DGA Detection System

1st Hugo Correia
NºMec: 108215
DETI, University of Aveiro
hf.correia@ua.pt

2nd Joaquim Rosa
NºMec: 109089
DETI, University of Aveiro
joaquimvr15@ua.pt

***Abstract*—Domain Generation Algorithms (DGAs) are widely used by malware to generate large volumes of pseudo-random domains for command-and-control communication, enabling evasion of static blocklists and complicating takedown efforts. This work presents an end-to-end DGA detection system designed for real-time, client-facing settings where only the queried domain string is reliably available at decision time. The platform implements a two-stage pipeline: (i) binary classification of domains as benign or DGA-generated and (ii) malware family attribution across 25 DGA families for domains flagged as malicious. We evaluate feature-based ensemble baselines (Random Forest, Gradient Boosting, and XGBoost) using engineered lexical and character-composition features, alongside sequence-based neural architectures (CNN-LSTM, Transformer) and a transfer-learning approach (DistilBERT) operating directly on character-level representations. Experimental results on a large, balanced dataset show that representation-learning models substantially outperform classical baselines: DistilBERT achieves the best binary detection performance (accuracy and F1-score of 0.961), while CNN-LSTM attains the strongest family classification results (accuracy 0.938, macro F1-score 0.936). Finally, the trained models are deployed behind a FastAPI service and integrated into a web dashboard and a browser extension that provides real-time warnings, manual scanning, and operational observability via Prometheus/Grafana monitoring.**

***Index Terms*—Domain Generation Algorithms, DGA detection, malicious domain classification, malware family classification, machine learning, deep learning, character-level models, CNN-LSTM, Transformer, DistilBERT, FastAPI, browser extension**

I. Introduction

Domain Generation Algorithms (DGAs) are a common technique used by malware to produce large volumes of pseudo-random domain names for command-and-control communication. By rapidly rotating candidate domains, DGA-enabled botnets can evade static blacklists and make infrastructure takedown more difficult, since only a small subset of generated domains needs to be registered and reachable at any time. As a result, accurate and timely identification of DGA-generated domains remains an important capability for network defenders and endpoint security tools [1].

The main challenge addressed in this work is the design of a practical DGA detection pipeline that can operate in real time when only the queried domain string is reliably available at decision time. While some detection strategies leverage passive DNS telemetry and auxiliary behavioral signals [2], such information is not always accessible in client-facing settings. This motivates detection methods based on lexical and character-level properties of domains, as well as sequence models that can learn discriminative representations directly from raw domain strings.

To address this problem, this project implements an end-to-end DGA detection system composed of client interfaces and a backend classification service. The platform supports both binary detection (benign versus DGA-generated) and, when a domain is flagged as malicious, a second-stage family classification over 25 DGA families. Multiple model families are evaluated under a unified experimental protocol, including feature-based ensemble baselines (Random Forest, Gradient Boosting, and XGBoost) and neural sequence learners (CNN-LSTM, Transformer, and DistilBERT). The system is exposed through a REST API and integrated into a browser extension for real-time warnings, as well as a web dashboard for manual scanning and performance inspection, with monitoring instrumentation to support deployment observability.

The remainder of this paper is organized as follows. Section II reviews related work on malicious domain and DGA detection. Section III describes the dataset construction, preprocessing, feature extraction, model design choices, and the overall system architecture. Section IV details the implementation of the machine learning models. Section V reports experimental results and an end-to-end demonstration of the platform. Section VI concludes the paper and outlines directions for future work.

II. Related Work

Early research on malicious domain identification frequently leveraged passive DNS telemetry and side-information beyond the domain string itself. Bilge et al. proposed EXPOSURE, a passive DNS analysis service that detects and reports malicious domains using a set of engineered features derived from observed DNS behavior (e.g., temporal and answer-based characteristics) and operational deployment feedback. [3] This line of work is effective when rich network context is available, but it assumes access to passive DNS streams and infrastructure capable of collecting and updating behavior-driven features at scale.

More recent DGA-focused systems reduce the dependency on extensive side channels by focusing on characteristics observable from DNS failures and the queried names. Schüppen et al. introduced FANCI, which targets DGA infections by monitoring NXDomain responses and classifying the corresponding non-existent domain queries with a feature-based machine learning pipeline. By extracting features from individual NXDomains, the approach supports online detection and demonstrates strong generalization across many DGA families, while still relying on DNS response visibility and a tailored feature engineering design. [4]

In parallel, deep learning approaches have shown that high-performing DGA detection can be achieved directly from the character sequence of a domain name, minimizing manual feature construction. Woodbridge et al. demonstrated an LSTM-based classifier that learns discriminative representations from raw domain strings for both binary detection and family attribution [5], highlighting the suitability of sequence models for DGA patterns.

Compared to these approaches, the system presented in this work is designed for real-time, client-facing use where only the domain string is guaranteed to be available at decision time. The proposed pipeline emphasizes lexical and character-level modeling that can be served through an API and integrated into a browser extension, avoiding dependencies on passive DNS feeds, TTL information, or NXDomain-only filtering.Rather than relying on a single classifier, this work benchmarks both feature-based classical models and end-to-end neural sequence models using the same dataset splits and evaluation metrics, allowing a direct and fair comparison for deployment.

III. Methodology

A. Data Acquisition

The data used in this work was obtained from multiple publicly available sources in order to construct a diverse corpus of benign and algorithmically generated domain names.

The core malicious and benign samples were obtained from the DGA Domains Dataset released on GitHub [6], which is based on domains extracted from the Netlab OpenData Project [7] and benign domains sourced from Alexa Top Sites [8]. This dataset contains a total of 675,000 domain names, with an equal class distribution consisting of 337,500 DGA-generated domains and 337,500 legitimate domains. The malicious samples are evenly distributed across 25 distinct DGA families, resulting in 13,500 domains per family. These families include both time-dependent and time-independent algorithms, ensuring diversity in generation patterns and temporal behavior. The balanced class distribution mitigates bias during training and allows for fair performance evaluation across classes. The dataset and its construction are consistent with the methodology described in the original study [9].

In addition to the primary dataset, an independent dataset published on Kaggle was consulted [10]. This dataset provides two CSV files containing 337,500 DGA domains and 337,337 legitimate domains, closely mirroring the size and balance of the primary corpus. The consistency across datasets reinforces the representativeness of the selected data and supports reproducibility of results across different sources.

Table I summarizes the datasets used in this study, including their origin, purpose, and class distribution.

The combination of a large-scale, balanced DGA corpus with auxiliary infrastructure-specific data ensures that the resulting dataset captures both algorithmic diversity and real-world deployment characteristics. This data acquisition strategy provides a robust foundation for training machine learning models capable of detecting DGA-based threats in practical environments.

TABLE I
Summary of datasets used in the DGA detection system

Dataset Source	Benign Samples	DGA Samples
DGA Domains Dataset [6]	337,500	337,500
Kaggle DGA Dataset [10]	337,337	337,500

To support domain preprocessing during inference, a curated list of dynamic DNS (DDNS) provider domains was incorporated. This auxiliary dataset was sourced from an automated and continuously updated repository that aggregates domains from known dynamic DNS providers [11]. At the time of data collection, this list contained 32,756 provider domains spanning multiple major DDNS services. These entries are not labeled as malicious or benign, as they represent legitimate DDNS service providers rather than training samples. Instead, this list serves a critical preprocessing role: when a submitted domain is identified as using a DDNS provider, for example ”xk7mq2p.ddns.net”, the system extracts the subdomain ”xk7mq2p” for analysis rather than the provider’s base domain ”ddns”. This ensures that potentially algorithmically generated subdomains, commonly used by malware leveraging DDNS infrastructure, are correctly identified and passed to the classification model.

B. Preprocessing

The preprocessing pipeline addresses the challenge of transforming raw domain name strings into a consistent format suitable for machine learning analysis. Upon loading the dataset, duplicate entries are removed to prevent data leakage and ensure that each domain contributes uniquely to the training process. Missing values are dropped to maintain data integrity, and all domain names are converted to lowercase to ensure case-insensitive feature extraction.

For domains submitted through URLs, a domain extraction utility strips protocol prefixes, path components, query parameters, and port numbers. The system employs the tldextract library for intelligent domain parsing, which correctly handles edge cases such as multi-level top-level domains and country code suffixes. A particularly important preprocessing step involves the handling of dynamic DNS providers. The system maintains a database of over 34,000 DDNS provider domains loaded from a curated CSV file. When a domain is identified as belonging to a DDNS provider, the system extracts the subdomain portion for analysis rather than the base domain, since the subdomain represents the algorithmically generated component in such cases.

The data is split into training and testing sets using stratified sampling with an 80/20 ratio to preserve the class distribution. Feature extraction is performed using a batch method for convenience when processing multiple domains.

C. Feature Extraction

The feature extraction process transforms domain name strings into numerical feature vectors suitable for machine learning classification. Two distinct approaches are employed depending on the model architecture: handcrafted feature engineering for ensemble methods and character-level encoding for deep learning models.

For the Random Forest, XGBoost, and Gradient Boosting classifiers, a set of 21 handcrafted features is extracted from each domain name. These features are organized into four categories: statistical features, character distribution features, linguistic features, and TF-IDF similarity scores.

The statistical features capture basic properties of the domain string. The length feature records the total number of characters, while unique characters counts the distinct characters present. The unique ratio normalizes this count by the domain length. Shannon entropy is calculated to quantify the randomness of the character distribution, with higher entropy values typically indicating algorithmically generated domains. A normalized entropy measure is also computed by dividing the raw entropy by the theoretical maximum for 26 characters.

