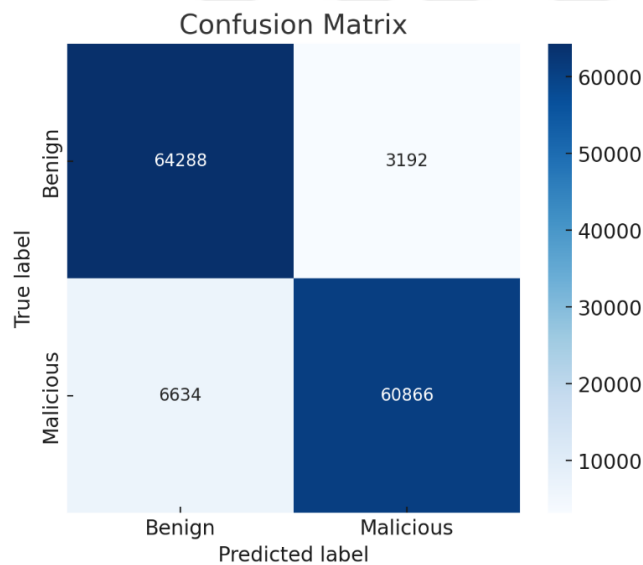


Figure 27: Confusion matrix illustrating the GRU model's classification performance on test data.

The results demonstrated that the GRU model was highly effective in capturing sequential dependencies within domain names for DGA detection. The steady improvement of both accuracy and ROC-AUC across epochs indicated that the model learned progressively without overfitting.

The relatively higher precision compared to recall suggested that the GRU was more effective in avoiding FPs than in minimizing FNs. In operational contexts, this implies that the model is less likely to flag legitimate domains incorrectly. This is desirable for reducing unnecessary alerts in security systems. However, the trade-off is a slightly lower sensitivity to malicious domains, as reflected by the FNs observed in the

confusion matrix. This limitation could potentially allow some DGA domains to bypass detection, which requires further optimization.

Overall, the GRU model achieved competitive performance compared to similar approaches reported in the literature.

5.1.4. Transformer Networks

The Transformer model was trained for 10 epochs with early stopping and dropout regularization to mitigate overfitting. During training, both training and validation performance improved steadily across epochs. The training ROC-AUC increased from 94.88% in the first epoch to 98.83% in the final epoch. Similarly, training loss consistently decreased from 0.2836 to 0.1367.

The final evaluation on the held-out test set confirmed the effectiveness of the Transformer network architecture. The test achieved an accuracy of 94.87%, precision of 93.89%, recall of 96.00%, F1-score of 94.93%, and an ROC-AUC of 98.98%. The training, validation and test results of Transformer model is illustrated in Table 19.

Table 19: The training, validation and test results of Transformer model.

	Accuracy	Precision	Recall	F1-score	ROC-AUC
Training	0.9651	0.9547	0.9742	0.9650	0.9883
Validation	0.9580	0.9482	0.9671	0.9572	0.9885
Test	0.9487	0.9389	0.9600	0.9493	0.9898

An AUC score of 0.9898 indicates that the Transformer model achieved a strong performance in distinguishing between DGA-generated and benign domains as seen in Figure 28. It reflects the model's effectiveness in capturing the underlying patterns of malicious domain names. This proves transformers as a reliable classifier for DGA detection task.

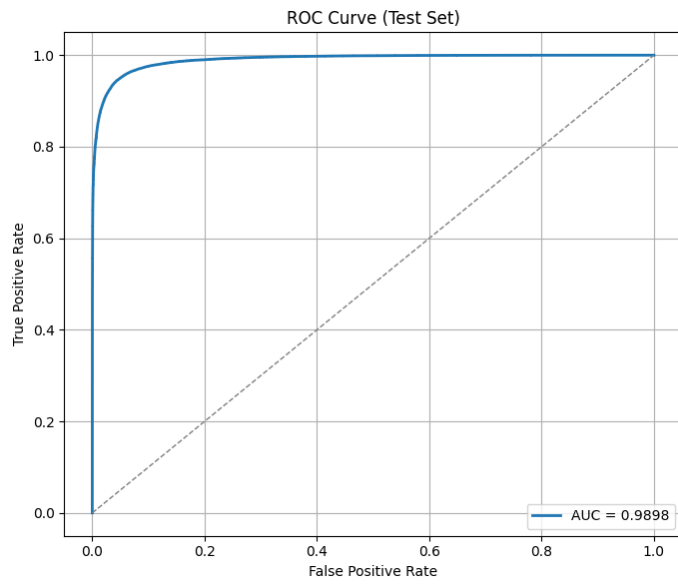


Figure 28: ROC curve of the Transformer model applied to DGA detection. The curve demonstrates high classification performance with an AUC score of 0.9898, indicating the model's strong ability to distinguish between legitimate and algorithmically generated domain names.

The transformer model confusion matrix, which is shown in Figure 29, illustrates the performance of the model in distinguishing between benign and DGA-generated domains.

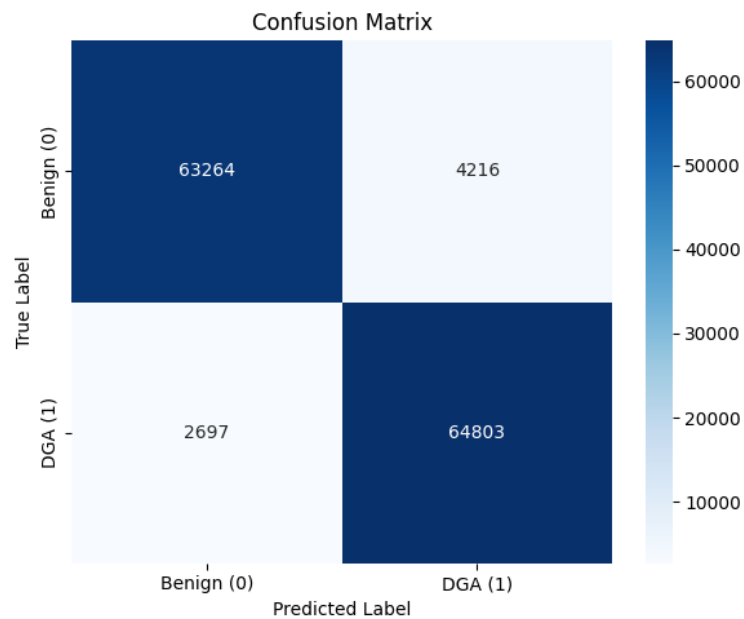


Figure 29: Confusion matrix illustrating the Transformer model's classification performance on the independent 20% test set.

Out of the total benign samples, 63,264 were correctly classified, while 4,216 were misclassified as DGAs. Similarly, for the malicious DGA domains, 64,803 were accurately identified, with 2,697 instances incorrectly labeled as benign. This distribution indicates that the model demonstrates strong classification capability, with high numbers of TPs and TNs compared to the relatively lower FP and FN counts. In practical terms, this means the model is effective in both detecting malicious domains and avoiding unnecessary misclassification of benign ones. This is crucial for reducing false alarms while maintaining robust detection accuracy.

Overall, Transformer model provided a powerful and flexible approach for DGA detection.

5.2. Hybrid Model

The hybrid model demonstrated exceptional performance in detecting DGA-generated domains. Across the training process, the ROC-AUC metric showed consistent improvement and reached a peak value of 0.9947 by final epoch. This progression indicates strong discriminative ability between legitimate and malicious domains. During the training phase, the model achieved an accuracy of 97.04%, a precision of 96.98%, and an F1-score of 97.04%.

The training confusion matrix which is shown in Figure 30 demonstrates that the model correctly classified 262,002 benign domains (TNs) and 261,934 malicious domains (TPs), while misclassifying 8,154 benign domains as malicious (FPs) and 7,821 malicious domains as benign (FNs).

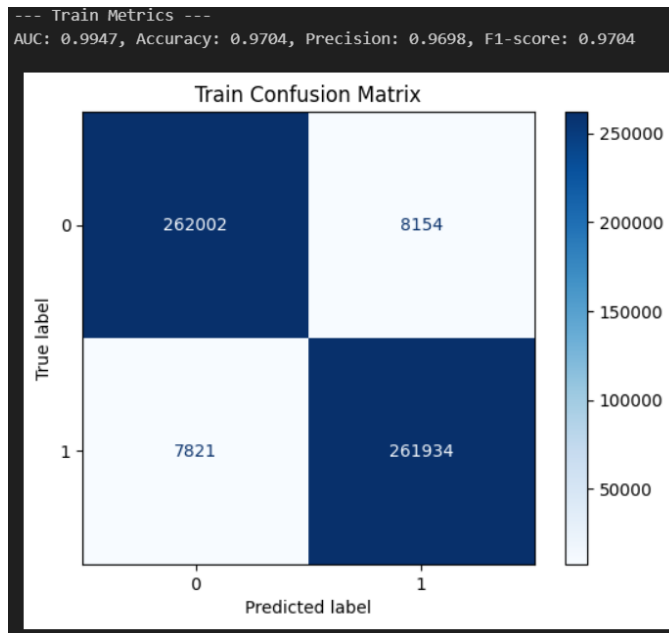


Figure 30: Confusion matrix illustrating the hybrid model's classification performance on training data.

