ISSN: 2977-0041

Research Article

Journal of Material Sciences and Engineering Technology

Simple but Smart: Against the Pursuit of Endless Complexity in Deep Learning Models for Detecting Phishing URLs

Musa Ibrahim Anda[1], Armaya'u Zango Umar[2*] and Barira Hamisu[2]

[1]*Dept. of Computer Science and IT Al-Qalam University Katsina, Nigeria*

[2]*Dept. of Software Engr. and Cybersec Al-Qalam University Katsina, Nigeria*

[*]Corresponding author
Armaya'u Zango Umar, Dept. of Software Engr. and Cybersec Al-Qalam University Katsina, Nigeria.

Received: January 08, 2026; **Accepted:** January 19, 2026; **Published:** January 26, 2026

ABSTRACT
Phishing attacks are one of the most prevalent and evolving cyber security threats. The attacks often employ a fraudulent Universal Resource Locator (URL) and a well-crafted social engineering tactic to mislead gullible individuals into releasing their sensitive information, such as bank or credit card details. Machine learning classification models have been proposed to proactively detect phishing URLs. In an attempt to improve the detection accuracy, deep learning models were also proposed. Yet, to push the performance of the models even further, more sophisticated deep learning architectures were proposed. Nonetheless, the sophistication in the architectures does not improve the performance commensurate with its complexity. To this end, this paper compared the performance of a simple Feedforward Neural Network (FNN) against more complex architectures: hybridized Deep Neural Network and Bidirectional Long Short-Term Memory (DNN-BiLSTM), hybridized Deep Neural Network and Bidirectional Long Short-Term Memory (DNN-BiLSTM) with a transformer, and hybridized Deep Neural Network and Bidirectional Long Short-Term Memory (DNN-BiLSTM) with semantic Natural Language Processing (NLP) features. The models were trained on a phishing dataset that had label noise corrected with Cleanlab – A Confidence learning framework. The results show that a simple Feedforward Neural Network, when trained on cleaned data, can equal or even surpass the performance of any complex deep learning architecture while maintaining significantly lower runtime.

Introduction

Phishing is a cyber-attack that exploits human vulnerabilities using a carefully crafted fraud known as social engineering. The exploitable human vulnerabilities include fear, trust, or curiosity. For example, attackers might launch a fraudulent airtime campaign impersonating one of the major telecom companies. In the campaign, the attackers will provide a fraudulent URL to redeem the purported offer. When a victim clicks on the provided link, certain sensitive information will be requested, and when provided, the attackers have succeeded in deceiving the victim. Another typical example is impersonating a major corporate organization, such as a central bank. In this case, the attackers' message to the victim might be claim to offer certain benefits, such as government's palliative or business support. As in the previous example, a fake URL will be provided which when clicked, will land the victim on a fake central bank website. Sensitive information such as bank and credit card details may be phished from the victim and when provided, such information is compromised. In both examples, the attackers exploit the trust the victims have in the reputable organizations and the fear of missing out from the purported benefits.

To prepare for phishing, the attackers create a domain name that mimics the real one. For example, if the real domain is cbn.gov.ng, of which the URL is https://www.cbn.gov.ng/, the fake domain created by the attackers might be cbn.com (with the URL as https://www.cbn.com). Similarly, if a real domain of a telecom service provider is mtn.ng, of which the URL is https://www.mtn.ng/, the fake one might be m-tn.ng, with the dash after m (with the URL as https://www.m-tn.ng/). After the creation of

Citation: Musa Ibrahim Anda, Armaya'u Zango Umar, Barira Hamisu. Simple but Smart: Against the Pursuit of Endless Complexity in Deep Learning Models for Detecting Phishing URLs. J Mat Sci Eng Technol. 2026. 4(1): 1-8. DOI: doi.org/10.61440/JMSET.2026.v4.96

the fake domain, the website of the impersonated organization will be cloned to have a fake website with the same look and feel (colour, logo, and aesthetics) as the genuine website.

There have been several attempts to use machine learning models to detect phishing URLs. These include both white box and deep learning (black box) models. In the category of white box models, the studies proposed smart models engineered with sophisticated ensembling strategies or elegant feature selection strategies [1-3]. Although a white box model has the advantage of being interpretable and computationally more efficient than its deep learning model counterpart, it has a limitation in its failure to recognize complex patterns that may be present in a typical phishing URL. Consequently, deep learning models were also proposed.