Character distribution features examine the composition of the domain name. The vowel ratio and consonant ratio measure the proportion of vowels and consonants relative to the total character count, respectively. The digit ratio captures the prevalence of numerical characters. Two features measure the maximum number of consecutive consonants and consecutive digits, as DGA domains often exhibit unusual character sequences. The character variety feature counts the number of distinct character types present, including uppercase letters, lowercase letters, and digits. Two binary features capture the presence of special characters in the original domain: has numbers indicates whether the domain contains any numerical digits, while has hyphen indicates the presence of hyphen characters. These features are computed before the cleaning process to detect formatting patterns commonly associated with either legitimate or algorithmically generated domains.

Linguistic features assess how closely a domain resembles natural language patterns. The bigram score measures the proportion of character pairs that appear in a predefined set of 28 common English bigrams, including frequent pairs such as th, he, in, en, and re. Similarly, the trigram score evaluates the presence of common three-character sequences. The dictionary word count feature searches for English words of four or more characters within the domain using a dictionary containing 354,986 words. The dictionary coverage feature calculates the percentage of the domain that can be covered by recognized dictionary words.

The TF-IDF features provide a statistical measure of similarity to known legitimate domains and English words. Two TfidfVectorizer instances are fitted on the legitimate domain corpus and the English word dictionary, respectively, using character-level n-grams ranging from three to five characters. For each input domain, the TF-IDF scores are computed against both vectorizers, yielding raw and log-transformed similarity scores. These features capture subtle patterns in character sequences that distinguish legitimate domains from randomly generated ones.

For the deep learning models, domain names are encoded at the character level without explicit feature engineering. A vocabulary of 40 characters is defined, comprising lowercase letters, digits, and common special characters including hyphens, underscores, and periods. Each character is mapped to an integer index, with zero reserved for padding. Domains are padded or truncated to a fixed maximum length of 63 characters, consistent with DNS label length limits.

D. Model Selection

To evaluate the effectiveness of different modeling paradigms for detecting algorithmically generated domain names, this work explores a diverse set of machine learning and deep learning algorithms. The selected models span from traditional ensemble methods based on handcrafted features to modern neural architectures capable of learning representations directly from raw character sequences. This selection enables a systematic comparison between interpretable feature-based approaches and end-to-end representation learning techniques.

Classical ensemble methods were chosen as strong baseline models due to their proven effectiveness in structured tabular data and prior success in DGA detection literature. A Random Forest classifier was implemented to leverage its robustness to noise, ability to model non-linear decision boundaries, and inherent feature importance estimation. Its ensemble of independently trained decision trees provides a reliable baseline while maintaining interpretability through feature contribution analysis.

Gradient Boosting and Extreme Gradient Boosting (XGBoost) were selected to investigate whether sequential boosting strategies could improve performance by focusing on hard-to-classify samples. Gradient Boosting incrementally builds trees that correct the errors of previous iterations, allowing for more refined decision boundaries. XGBoost extends this idea by incorporating regularization, optimized tree construction, and efficient handling of feature interactions, making it particularly suitable for high-dimensional handcrafted feature spaces. These models were expected to outperform Random Forest in scenarios where subtle statistical and linguistic patterns differentiate legitimate and DGA-generated domains.

In addition to feature-based models, recurrent and attention-based neural architectures were explored to assess the benefits of learning directly from character-level representations. A hybrid CNN-LSTM network was implemented to capture both local character patterns and sequential dependencies in domain name strings, as DGA-generated domains often exhibit unnatural character transitions that span multiple positions. The architecture combines convolutional layers for local n-gram-like pattern extraction with a Bidirectional LSTM for long-range sequence modeling. This model eliminates the need for manual feature engineering and can generalize across unseen DGA families.

To further enhance sequence modeling capabilities, a Transformer-based architecture was evaluated. By relying on multi-head self-attention rather than recurrence, the Transformer model is able to capture long-range dependencies and global character interactions more efficiently. This architecture is particularly well-suited for domain names where informative patterns may appear at arbitrary positions within the string. The Transformer serves as a modern deep learning baseline and allows comparison against recurrent approaches under similar input representations.

Finally, a DistilBERT-based classifier was included to examine whether pre-trained language models can be effectively adapted to the domain name classification task. Although domain names are not natural language sentences, fine-tuning DistilBERT enables the model to leverage contextual embedding capabilities learned from large-scale corpora. This approach represents the most expressive model evaluated in this study and provides insight into the trade-offs between model complexity, training cost, and detection performance.

E. System Architecture

The system follows a multi-tier architecture consisting of five primary components: client, API, ML classification engine, data, and monitoring, as illustrated in Fig. 1. At the client layer, two interfaces enable user interaction: a Chrome browser extension that intercepts navigation events to perform real-time URL analysis and displays warning overlays when DGA domains are detected, and a React-based [12] dashboard that provides detection analytics, a domain scanner for manual queries, and model comparison tools for evaluating classifier performance. Both clients communicate with the backend through RESTful HTTP requests.

The API layer is implemented as a FastAPI [13] backend exposing endpoints for single-domain prediction, batch classification of multiple domains, and detailed analysis with feature-level explanations. Authentication endpoints secure access to protected resources, while dedicated routes expose model metadata including accuracy metrics and feature importance rankings. The API orchestrates the ML Classification Engine, which implements a two-stage cascading pipeline.

In the first stage, binary classification determines whether a domain is DGA-generated or legitimate using any of the six model architectures described previously. When operating in automatic mode, the system selects the best available model following a priority order based on expected accuracy: DistilBERT, Transformer, CNN-LSTM, XGBoost, Gradient Boosting, and Random Forest. Users may alternatively specify a particular model via API parameters.

For domains classified as DGA-generated, the second stage performs family classification to identify the specific malware family among 25 known families. Users may select a specific model for this stage or rely on automatic selection. When multiple family classifiers are available and automatic mode is enabled, the system employs an ensemble voting mechanism that aggregates predictions from all loaded classifiers. Each model's prediction is weighted by its classification accuracy, computed as $w_i = a_i / \sum_j a_j$, where a_i denotes the accuracy of model i. The weighted confidence for each candidate family is accumulated across models, and the family with the highest aggregated score is selected as the final prediction. Agreement analysis further adjusts confidence: unanimous model agreement increases confidence by a factor of 1.2, while disagreement among models reduces it by 0.85, providing calibrated uncertainty estimates. When a specific model is requested or only one classifier is available, the system returns that model's prediction directly without ensemble aggregation.

The data layer persists detection records and user information in a SQLite [14] database, while trained model weights are stored as serialized files—joblib format for tree-based models and Keras format with associated metadata for neural architectures. For observability, the API exposes Prometheus-compatible [15] metrics tracking prediction counts partitioned by model and result, family detection counts by threat level, and prediction latency histograms. These metrics are collected by a Prometheus server and visualized through pre-configured Grafana [16] dashboards, enabling real-time monitoring of system throughput, response times, and model utilization.

IV. Implementation

A. Random Forest

The Random Forest classifier is implemented using the `sklearn.ensemble.RandomForestClassifier` from the Scikit-learn library [17]. This ensemble method constructs multiple decision trees trained independently on bootstrap samples of the training data, with final predictions determined by majority voting across all trees.

The model is configured with 200 estimators, a maximum tree depth of 20 levels, and a minimum of 5 samples required for node splitting. Class weighting is set to `balanced` to automatically adjust weights inversely proportional to class frequencies, addressing potential class imbalance in the training data. Parallel processing is enabled via `njobs=-1` to utilize all available CPU cores during training.

Listing 1. Random Forest classifier configuration

```
RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    min_samples_split=5,
    min_samples_leaf=2,
    class_weight='balanced',
    n_jobs=-1,
    random_state=42
)
```

The input to the Random Forest model consists of the 21 handcrafted features extracted by the `DomainFeatureExtractor` class, which is responsible for extracting statistical features from domain names. These features include statistical measures such as domain length and Shannon entropy, character distribution ratios, n-gram scores based on common English bigrams and trigrams, dictionary word coverage, and TF-IDF similarity scores against legitimate domain corpora. The model outputs both

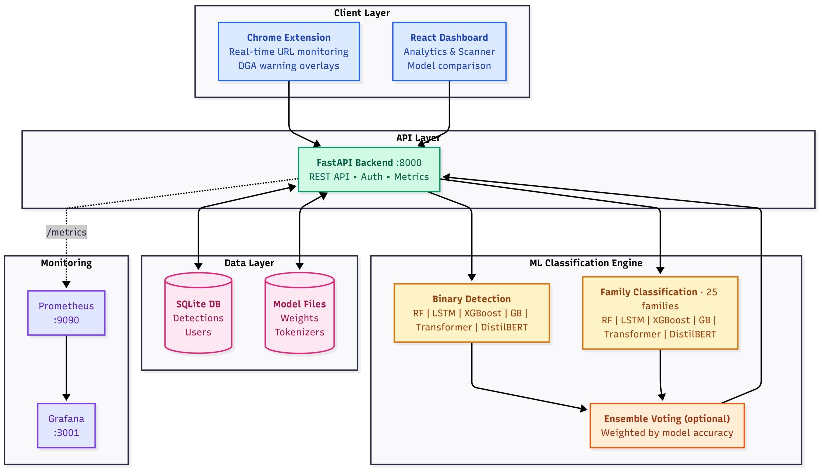


Fig. 1. System architecture of the DGA detection platform, showing the client interfaces, API layer, ML classification engine, data persistence, and monitoring stack.

a binary classification label and a confidence score derived from the proportion of trees voting for each class.

Training employs stratified 5-fold cross-validation to ensure robust performance estimation across different data partitions. Model serialization is handled through `joblib`, which efficiently stores the trained ensemble along with feature importance scores that enable interpretability analysis.

