

Email Phishing Detection Using NLP and Machine Learning

Norah Aljurbua¹, Asma A. Alhashmi², Abdulbasit A. Darem^{3*}

Department of Computer Science, College of Science, Northern Border University, Arar 73213, Saudi Arabia^{1,2}

Center for Scientific Research and Entrepreneurship, Northern Border University, Arar 73213, Saudi Arabia³

Abstract—Phishing emails imitate legitimate correspondence and evolve to evade rule-based filters, yet many reported detection gains rest on single data splits, untested differences, and unexamined benchmark artifacts. This study compares TF-IDF classifiers (Logistic Regression, Decision Tree, SGD, and XGBoost) with recurrent networks (Simple RNN, GRU, Bi-LSTM, and Bi-LSTM with attention) on 17,085 deduplicated emails from the Phishing Email Detection dataset, using one leakage-controlled pipeline and paired significance tests. SGD achieved the highest held-out F1-score (97.33%; ROC-AUC 0.9979) and made significantly fewer errors than Logistic Regression, Bi-LSTM, and the attention model (McNemar, Holm-adjusted $p \leq 0.012$). Under stratified cross-validation, however, the five leading models were statistically indistinguishable, and tuned linear models reached the highest mean F1-score (97.2%). A three-seed ablation found no gain from attention over parameter-free pooling. Provenance analysis showed that corpus-identifying tokens inflate scores: removing them lowered classical F1-scores by up to 3.5 points, and holding out Enron-like emails raised SGD’s false-positive rate on them from 0.5% to 3.8%. Under nine perturbations of 2,000 held-out emails, character-level edits altered fewer than 1.2% of SGD, GRU, and Bi-LSTM detections, whereas appended benign text evaded 10–53% of detections for every model except the Simple RNN (recall 14.9%); a heuristic context-aware boost did not mitigate this. A two-stage cascade escalating 5.6% of emails from SGD to the attention model achieved 98.52% recall at a 2.77% false-positive rate and 0.49 ms median latency, significantly outperforming the attention model alone while matching SGD alone. Well-tuned linear TF-IDF models therefore remain highly competitive.

Keywords—Phishing detection; natural language processing; TF-IDF; recurrent neural networks; bidirectional LSTM; attention mechanism; statistical significance testing; dataset provenance; email security

I. INTRODUCTION

Email is one of the most widely used means of communication among individuals and organizations, which makes it a primary target of phishing, a form of social engineering. In phishing campaigns, attackers send deceptive emails that closely resemble genuine ones to trick recipients into revealing sensitive information, such as passwords, or into performing other harmful actions. Phishing remains one of the most persistent cybersecurity threats [1], and large language models (LLMs) escalate it further: they can readily produce persuasive, context-appropriate, human-like text, which helps attackers bypass fixed-rule defenses and makes machine-learning-based detection harder [2]. Detection therefore

involves a trade-off: stricter filtering reduces the chance of missing a fraudulent message but increases the chance of wrongly flagging a legitimate one, which carries its own operational cost. Managing this trade-off against sophisticated and constantly changing attacks is one of the foremost challenges in cybersecurity [3].

From a machine-learning perspective, phishing email detection is a supervised binary text-classification task. A major obstacle to building reliable classifiers is the scarcity of high-quality labeled email datasets: real-world corpora cannot be shared publicly without sanitization because of privacy and corporate restrictions, which reduces corpus diversity and limits the ability of models to generalize [4]. These restrictions call for data-efficient methods evaluated with metrics that reflect the asymmetric cost of misclassification, because missing an attack (a false negative) is usually far more serious than flagging a legitimate message (a false positive).

Natural language processing (NLP) provides the core techniques for filtering phishing attempts by their message content: it converts raw email text into numerical representations that a supervised classifier can process. Term frequency–inverse document frequency (TF-IDF) builds sparse, high-dimensional vectors in which each term is weighted by its frequency in a message and its rarity across the corpus. It is computationally inexpensive yet effective and has achieved results competitive with the state of the art in phishing detection [5]. Its main limitation is that it treats a message as an unordered collection of terms and therefore ignores word order and the sequential dependencies that may signal deceptive intent. Recurrent architectures, such as recurrent neural networks (RNNs), gated recurrent units (GRUs), long short-term memory networks (LSTMs), and their bidirectional variants, were introduced to model these dependencies; bidirectional variants combine forward and backward hidden states, which can improve the modeling of deceptive language [3]. Although both model families are mature, to the best of our knowledge, direct comparisons in which every model undergoes the same feature extraction and evaluation procedure remain limited, and many studies report accuracy alone without the false-negative rate, which makes cross-study comparisons unreliable [6].