In the category of deep learning models, proposed a Long Short-Term Memory (LSTM) neural network architecture to capture sequential patterns in URLs [4]. extended this study using character-level Convolutional Neural Networks (CNNs) to reduce the computational demand of word-level encoding. The fact that a URL is a sequential text that often contains information with a semantic pattern, a study introduced CNN-Fusion, which used multiple one-layer CNNs with different kernel sizes to capture spatial patterns at varying granularities [5,6].

In machine learning, there are two (2) popular approaches to capturing features for model training: (i) character embedding and (ii) manually hand-crafted. Character embedding is a technique used to represent individual characters (letters, digits, symbols, etc.) as dense, low-dimensional vectors in a continuous vector space. Thus, each individual character in a text sequence is represented as a numerical vector. These embedding capture semantic and syntactic similarities between characters, allowing models to generalize better when processing text data at the character level [7]. In contrast to character embedding, manually hand-crafted features refer to features that are explicitly designed by domain experts or data scientists based on prior knowledge, intuition, and data analysis. Hand-crafted features were dominant before the advent of deep learning. In the context of phishing detection, examples of hand-crafted features include URL length, the presence of suspicious characters, or domain age.

Although deep learning models can capture complex patterns in URLs, existing studies have mainly focused on either hand-crafted features (high-level connections) or character embedding-based features but not both [5,4]. Consequently, a hybrid approach, which combines different methods to improve accuracy and precision, was proposed [8]. The hybrid model integrates the strengths of both Deep Neural Networks and Bidirectional Long Short-Term Memory (DNN-BiLSTM) networks to enhance phishing URL detection. The work in is state-of-the art but did not benefit from Confident Learning, leading to a very complicated architecture [8-10]. CL is a framework in machine learning that focuses on identifying, quantifying, and correcting label errors in datasets. It is particularly useful in weakly supervised learning, where training labels may be noisy or unreliable.

To this end, this paper compared the performance of a simple Feedforward Neural Network (FNN), engineered with Confidence Learning, against the complex architecture proposed in as it was, and against other modifications to the architecture proposed in [8]. The results show that a simple Feedforward Neural Network, when trained on cleaned data, with label noise corrected, can equal or even surpass the performance of any complex deep learning architecture while maintaining significantly lower runtime.

Literature Review

Phishing URL detection has witnessed a variety of methodological approaches, ranging from traditional machine learning to advanced deep learning architectures. Early models focused heavily on character-level features derived from raw URLs. For instance, employed a Long Short-Term Memory (LSTM) neural network to capture sequential patterns in URLs, achieving an accuracy of 98.7% and outperforming Random Forest classifiers [4]. extended this line of work using character-level Convolutional Neural Networks (CNNs), emphasizing speed and lightweight computation by eliminating the need for external content retrieval [5]. Their model effectively captured syntactic URL patterns and performed competitively against benchmark datasets.

In a similar vein, introduced CNN-Fusion, which used multiple one-layer CNNs with different kernel sizes to capture spatial patterns at varying granularities [6]. To improve robustness, they applied spatial dropout and pooling techniques. Their architecture proved lightweight and suitable for resource-constrained devices. However, like previous character-level models, it lacked semantic awareness, making it vulnerable to more sophisticated or obfuscated phishing strategies.

Other researchers attempted to bridge this gap by integrating contextual and content-based features. Designed fusion models that analysed URLs along with webpage titles, body text, and hidden HTML tags [6]. This allowed their models to extract semantic relationships beyond syntactic URL patterns. Although their multi-component architecture provided nuanced insight into phishing content, its reliance on complete webpage data makes real-time or large-scale deployment less feasible. A study also leveraged both URL and HTML data through character and word embedding, achieving a high detection accuracy of 98.1%. Nevertheless, the overhead of processing and embedding HTML content remained a practical bottleneck [11].

Seeking a middle ground, Sudar. Focused on dynamic features of URLs — such as domain age and live web-scraped content — and applied forward selection and LASSO regularization to retain only the most informative attributes. While this method addressed the limitations of static analysis, it too suffered from reliance on live web availability, which may be inconsistent or time-sensitive in real-world phishing cases.

Meanwhile, Wei and Sekiya compared a variety of ensemble machine learning methods (e.g., Random Forest, AdaBoost, LightGBM) and neural architectures (FCNN, LSTM, CNN), ultimately concluding that traditional ensemble methods outperformed deep learning approaches on their dataset. While insightful, the study did not explore newer hybrid or attention-based architectures that may close this performance gap.