B. Gradient Boosting

The Gradient Boosting classifier extends the ensemble approach through sequential tree construction, where each subsequent tree is trained to correct the residual errors of its predecessors. The implementation uses `sklearn.ensemble.GradientBoostingClassifier` with gradient-based optimization of a log-loss objective function.

Unlike the Random Forest configuration, Gradient Boosting employs shallower trees with a maximum depth of 5 to prevent overfitting during the boosting process. A learning rate of 0.1 controls the contribution of each tree to the final prediction, providing a trade-off between model accuracy and training stability. Subsample ratio of 0.8 introduces stochastic gradient boosting by training each tree on a random subset of the training data.

Listing 2. Gradient Boosting classifier configuration

```
GradientBoostingClassifier(
    n_estimators=200,
    max_depth=5,
    learning_rate=0.1,
    subsample=0.8,
    min_samples_split=5,
    min_samples_leaf=2,
    max_features='sqrt',
    n_iter_no_change=10,
    validation_fraction=0.1,
    tol=1e-4,
    random_state=42
)
```

Early stopping is enabled through `niternochange=10`, which monitors validation performance and terminates training when no improvement is observed for 10 consecutive iterations. This mechanism prevents overfitting and reduces unnecessary computation. The `maxfeatures='sqrt'` parameter introduces additional regularization by considering only a random subset of features at each split, promoting diversity among the boosted trees.

C. XGBoost

The Extreme Gradient Boosting (XGBoost) implementation leverages the `xgboost.XGBClassifier` library [18], which provides optimized gradient boosting with advanced regularization capabilities. XGBoost extends traditional gradient boosting through second-order gradient information, column subsampling, and built-in L1/L2 regularization terms.

The model configuration specifies 200 boosting rounds with a maximum tree depth of 10, balancing model complexity against generalization. Column subsampling (`colsamplebytree=0.8`) is applied at each tree construction, preventing over-reliance on any single feature. The `scaleposweight` parameter is dynamically computed as the ratio of negative to positive samples, providing automatic class imbalance correction without requiring explicit sample weighting.

Listing 3. XGBoost classifier configuration

```
XGBClassifier(
    n_estimators=200,
    max_depth=10,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    min_child_weight=1,
    gamma=0,
    reg_alpha=0,
    reg_lambda=1,
    scale_pos_weight=neg_count/pos_count,
    eval_metric='logloss',
    random_state=42
)
```

The regularization framework includes L2 regularization (`reglambda=1`) on leaf weights to penalize model complexity. Training utilizes an evaluation set for early stopping based on log-loss, terminating after 10 rounds without improvement. XGBoost's native handling of missing values and efficient sparse-aware algorithms contribute to faster training compared to standard gradient boosting implementations.

D. CNN-LSTM Hybrid Network

The CNN-LSTM hybrid model combines convolutional neural networks for local pattern extraction with Long Short-Term Memory layers for sequential dependency modeling. This architecture processes domain names as character sequences without requiring handcrafted features, learning discriminative representations directly from raw character inputs.

The input pipeline encodes each domain character into an integer index using a vocabulary of 40 characters comprising lowercase letters, digits, and common special characters (hyphen, underscore, period). Domains are padded or truncated to a fixed length of 63 characters, consistent with DNS label length constraints. An embedding layer maps each character index to a 64-dimensional dense vector, forming the initial representation.

Listing 4. CNN-LSTM hybrid architecture
```
Input(shape=(63,))
    |
Embedding(vocab_size=40, output_dim=64)
    |
Conv1D(filters=128, kernel_size=3, activation='relu')
    |
MaxPooling1D(pool_size=2)
    |
Conv1D(filters=64, kernel_size=3, activation='relu')
    |
MaxPooling1D(pool_size=2)
    |
Bidirectional(LSTM(units=64, dropout=0.3))
    |
Dense(64, activation='relu')
    |
Dropout(0.3)
    |
Dense(1, activation='sigmoid')
```

Two convolutional layers with 128 and 64 filters respectively extract local n-gram-like patterns from the embedded character sequences. Each convolutional layer uses a kernel size of 3 and ReLU activation, followed by max-pooling with pool size 2 to reduce dimensionality and introduce translation invariance. The Bidirectional LSTM layer processes the convolutional features in both forward and backward directions, capturing sequential dependencies and contextual information from the entire domain string.

The implementation uses TensorFlow/Keras [19], [20] with callbacks for early stopping (patience of 5 epochs monitoring validation loss) and learning rate reduction on plateau (factor of 0.5 after 3 epochs without improvement). The Adam optimizer with a learning rate of 0.001 is used for training over 50 epochs with a batch size of 256.

E. Transformer

The Transformer-based classifier employs multi-head self-attention mechanisms to model global character interactions without recurrence. This architecture enables parallel processing of all character positions and captures long-range dependencies more effectively than sequential models.

The implementation defines custom `TransformerBlock` and `PositionalEncoding` layers within the TensorFlow/Keras framework. The positional encoding layer adds sinusoidal position information to the character embeddings, preserving sequence order information that would otherwise be lost in the attention mechanism:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right). \end{aligned} \tag{1}$$

Each Transformer block consists of a multi-head self-attention sublayer followed by a position-wise feed-forward network. Residual connections and layer normalization are applied after each sublayer to stabilize training and enable gradient flow through deep architectures.

Listing 5. Transformer architecture for domain classification
```
Input(shape=(63,))
    |
Embedding(vocab_size=40, output_dim=64)
    |
PositionalEncoding(max_length=63)
    |
TransformerBlock(num_heads=4, ff_dim=128, dropout=0.1)
    |
TransformerBlock(num_heads=4, ff_dim=128, dropout=0.1)
    |
GlobalAveragePooling1D()
    |
Dense(64, activation='relu') -> Dropout(0.1)
    |
Dense(32, activation='relu') -> Dropout(0.05)
    |
Dense(1, activation='sigmoid')
```

The model employs 2 Transformer blocks, each with 4 attention heads and a feed-forward dimension of 128. The key dimension per head is computed as `embeddim // numheads = 16`. Global average pooling aggregates the sequence representations into a fixed-size vector for classification. Training configuration mirrors the CNN-LSTM model with identical callbacks and optimizer settings.

F. DistilBERT

The DistilBERT classifier leverages transfer learning from a pre-trained language model, fine-tuning the `distilbert-base-uncased` checkpoint from the Hugging Face Transformers library. DistilBERT is a distilled version of BERT that retains 97 of its language understanding capability while being 40 smaller and 60 faster.

Domain names are preprocessed by inserting spaces between characters (e.g., example" becomes e x a m p l e") before tokenization. This character-level representation allows the pre-trained subword tokenizer to process individual characters while benefiting from the model's contextual embedding capabilities. Tokenization produces input sequences padded to a maximum length of 128 tokens, along with attention masks indicating valid positions.

Listing 6. DistilBERT fine-tuning configuration

```
model =
    DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels=2
)

optimizer = AdamW(model.parameters(), lr=2e-5)
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=total_steps // 10,
    num_training_steps=total_steps
)
```

The fine-tuning process uses PyTorch [21] with the AdamW optimizer at a learning rate of 2×10^{-5}, which is typical for pre-trained transformer fine-tuning. A linear learning rate schedule with warmup over 10 of the total training steps helps stabilize early training. Gradient clipping with a maximum norm of 1.0 prevents exploding gradients. Training proceeds for 3 epochs with a batch size of 32, employing early stopping with a patience of 2 epochs based on validation performance.

The DistilBERT architecture consists of 6 Transformer encoder layers with 12 attention heads each, a hidden dimension of 768, and a feed-forward dimension of 3072. The classification head takes the pooled output from the `[CLS]` token position and maps it to 2 output logits through a linear layer. This pre-trained architecture provides powerful contextual representations that can distinguish subtle patterns differentiating legitimate domains from algorithmically generated ones.

V. Results and Evaluation

A. Model Performance

*1) **Binary DGA Detection**:* This subsection reports the performance of the binary DGA detection task, where models classify each domain as benign or DGA-generated. Performance is summarized using accuracy and the precision, recall, and F1-score computed for the DGA class, as shown in Table II.

Overall, the neural and transformer-based approaches outperform the classical machine learning baselines. DistilBERT achieves the best performance across all reported metrics, reaching an accuracy of 0.961 and an F1-score of 0.961, indicating a strong balance between detecting DGA domains and controlling false alarms. The CNN-LSTM model provides comparable results (accuracy 0.957, F1-score 0.957), while the Transformer model remains competitive but slightly below the top-performing architecture (accuracy 0.939, F1-score 0.938).

Among the tree-based baselines, XGBoost yields the strongest overall results (accuracy 0.884, recall 0.848, F1-score 0.880), with Random Forest and Gradient Boosting producing similar but slightly lower scores. These results suggest that learned sequence representations in deep and transformer-based models provide a clear advantage for discriminating algorithmically generated domains.

The confusion matrices of the tree-based baselines indicate that the dominant error type is false negatives, i.e., DGA domains misclassified as legitimate, which is consistent with their comparatively lower recall values (Appendix A-A). Among these models, XGBoost reduces the number of false

TABLE II
Binary DGA detection performance (DGA as the positive class). Best values per column are highlighted.

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.883	0.918	0.840	0.877
Gradient Boosting	0.879	0.914	0.837	0.874
XGBoost	0.884	0.914	0.848	0.880
CNN-LSTM	0.957	0.963	0.950	0.957
Transformer	0.939	0.952	0.925	0.938
DistilBERT	**0.961**	**0.964**	**0.957**	**0.961**

negatives relative to Random Forest and Gradient Boosting, which explains its higher recall and F1-score, while exhibiting a slightly higher number of false positives. This suggests that XGBoost operates with a marginally less conservative decision boundary, improving detection of DGA samples at the cost of more benign domains being flagged.