This study compares classical machine-learning models with sequential deep-learning models for phishing email detection through a single, repeatable experimental procedure. The classical branch trains Logistic Regression, a Decision Tree, an SGD classifier, and XGBoost on TF-IDF features; the deep-learning branch trains a Simple RNN, a GRU, a Bi-

*Corresponding author.

LSTM, and the same Bi-LSTM with an attention layer on tokenized, padded sequences. Both branches use the same preprocessed dataset, the same stratified train-test split, and the same phishing-positive metrics, and the resulting differences are tested statistically.

The contributions of this work are empirical rather than architectural; no new neural architecture is proposed. First, it provides a leakage-controlled comparison of four TF-IDF classifiers and four recurrent networks on a cleaned and deduplicated benchmark, and it goes beyond single-split reporting by testing the resulting ranking with cross-validation, bootstrap confidence intervals, and paired significance tests with multiple-comparison correction; these tests show that a linear SGD classifier makes significantly fewer errors than three of the four strongest alternatives on the held-out split. Second, it quantifies a corpus-provenance shortcut in this widely used dataset: corpus-identifying tokens alone predict the class with an ROC-AUC of up to 0.85, deleting them lowers classical F1-scores by up to 3.5 points, and holding Enron-like emails out of training raises the false-positive rate on them sevenfold, which qualifies the near-perfect scores reported on this benchmark. Third, a multi-seed pooling ablation shows that the attention layer adds no measurable benefit over parameter-free pooling, and that the published attention layer places about one fifth of its weight on padding. Fourth, it introduces a reproducible benchmark of 2,000 held-out emails under nine synthetic perturbations, which shows that all models resist character-level edits but are evaded by appended benign text, and that a heuristic context-aware boost does not mitigate this. Fifth, it implements and evaluates end-to-end a two-stage cascade that escalates only 5.6% of the emails from an SGD filter to a Bi-LSTM with attention, significantly reducing the errors of the attention model at a median latency of 0.49 ms, and it profiles the latency and throughput of all models. The code, configurations, split indices, and result tables are released for independent verification.

The remainder of this study is organized as follows. Section II reviews related studies on phishing email detection with NLP. Section III describes the dataset, the preprocessing, the models, and the evaluation metrics. Section IV reports the held-out results, the statistical validation, the corpus provenance and attention analyses, the adversarial benchmark, the cascade evaluation, the latency and throughput profile, and the interactive demonstrator (Section IV-K). Section V discusses the findings, the comparison with previous work, and the limitations of the study. Section VI concludes the study and outlines future research.

II. RELATED WORK

Several peer-reviewed research papers have tackled phishing email detection as a key security classification task. One thing is clear: defenders have to balance making things user-friendly with keeping the risk level low because attackers' tactics are constantly changing. The phishing detection techniques mentioned in the literature range from simple lexical methods such as TF-IDF combined with classical classification models to sequential and contextual deep learning models. In addition, certain works also discuss

practical implementation issues faced when deploying these systems in real-world settings.

In phishing detection, there have been several systematic studies that compare model families and analyze their performance landscape. Alhuzali et al. [6] conducted a systematic evaluation of the performance of different machine learning and deep learning models on multiple benchmark datasets. Based on their evaluation, they suggested that no single model family is consistently better than the rest; performance depends heavily on the selected dataset and, to some extent, on the preprocessing methods used. In another related study, Altwaijry et al. [3] focused mainly on the comparison and evaluation of different types of deep learning architectures (such as recurrent layers and convolutions). The results of the research indicated that bidirectional and attention-integrated models were better at contextual encoding and hence could provide a better representation of a given input than unidirectional recurrent models. However, they also concluded that although some methods showed improvements, these were not always substantial compared to well-tuned baseline models. Gupta et al. [7] compared classical approaches with a set of more advanced NLP and transformer-based approaches like RoBERTa and GPT-4 that have recently performed strongly on NLP benchmarks. Their study revealed that these newer models could outperform classical approaches, but their review also pointed out that they had combined results from a number of different studies with different experimental protocols, setups, and data, making it difficult to fully interpret the comparative analysis of these works.