A more integrative approach was proposed by van Geest who developed a hybrid phishing detection framework combining URL-based models with HTML content and DOM structure analysis, merged through a stacking ensemble. This demonstrated improved detection robustness and real-world applicability, but introduced significant computational complexity, making it less suitable for scalable or low-latency applications.

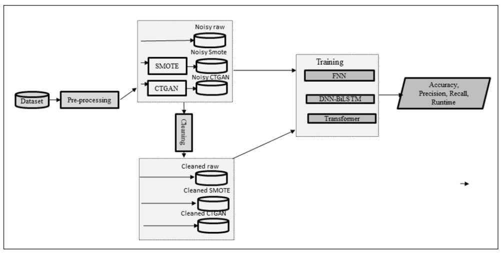
Taken together, these studies highlight key gaps in the phishing detection literature. First, many models focus exclusively on either character-level or content-level information, often neglecting the middle ground of semantic URL structures, such as suspicious tokens, domain heuristics, or entropy scores. Second, although complex fusion or ensemble models often show improved accuracy, they tend to suffer from overfitting, poor scalability, or impractical latency.

Methodology

This section outlines the research design, data pre-processing techniques, synthetic data augmentation strategies, model development, and performance evaluation methods used in this study. The methodology is designed to evaluate the trade-offs between architectural complexity and performance in phishing URL detection, with a focus on simplifying models and optimizing data quality. The methodological workflow is illustrated in Figure 1.

The Dataset

The dataset employed in this study comprises a total of 11,430 records. It was curated for training and evaluating machine learning models in the context of phishing detection, with the primary goal of distinguishing between legitimate and malicious websites based on a wide array of syntactic, lexical, and structural characteristics extracted from URLs and their associated web content [12].



Synthetic Minority Over-sampling Technique (SMOTE)

As class imbalance often affects the performance of machine learning models, Synthetic Minority Over-sampling Technique (SMOTE) is a widely used data augmentation method designed to mitigate class imbalance in machine learning datasets [13]. When one class (the majority) dominates another (the minority), models tend to bias predictions toward the majority class, degrading performance on the minority class. SMOTE counteracts this by generating synthetic minority-class samples, thereby balancing the dataset and enhancing model performance. Data balancing with SMOTE was used in this study to compare its impact with label noise correction using clean lab.

Conditional Tabular Generative Adversarial Network (CTGAN),

The Conditional Tabular Generative Adversarial Network (CTGAN) is a synthetic data generation technique introduced by [14]. It enhances the traditional GAN architecture to better handle the generation of structured tabular data [15]. It ensures that the synthetic data not only mirrors the marginal distributions of the original dataset but also retains the conditional relationships among different features. CTGAN begins by selecting a condition from the actual dataset, which is then combined with random noise and fed into the conditional generator. This strategy allows the model to preserve the attribute dependencies present in the original data.

Confidence Learning with Clean Lab

To address the challenge of label noise, which can significantly degrade model performance, this study employed the clean lab framework to automatically identify and remove potentially mislabelled examples from the training data. The process is presented in Algorithm 1. The method began by loading the data and encoding the target feature (Line 3 and 4 of Algorithm 1). The Algorithm then proceeds with the encoding of the remaining features (Line 5 of Algorithm 1). Then, a baseline classification model was built (Line 6 of Algorithm 1). The baseline model plays a foundational and diagnostic role. It serves as the primary tool through which clean lab estimates the reliability of each labelled instance in the dataset. It estimates the probability that a given sample's label is incorrect based on the model's understanding of the data distribution [16]. In this study, Logistic Regression was used as the baseline model. Five (5) cross-validated probability estimates were used to ensure that the detection of label issues was based on fair and generalizable model behaviour (Line 7 of Algorithm 1). The labels with noise are detected and removed (Line 8 and 9 of Algorithm 1). The cleaned dataset is then saved to a file. The label correction was applied to the three different datasets: (i) raw dataset, (ii) balanced dataset using SMOTE, and (iii) augmented dataset using CTGAN. The operation resulted in producing three (3) cleaned datasets: (i) Cleaned Raw, Cleaned SMOTE, and (iii) Cleaned CTGAN.