Feature-importance rankings provide a consistent interpretability signal across the three tree-based models (Appendix A-F). The most influential variables are related to lexical plausibility and structural properties of domains, including dictionary coverage and dictionary word statistics, domain length, character diversity, n-gram scores, and indicators of atypical character runs (e.g., consecutive consonants or digits). In contrast, the TF-IDF-derived features contribute negligibly in these models, suggesting that the engineered lexical and character-composition features dominate the decision process for the classical baselines.

For the deep learning models, the confusion matrices show a clear reduction in both false positives and false negatives compared to the classical baselines, which aligns with the higher precision–recall balance reported in Table II (Appendix A-C). Among these models, DistilBERT yields the most favorable error profile, exhibiting the lowest counts of misclassified benign domains and missed DGA samples, while the CNN-LSTM model remains close in performance. The Transformer model produces slightly more missed detections (false negatives) than the other two, which is consistent with its comparatively lower recall.

Training curves further indicate that optimization is stable for all deep models (Appendix A-D). The CNN-LSTM exhibits a small generalization gap in later epochs, suggesting mild overfitting, whereas the Transformer curves remain closely aligned between training and validation, indicating good generalization throughout training. DistilBERT reaches strong validation performance within a small number of fine-tuning epochs, suggesting rapid convergence and limited benefit from prolonged training under the chosen setup.

*2) **Family DGA Detection**:* We next evaluate the multi-class setting, where each DGA domain is assigned to its generating family. The performance is reported using overall accuracy and macro-averaged precision, recall, and F1-score (Table III). Macro-averaging weights each family equally, providing a more faithful view of performance across both frequent and rare families.

Across the classical baselines, performance remains in

TABLE III
DGA FAMILY CLASSIFICATION PERFORMANCE. BEST VALUES PER COLUMN ARE HIGHLIGHTED (MACRO METRICS ACROSS FAMILIES).

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.795	0.787	0.795	0.786
Gradient Boosting	0.804	0.799	0.804	0.795
XGBoost	0.810	0.804	0.810	0.801
CNN-LSTM	**0.938**	**0.946**	**0.938**	**0.936**
Transformer	0.936	0.943	0.936	0.933

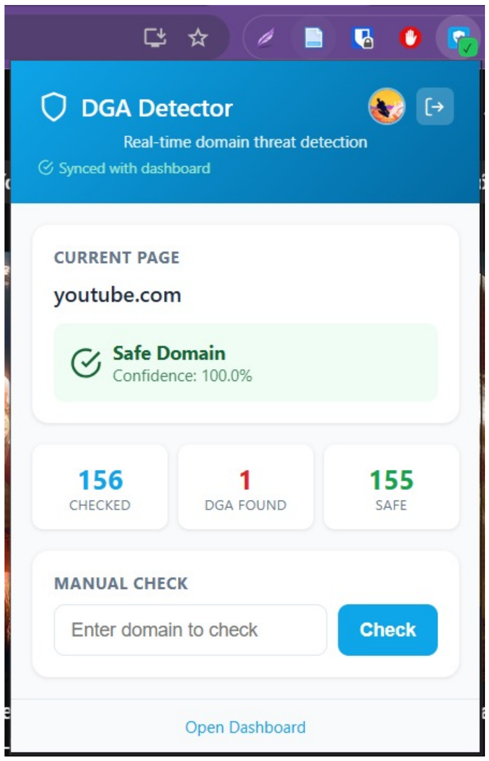
the 0.79–0.81 accuracy range, with XGBoost yielding the strongest results among the tree-based methods (accuracy 0.810, macro-F1 0.801). In contrast, the deep sequence models provide a substantial improvement, reaching approximately 0.94 accuracy and macro-F1 above 0.93. The CNN-LSTM achieves the best values across all reported metrics, indicating both strong overall correctness and consistent performance across families. The Transformer model performs comparably, with a small decrease in macro recall and macro F1, suggesting slightly more confusion between certain families despite high overall accuracy.

The confusion matrices of the Random Forest, Gradient Boosting, and XGBoost baselines are strongly diagonal, suggesting that most families are learned reliably, with errors concentrated in a small number of visually similar classes (Appendix A-E). These residual confusions are consistent with families exhibiting overlapping character distributions and structural patterns, where purely surface-level cues provide limited separation.

Feature-importance rankings further highlight which lexical signals drive the classical models (Appendix **??**). Across Random Forest and Gradient Boosting, domain length and character-composition indicators (e.g., vowel/consonant ratios, consecutive consonant runs, entropy and dictionary-coverage proxies) dominate, reflecting a reliance on phonotactic plausibility and structural regularities. XGBoost, in contrast, assigns substantial weight to digit-related structure (notably digit ratio) and delimiter patterns (e.g., hyphens), suggesting that some families are primarily distinguished by templated numeric formatting rather than broader linguistic plausibility.

For the CNN-LSTM family classifier, the confusion matrix is strongly diagonal, indicating consistently high per-family recognition and aligning with the best macro-averaged performance reported in Table III (Appendix A-G). The remaining errors are sparse and concentrated in a small number of visually similar families, where overlapping character-level patterns and templated structures reduce separability; these appear as faint off-diagonal blocks rather than systematic confusion across many classes.

Training dynamics further suggest stable optimization and good generalization (Appendix A-H). Both loss and accuracy converge quickly within the first epochs, and the validation curves largely track the training curves. While the validation loss exhibits brief transient spikes mid-training, it returns to the same downward trend without a sustained divergence. Overall, the small train–validation gap at convergence indicates only mild overfitting under the chosen setup.


DGA Detector
Real-time domain threat detection
Synced with dashboard

CURRENT PAGE
youtube.com
Safe Domain
Confidence: 100.0%

156 CHECKED
1 DGA FOUND
155 SAFE

MANUAL CHECK
Enter domain to check
Check

Open Dashboard

Fig. 2. Extension popup UI for `youtube.com`, showing a *Safe Domain* classification with 100.0% confidence, session counters, and manual domain checking.

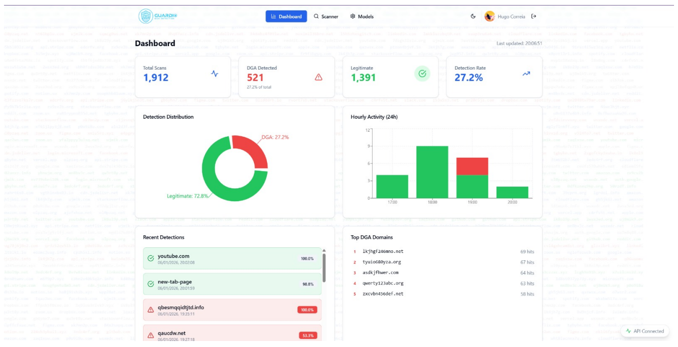
B. End-to-end System Demonstration

*1) **Browser Extension:*** To support an end-to-end demonstration in a realistic browsing setting, we provide a lightweight browser extension that surfaces DGA risk assessments directly in the user's workflow. The extension is designed to be minimally intrusive during normal navigation while still giving clear, immediate visual feedback about the security status of the current domain.

Figure 2 shows the primary popup interface. At the top, a compact header identifies the tool and indicates whether the extension is synchronized with the dashboard. The "Current Page" card prominently displays the active domain and a color-coded status panel. For benign domains, the interface uses a green checkmark and a "Safe Domain" label, accompanied by an explicit confidence percentage for transparency.

Below the status card, the popup presents session-level summary counters that provide at-a-glance situational awareness: the total number of domains checked, the number of DGA detections, and the number classified as safe. This quick summary enables users to understand how frequently checks are occurring and whether threats have been encountered during the session.

The popup also includes a *Manual Check* capability, allowing users to enter any domain and request an on-demand classification without first visiting the site. Results are displayed



Dashboard

Last updated: 20:06:51

Total Scans	DGA Detected	Legitimate	Detection Rate
1,912	521	1,391	27.2%
	27.2% of total		

Recent Detections

youtube.com	06/01/2026, 20:02:08	100.0%
new-tab-page	06/01/2026, 20:01:59	98.8%
qbesmqqidtjtd.info	06/01/2026, 19:35:11	100.0%
qaucdw.net	06/01/2026, 19:27:18	53.3%

Top DGA Domains

1	lkjhgf246mno.net	69 hits
2	tyuio680yza.org	67 hits
3	asdkjfhwer.com	64 hits
4	qwerty123abc.org	63 hits
5	zxcvbn456def.net	58 hits

Fig. 3. Dashboard interface showing summary statistics, detection distribution, hourly activity chart, and recent detections.

using the same visual language as automatic checks (status label, iconography, and confidence), ensuring consistency and easy interpretation.

When a suspicious domain is detected, the extension escalates the visual treatment to emphasize risk (e.g., red warning indicators) and can present an interruptive warning experience on the page to prevent accidental continuation. Finally, a persistent "Open Dashboard" link provides a direct transition from the lightweight popup view to the richer dashboard interface for deeper inspection and historical review.

*2) **DGA Detector Platform**:* The DGA Detector Platform provides a web-based interface for domain analysis, system monitoring, and model management. The application follows a modern single-page architecture with responsive design, supporting both light and dark themes for user comfort across different viewing conditions.

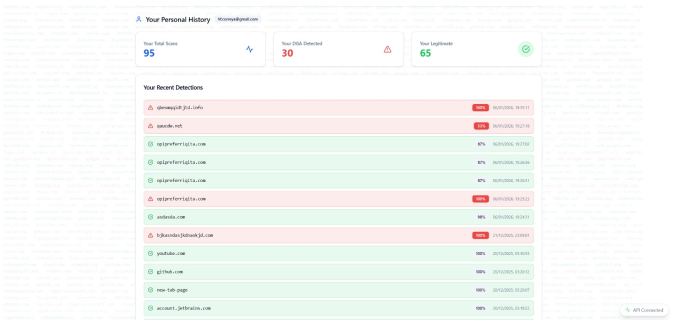
The dashboard, presented in Fig. 3, serves as the central monitoring hub, presenting aggregated statistics and recent activity at a glance. The top row displays four summary cards showing total scans, DGA detections, legitimate classifications, and the overall detection rate.