Many studies have demonstrated the effectiveness of TF-IDF-based classical machine learning in phishing detection. Fares et al. [5] trained a collection of models such as SVM, LSTM, Random Forest, and Naive Bayes variants using TF-IDF-based features extracted from the Kaggle Phishing Email Detection dataset, which was also used in our research. Among those models, SVM was the top performer in terms of accuracy. The work further highlighted runtime as an important operational consideration, noting that the effectiveness of detection should not be measured solely based on accuracy. Rabbi et al. [8] conducted a performance comparison of six classification algorithms on the TF-IDF feature vectors extracted from the Ling and TREC publicly available email corpora; Random Forest became the top performer on the much bigger TREC dataset, and the authors explicitly prioritized minimizing false negatives, while explaining that including email subject lines led to higher accuracy on the more varied corpora. Alkhodhairy and Saleem [9] applied an NLP preprocessing pipeline with an optimized SVC technique on the Kaggle spam classification dataset and obtained high scores in terms of precision and recall with just a small number of misclassifications, outperforming Decision Tree, Neural Network, and Naive Bayes. Al-Subaiey et al. [10] consolidated six publicly available email corpora into over 82,000 messages, trained SVM with TF-IDF as well as Multinomial Naive Bayes and Random Forest, and deployed the best model with LIME-based explanations. This addressed the gap between benchmark accuracy and interpretable real-user deployment. Meléndez et al. [11] performed an experimental study on more than 119,000 emails collected from different sources. They

compared classical machine learning models, such as Logistic Regression, Random Forest, SVM, and Naive Bayes, with modern transformer-based architectures such as RoBERTa, BERT, XLNet, and ALBERT. In their results, the transformer models consistently outperformed classical baselines and made substantially fewer errors on average. Sahit et al. [12] compared Logistic Regression, SVM, Naive Bayes, Random Forest, and a voting ensemble using engineered link, tag, content, and behavioral features. Random Forest and ExtraTrees scored the highest accuracy in the experiment. Mohammed et al. [13] created a majority-voting ensemble system that they trained on TF-IDF features after feature selection using mRMR and chi-square. They reported strong accuracy and ROC-AUC figures and deployed the system for real-time inference.

Research has also focused on expanding detection beyond TF-IDF by including lexical, structural, and metadata features. Zhang et al. [4] gathered more than 660,000 emails and extracted 79 static features from the email headers and bodies alongside TF-IDF and Word2Vec representations. TF-IDF with a Random Forest classifier achieved very high accuracy and an extremely low false-positive rate, showing that with careful feature selection, very high performance can be achieved if the training set is diverse enough. Franchina et al. [14] combined offline email features, including sender domain indicators, suspicious URLs, and misspellings, with a text-mining pipeline to build a risk-based model. Through this model, each email gets a risk score ranging from 0 to 10 based on the features in the message. In their case, the system was highly effective, achieving high accuracy and a low false-positive rate on a dataset of more than 4,000 emails. Naswir et al. [15] analyzed email-based and human-behavior features on the IWSPA-AP 2018 and PECORP datasets and concluded that combined feature sets performed better than either category alone, with URL length and misspelling count being among the most consistently important features. Somesha and Pais [16] conducted a comparison of TF-IDF, Word2Vec, and FastText using five different classifiers. Among them, FastText-CBOW combined with Random Forest performed the best in terms of accuracy and also had a very low false positive rate. Mittal et al. [17] proposed the DARTH framework, which uses BERT-based analysis of the email body text coupled with URL and metadata neural networks in an ensemble. This ensemble was trained using a dataset of over 150,000 emails, and it achieved very high accuracy and precision. Naik et al. [18] utilized a BERT-based text classifier along with a graph component encoding communication links and domain reputation. This hybrid model not only produced higher accuracy but also a lower false positive rate than the text-only baselines on a dataset of over 18,000 emails. Jeevitha et al. [19] developed PhishDefender-X as a multi-feature phishing detection system by combining TF-IDF, Word2Vec, BERT, graph analysis, blockchain, and federated learning. However, due to the lack of exact numerical performance metrics, it is difficult to reliably evaluate the approach.

Sequential models have been the focus of considerable research as they are good at representing word order, which is a very important factor in understanding the message. Atawneh and Aljehani [20] experimented with CNN, RNN, LSTM, and

BERT-LSTM models on public phishing datasets and found that the BERT-LSTM hybrid performed best in terms of accuracy and precision. They argued that BERT's main strength lies in modeling context in both directions, which regular RNN models do not support. Pimpason et al. [21] ran experiments with LSTM, GRU, BERT, CNN, and RNN on a Figshare corpus (a collection of over 100,000 emails). LSTM was the best performer overall, but a simple RNN gave the lowest precision, which is consistent with the well-known vanishing-gradient problem. Shrivastava et al. [22] integrated a BERT encoder, a Bi-LSTM layer, and a metadata fusion component that drew on sender domain reputation, hyperlink patterns, and header anomalies. Their model outperformed the Naive Bayes, TF-IDF, and Random Forest baselines on all reported metrics on the Enron corpus while maintaining an average inference latency of less than 400 milliseconds per email.