Algorithm 2: Label Noise Correction with Cleanlab

1	Input: raw dataset
2	Output: cleaned_dataset
3	Load dataset
4	Encode target labels
5	Set up preprocessing pipeline: preprocessor ← Column Transformer with: - StandardScaler for numeric features - OneHotEncoder for categorical features
6	Build classification pipeline: model_pipeline ← Pipeline with: - 'preprocessor': preprocessor - 'classifier': LogisticRegression(mixite=1000)
7	Perform cross-validated prediction: predicted probabilities ← cross_val_predict(model pipeline, X, y, cv=5)
8	Detect potential label issues
9	Remove noisy records
10	Save cleaned dataset

Training the models

Feedforward Neural Network Architecture (FFN)

In this study, a Feedforward Neural Network (FNN) was implemented due to its proven capability in modelling complex nonlinear relationships between input features and target labels in structured datasets [17]. Prior to feeding the data into the model, features were standardized using the StandardScaler method, which transforms each feature to have zero mean and unit variance. The network consists of three hidden layers; each composed of fully connected (dense) neurons. The first layer includes 256 neurons, followed by 128 neurons in the second layer and 64 neurons in the third. Each layer applies the Rectified Linear Unit (RLU) activation function to introduce non-linear transformations and enhance model expressiveness. To mitigate overfitting, each dense layer is followed by a Batch Normalization layer and a Dropout layer. Dropout randomly disables a fraction of neurons during training, preventing co-adaptation of features [18,19]. Dropout rates were set to 0.3 for the first two layers and 0.2 for the third, reflecting a decreasing regularization strategy as the network deepens.

The final layer consists of a single neuron with a sigmoid activation function, producing a probability output between 0 and 1. The model was compiled using the binary cross-entropy loss function, which is appropriate for binary classification problems, and optimized using the Adam optimizer, known for its robustness and adaptive learning rate properties [20].

DNN-BiLSTM

This study replicated the hybrid deep learning model that integrates a Deep Neural Network (DNN) with a Bidirectional Long Short-Term Memory (BiLSTM) network as proposed in [8]. For this architecture, the features fed were the raw URL and the set of manually engineered features. These features capture statistical, structural, and lexical characteristics of URLs and domains, such as token counts, entropy values, and the presence of certain keywords or patterns. The target variable is transformed into binary format using label encoding, with "phishing" represented as 1 and "legitimate" as 0. To ensure consistent feature scale and improve convergence speed during training, all numerical features are standardized using z-score normalization.

The raw URL strings, were processed through a character-level tokenizer. Unlike word-level tokenization, which may miss obfuscated character patterns used in phishing attacks, character-level tokenization enables the model to detect subtle manipulations and irregular character sequences [21]. Each URL is converted into a sequence of integers and padded to a fixed length of 75 characters to standardize input dimensions for the sequential model [22].

The architecture is composed of two parallel branches: (i) 1. DNN Branch (Processing Hand-Crafted Features) and (ii) BiLSTM Branch (Processing Character-Level Embeddings). The first branch (DNN branch) begins with a dense layer comprising 64 neurons activated by the ReLU function, followed by a dropout layer with a rate of 0.3 to reduce overfitting. Another layer in the first branch is the second dense layer with 32 neurons to further abstract the learned representations [23].

The second branch (BiLSTM branch) receives raw URL sequences in the form of character indices. These are passed through an embedding layer, which transforms each character index into a 128-dimensional vector, thereby capturing distributed representations of characters based on usage context. The embedded sequences are then fed into a Bidirectional LSTM layer with 64 units. This recurrent layer processes the sequence in both forward and backward directions, enabling the model to capture patterns that may emerge from any part of the URL string.

The outputs of the DNN and BiLSTM branches are combined, creating a unified feature vector that combines high-level semantic signals with low-level sequential patterns. This merged representation is passed through an additional dense layer of 32 neurons with ReLU activation, followed by another dropout layer to further improve generalization. The final layer consists of a single neuron activated by the sigmoid function, producing a probability score for binary classification.

The model is compiled using the binary cross-entropy loss function, appropriate for two-class problems, and trained using the Adam optimizer, known for its adaptive learning rate and computational efficiency [13]. The training process spans 10 epochs with a batch size of 32 and includes a validation split to monitor performance.

In summary, in the DNN–BiLSTM hybrid architecture, the DNN branch learns from hand-crafted semantic features, while the BiLSTM branch processes character-level embedding of URLs to capture syntactic irregularities and sequential patterns.