The middle section contains two visualization panels arranged side by side. A donut chart displays the detection distribution between DGA and legitimate domains with percentage labels, while a stacked bar chart shows hourly activity over the past 24 hours, allowing users to identify temporal patterns in scanning behavior and threat detection.

The lower section presents two scrollable lists: recent detections on the left and top DGA domains on the right. Each detection entry is color-coded based on its classification—red background tints for DGA detections and green for legitimate domains—with the domain name, timestamp, and confidence percentage clearly visible. The top DGA domains list ranks the most frequently detected malicious domains with hit counts, helping identify persistent threats.

For authenticated users, an additional "Personal History" section appears below, displaying individual scanning statistics and a personalized list of recent detections associated with their account, as shown in Fig. 4.

The scanner page provides the primary interface for domain analysis. A tabbed interface at the top allows switching be-



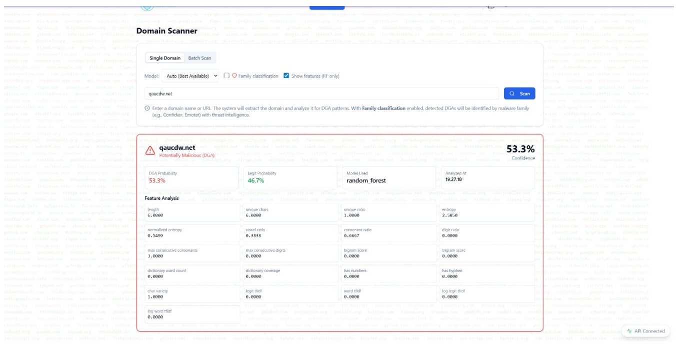
Your Personal History hf.correya@gmail.com

Your Total Scans	Your DGA Detected	Your Legitimate
95	30	65

Your Recent Detections

qbesmqqidtjtd.info	100%	06/01/2026, 19:35:11
qaucdw.net	53%	06/01/2026, 19:27:18
opipreferriqita.com	87%	06/01/2026, 19:27:02
opipreferriqita.com	87%	06/01/2026, 19:26:56
opipreferriqita.com	87%	06/01/2026, 19:26:51
opipreferriqita.com	100%	06/01/2026, 19:25:23
asdasda.com	98%	06/01/2026, 19:24:31
bjkasndasjkdnaskjd.com	100%	21/12/2025, 23:09:01
youtube.com	100%	20/12/2025, 03:30:55
github.com	100%	20/12/2025, 03:20:12
new-tab-page	100%	20/12/2025, 03:20:07
account.jetbrains.com	100%	20/12/2025, 03:19:53

Fig. 4. Personal detection statistics and history for authenticated users.



Domain Scanner

Model: Auto (Best Available) ☐ Family classification ☑ Show features (RF only)

qaucdw.net

Enter a domain name or URL. The system will extract the domain and analyze it for DGA patterns. With **Family classification** enabled, detected DGAs will be identified by malware family (e.g., Conficker, Emotet) with threat intelligence.

qaucdw.net
Potentially Malicious (DGA)
53.3%
Confidence

DGA Probability	Legit Probability	Model Used	Analyzed At
53.3%	46.7%	random_forest	19:27:18

Feature Analysis

length 6.0000	unique chars 6.0000	unique ratio 1.0000	entropy 2.5850
normalized entropy 0.5499	vowel ratio 0.3333	consonant ratio 0.6667	digit ratio 0.0000
max consecutive consonants 3.0000	max consecutive digits 0.0000	bigram score 0.0000	trigram score 0.0000
dictionary word count 0.0000	dictionary coverage 0.0000	has numbers 0.0000	has hyphen 0.0000
char variety 1.0000	legit tfidf 0.0000	word tfidf 0.0000	log legit tfidf 0.0000
log word tfidf 0.0000			

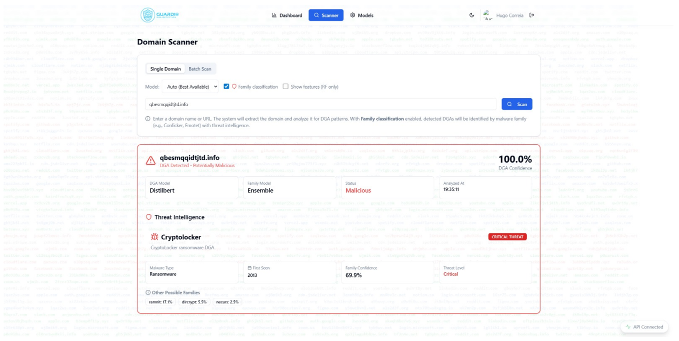
Fig. 5. Domain scanner interface with the "Show features" option activated.

tween single-domain and batch-scan modes. In single-domain mode, users enter a domain name in a text field and click a "Scan" button to initiate analysis. In batch mode, a multi-line text area accepts multiple domains separated by newlines for bulk processing.

Configuration options appear above the input area, including a model selector dropdown for choosing among the available detection models (with an "Auto" option that selects the best available model), a "Family classification" checkbox to enable malware family identification, and a "Show features" checkbox for displaying the extracted features when using the Random Forest model, the Fig. 5 shows this option activated.

Scan results appear below the input form with prominent visual treatment. For standard scans, a result card displays the domain name, classification status ("Potentially Malicious" or "Likely Legitimate"), and confidence percentage. A grid of four metric cards shows the DGA probability, legitimate probability, model used, and analysis timestamp. The card's border and background color reflect the classification—red tones for DGA detections and green for legitimate domains.

When family classification is enabled and a DGA is detected, an extended result card appears with a "Threat Intelligence" section, as shown in Fig. 6. This section prominently displays the identified malware family name alongside a color-coded threat level badge (Critical, High, Medium, or Low). Additional details include the malware type (e.g., Ransomware, Banking Trojan, Botnet), first-seen date, family confidence score, and a brief description of the malware family's characteristics. Alternative family classifications with



Domain Scanner

Single Domain | Batch Scan

Model: Auto (Best Available) | ☑ Family classification | ☐ Show features (RF only)

qbesmqqidtjtd.info | Scan

Enter a domain name or URL. The system will extract the domain and analyze it for DGA patterns. With **Family classification** enabled, detected DGAs will be identified by malware family (e.g., Conficker, Emotet) with threat intelligence.

qbesmqqidtjtd.info
DGA Detected - Potentially Malicious

100.0%
DGA Confidence

DGA Model	Family Model	Status	Analyzed At
Distilbert	Ensemble	Malicious	19:35:11

Threat Intelligence

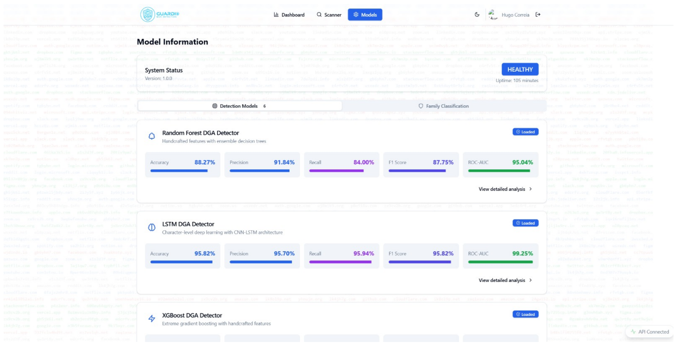
Cryptolocker — CRITICAL THREAT
CryptoLocker ransomware DGA

Malware Type	First Seen	Family Confidence	Threat Level
Ransomware	2013	69.9%	Critical

Other Possible Families
ramnit: 17.1% | dircrypt: 5.5% | necurs: 2.5%

API Connected

Fig. 6. Family classification result showing threat intelligence with malware family identification, threat level, and description.



Model Information

System Status
Version: 1.0.0
HEALTHY
Uptime: 105 minutes

Detection Models 6 | Family Classification

Random Forest DGA Detector — Loaded
Handcrafted features with ensemble decision trees

Accuracy	Precision	Recall	F1 Score	ROC-AUC
88.27%	91.84%	84.00%	87.75%	95.04%

View detailed analysis

LSTM DGA Detector — Loaded
Character-level deep learning with CNN-LSTM architecture

Accuracy	Precision	Recall	F1 Score	ROC-AUC
95.82%	95.70%	95.94%	95.82%	99.25%

View detailed analysis

XGBoost DGA Detector — Loaded
Extreme gradient boosting with handcrafted features

API Connected

Fig. 7. Models page displaying detection and family classification models with performance metrics.

their confidence scores appear at the bottom, providing transparency about the model's uncertainty.

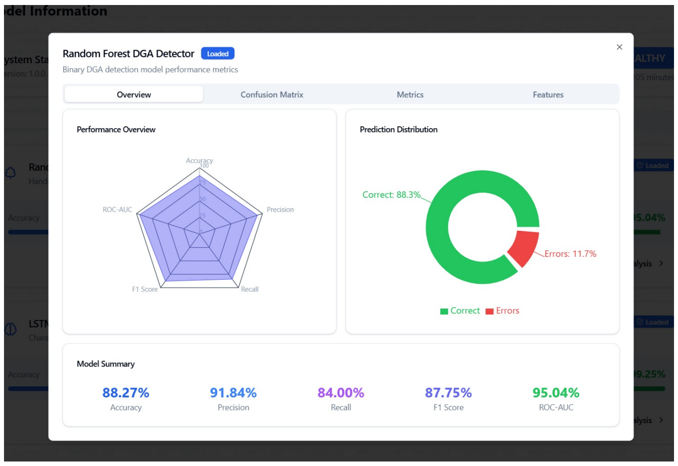
The models page, shown in Fig. 7, provides visibility into the loaded machine learning models and their performance characteristics. A system status card at the top displays the API health status, version number, and uptime duration, using a color-coded badge to indicate healthy or unhealthy states.