In the training phase, the proportion of FPs was 1.53%, while the proportion of FNs was 1.47%. This indicates that the model incorrectly flagged a small part of benign domains as malicious, which may result in unnecessary blocking of legitimate services. On the other hand, a nearly equal proportion of malicious domains were misclassified as benign. This may potentially allow a limited number of DGA-generated domains to bypass detection.

The final evaluation on the held-out test set confirmed the effectiveness of the proposed hybrid model architecture. On the test dataset, the model achieved an accuracy of 96.21% and a precision of 96.2%, resulting in a balanced F1-score of 96.23% and an ROC-AUC of 0.9917.

The test confusion matrix which demonstrates that the model correctly classified 64,655 benign domains (TNs) and 65,208 malicious domains (TPs), while misclassifying 2578 benign domains as malicious (FPs) and 2537 malicious domains as benign (FNs) as seen in Figure 31.

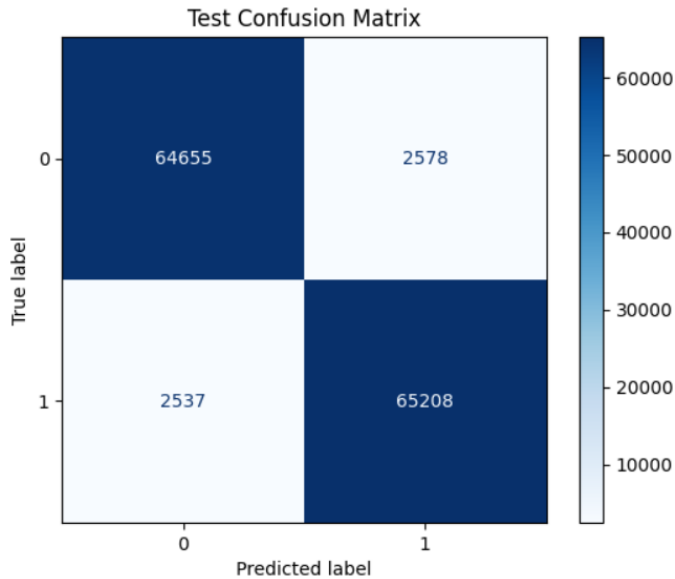


Figure 31: Confusion matrix illustrating the hybrid model's classification performance on the independent 20% test set.

The ROC curve of hybrid model which is shown in Figure 32, illustrates the trade-off between the TPR and the FPR for the test dataset. The curve is positioned well above the diagonal baseline. This indicates that the model achieves a high level of discriminative capability between benign and malicious domains. The ROC-AUC is measured at 0.9917, which is very close to 1.0. This value reflects excellent classification performance. This score confirms that the model maintains a high probability of correctly distinguishing malicious domains from benign ones.

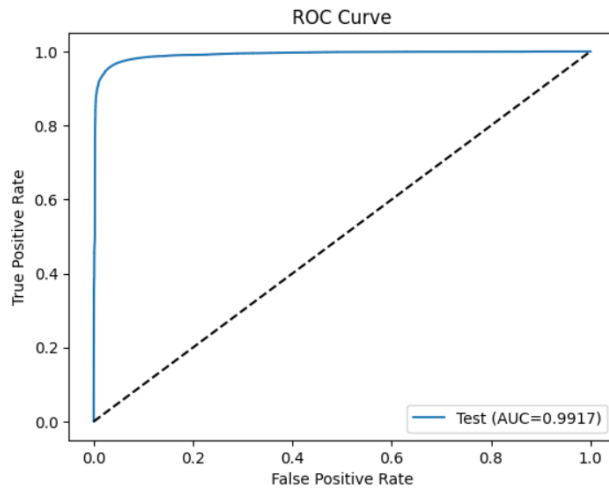


Figure 32: ROC curve of the hybrid model applied to DGA detection. The curve demonstrates high classification performance with an AUC score of 0.9917, indicating the model's strong ability to distinguish between legitimate and algorithmically generated domain names.

Figure 33 presents a comparison of model performance metrics on the training and test datasets. The model demonstrates strong generalization capability, as the results across both datasets are highly consistent. The minimal differences between the train and test metrics suggest that the model is neither overfitting nor underfitting and maintains robust predictive performance when applied to unseen data. Overall, the almost equal scores across all evaluation metrics confirm the reliability of the proposed model in DGA detection task.



Figure 33: Bar chart comparing the hybrid model's performance metrics between training and test datasets. The results demonstrate strong generalization capability, with minimal performance drop across evaluation phases.

5.3. Hybrid Model Validation

To ensure robustness, 5-fold cross-validation was applied. The validation results yield a mean accuracy of 96.88% and a mean AUC of 0.9941 across all folds as seen in Figure 34.

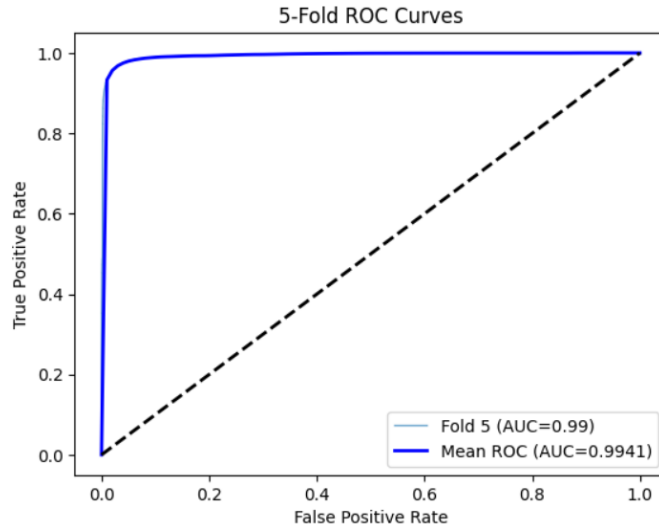


Figure 34: ROC curves generated from 5-fold cross-validation, illustrating the model's classification performance across multiple validation splits. The mean AUC of 0.9941 confirm the model's robustness and generalization capability in detecting DGA-generated domains.

Table 20 presents the performance metrics obtained through 5-fold cross-validation of the proposed hybrid model. The stability of the metrics further confirmed the generalization capability of our hybrid model.

Table 20: Performance metrics obtained through 5-fold cross-validation of the proposed hybrid model.

	AUC	Accuracy	Precision	F1_Score
Fold 1	0.9940	0.9685	0.9677	0.9685
Fold 2	0.9941	0.9688	0.9684	0.9688
Fold 3	0.9942	0.9688	0.9679	0.9689
Fold 4	0.9939	0.9687	0.9685	0.9687
Fold 5	0.9943	0.9691	0.9687	0.9691
Mean	0.9941	0.9688	0.9682	0.9688

5.4. Comparison of Models

To comprehensively assess the effectiveness of our proposed hybrid model, its performance was compared against standalone CNN, LSTM, GRU and Transformer models. Each model was trained and evaluated on the same dataset under identical experimental settings to ensure fairness in comparison. The evaluation focuses on key metrics, including accuracy, precision, recall, F1-score and ROC-AUC analysis. Table 21 summarizes the key evaluation metrics for each of the experimented models.

Table 21: Comparison of 5 models' experiment results. The highest scores are shown in green colors and the lowest scores are shown in red color, illustrating that the hybrid model achieved the top scores in ROC-AUC and recall, and performed second in other metrics just with minimal differences.

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
CNN	0.9403	0.9290	0.9535	0.9411	0.9857
LSTM	0.9688	0.9718	0.9656	0.9687	0.9688
GRU	0.9272	0.9502	0.9017	0.9253	0.9803
Transformer	0.9487	0.9389	0.9600	0.9493	0.9898
Hybrid Model	0.9621	0.9620	0.9673	0.9623	0.9917

The CNN model, which relies primarily on convolutional layers for feature extraction, achieved accuracy of 0.9403, precision of 0.9290, recall of 0.9535, F1-score of 0.9411 and ROC-AUC of 0.9857. The model's recall value is relatively strong and shows that CNN was effective in detecting malicious domains. The main limitation of CNN lies in its inability to fully capture long-term dependencies within domain names. Although n-gram patterns extracted by convolutional filters provide useful local features, they fail to model long-term dependencies effectively, which are critical for distinguishing AGDs from legitimate ones.

The LSTM model significantly improved performance and achieved the highest accuracy, precision and F1-score among all models. It also recorded a recall of 0.9656 and ROC-AUC of 0.9688. These results highlight LSTM's strength in learning long-term dependencies. This is essential for detecting subtle variations in DGA-generated domain patterns. The high recall indicates that LSTM was particularly effective in minimizing FNs, ensuring that most malicious domains were correctly identified.

The GRU model also delivered competitive results, with an accuracy of 0.9272 and ROC-AUC of 0.9803. Its precision of 0.9502 was strong, and the model showed balanced performance across other metrics. GRU's simpler gating mechanism compared to LSTM contributed to faster training with lower computational cost. This

capability made GRU as a good alternative in resource-constrained environments. However, its recall value of 0.9017 was notably the lowest, which indicates a higher tendency to miss malicious domains. GRU also performed the lowest results in terms of accuracy and F1-score among other models. This limitation suggests that while GRU is efficient, it stands out as a lightweight architecture and may not be as robust as other models in capturing the intricate sequential patterns present in domain name structures.