Semantic NLP

This model builds upon the previous DNN–BiLSTM design by incorporating lexical representations of tokenized URL segments as part of what is here referred to as Semantic NLP features. These Semantic NLP features are extracted from URLs by decomposing them into meaningful tokens using common delimiters such as slashes, dots, and hyphens. The resulting tokens are vectorised using TF-IDF, which measures the relative importance of each term based on its frequency and uniqueness across the corpus. By retaining only the top 100 tokens, the model captures essential lexical indicators of phishing behaviour, such as misleading subdomains, obfuscated brand names, and suspicious keyword combinations.

These semantic features were combined with hand-crafted features to form a rich structured input. This combined feature set is normalized and passed into a deep neural network, which constitutes the DNN branch of the model. The DNN is thus responsible for learning high-level patterns from both quantitative behaviours, represented by hand-crafted features, and semantic word cues from the URL tokens.

In parallel, the model processes the raw URL strings using character-level tokenization on the BiLSTM branch. The outputs of the Semantic NLP + hand-crafted DNN branch and the BiLSTM branch are combined, fused through an additional dense layer, and passed into a sigmoid-activated output neuron for binary classification. The model is trained using binary cross-entropy loss, optimized via Adam.

Transformer Architecture

This model added a URL Transformer encoder to the DNN-BiLSTM architecture. The URL Transformer block defines multi-head self-attention, residual connections, and feed-forward layers, resembling the encoder module of the original Transformer architecture. The sequence is passed through two BiLSTM layers: the first with 128 units returning sequences, and the second with 64 units reducing it to a fixed-length output vector. This two-stage BiLSTM stack captures both local and long-range dependencies in the URL string, a critical advantage when detecting manipulations that occur across various positions in the URL.

The outputs of the DNN branch and the BiLSTM branch are concatenated to form a unified latent representation. This combined vector is passed through a dense layer with 64 GELU-activated neurons, followed by dropout. The final classification is performed using a sigmoid-activated neuron that outputs the probability of the instance being phishing or legitimate. The model is compiled using binary cross-entropy loss and optimized with the Adam optimizer, utilizing a custom exponential decay learning rate schedule. This schedule starts with a learning rate of 0.001 and decays over time, facilitating better convergence during prolonged training. To avoid overfitting and ensure optimal performance on phishing detection (class 1), early stopping is configured to monitor validation recall with a patience of five epochs. Furthermore, class weights are applied to slightly prioritize the minority class, addressing mild label imbalance.

Evaluation

In the context of this study, the evaluation process involves various metrics and techniques to measure the model's predictive accuracy and generalization capabilities.

Precision: Calculates the proportion of true positive predictions among all positive predictions. It's a measure of the model's ability to avoid false positives. The formula is given as:

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

Where TP means True Positive and FP means False Positive

Recall (Sensitivity): Calculates the proportion of true positive predictions among all actual positives. It's a measure of the model's ability to capture all positives. The formula is given as:

$$\mathrm{Re}\,call = \frac{TP}{TP + FN}$$

Where FN means False Negative

F1-Score: Harmonic mean of precision and recall, providing a balance between the two metrics. Useful when the class distribution is imbalanced. The formula is:

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

Runtime. Runtime refers to the total amount of time a model takes to complete a specific task, such as training or inference. In the context of phishing URL detection, runtime is typically measured in seconds (s) and serves as an essential metric for evaluating a model's computational efficiency and practicality.

Results and Discussions

This section presents a comprehensive analysis of experimental results evaluating the performance, computational efficiency, and robustness of four machine learning models across diverse dataset conditions. The investigation focuses on three critical dimensions: (1) model effectiveness on noisy versus cleaned datasets, measured by F1-scores; (2) computational runtime efficiency; and (3) the trade-offs between training time and classification accuracy, particularly in the context of phishing detection.

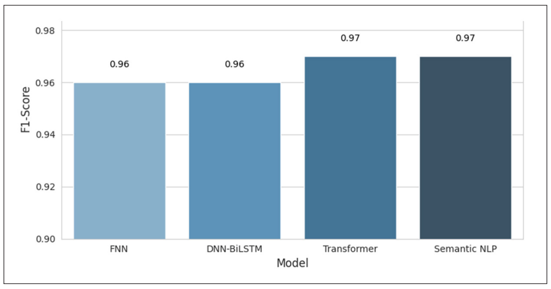


Figure 2: Comparison of FNN, DNN-BiLSTM, Transformer, and Semantic NLP before label noise correction using the F1-score metric

The bar graph, in Figure 2 provides a quantitative comparison of four models: Feedforward Neural Network (FNN), Deep Neural Network with Bidirectional Long Short-Term Memory (DNN-BiLSTM), Transformer, and Semantic NLP, using the F1-score metric. The results reveal distinct disparities in the models' ability to handle noise, with scores as follows: FNN: 0.96, DNN-BiLSTM: 0.96, Transformer: 0.96, and Semantic NLP: 0.97.