Below the status card, a tabbed interface separates detection models (for binary DGA classification) from family classification models. Each tab displays a count badge indicating the number of loaded models in that category. Model cards present individual models with a type-specific icon (e.g., a tree for Random Forest, a brain for LSTM, a sparkle for Transformer), the model name, a brief description of its architecture, and a loading status badge.

For trained models, a metrics grid displays five key performance indicators: accuracy, precision, recall, F1 score, and ROC-AUC. Each metric appears in a compact card with color-coded typography for visual differentiation. A "View detailed analysis" button provides access to extended model information including confusion matrices and feature importance visualizations through a modal dialog, as shown in Fig. 8.

VI. Conclusion

This work presented an end-to-end framework for detecting domains generated by DGAs and, when malicious, attributing them to a likely DGA family. Beyond model development,



Random Forest DGA Detector — Loaded
Binary DGA detection model performance metrics

Model Summary

Accuracy	Precision	Recall	F1 Score	ROC-AUC
88.27%	91.84%	84.00%	87.75%	95.04%

Fig. 8. Models page displaying detection and family classification models with performance metrics.

the project emphasizes practical deployability by integrating trained classifiers into a complete detection pipeline with a user-facing dashboard, browser extension, and operational monitoring. This framing targets the core constraint of real-world DGA defense: maintaining high detection quality while remaining usable, observable, and maintainable in realistic browsing and analyst workflows.

Empirically, the evaluation highlights a clear advantage for representation-learning approaches over classical lexical feature baselines. In the binary detection task, DistilBERT achieved the strongest overall performance (accuracy and F1-score of 0.961 in Table II), with CNN-LSTM also performing competitively (accuracy 0.957). In the multi-class family setting, CNN-LSTM delivered the best macro results (accuracy 0.938 and macro F1-score 0.936 in Table III), narrowly outperforming the transformer model (accuracy 0.936, macro F1-score 0.933). By contrast, the tree-based baselines clustered near 0.80 accuracy in family classification, indicating persistent confusion across families when relying primarily on surface-level lexical cues. Additional confusion matrices, feature-importance rankings, and training curves that support these findings are provided in Appendix A.

References

[1] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From Throw-Away traffic to bots: Detecting the rise of DGA-Based malware," in *21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX Association, Aug. 2012, pp. 491–506. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/antonakakis

[2] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: Dga-based botnet tracking and intelligence," vol. 8550, 07 2014, pp. 192–211.

[3] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A passive dns analysis service to detect and report malicious domains," *ACM Trans. Inf. Syst. Secur.*, vol. 16, no. 4, Apr. 2014. [Online]. Available: https://doi.org/10.1145/2584679

[4] S. Schüppen, D. Teubert, P. Herrmann, and U. Meyer, "FANCI : Feature-based automated NXDomain classification and intelligence," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp.

1165–1181. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/schuppen

[5] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," *CoRR*, vol. abs/1611.00791, 2016. [Online]. Available: http://arxiv.org/abs/1611.00791

[6] C. Morbidoni, "Dga domains dataset," GitHub repository, Jan. 2026. [Online]. Available: https://github.com/chrmor/DGA_domains_dataset

[7] 360 Netlab, "Netlab opendata project: Dga feed," Online dataset/feed, Jan. 2026. [Online]. Available: https://data.netlab.360.com/dga/

[8] Alexa Internet, "Alexa top sites," Website, Jan. 2026. [Online]. Available: https://www.alexa.com/topsites

[9] A. Cucchiarelli, C. Morbidoni, L. Spalazzi, and M. Baldi, "Algorithmically generated malicious domain names detection based on n-grams features," *Expert Systems with Applications*, vol. 170, p. 114551, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420311957

[10] Ö. Tatar, "Dga data sets," Kaggle dataset, Oct. 2022. [Online]. Available: https://www.kaggle.com/datasets/omurcantatar/dga-data-sets

[11] alexandrosmagos, "dyn-dns-list: Dynamic dns domain list," GitHub repository, Jan. 2026. [Online]. Available: https://github.com/alexandrosmagos/dyn-dns-list

[12] Meta Platforms, Inc., "React documentation," Online documentation, Jan. 2026. [Online]. Available: https://react.dev/

[13] FastAPI Contributors, "Fastapi documentation," Online documentation, Jan. 2026. [Online]. Available: https://fastapi.tiangolo.com/

[14] SQLite Project, "Sqlite documentation," Online documentation, Jan. 2026. [Online]. Available: https://sqlite.org/docs.html

[15] Prometheus Authors, "Prometheus documentation," Online documentation, Jan. 2026. [Online]. Available: https://prometheus.io/docs/introduction/overview/

[16] Grafana Labs, "Grafana documentation," Online documentation, Jan. 2026. [Online]. Available: https://grafana.com/docs/grafana/latest/

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[18] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang, "Tensorflow: A system for large-scale machine learning," *CoRR*, vol. abs/1605.08695, 2016. [Online]. Available: http://arxiv.org/abs/1605.08695

[20] F. Chollet and others, "Keras: The Python Deep Learning library," Astrophysics Source Code Library, record ascl:1806.022, Jun. 2018.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *CoRR*, vol. abs/1912.01703, 2019. [Online]. Available: http://arxiv.org/abs/1912.01703

Appendix A
Supplementary Evaluation Plots

A. *Binary Confusion Matrices for Tree-Based Models*

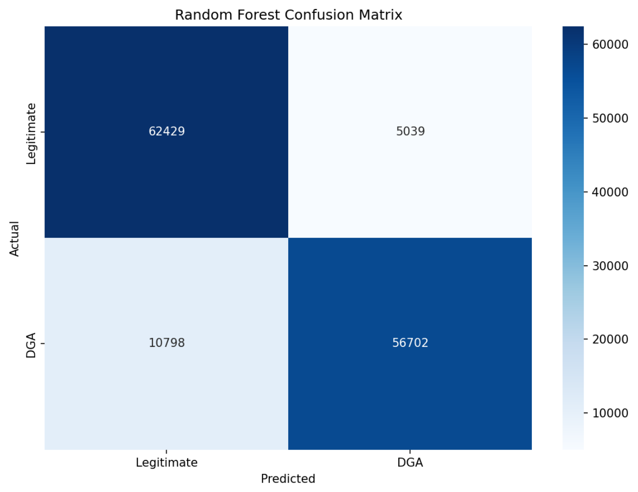


Fig. 9. Random Forest confusion matrix for the binary DGA detection task.

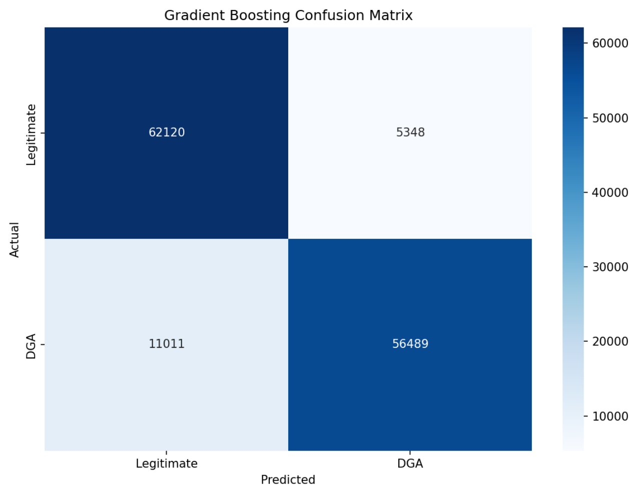


Fig. 10. Gradient Boosting confusion matrix for the binary DGA detection task.

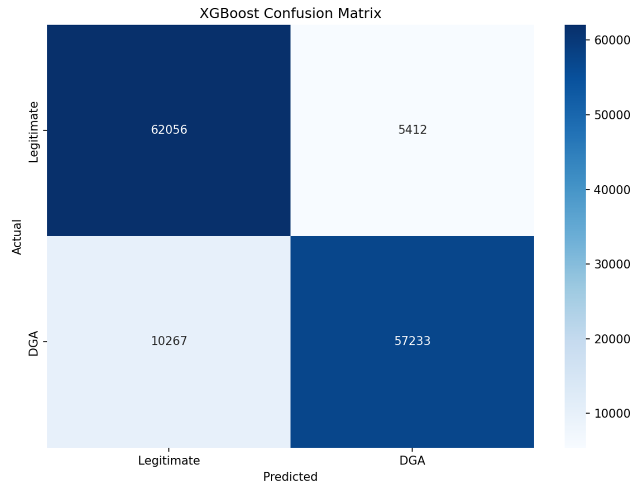


Fig. 11. XGBoost confusion matrix for the binary DGA detection task.

B. Feature Importance for Tree-Based Models in Binary Classification

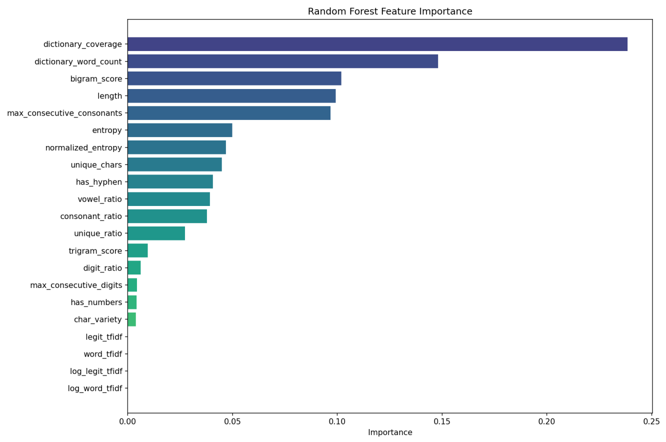


Fig. 12. Random Forest feature importance scores for the engineered lexical and character-level features.