The transformer model achieved the second highest performance in terms of ROC-AUC metric among other models with 0.9898. Considering that ROC-AUC is the most important metric with regard to DGA detection, Transformer model proved to be highly effective in separating DGA generated domains from benign domains. In addition, transformer model scored above 0.96 in recall metric, illustrating that it catches a great majority of the malicious domain samples.

Finally, the proposed hybrid model delivered strong results, with an accuracy of 0.9621, precision of 0.9620, recall of 0.9673, and F1-score of 0.9623. Its ROC-AUC of 0.9917 and recall of 0.9673 were the highest among all models, demonstrating superior discriminative capability. In addition, our hybrid model ranked second in accuracy, precision and F1-score, with only minimal differences compared to the top-performing models in terms of these evaluation metrics. By integrating CNN, LSTM, LR and the TLD side information, our hybrid model combined the strengths of each architecture.

It is important to emphasize that ROC-AUC is the most critical metric in the context of DGA detection. Since domain classification tasks often involve highly imbalanced datasets, accuracy alone may be misleading. In this context, our hybrid model not only achieved the highest ROC-AUC score of 0.9917, but it also recorded the highest recall of 0.9673. This is particularly important since recall reflects the ability to detect as many malicious domains as possible. Although our hybrid model ranked second in accuracy, precision and F1-score, these differences compared to other top-performing models were minimal. Overall, the hybrid model outperformed the other models with regard to quantitative performance metrics.

5.5. Comparison with Previous Studies on DGA Detection

Previous DGA detection studies leveraged several ML and DL methods. Table 22 illustrates the comparison of DL-based DGA detection models from the existing literature alongside the results of our experiments.

Table 22: Performance comparison of DL-based DGA detection models across different studies, with author's experiment outcomes illustrated in red color.

Model	Study	Year	Dataset Size	DGA	Benign	Accuracy	Precision	Recall	F1-Score	ROC-AUC
CNN	Chethana et al.	2022	Not reported			0.9430	0.9510	0.9360	0.9430	-
	Lu et al.	2025	Not reported			0.9500	0.9350	0.9130	0.9240	-
	Düztaş	2025	674.898	337.500	337.398	0.9403	0.9290	0.9535	0.9411	0.9857
LSTM	Woodbridge et al.	2016	1.750.000	750.000	1.000.000	-	0.9942	0.9937	0.9906	0.9993
	Li et al.	2019	1.942.000	942.000	1.000.000	0.9822	0.9826	0.9819	0.9822	0.9970
	Chethana et al.	2022	Not reported			0.9860	0.9900	0.9820	0.9860	-
	Lu et al.	2025	Not reported			0.9490	0.9280	0.9170	0.9220	-
	Düztaş	2025	674.898	337.500	337.398	0.9688	0.9718	0.9656	0.9687	0.9688
GRU	Chethana et al.	2022	Not reported			0.9920	0.9970	0.9870	0.9920	-
	Lu et al.	2025	Not reported			0.9510	0.9330	0.9020	0.9170	-
	Düztaş	2025	674.898	337.500	337.398	0.9272	0.9502	0.9017	0.9253	0.9803
Transformer Model	Gogoi et al.	2023	60.000	30.000	30.000	0.9900	0.9900	0.9900	0.9900	0.9800
	Lu et al.	2025	Not reported			0.9580	0.9410	0.9130	0.9260	-
	Düztaş	2025	674.898	337.500	337.398	0.9487	0.9389	0.9600	0.9493	0.9898
Hybrid Models	Li et al. (Ensemble model)	2019	1.942.000	942.000	1.000.000	0.9982	0.9982	0.9982	0.9982	0.9999
	Chethana et al. (CNN-LSTM)	2022	Not reported			0.9920	0.9970	0.9870	0.9920	-
	Rao et al. (BERT+RNN)	2024	160.000	Not reported		0.9800	-	0.9750	-	-
	Rao et al. (BERT+LSTM)		160.000	Not reported		0.9900	-	0.9900	-	-
	Pan et al. (AN+LSTM)	2025	500.000	250.000	250.000	-	0.9550	0.9428	0.9495	-
	Pan et al. (CNN+BiGRU)					-	0.9438	0.9325	0.9385	-
	Pan et al. (LSTM+Attention)					-	0.9502	0.9491	0.9504	-
	Pan et al. (CNN-BiLSTM-M)					-	0.9772	0.9868	0.9820	-
	Düztaş (Proposed Hybrid Model)	2025	674.898	337.500	337.398	0.9621	0.9620	0.9673	0.9623	0.9917

Note: “-” indicates that the corresponding metric was not reported in the original study.

As shown in Table 22, previous works of other authors, studying CNN, LSTM, GRU, Transformer and Hybrid models, have conducted their experiments with varying size of datasets. Furthermore, some studies have not reported their dataset size or composition. In addition to datasets, the model evaluation metrics of previous studies varied with some models focusing only on accuracy and recall. Considering that the ROC-AUC metric is critical in terms of distinguishing DGA generated domains from benign ones, the models which did not calculate this metric do not yield healthy results for direct comparison.

When compared with our study, some previous studies reported higher performance metrics by even achieving 0.9999 ROC-AUC and 0.9982 accuracy, precision, recall and F1-scores. These scores mean almost perfect discrimination capability between DGA generated domains and benign domains. However, the difference between our implemented models and the previous studies may stem from variations in datasets or experimental configurations.

Since the majority of the DGA detection studies did not share their model codes or datasets publicly to validate the results, this makes direct comparison between models extremely challenging. On the other hand, in our thesis we shared the python codes we used for model implementations along with the dataset used, in order to enable future studies to replicate and validate our findings.

Nevertheless, when evaluated in the context of the existing literature, our findings appear consistent, with no significant deviations.

5.6. Comparison of 5 models' Qualitative Aspects

In addition to quantitative performance metrics, it is essential to compare the models in terms of architectural characteristics, computational efficiency and practical applicability. Accuracy, precision, recall, F1-score and ROC-AUC provide insight into predictive performance. However, factors such as training time, resource requirements and generalization capability also play a crucial role in determining the feasibility of deploying these models in real-world environments. Table 23 summarizes these qualitative aspects for experimented models, by highlighting their strengths, limitations and best-use scenarios.

Table 23: Comparison of 5 models' qualitative aspects.

	CNN	LSTM	GRU	Transformer	Hybrid
Training Time	Faster than other models.	Longer training time due to complex gating mechanisms.	Slower than CNN, faster than other models.	Slower than CNN and GRU.	Longer due to architecture complexity.
Computational Cost	Low, fewer resources required.	Very high, requires more memory and processing power.	Moderate, less resource-intensive than LSTM and Hybrid model.	High, requires memory and processing power.	Very high, demanding both memory and computational power.
Primary Strength	Captures localized patterns through n-grams.	Long-range dependencies and contextual relationships can be modelled.	Efficient sequence modeling with simpler architecture than LSTM and Hybrid model.	Excellent for long-range global dependencies.	Combines local pattern extraction with long-range dependency modeling and TLD information for comprehensive feature learning.
Generalization	Limited, struggles with complex or irregular domain patterns.	Strong, effectively handles sophisticated or irregular structures in DGA domains.	Good, but weaker than LSTM and Hybrid model in handling complex patterns.	Very high, especially for new DGA families.	Excellent, robust across both simple and complex DGAs due to hybrid feature extraction.
Best For	Simpler DGA families with repetitive or localized patterns.	Complex DGAs that require understanding of sequence-wide dependencies.	Medium-complexity DGAs requiring balance of speed and accuracy.	Complex DGAs, large datasets.	All types of DGAs, particularly those with mixed complexity patterns.

With regards to qualitative aspects, the CNN model is advantageous for scenarios requiring faster training and low computational power. This capability makes it suitable for resource-constrained environments. However, its generalization capability is limited. Because it struggles to capture long-range dependencies. This feature makes CNNs effective for simpler DGA families with repetitive or localized patterns.

The LSTM model, in contrast, excels in modeling long-range dependencies within domain sequences. This aspect results in strong generalization for complex DGA structures. This strength comes at the cost of longer training times and higher computational requirements.

The GRU model offers a middle ground. It achieves efficient sequence modeling with faster training time compared to LSTM. Its performance, however, slightly falls behind LSTM in handling highly sophisticated DGAs. This makes it better suited for medium-complexity DGA families where both accuracy and speed are important considerations.

The advantages of Transformer for DGA detection include strong generalization capability to previously unseen DGA families. In addition, Transformer models perform best on large-scale datasets. They consistently outperform traditional ML models and often surpass some DL methods such as CNN and GRU, especially when the dataset includes diverse and highly irregular DGAs. The architecture's parallel processing ability also allows for faster training on GPUs compared to recurrent networks. However, Transformer models require more memory and computational resources during training, and may underperform on extremely small datasets.