Semantic NLP emerged as the top-performing model, achieving an F1-score of 0.97. This superior performance can be attributed to its focus on contextual and semantic understanding, which enables it to discern meaningful patterns despite noise. By leveraging advanced linguistic features, Semantic NLP mitigates the distortions introduced by noisy data, making it particularly effective for tasks requiring nuanced interpretation.

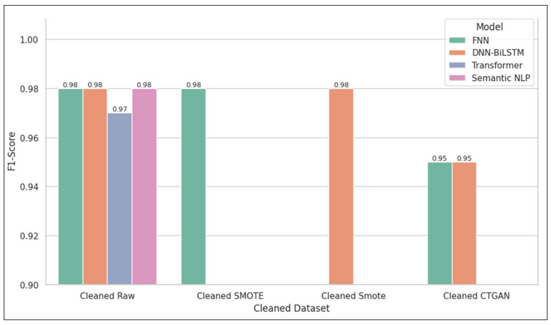


Figure 3: Comparison of FNN, DNN-BiLSTM, Transformer, and Semantic NLP after label noise correction on three datasets: (i) Raw dataset, (ii) Dataset balanced with SMOTE, and (iii) Dataset augmented with CTGA- using the F1-score metric

The graph in Figure 3 shows that the FNN consistently outperformed other models across all cleaned datasets, achieving the highest score (0.98) on the dataset with label noise corrected (Cleaned Raw). This suggests that for cleaned datasets, simpler architectures may sometimes outperform more complex ones. DNN-BiLSTM maintained strong performance, closely following FNN, indicating that its bidirectional processing remains effective even with cleaned data. The Transformer showed moderate performance, while Semantic NLP, despite its sophistication, ranked lowest in this comparison. The FNN's strong performance challenges the common assumption that complex models always outperform simpler ones.

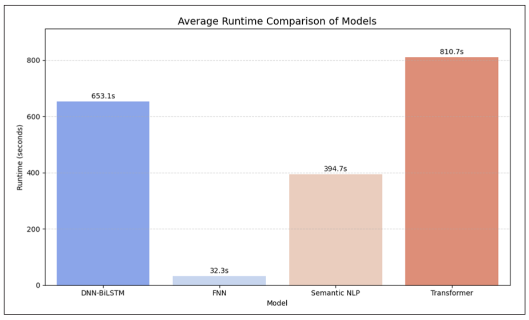


Figure 4: Comparison of computational efficiency of FNN, DNN-BiLSTM, Transformer, and Semantic NLP after label correction

The bar graph in Figure 4 shows the following results: DNN-BiLSTM: 653.1 seconds, FNN: 39.47 seconds, Semantic NLP: 32.3 seconds, Transformer: 810.7 seconds. Thus, FNN Semantic NLP, despite its advanced semantic processing capabilities, also demonstrates impressive efficiency. In contrast, DNN-BiLSTM and Transformer exhibited significantly higher runtimes (653.1 seconds and 810.7 seconds, respectively). The DNN-BiLSTM's bidirectional recurrent layers require sequential processing of data in both forward and backward directions, leading to increased computational time. The Transformer, while powerful, suffers from the quadratic complexity of its self-attention mechanisms, particularly for longer input sequences, which explains its status as the slowest model in this comparison.

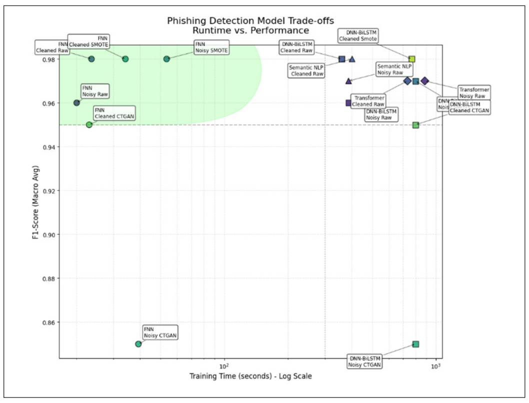


Figure 5: The trade-offs between model training time and classification performance for various phishing detection models and datasets