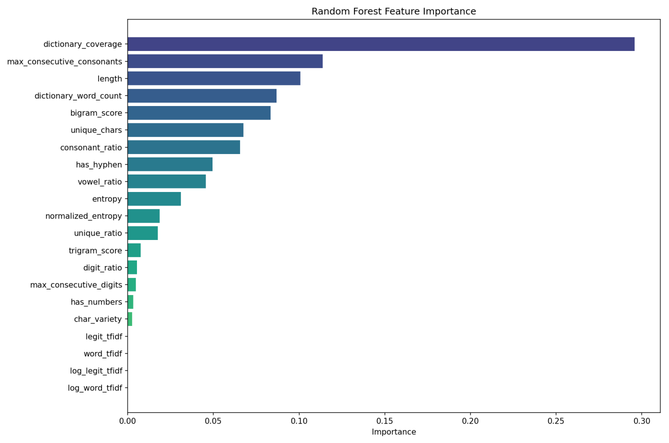


Fig. 13. Gradient Boosting feature importance scores for the engineered lexical and character-level features.

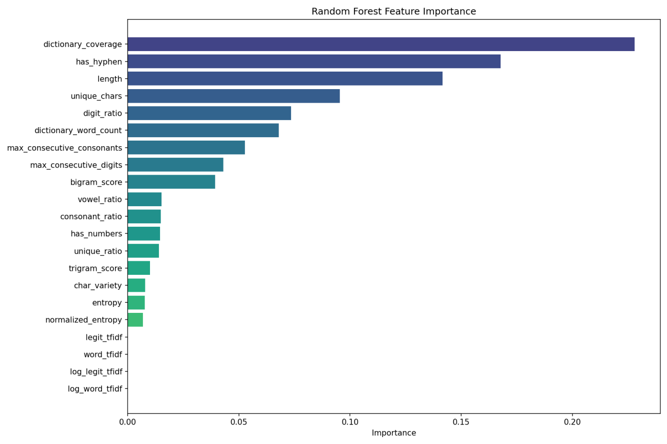


Fig. 14. XGBoost feature importance scores for the engineered lexical and character-level features.

C. Binary Confusion Matrices for Deep Learning Models

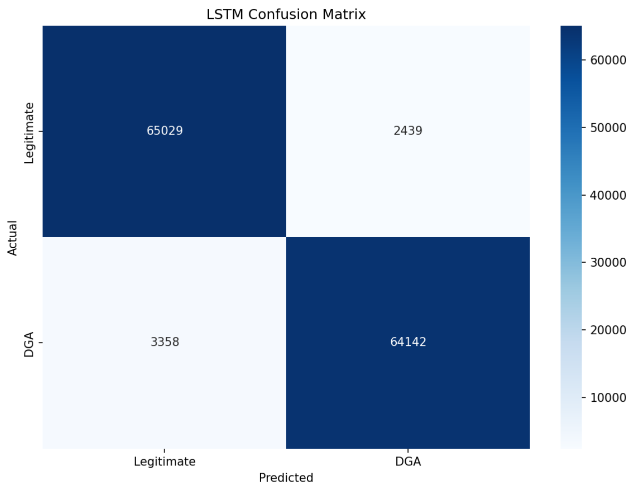


Fig. 15. CNN-LSTM confusion matrix for the binary DGA detection task.

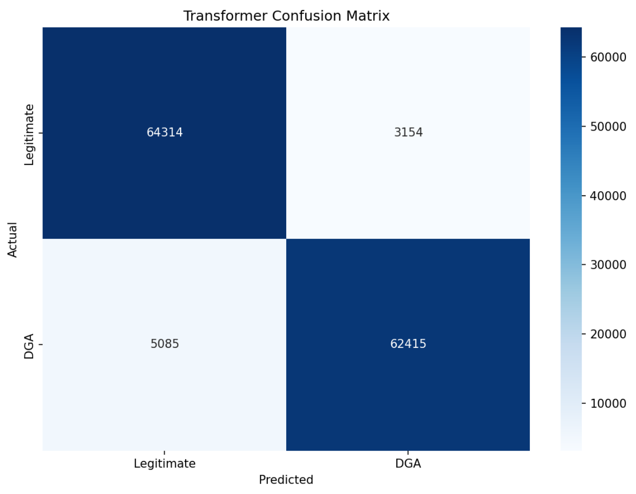


Fig. 16. Transformer confusion matrix for the binary DGA detection task.

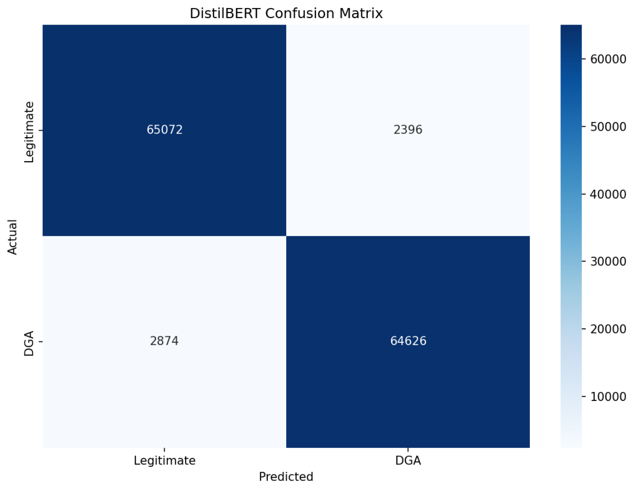


Fig. 17. DistilBERT confusion matrix for the binary DGA detection task.

D. Training Curves for Deep Learning Models

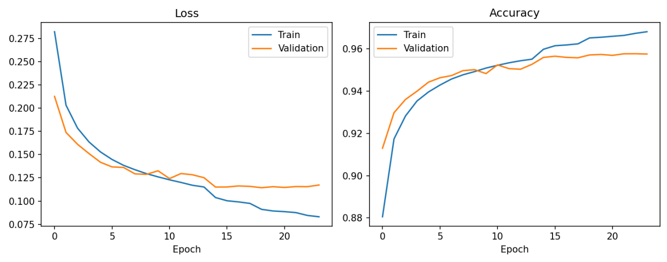


Fig. 18. CNN-LSTM training and validation loss and accuracy across epochs.

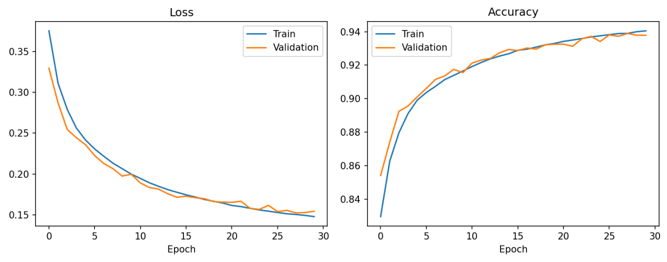


Fig. 19. Transformer training and validation loss and accuracy across epochs.

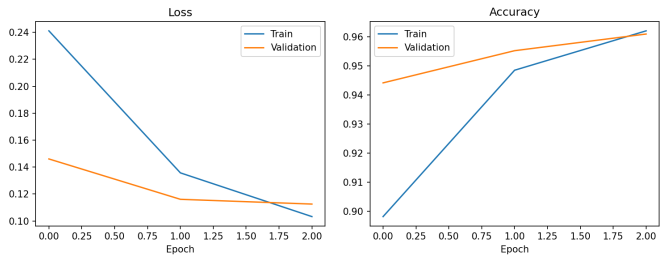


Fig. 20. DistilBERT training and validation loss and accuracy across fine-tuning epochs.

E. Family Confusion Matrices for Tree-Based Models

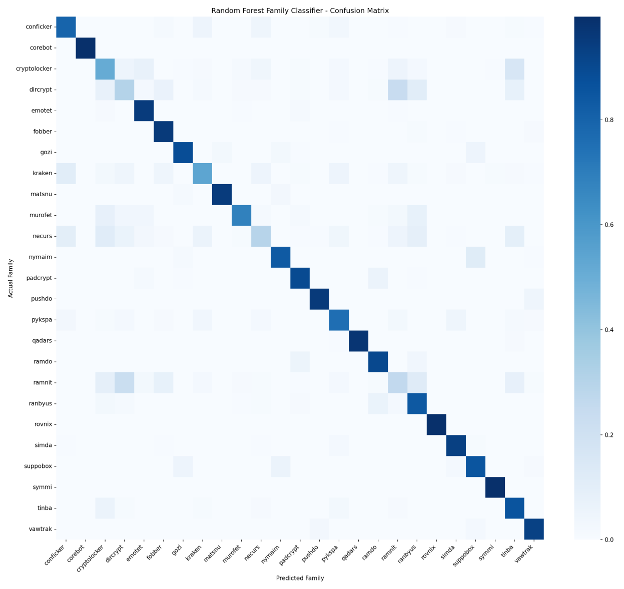


Fig. 21. Random Forest confusion matrix for the family DGA detection task.