Finally, the hybrid model delivers the most robust performance by combining CNN's ability to extract localized n-gram patterns with LSTM's capability to model global sequential dependencies. In simple words, CNN is like a microscope which looks closely at tiny parts of the domain. On the other hand, LSTM is like a storyteller which looks at how the entire sequence flows. Together, they help the model understand both the small details and the big picture, which makes hybrid model powerful for detecting DGAs. In addition, the feature set is further enriched by TLD side information and set to binary classification with LR. This architecture offers excellent generalization across both simple and complex DGA families. Our proposed hybrid model proves to be ideal for high-risk security environments. The trade-off is increased training time and computational cost.

5.7. Advantages of DL Use for DGA Detection

DL models outperform traditional ML models in DGA detection. Because they can automatically learn complex patterns from raw domain name data without relying on manually engineered features. While ML models require explicit feature extraction, such as domain length, character distribution or n-gram statistics, DL architectures use multiple hidden layers to capture complex patterns. These hidden layers act as a black box. It means that the feature extraction process is not interpretable.

Moreover, DL models handle large-scale datasets more effectively and can generalize better to previously unseen DGAs. Their ability to learn directly from sequences allows them to detect subtle structural patterns in malicious domains. The combination of automatic feature learning and black-box behavior makes DL highly effective for adaptive DGA detection in dynamic cyber threat environment.

5.8. Summary

Our proposed hybrid model outperformed all standalone models by achieving the best detection performance during our experiments. In addition, it also maintained a balance between performance and resource use. Its ability to leverage both localized pattern recognition and long-term dependency modeling makes it a robust solution for diverse DGA detection scenarios. This demonstrates the advantage of combining complementary DL techniques rather than relying on a single architecture. And it also sets the foundation for exploring even more advanced hybrid or attention-based models in future research.



CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. Conclusion

DGA detection remains a critical and evolving area within cybersecurity domain, since adversaries continue to rely on DGAs to maintain covert communication channels with C2 servers. Even though substantial progress has been made through the use of ML and DL techniques, the dynamic nature of DGAs poses ongoing challenges.

One of the concerns in DGA detection is the adaptive behavior of attackers. CTAs seek to exploit the known behaviors of detection algorithms. This creates a persistent arms race between defenders and CTAs. So, detection models must not only be accurate but also must be adaptable to new adversarial strategies. Traditional ML models are dependent on manual feature engineering and this reliance introduces a level of transparency that can be exploited by CTAs. By analyzing model architectures, CTAs may reverse-engineer the detection algorithms and eventually they can develop more resilient DGAs.

In contrast, DL approaches mitigate these risks with their ability to automatically learn DGA patterns directly from raw data without relying on manual features. As complex “black-box” models, DL models obscure their internal decision-making process. This makes DL models more difficult for CTAs to reverse-engineer.

IoT environments present a unique and growing attack surface for malware employing DGAs. IoT devices are typically resource-constrained in terms of processing power, memory and storage, and therefore often lack traditional security mechanisms such as antivirus software and endpoint detection systems. As a result, compromised IoT devices can be infected with malware that leverages DGAs for resilient C2 communication, making this an area of significant concern. However, not all DGA families are equally suitable for deployment in IoTs. Advanced DGA variants, such as adversarial or dictionary-based DGAs, generally require higher computational and memory resources and may not execute efficiently on constrained IoT hardware. In contrast, lightweight arithmetic-based DGAs can operate effectively under such limitations. Despite these constraints, the use of DGAs in IoT environments remains a serious security issue and should be explicitly considered in future DGA detection and mitigation strategies.

Moreover, the real performance of DGA detection approaches can only be validated through testing against the most recent DGA families. Outdated DGA datasets may lead to overfitting and inflated performance metrics. These test environments do not reflect real-world threat landscape. Therefore, retaining up-to-date threat intelligence and updating datasets is essential for maintaining the reliability of detection models.

Furthermore, the identification of domains generated by DGAs carries significant implications for cybersecurity operations. When such a domain is detected, it may serve as a critical indicator of compromise (IoC). This suggests that a system or network entity could be part of malicious activities such as botnet participation, DDoS attacks or malware infections. In this context, the detection of a DGA reveals potential adversarial activity and requires comprehensive system health check and forensic scans to evaluate the security posture.

DGA detection systems should not function in isolation but they should be integrated into broader cyber defense ecosystems to maximize their effectiveness. Standalone DGA detection offers valuable insights, but true operational value emerges when coupled with security information and event management (SIEM) platforms, threat intelligence feeds and automated incident response mechanisms like security orchestration, automation and response (SOAR) tools. Through such integration, detected DGA domains can be correlated with other network and endpoint events. This method enriches situational awareness and enables faster, more context-aware decision-making.

When a DGA-generated domain is identified, the detection result must be forwarded to the SIEM for correlation. This event can then trigger an automated response workflow. Afterwards, the domain is blocked at the firewall, the affected endpoint is quarantined, and an alert is escalated to the security operations center (SOC) for deeper investigation. Such workflows ensure that DGA detection contributes not only to identifying malicious activity but also to the timely containment and remediation of broader cyber threats. DGA response workflow is illustrated in Figure 35.

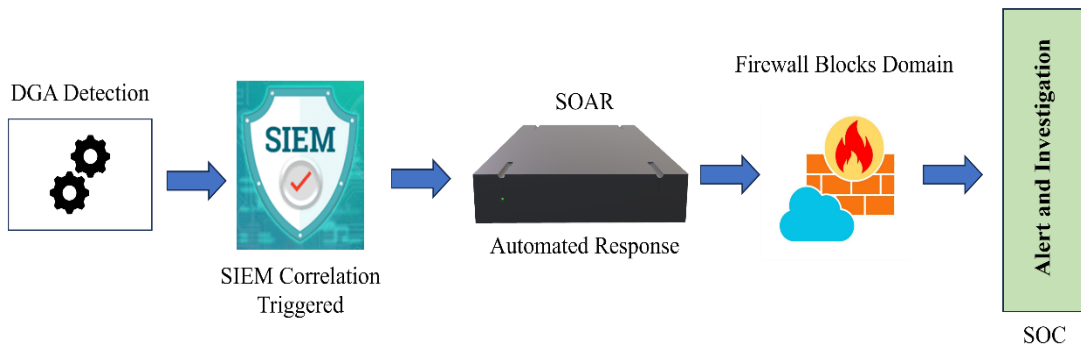


Figure 35: Workflow illustrating the automated response to DGA-based threats. The process begins with DGA detection, followed by SIEM correlation, SOAR-triggered actions, firewall intervention, and concludes with alerting and investigation in the SOC.

Building on these foundations, we focused on DL models in our thesis. We proposed a hybrid framework designed to leverage the complementary strengths of different architectures. By achieving superior detection performance compared to other DL models, we fulfilled initial aim of this thesis.

6.2. Future Work

Current DGA detection methods have demonstrated significant effectiveness. However, there are several points that require future exploration in order to enhance detection capabilities.

One notable limitation is that, existing approaches focus primarily on domains composed of Latin alphabet characters. This approach leaves domains in non-Latin scripts, such as Cyrillic, Arabic, Chinese or other regional scripts, unexplored. Extending detection methods to include these alphabets is essential for improving global threat coverage.

One other promising direction is the application of Graph Neural Networks (GNNs) to the DGA detection problem. By modeling relationships between domains, IP addresses and hosts as graph structures, GNNs can leverage network relational data to enhance detection accuracy. This approach could help identify DGA domains not only based on fundamental verbal characteristics but also through their interactions within DNS traffic or network topology.

Another area worth exploring is the integration of adversarial training to improve the model's robustness. Since attackers may attempt to generate adversarial domain names that evade existing models, training with adversarial examples can help to create more resilient DGA detection systems.

Finally, the combined use of multi-modal data, such as combining DNS metadata, WHOIS records, HTTP headers and threat intelligence feeds, may also yield richer feature sets and improve classification performance in future studies.



REFERENCES

- Ahluwalia, A. (2018). Impact Study of Length in Detecting Algorithmically Generated Domains. University of Victoria .
- Ali Soleymani, F. A. (2021). A Novel Approach for Detecting DGA-Based Botnets in DNS Queries Using Machine Learning Techniques. *Journal of Computer Networks and Communications*, 1-13.
- Biros, H. &. (2025). Enhancing DGA Detection with Machine Learning Algorithms. *Journal of Telecommunications and Information Technology*.
- Bogdan Cebere, J. F. (2024). Downtoearth! Guidelines for DGA-based Malware Detection. *RAID '24: Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses* (pp. 147-165). RAID.
- Chen Yijing and Pang Bo and Shao Guolin, W. G. (2021, August). DGA-based botnet detection toward imbalanced multiclass learning. *Tsinghua Science and Technology*, 26(4), 387-402.
- Chethana, C. P. (2022). Improved Domain Generation Algorithm to Detect Cyber-Attack with Deep Learning Techniques. *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)* (pp. 1-8). IEEE.
- Curtin, R. R. (2019). Detecting DGA Domains with Recurrent Neural Networks and Side Information. *Proceedings of the 14th International Conference on Availability, Reliability and Security* (pp. 1-10). ACM.
- Daniel Plohmann, K. Y. (2016). A Comprehensive Measurement Study of Domain Generating Malware. In *25th USENIX Security Symposium (USENIX Security 16)*., (pp. 263-278).
- De Luca, L. (2022). DGA Detection with Big Data Approaches. Politecnico di Torino.
- Giorgio Piras, M. P. (2022). Explaining Machine Learning DGA Detectors from DNS Traffic Data. *ITASEC'22: Italian Conference on Cybersecurity*. Rome, Italy: CEUR Workshop Proceedings.
- Gogoi, B. (2023). DGA domain detection using pretrained character based transformer models. *2023 IEEE Guwahati Subsection Conference (GCON)*, (pp. 1-6).
- Haleh Shahzad, A. R. (2021). DGA Domain Detection using Deep Learning. *5th International Conference on Cryptography, Security and Privacy* (pp. 139-143). IEEE.