Figure 5 presents the trade-offs between model training time and classification performance for various phishing detection models and datasets. The x-axis, scaled logarithmically, quantifies the training duration in seconds, reflecting computational resource allocation. The y-axis represents the F1-Score, a robust metric for evaluating classification models, particularly in scenarios with imbalanced class distributions, as it harmonically averages precision and recall. The shaded light green region at the upper-left quadrant of the plot signifies the optimal zone. This area delineates models that concurrently achieve high F1-Scores (generally exceeding 0.95) and relatively expedient training times (approximately below 200 seconds). This zone represents the ideal balance between predictive accuracy and computational efficiency.

Notably, several FNN configurations, especially those trained on Cleaned Raw, Cleaned SMOTE, and Noisy SMOTE datasets, are situated within or in close proximity to this optimal zone. These models consistently yielded F1-Scores above 0.97 with training durations significantly less than 100 seconds. This finding underscores the potential of FNNs, when coupled with Confidence Learning, to serve as highly efficient and effective solutions for phishing detection. Models such as DNN-BiLSTM Cleaned Raw and Semantic NLP Noisy Raw demonstrate commendable F1-Scores (approaching 0.97-0.98), indicating strong classification capabilities. However, their placement further to the right on the x-axis reveals substantially longer training times, extending into several hundred seconds. This suggests that while these models offer high accuracy, their computational demands for training may be considerable.

Conversely, certain model-dataset combinations exhibit suboptimal F1-Scores. For instance, FNN Noisy CTGAN yields a very low F1-Score (around 0.85) despite a moderate training time. Even more critically, DNN-BiLSTM Noisy CTGAN not only demonstrates poor performance but also incurs an exceedingly long training time, approaching 1000 seconds. Such instances highlight the critical interplay between model choice, data quality, and the efficacy of augmentation techniques.

Discussions

The effectiveness of CleanLab as a noise-removal tool becomes evident when evaluating its impact across various models and datasets, particularly on FNN and the replicated DNN-BiLSTM architecture. For instance, the FNN trained on noisy raw data achieved a baseline F1-score of 0.96. After applying CleanLab to this same dataset, the accuracy improved to 0.98, marking a 2% increase. This improvement demonstrates the value of label noise correction. This enhancement also becomes more notable when compared to the replicated DNN-BiLSTM architecture, which achieved 0.97 on the same cleaned dataset. Despite the architectural sophistication of the hybrid model, the cleaned FNN still outperformed it by a margin of 1%, indicating that data quality can rival-or even outweigh complex model when it comes to performance gains.

An even more dramatic effect of label cleaning is observed on synthetically augmented datasets, particularly those generated using CTGAN. CTGAN is known to introduce a high degree of variability, and occasionally noise, due to its generative adversarial nature. When FNN was trained on the raw CTGAN-generated dataset, it recorded a poor performance of 0.85, highlighting the detrimental effect of training on synthetic samples with noisy or ambiguous labels. However, after the data was cleaned using CleanLab, the model's performance surged to 0.95, reflecting a 10% absolute improvement. This underscores the critical importance of post-synthetic label verification when using generative data augmentation strategies.

Similarly, the replicated DNN-BiLSTM architecture showed improvement when trained on SMOTE-augmented data. On raw SMOTE data, it achieved an F1-score of 0.97. After applying CleanLab, this score nudged upward to 0.98. While the gain here is modest (1%), it confirms that even high-quality oversampling methods like SMOTE can benefit from noise filtering, especially by removing subtle outliers or mislabelled data that may otherwise misguide gradient updates during training.

Conclusion

This study demonstrates that a simple Feedforward Neural Network (FNN), when trained on a dataset with label noise corrected using the Cleanlab framework, can achieve performance comparable to or surpassing more complex deep learning architectures, such as the hybridized Deep Neural Network and Bidirectional Long Short-Term Memory (DNN-BiLSTM), DNN-BiLSTM with a transformer, and DNN-BiLSTM with semantic Natural Language Processing (NLP) features, in the task of phishing URL detection. The FNN achieved an impressive F1-score of 0.98 on the cleaned raw dataset, outperforming the more sophisticated models while requiring significantly lower computational resources, with a training time of approximately 39.47 seconds compared to over 600 seconds for DNN-BiLSTM and 810.7 seconds for the transformer-based model. The application of Cleanlab for label noise correction proved critical, enhancing model performance by up to 10% on synthetically augmented datasets like those generated by CTGAN. These findings challenge the prevailing trend of pursuing increasingly complex architectures for marginal performance gains, highlighting the importance of data quality over model complexity. Future research could explore the generalizability of these findings across diverse datasets and cybersecurity tasks.