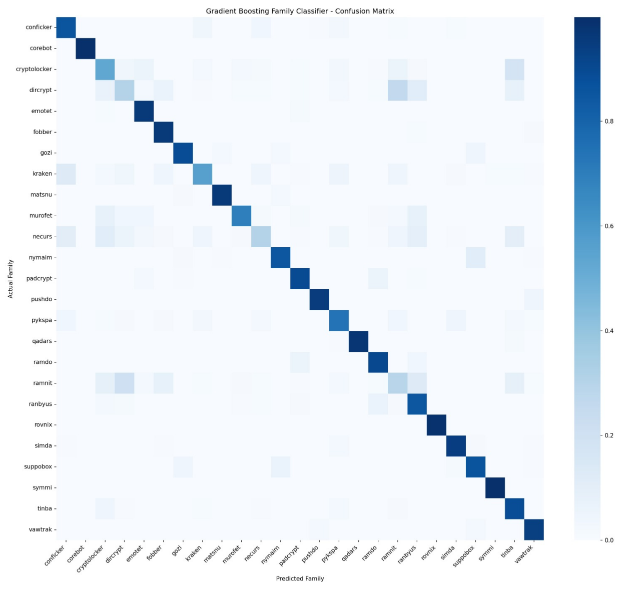


Fig. 22. Gradient Boosting confusion matrix for the family DGA detection task.

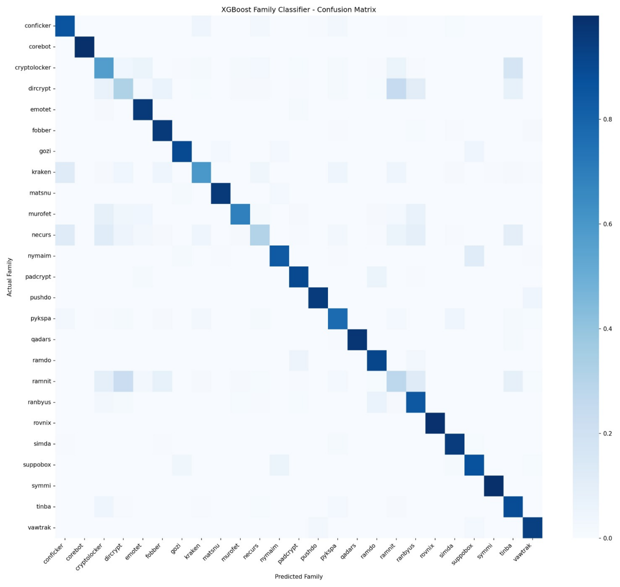


Fig. 23. XGBoost confusion matrix for the family DGA detection task.

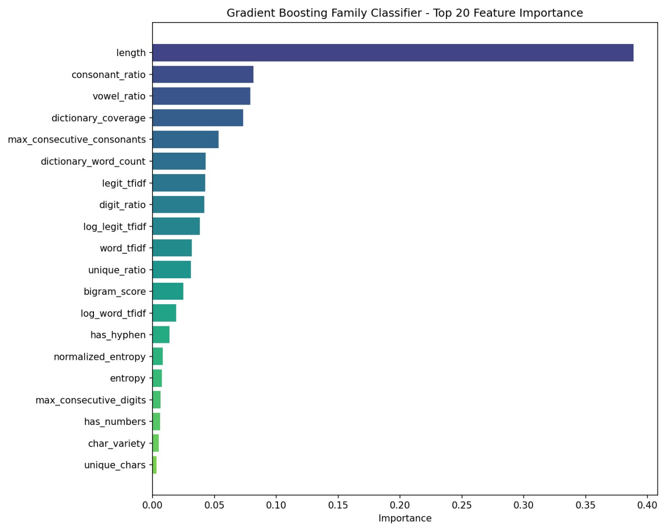


Fig. 25. Gradient Boosting feature importance scores for the engineered lexical and character-level features.

F. Feature Importance for Tree-Based Models in Family Classification

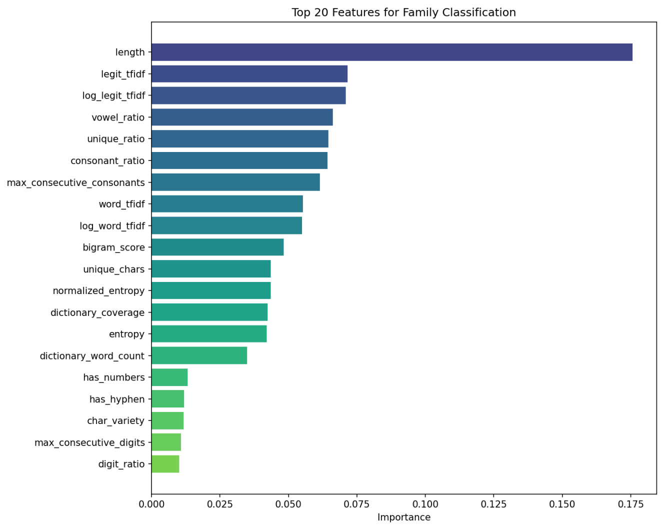


Fig. 24. Random Forest feature importance scores for the engineered lexical and character-level features.

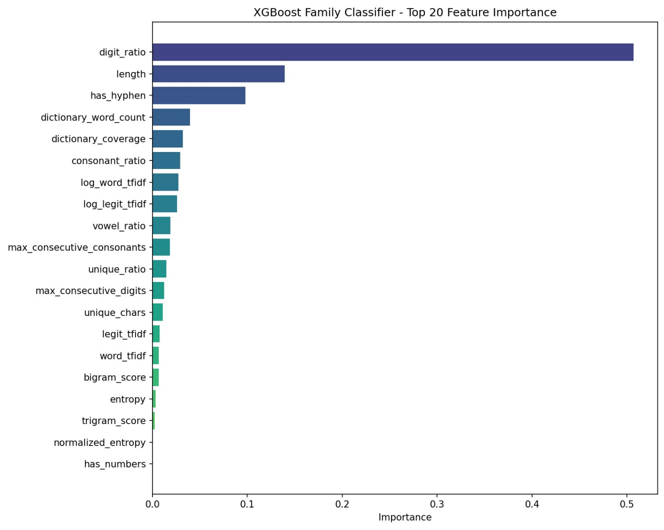


Fig. 26. XGBoost feature importance scores for the engineered lexical and character-level features.

G. Family Confusion Matrices for Deep Learning Models

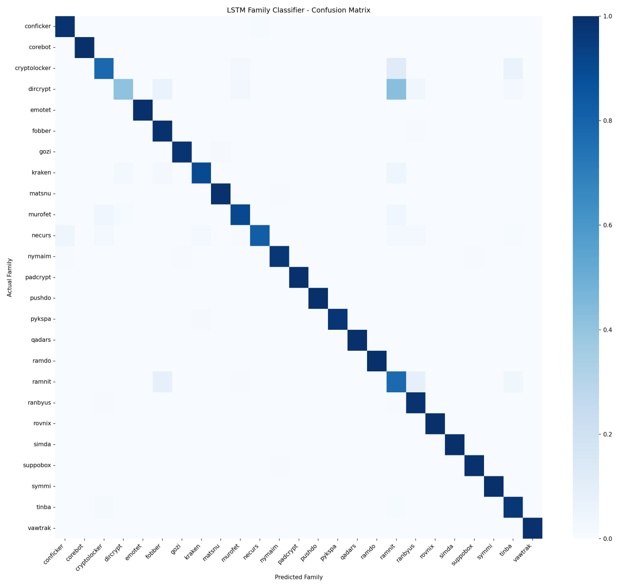


Fig. 27. CNN-LSTM confusion matrix for the family DGA detection task.

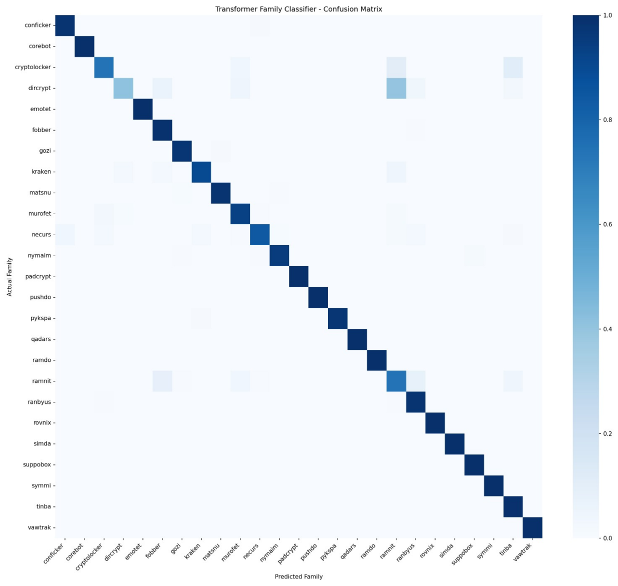


Fig. 28. Transformer confusion matrix for the family DGA detection task.

H. Family Training Curves for Deep Learning Models

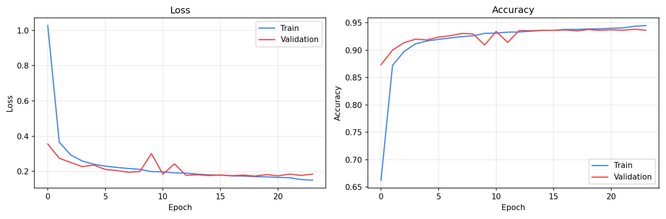


Fig. 29. CNN-LSTM training and validation loss and accuracy across epochs for family classification.

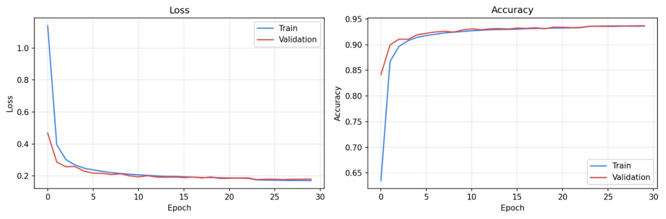


Fig. 30. Transformer training and validation loss and accuracy across epochs for family classification.