- Ji Huang, P. W. (2018). Detecting Domain Generation Algorithms With Convolutional Neural Language Models. *12th IEEE International Conference On Big Data Science And Engineering*, (pp. 1360-1367).
- Jianbing Liang, S. C. (2022, September). Heterogeneous DGA Domain Name Detection Model. *Computers & Security*, 120, 1-16.
- Jonghoon Lee, J. K. (2019). Cyber Threat Detection Based on Artificial Neural Networks Using Event Profiles. *IEEE Access*, 7, 165607–165626. doi:10.1109/ACCESS.2019.2953095
- Kusumi, R. T. (2025). Malicious Domain Detection Using Statistical Features of Domain Strings, Public Information and DNS Logs. *2025 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC)* (pp. 49-54). IEEE.
- Li, S. H. (2019). Domain Generation Algorithms detection through deep neural network and ensemble. *In Companion Proceedings of the 2019 World Wide Web Conference*, (pp. 189-196). doi:https://doi.org/10.1145/3308558.3316498
- Li, Y. X. (2019). A Machine Learning Framework for Domain Generation Algorithm-based Malware Detection. *IEEE Access*, 32765–32782.
- Liew, S. R. (2023). Use of Subword Tokenization for Domain Generation Algorithm Classification. *Cybersecurity*, 6(1), 49.
- Lu, T. (2025). Transformer-Based Detection of DGA Domains Enhanced by Semantic and Structural Branches. *4th International Conference on Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIOTC)*, (pp. 214-218).
- M. Grill, I. N. (2015). Detecting DGA Malware Using NetFlow. (pp. 1304–1309). IEEE/International Symposium on Integrated Network Management (IM).
- Patsakis, C. &. (2021). Exploiting Statistical and Structural Features for the Detection of Domain Generation Algorithms. *Journal of Information Security and Applications*, 28.
- Raaghavi Sivaguru, C. C. (2018). An Evaluation of DGA Classifiers. *2018 IEEE International Conference on Big Data*, (pp. 5058-5067).
- Raaghavi Sivaguru, J. P. (2020, August). Inline Detection of DGA Domains Using Side Information. *IEEE*, 8, 141910-141922.
- Rao, S. U. (2024). A Novel Approach to DGA Detection Combining Machine Learning and Transformer-based Techniques. *International Conference on IoT Based Control Networks and Intelligent Systems*, (pp. 1416-1420).

- Ravi Vinayakumar and Alazab Mamoun and Srinivasan Sriram and Arunachalam Ajay, S. K. (2023, January). Adversarial defense: DGA-based botnets and DNS homographs detection through integrated deep learning. *IEEE transactions on engineering management*, 70(1), 249-266.
- Reo Kusumi, M. T. (2025). Malicious Domain Detection Using Statistical Features of Domain Strings, Public Information and DNS Logs. *ICAIIIC* (pp. 49-54). IEEE.
- Rui Pan, Y. W. (2025, January). A DGA Domain Name Detection Model Based on a Hybrid Deep Neural Network with Multi-dimensional Features. *IAENG International Journal of Computer Science*, 52(1), 11-22.
- S. Yadav, A. K. (2012, October). Detecting Algorithmically Generated Domain-Flux Attacks with DNS Traffic Analysis. *IEEE/ACM Transactions on Networking*, 20(5), 1663–1677.
- Saeid Salehi, H. S. (2018). A Novel Approach for Detecting DGA-based Ransomwares. *15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*. Tehran, Iran: IEEE. doi:10.1109/ISCISC.2018.8546941
- Sherali, S. A. (2016). A taxonomy of domain-generation algorithms. *IEEE Security & Privacy*, 14(4), 46-53.
- Tong Anh Tuan, H. V. (2022, February). On Detecting and Classifying DGA Botnets and their Families. *Computers & Security*, 113, 1-17.
- Wang, Z. J. (2018). A Detection Scheme for DGA Domain Names based on SVM. *2018 International Conference on Mathematics, Modelling, Simulation and Algorithms (MMSA 2018)* (pp. 257–263). Atlantis Press.
- Wenxuan He, G. G. (2019). Malicious Domain Detection via Domain Relationship and Graph Models. *IEEE*.
- Woodbridge, J. A. (2016, November). Predicting Domain Generation Algorithms with Long Short-Term Memory Networks. *arXiv.org (preprint)*.
- Xiaoyan Hu, M. L. (2022). Towards Accurate DGA Detection based on Siamese Network with Insufficient Training Samples. *International Conference on Communications* (pp. 2670-2675). IEEE.
- Xin Fang, X. S. (2020). Domain-Embeddings Based DGA Detection with Incremental Training Method. *Symposium on Computers and Communications (ISCC)*. Rennes, France: IEEE. doi:10.1109/ISCC50000.2020.9219664

- Y. Zhou, Q. S. (2013). DGA-Based Botnet Detection Using DNS Traffic. *J. Internet Serv. Inf. Secur.*, 3(3/4), 116-123.
- Yang Luhui and Liu Guangjie and Dai Yuewei and Wang Jinwei, Z. J. (2020). Detecting stealthy domain generation algorithms using heterogeneous deep neural network framework. *IEEE Access*, 8, 82876–82889.
- Yu Fu, L. Y. (2017, June). Stealthy Domain Generation Algorithms. *IEEE Transactions on Information Forensics and Security*, 12(6), 1430–1442.
- Yu, B. G. (2017). Inline DGA detection with deep networks. *IEEE international conference on data mining workshops (ICDMW)* (pp. 683-692). IEEE.
- Yu, B. P. (2019). Weakly Supervised Deep Learning for the Detection of Domain Generation Algorithms. *IEEE Access*, 51542–51556.
- Zhao Hong and Chang Zhaobin and Bao Guangbin, Z. X. (2019). Malicious domain names detection algorithm based on N-gram. *Journal of Computer Networks and Communications*, 1-8.

APPENDICES

APPENDIX A – STANDALONE CNN MODEL TRAINING PYTHON CODE

```
# import packages
import pandas as pd
import numpy as np
import os
r = 'C:\CNN-DGA'
os.chdir(r)

dga = pd.read_csv('dga-dataset.txt', sep = ',', header = None)
dga.columns = ['label','origin','domain']

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc,
precision_recall_curve, average_precision_score
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense,
Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import pickle
from tensorflow.keras.regularizers import l2

# Parameters
MAXLEN = 75    # max domain length
MAX_VOCAB = 40 # max number of distinct chars
EMBED_DIM = 32
BATCH_SIZE = 128
EPOCHS = 10

# Preprocessing

X = dga['domain'].astype(str).str.lower().values
```

```

# map string labels to integers
label_map = {label: idx for idx, label in enumerate(dga['label'].unique())}
print("Label mapping:", label_map)

y = dga['label'].map(label_map).values

# char-level tokenizer
char_texts = [" ".join(list(x)) for x in X]
tok = Tokenizer(num_words=MAX_VOCAB, char_level=False, filters="", lower=False,
oov_token="<OOV>")
tok.fit_on_texts(char_texts)

seqs = tok.texts_to_sequences(char_texts)
X_pad = pad_sequences(seqs, maxlen=MAXLEN, padding='post', truncating='post')

# split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y, test_size=0.2, stratify=y,
random_state=42)

# CNN Model
vocab_size = min(MAX_VOCAB, len(tok.word_index) + 1)

model = Sequential([
    Embedding(input_dim=vocab_size+1,                                output_dim=EMBED_DIM,
input_length=MAXLEN),

    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same',
        kernel_regularizer=l2(0.001)), # L2 regularization
    BatchNormalization(),

    Conv1D(filters=64, kernel_size=5, activation='relu', padding='same',
        kernel_regularizer=l2(0.001)),
    GlobalMaxPooling1D(),

    Dropout(0.5),           # stronger dropout
    Dense(32, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.3),

    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(1e-3), loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

```

```

# Training
es = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True,
verbose=1)

history = model.fit(
    X_train, y_train,
    validation_split=0.15,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[es],
    verbose=1
)

# Evaluation

from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)

# Predictions
y_pred_proba = model.predict(X_test).ravel()
y_pred = (y_pred_proba >= 0.5).astype(int)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=4))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
print(f"Confusion Matrix: TN={tn}, FP={fp}, FN={fn}, TP={tp}")

# Metrics
test_accuracy = accuracy_score(y_test, y_pred)
test_precision = precision_score(y_test, y_pred)
test_recall = recall_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred)
test_auc = roc_auc_score(y_test, y_pred_proba)