References

1. Alsariera YA, Balogun AO, Adeyemo VE, Tarawneh OH, Mojeed HA. Intelligent tree-based ensemble approaches for phishing website detection. Jestec.Taylors.Edu.MyYA Alsariera, AO Balogun, VE Adeyemo, OH Tarawneh, HA MojeedJ. Eng. Sci. Technol. 2022. 17: 563-582.
2. Ahammad SH, Kale SD, Upadhye GD, Pande SD, Babu EV, et al. Phishing URL detection using machine learning methods. Advances in Engineering Software. 2022.
3. Balogun AO, Mojeed HA, Adewole KS, Akintola AG, Salihu SA, et al. Optimized decision forest for website phishing detection. InProceedings of the Computational Methods in Systems and Software Cham: Springer International Publishing. 2021. 568-582.
4. Bahnsen AC, Bohorquez EC, Villegas S, Vargas J, Gonzalez FA. Classifying phishing URLs using recurrent neural networks. ECrime Researchers Summit, ECrime. 2017.
5. Aljofey A, Jiang Q, Qu Q, Huang M, Niyigena JP. An effective phishing detection model based on character level convolutional neural network from URL. Electronics (Switzerland). 2020.
6. Liu DJ, Geng GG, Zhang, XC. Multi-scale semantic deep fusion models for phishing website detection. Expert Systems with Applications. 2022. 209: 118305.
7. Ballesteros M, Dyer C, Smith NA. Improved transition-based parsing by modeling characters instead of words with LSTMs. Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing. 2015.
8. Ozcan A, Catal C, Donmez E, Senturk B. A hybrid DNN–LSTM model for detecting phishing URLs. Neural Computing and Applications. 2023. 35: 4957-4973.
9. Northcutt C, Jiang L, Chuang I. Confident learning: Estimating uncertainty in dataset labels. Journal of Artificial Intelligence Research. 2021. 70: 1373-411.
10. Zhang M, Gao J, Lyu Z, Zhao W, Wang Q, et al. Characterizing label errors: confident learning for noisy-labeled image segmentation. InInternational conference on medical image computing and computer-assisted intervention. Cham: Springer International Publishing. 2020. 29: 721-730.
11. Opara C, Chen Y, Wei B. Look before you leap: Detecting phishing web pages by exploiting raw URL and HTML characteristics. Expert Systems with Applications. 2024. 236.
12. Abdelhakim H, Salima Y. Web page phishing detection. Engineering Applications of Artificial Intelligence. 2021.
13. Elreedy D, Atiya AF. A Comprehensive Analysis of Synthetic Minority Oversampling Technique (SMOTE) for handling class imbalance. Information Sciences. 2019. 505: 32-64.
14. Xu L, Skoularidou M, Cuesta-Infante A, Veeramachaneni K. Modeling tabular data using conditional GAN. Advances in Neural Information Processing Systems. 2019. 32.
15. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, et al. Generative adversarial networks. Communications of the ACM. 2020. 63: 139-144.
16. Northcutt CG, Athalye A, Mueller J. Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks. Advances in Neural Information Processing Systems. 2021.
17. Lecun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015. 521: 436-444.
18. Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 32nd International Conference on Machine Learning, ICML. 2015. 1: 448-456.
19. SrivastavaNitish, HintonGeoffrey, KrizhevskyAlex, SutskeverIlya, SalakhutdinovRuslan. Dropout. The Journal of Machine Learning Research. 2014. 15: 1929-1958.
20. Kingma DP, Ba JL. Adam: A method for stochastic optimization. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. 2015.
21. Correa Bahnsen A. DeepPhish Simulating Malicious AI. Proceedings of the Symposium on Electronic Crime Research, San Diego, CA, USA. 2018. 1-8.

22. Karim A, Shahroz M, Mustofa K, Belhaouari SB, Joga SR. Phishing detection system through hybrid machine learning based on URL. IEEE Access. 2023. 11: 36805-36822.
23. Zhu E, Ju Y, Chen Z, Liu F, Fang X. DTOF-ANN: an artificial neural network phishing detection model based on decision tree and optimal features. Applied Soft Computing. 2020. 95: 106505.