```

```

print("\n--- Test Metrics ---")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"AUC: {test_auc:.4f}")

# Visualization

import matplotlib.pyplot as plt
plt.style.use("seaborn-v0_8-darkgrid") # modern equivalent

# Training curves
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend(); plt.title("Loss Curve")

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.legend(); plt.title("Accuracy Curve")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.4f})", linewidth=2)
plt.plot([0,1], [0,1], 'k--', alpha=0.7)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.show()

# Precision-Recall
prec, rec, _ = precision_recall_curve(y_test, y_pred_proba)
ap = average_precision_score(y_test, y_pred_proba)

plt.figure()
plt.plot(rec, prec, label=f"AP={ap:.4f}")
plt.xlabel("Recall"); plt.ylabel("Precision")

```

```
plt.title("Precision-Recall Curve"); plt.legend(); plt.show()

# Confusion matrix heatmap
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Benign","DGA"],
yticklabels=["Benign","DGA"])
plt.title("Confusion Matrix"); plt.ylabel("Actual"); plt.xlabel("Predicted")
plt.show()

# Save model + tokenizer

model.save("cnn_dga_model")
with open("cnn_tokenizer.pkl", "wb") as f:
    pickle.dump(tok, f)

print("Model and tokenizer saved.")
```



APPENDIX B – STANDALONE LSTM MODEL TRAINING PYTHON CODE

```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, classification_report
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional,
SpatialDropout1D
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Load dataset
r = r"C:\LSTM-DGA"
os.chdir(r)
dga = pd.read_csv("dga-dataset.txt", sep=",", header=None)
dga.columns = ["label", "origin", "domain"]
# Encode labels (assuming benign=0, dga=1)
y = dga["label"].map({"benign": 0, "dga": 1}).values
# Normalize labels
dga['label'] = dga['label'].astype(str).str.lower()

# Map to binary
y = dga['label'].map({
    'benign': 0,
    'legit': 0,
    'normal': 0,
    'dga': 1,
    'malicious': 1
})

# Drop NaNs
mask = ~y.isna()
X = X[mask]
y = y[mask].astype(int).values

# Tokenize characters
MAXLEN = 75
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(dga["domain"])
```

```

X = tokenizer.texts_to_sequences(dga["domain"])
X = pad_sequences(X, maxlen=MAXLEN, padding="post")

VOCAB_SIZE = len(tokenizer.word_index) + 1 # FIXED

# Train-test split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=42)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.15, stratify=y_train_val, random_state=42

model = Sequential([
    Embedding(input_dim=VOCAB_SIZE, output_dim=64, input_length=MAXLEN),
    SpatialDropout1D(0.2),
    Bidirectional(LSTM(128, dropout=0.3, recurrent_dropout=0.3)),
    Dense(64, activation="relu"),
    Dropout(0.5),
    Dense(1, activation="sigmoid")
])

model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
callbacks = [
    EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=True),
    ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=2)
]

# Train
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=10,
    batch_size=128,
    callbacks=callbacks,
    verbose=1
)

# Evaluation
y_pred = (model.predict(X_val) > 0.5).astype("int32")
print("Accuracy:", accuracy_score(y_val, y_pred))
print("Precision:", precision_score(y_val, y_pred))
print("Recall:", recall_score(y_val, y_pred))
print("F1:", f1_score(y_val, y_pred))
print("ROC-AUC:", roc_auc_score(y_val, y_pred))
print(classification_report(y_val, y_pred))

```

APPENDIX C – STANDALONE GRU MODEL TRAINING PYTHON CODE

```
# Imports
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# r = r'C:\DGA-GRU'
# os.chdir(r)

from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, confusion_matrix, classification_report, roc_curve
)

import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Embedding, GRU, Bidirectional, Dense, Dropout,
    SpatialDropout1D, GlobalMaxPooling1D
)
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam

SEED = 42
np.random.seed(SEED)
tf.random.set_seed(SEED)

# Load dataset
# dga = pd.read_csv('dga-dataset.txt', sep=',', header=None)
# dga.columns = ['label','origin','domain']
# dga = dga.apply(lambda x: x.astype(str).str.lower())
# dga['domain_list'] = dga['domain'].apply(lambda x: x.split('.'))

dga = pd.read_csv('dga-dataset.txt', sep=',', header=None,
names=['label','origin','domain'])
dga = dga.astype(str)
for col in dga.columns:
```

```

dga[col] = dga[col].str.strip().str.lower()

dga = dga[dga['domain'].str.len() > 0].dropna(subset=['domain'])

def get_sld(domain: str):
    parts = domain.split('.')
    return parts[0] if len(parts) > 0 else domain

dga['domain_sld'] = dga['domain'].apply(get_sld)

# Encode labels
def to_binary_label(x: str):
    x = x.strip().lower()
    # Numeric 0/1 as strings
    if x in {'0', '1'}:
        return int(x)
    # Common text labels
    if x in {'benign', 'good', 'legit', 'normal', 'clean', 'ben'}:
        return 0
    if x in {'malicious', 'dga', 'bad', 'malware', 'mal', 'botnet'}:
        return 1

    raise ValueError(f"Unrecognized label: {x}")

y = dga['label'].apply(to_binary_label).astype(int).values

# Character-level tokenization
# Build a vocabulary of characters present in domains
all_text = dga['domain_sld']
chars = sorted(set("".join(all_text)))
# Reserve 0 for padding; start indexing at 1
char2idx = {c: i+1 for i, c in enumerate(chars)}
vocab_size = len(char2idx) + 1 # +1 for padding index 0

def text_to_seq(s: str):
    return [char2idx.get(ch, 0) for ch in s] # unknowns -> 0 (same as pad)

sequences = all_text.apply(text_to_seq).tolist()

# Set max length conservatively (95th percentile) to avoid overfitting on long tails
lengths = np.array([len(seq) for seq in sequences])
max_len = int(np.percentile(lengths, 95))
max_len = max(10, max_len) # minimum sensible length

X = pad_sequences(sequences, maxlen=max_len, padding='post', truncating='post')

```

```

# Train/Val/Test split
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.2, random_state=SEED, stratify=y
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.15, random_state=SEED, stratify=y_train_full
)
print(f"Train: {X_train.shape}, Val: {X_val.shape}, Test: {X_test.shape}")
print(f"Vocab size: {vocab_size}, Max len: {max_len}")

# Build a regularized BiGRU
# (anti-overfitting: dropout, l2, modest capacity, early stopping)
embedding_dim = 64

model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=max_len),
    SpatialDropout1D(0.2),
    Bidirectional(GRU(64, return_sequences=True, dropout=0.3,
recurrent_dropout=0.2)),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu', kernel_regularizer=l2(1e-4)),
    Dropout(0.4),
    Dense(1, activation='sigmoid')
])

optimizer = Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(
    optimizer=optimizer,
    loss='binary_crossentropy',
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.AUC(name='auc')
    ]
)

model.summary()
# Callbacks to prevent overfitting
callbacks = [
    EarlyStopping(monitor='val_auc', patience=4, mode='max',
restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbose=1),
    ModelCheckpoint('best_gru.keras', monitor='val_auc', mode='max',
save_best_only=True)

```

```

]

# Train
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=10,          # generally won't need all due to EarlyStopping
    batch_size=128,
    verbose=1,
    callbacks=callbacks
)

# Visualizations (history)
# Accuracy
plt.figure()
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.title('Training vs Validation Accuracy');
plt.legend(); plt.grid(True)
plt.show()

# Loss
plt.figure()
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Training vs Validation Loss'); plt.legend();
plt.grid(True)
plt.show()

# AUC
plt.figure()
plt.plot(history.history['auc'], label='train_auc')
plt.plot(history.history['val_auc'], label='val_auc')
plt.xlabel('Epoch'); plt.ylabel('AUC'); plt.title('Training vs Validation AUC');
plt.legend(); plt.grid(True)
plt.show()

# Show final validation results
final_val_acc = history.history['val_accuracy'][-1]
final_val_loss = history.history['val_loss'][-1]
final_val_auc = history.history['val_auc'][-1]

print("Final Validation Results:")
print(f"Validation Accuracy : {final_val_acc:.4f}")
print(f"Validation Loss      : {final_val_loss:.4f}")

```

```

print(f"Validation AUC    : {final_val_auc:.4f}")

# Plot training vs validation metrics
import matplotlib.pyplot as plt

# Accuracy
plt.figure()
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Training vs Validation Accuracy")
plt.xlabel("Epochs"); plt.ylabel("Accuracy")
plt.legend(); plt.grid(True)
plt.show()

# Loss
plt.figure()
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Training vs Validation Loss")
plt.xlabel("Epochs"); plt.ylabel("Loss")
plt.legend(); plt.grid(True)
plt.show()

# AUC
plt.figure()
plt.plot(history.history['auc'], label='Train AUC')
plt.plot(history.history['val_auc'], label='Validation AUC')
plt.title("Training vs Validation AUC")
plt.xlabel("Epochs"); plt.ylabel("AUC")
plt.legend(); plt.grid(True)
plt.show()

# Evaluation on test set
y_prob = model.predict(X_test).flatten()
y_pred = (y_prob >= 0.5).astype(int)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, zero_division=0)
rec = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
auc = roc_auc_score(y_test, y_prob)

print(f"\nTest Accuracy : {acc:.4f}")
print(f"Test Precision: {prec:.4f}")
print(f"Test Recall   : {rec:.4f}")

```

```

print(f"Test F1-score : {f1:.4f}")
print(f"Test ROC-AUC : {auc:.4f}\n")

print("Classification report:\n", classification_report(y_test, y_pred, digits=4))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", cm)

plt.figure()
plt.imshow(cm, interpolation='nearest')
plt.title('Confusion Matrix')
plt.xlabel('Predicted'); plt.ylabel('True')
for (i, j), v in np.ndenumerate(cm):
    plt.text(j, i, str(v), ha='center', va='center')
plt.colorbar()
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure()
plt.plot(fpr, tpr)
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate'); plt.title('ROC Curve')
plt.grid(True)
plt.show()

```

APPENDIX D – TRANSFORMER MODEL TRAINING PYTHON CODE

```
import os
import numpy as np
import pandas as pd
import torch
import torch.nn as nn

from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, confusion_matrix
)
r = r"C:\Transformers-DGA"
os.chdir(r)
DATA_PATH = "dga-dataset.txt"

MAX_LEN = 75 # max domain length
BATCH_SIZE = 128
EPOCHS = 10
LR = 1e-3

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Basic settings
SEED = 42
np.random.seed(SEED)
torch.manual_seed(SEED)

# Load dataset
print("Loading dataset...")
dga = pd.read_csv(DATA_PATH, sep=",", header=None)
dga.columns = ["label", "origin", "domain"]

dga["domain"] = dga["domain"].astype(str).str.lower()
dga["label"] = dga["label"].astype(int)

print(dga.head())

# Build character vocabulary
```

```

all_domains = dga["domain"].tolist()
all_chars = set("".join(all_domains))

char2idx = {c: i + 1 for i, c in enumerate(sorted(all_chars))}
PAD_IDX = 0
vocab_size = len(char2idx) + 1

print("Vocab size:", vocab_size)
print("Example chars:", list(char2idx.keys())[:30])

def encode_domain(domain, max_len=MAX_LEN):
    seq = [char2idx.get(c, 0) for c in domain[:max_len]]
    if len(seq) < max_len:
        seq += [PAD_IDX] * (max_len - len(seq))
    return seq

# Encode all domains
X = np.array([encode_domain(d) for d in dga["domain"]])
y = dga["label"].values.astype(np.float32)

print("X shape:", X.shape)
print("y shape:", y.shape)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=SEED, stratify=y
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.15, random_state=SEED, stratify=y_train_full
)
print("Train size:", X_train.shape[0])
print("Test size :", X_test.shape[0])

# Dataset
class DomainDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.long)
        self.y = torch.tensor(y, dtype=torch.float32)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_dataset = DomainDataset(X_train, y_train)

```

```

test_dataset = DomainDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

# Transformer model
class DGATransformer(nn.Module):
    def __init__(
        self,
        vocab_size,
        emb_dim=64,
        n_heads=4,
        n_layers=2,
        ff_dim=128,
        max_len=MAX_LEN,
        dropout=0.1,
        pad_idx=PAD_IDX,
    ):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim, padding_idx=pad_idx)
        self.pos_embedding = nn.Embedding(max_len, emb_dim)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model=emb_dim,
            nhead=n_heads,
            dim_feedforward=ff_dim,
            dropout=dropout,
            batch_first=True # important: (batch, seq, feature)
        )
        self.transformer_encoder = nn.TransformerEncoder(encoder_layer,
            num_layers=n_layers)

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(emb_dim, 1)

    def forward(self, x):
        # x: (batch, seq_len)
        batch_size, seq_len = x.size()
        positions = torch.arange(0, seq_len,
            device=x.device).unsqueeze(0).expand(batch_size, seq_len)
        x = self.embedding(x) + self.pos_embedding(positions)
        x = self.transformer_encoder(x)
        # Simple mean pooling over sequence
        x = x.mean(dim=1)
        x = self.dropout(x)
        logits = self.fc(x).squeeze(1)

```

```

        probs = torch.sigmoid(logits)
        return probs

model = DGATransformer(vocab_size=vocab_size).to(device)
print(model)

criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LR)

# Training
print("Starting training...")
for epoch in range(EPOCHS):
    model.train()
    epoch_loss = 0.0
    all_train_labels = []
    all_train_scores = []

    for X_batch, y_batch in train_loader:
        X_batch = X_batch.to(device)
        y_batch = y_batch.to(device)

        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

    epoch_loss += loss.item() * X_batch.size(0)
    all_train_labels.extend(y_batch.detach().cpu().numpy())
    all_train_scores.extend(outputs.detach().cpu().numpy())

avg_loss = epoch_loss / len(train_dataset)
try:
    train_auc = roc_auc_score(all_train_labels, all_train_scores)
except ValueError:
    train_auc = float("nan") # in case only one class shows up in a batch

    print(f"Epoch {epoch+1}/{EPOCHS} - Loss: {avg_loss:.4f} - Train AUC: {train_auc:.4f}")

# Evaluation on test set
print("\nEvaluating on test set...")
model.eval()
all_labels = []
all_probs = []

```

```

with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch = X_batch.to(device)
        y_batch = y_batch.to(device)

        outputs = model(X_batch)
        all_labels.extend(y_batch.cpu().numpy())
        all_probs.extend(outputs.cpu().numpy())

all_labels = np.array(all_labels)
all_probs = np.array(all_probs)
all_preds = (all_probs >= 0.5).astype(int)

acc = accuracy_score(all_labels, all_preds)
prec = precision_score(all_labels, all_preds)
rec = recall_score(all_labels, all_preds)
f1 = f1_score(all_labels, all_preds)
auc = roc_auc_score(all_labels, all_probs)

cm = confusion_matrix(all_labels, all_preds)

print("TEST RESULTS ===")
print("Accuracy :", acc)
print("Precision:", prec)
print("Recall  :", rec)
print("F1-score :", f1)
print("ROC-AUC  :", auc)
print("Confusion Matrix:")
print(cm)

```



APPENDIX E – HYBRID MODEL TRAINING PYTHON CODE

```
import os
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import LabelBinarizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    roc_auc_score, accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report, roc_curve, ConfusionMatrixDisplay
)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, LSTM, Dropout, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Loading dataset

def load_tld_list(tld_path: str) -> set:
    tld = pd.read_csv(tld_path, comment="#",
header=None)[0].dropna().astype(str).str.lower().tolist()
    return set(tld)

def load_dataset(data_path: str) -> pd.DataFrame:
    df = pd.read_csv(data_path, sep=",", header=None, names=["label", "origin",
"domain"])
    df = df.apply(lambda x: x.astype(str).str.lower())
    return df

def normalize_labels(df: pd.DataFrame) -> pd.DataFrame:
    def fix_label(x: str) -> str:
        if x in ("dg", "da", "dgb", "dgs", "dgf", "dha"):
            return "dga"
        if x in ("lgit", "lenit", "lefit", "legip"):
            return "legit"
        return x

    df = df.copy()
    df["label"] = df["label"].apply(fix_label)
```

```

df = df[df["label"] != "nan"].copy()
return df

# Domain parsing into primary and tld columns

def parse_domain_parts(domain: str, common_tlds: set) -> tuple:

    parts = domain.split(".")
    parts = [p for p in parts if p != ""] # safety

    primary1 = primary2 = tld1 = tld2 = ""

    if len(parts) == 2:
        primary1, tld1 = parts[0], parts[1]

    elif len(parts) == 3:
        a, b, c = parts
        # pri + tld1 + tld2 (both tlds in common list)
        if (b in common_tlds) and (c in common_tlds):
            primary1, tld1, tld2 = a, b, c
        # sub + pri + tld
        else:
            primary1, tld1 = b, c

    elif len(parts) == 4:
        a, b, c, d = parts
        # sub + pri + tld1 + tld2 (both common)
        if (c in common_tlds) and (d in common_tlds):
            primary1, tld1, tld2 = b, c, d
        # sub + pri2 + pri1 + tld
        else:
            primary2, primary1, tld1 = b, c, d

    else:
        # if domain has unexpected length, fallback:
        # take last part as tld1, and one before it as primary1 if available
        if len(parts) >= 2:
            tld1 = parts[-1]
            primary1 = parts[-2]
        else:
            primary1 = domain

    return primary1, primary2, tld1, tld2

def build_parsed_dataframe(df: pd.DataFrame, tld_path: str) -> pd.DataFrame:

```

```

common_tlds = load_tld_list(tld_path)

out = df.copy()
parts = out["domain"].apply(lambda d: parse_domain_parts(d, common_tlds))
out[["primary1", "primary2", "tld1", "tld2"]] = pd.DataFrame(parts.tolist(),
index=out.index)

out = normalize_labels(out)
return out

# CNN(n-gram)+LSTM training

def build_char_vocab(strings: np.ndarray) -> dict:
    vocab = sorted(set("".join(strings)))
    return {ch: i + 1 for i, ch in enumerate(vocab)} # 0 reserved for padding/unknown

def encode_domains(domains: np.ndarray, char2idx: dict, maxlen: int) -> np.ndarray:
    X = [[char2idx.get(ch, 0) for ch in d] for d in domains]
    return pad_sequences(X, maxlen=maxlen)

def build_cnn_lstm_ngram(vocab_size: int, maxlen: int, n_gram: int = 5) -> Sequential:
    model = Sequential([
        Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen),
        Conv1D(filters=128, kernel_size=n_gram, activation="relu"),
        LSTM(128),
        Dropout(0.5),
        Dense(1, activation="sigmoid")
    ])
    model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=["accuracy"])
    return model

def train_cnn_lstm_with_holdout(
    df: pd.DataFrame,
    max_epoch: int = 10,
    batch_size: int = 128,
    n_gram: int = 5,
    patience: int = 2,
    seed: int = 0
)

    df = df.copy()
    labels = df["label"].values
    X_raw = df["primary1"].values.astype(str)

```

```

char2idx = build_char_vocab(X_raw)
vocab_size = len(char2idx) + 1
maxlen = max(len(x) for x in X_raw)

X = encode_domains(X_raw, char2idx, maxlen)
y = np.array([0 if x == "legit" else 1 for x in labels], dtype=np.int32)

X_train, X_test, y_train, y_test, labels_train, labels_test = train_test_split(
    X, y, labels, test_size=0.2, random_state=seed, stratify=y
)

X_train, X_hold, y_train, y_hold = train_test_split(
    X_train, y_train, test_size=0.15, random_state=seed, stratify=y_train
)

model = build_cnn_lstm_ngram(vocab_size, maxlen, n_gram=n_gram)

best_auc = -1.0
best_epoch = -1
best_out = None

for ep in range(max_epoch):
    print(f"\n=== Epoch {ep+1}/{max_epoch} (n-gram={n_gram}) ===")
    model.fit(X_train, y_train, epochs=1, batch_size=batch_size, verbose=1)

    hold_probs = model.predict(X_hold, verbose=0).ravel()
    hold_auc = roc_auc_score(y_hold, hold_probs)
    print(f"[Holdout] AUC = {hold_auc:.6f} (best={best_auc:.6f})")

    if hold_auc > best_auc:
        best_auc = hold_auc
        best_epoch = ep

    test_probs = model.predict(X_test, verbose=0).ravel()
    cm = confusion_matrix(y_test, test_probs > 0.5)

    best_out = {
        "epoch": ep,
        "holdout_auc": hold_auc,
        "y_test": y_test,
        "labels_test": labels_test,
        "test_probs": test_probs,
        "confusion_matrix": cm
    }

```

```

        print("[Test] Confusion matrix:")
        print(cm)

    elif (ep - best_epoch) > patience:
        print("Early stopping triggered.")
        break

    return char2idx, model, best_out

# TLD one-hot encoding

def onehot_top_tlds(tld_series: pd.Series, top_k: int = 250):
    top_tld = tld_series.value_counts().head(top_k).index.astype(str).tolist()
    lb = LabelBinarizer()
    lb.fit(top_tld)

    X_tld = lb.transform(tld_series.astype(str))
    return top_tld, lb, X_tld

# LR training

def evaluate_lr(model, X_train, y_train, X_test, y_test, title_prefix=""):
    for name, X, y in [("Train", X_train, y_train), ("Test", X_test, y_test)]:
        probs = model.predict_proba(X)[:, 1]
        pred = (probs > 0.5).astype(int)

        auc = roc_auc_score(y, probs)
        acc = accuracy_score(y, pred)
        prec = precision_score(y, pred)
        rec = recall_score(y, pred)
        f1 = f1_score(y, pred)

        print(f"\n{title_prefix}{name} metrics:")
        print(f"AUC={auc:.4f} ACC={acc:.4f} PREC={prec:.4f} REC={rec:.4f} F1={f1:.4f}")

        cm = confusion_matrix(y, pred)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp.plot()
        plt.title(f"{title_prefix}{name} Confusion Matrix")
        plt.show()

        fpr, tpr, _ = roc_curve(y, probs)
        plt.plot(fpr, tpr, label=f"{name} (AUC={auc:.4f})")
        plt.plot([0, 1], [0, 1], "k--")

```

```

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title(f"{title_prefix}ROC Curve")
plt.legend()
plt.show()

def kfold_lr_auc(X, y, k=5, seed=0):
    skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=seed)

    aucs, accs, precs, recs, f1s = [], [], [], [], []
    mean_fpr = np.linspace(0, 1, 200)
    tprs = []

    plt.figure(figsize=(8, 6))

    for fold, (tr, te) in enumerate(skf.split(X, y), start=1):
        model = LogisticRegression(max_iter=500, n_jobs=-1, random_state=seed)
        model.fit(X[tr], y[tr])

        probs = model.predict_proba(X[te])[:, 1]
        pred = (probs > 0.5).astype(int)

        auc = roc_auc_score(y[te], probs)
        acc = accuracy_score(y[te], pred)
        prec = precision_score(y[te], pred)
        rec = recall_score(y[te], pred)
        f1 = f1_score(y[te], pred)

        aucs.append(auc); accs.append(acc); precs.append(prec); recs.append(rec);
f1s.append(f1)

        fpr, tpr, _ = roc_curve(y[te], probs)
        tprs.append(np.interp(mean_fpr, fpr, tpr))
        tprs[-1][0] = 0.0

        plt.plot(fpr, tpr, lw=1, alpha=0.7, label=f"Fold {fold} (AUC={auc:.3f})")

    mean_tpr = np.mean(tprs, axis=0)
    mean_tpr[-1] = 1.0

    plt.plot(mean_fpr, mean_tpr, lw=2, label=f"Mean ROC
(AUC={np.mean(aucs):.4f})")
    plt.plot([0, 1], [0, 1], "k--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")

```

```

plt.title(f"{k}-Fold ROC Curves (LogReg)")
plt.legend(loc="lower right")
plt.show()

print("\nK-Fold mean metrics:")
print(f"AUC={np.mean(aucs):.4f} ACC={np.mean(accs):.4f}
PREC={np.mean(precs):.4f} REC={np.mean(recs):.4f} F1={np.mean(f1s):.4f}")

return {
    "AUC": aucs, "Accuracy": accs, "Precision": precs, "Recall": recs, "F1": f1s
}

# Main pipeline
def main():

    WORKDIR = r"C:\DGA-Hybrid"
    os.chdir(WORKDIR)

    DATA_PATH = "dga-dataset.txt"
    TLD_PATH = "TLDlist.txt"

    print("[INFO] Loading and parsing dataset...")
    df_raw = load_dataset(DATA_PATH)
    df_parsed = build_parsed_dataframe(df_raw, TLD_PATH)

    df_for_dl = df_parsed[["label", "primary1"]].copy()

    print("[INFO] Training CNN(n-gram)+LSTM...")
    char2idx, cnn_lstm_model, best_out = train_cnn_lstm_with_holdout(
        df_for_dl,
        max_epoch=10,
        batch_size=128,
        n_gram=5,
        patience=2,
        seed=0
    )

    print("\n[INFO] Best CNN-LSTM epoch:", best_out["epoch"])
    print("[INFO] Best holdout AUC:", best_out["holdout_auc"])
    print("[INFO] Best test confusion matrix:\n", best_out["confusion_matrix"])

    # Save model
    cnn_lstm_model.save("cnn_lstm_hybrid_ngram_model")
    with open("valid_chars.pkl", "wb") as f:

```

```

pickle.dump(char2idx, f)

# One-hot encode top TLDs
print("[INFO] One-hot encoding top TLDs...")
top_tld, lb, X_tld = onehot_top_tlds(df_parsed["tld1"], top_k=250)
with open("labelbinarizer.pkl", "wb") as f:
    pickle.dump(lb, f)
with open("top_tld.pkl", "wb") as f:
    pickle.dump(top_tld, f)

print("[INFO] Generating hybrid probabilities for all samples...")
maxlen = cnn_lstm_model.input_shape[1]
X_all = encode_domains(df_parsed["primary1"].astype(str).values, char2idx, maxlen)
hybrid_probs = cnn_lstm_model.predict(X_all, verbose=0).ravel().reshape(-1, 1)

X_features = np.hstack([hybrid_probs, X_tld])

# Binary labels
y = (df_parsed["label"].values != "legit").astype(int)

# Train/test split for LR
X_train, X_test, y_train, y_test = train_test_split(
    X_features, y, test_size=0.2, random_state=0, stratify=y
)

print("\n[INFO] Training Logistic Regression ensemble...")
lr = LogisticRegression(max_iter=500, n_jobs=-1, random_state=0)
lr.fit(X_train, y_train)

# Metrics visuals
evaluate_lr(lr, X_train, y_train, X_test, y_test, title_prefix="Ensemble LR - ")

print("\n[INFO] Classification report (Test):")
probs_test = lr.predict_proba(X_test)[: , 1]
pred_test = (probs_test > 0.5).astype(int)
print(classification_report(y_test, pred_test, target_names=["Legit", "DGA"]))

# K-fold evaluation
kfold_lr_auc(X_features, y, k=5, seed=0)

```