Transformer-based and operational approaches have added new dimensions to the discourse. Ugwunna et al. [23] conducted experiments involving fine-tuned BERT and zero-shot GPT-4o versus classical baselines on the Enron Spam dataset, with the fine-tuned BERT model coming out on top in terms of combined performance. The cross-dataset experiments on three other corpora showed a great deal of performance variation, which highlights the risks of relying on a single benchmark for evaluation. Embedding detection in an enterprise anti-phishing workflow, Gallo et al. [24] used classifiers trained on over 22,000 reported emails with composite content and security context features. Random Forest, with 36 selected features, achieved the best precision and recall, and these results were validated through large-scale internal deployment. Kumar et al. [25] created a GDPR-compliant inbox-level framework using the Gmail API and a feedback retraining loop, reporting approximately 92% accuracy in real-time tests. Ahmed et al. [26] developed a hybrid NLP and RNN/LSTM model for Arabic phishing detection on a dataset of nearly 100 messages. Their model outperformed the Random Forest and kNN baseline models. However, the very small size of the dataset limits the generalizability of the results. Himdi and Alhayan [27] compiled the Arabic_Fraud_Phishing dataset consisting of 1,000 emails. Subsequently, they used TF-IDF-based classifiers and stacking ensembles. The full stacking meta-classifier performed best in terms of accuracy, and this was validated by testing on a second corpus of Arabic phishing emails. Lappas et al. [28] developed a Bayesian network training module using 164 phishing and 200 legitimate emails. This module provides probability-based detection insights within a cybersecurity education context.

The emergence of LLM-generated phishing messages has introduced new challenges in phishing detection. For example, Han et al. [2] introduced Phish-Master, which works through chain-of-thought prompting, metaprompt refinement, and iterative LLM generation with SpamAssassin feedback. Phish-Master succeeded in producing high-quality phishing messages. TF-IDF-trained Random Forest classifiers achieved high accuracy on LLM-generated test sets, and the generated messages achieved a high delivery rate in real campus phishing simulations, which clearly illustrates both the severity of the

threat and the potential of existing classifiers when representative training data are available. Research by Eze and Shamir [29] explored text analysis methodologies on 865 phishing emails generated by AI. They concluded that current detection systems can maintain high performance if their training data also contain AI-generated content, which implies that core textual features remain useful. Weinz et al. [30] carried out phishing simulations across three companies involving more than 71,000 emails. Their results showed that OSINT-assisted LLM-generated messages led to substantially higher credential-submission rates than conventional phishing, illustrating the real-world growth of AI-assisted social engineering attacks.

The analyses reveal that both 1) classical machine learning based on TF-IDF and 2) sequence-based deep learning work very well on standard benchmarks. For instance, many studies only test one model family or one type of data representation, so there is no controlled basis for comparing statistical and contextual approaches under the same conditions [3], [6]. Some of them use very small or non-public datasets, which limit external validity [26], [28]. Others use URL, header, and sender features, which are easily manipulated by attackers [12], [14]. Besides, accuracy is most often reported, whereas false-negative behavior, the most operationally critical dimension in a security context, is often underreported [8], [11]. Different preprocessing procedures, class distributions, and train-test partitioning strategies make cross-study comparisons unreliable [3], [5]. Transformer and LLM-based approaches [7], [23], despite their high performance, have substantial computational costs, and the growing prevalence of AI-generated phishing challenges systems trained on fixed-language corpora [2], [29].

The present study tackles the most immediately relevant of these gaps by implementing TF-IDF-based classical machine learning models and sequential deep learning architectures within a single unified pipeline that shares the same dataset, preprocessing procedure, evaluation metrics, and train-test

partition. Classical models are kept as efficient, interpretable, and competitive baselines [5], [9], whereas sequential and attention-based architectures are added to assess the value of contextual modeling for phishing language patterns such as credential-request sequences and urgency-action phrasing [3], [22]. The controlled experimental design provides evidence that can be directly interpreted for comparative analysis. It will help to strengthen phishing email detection systems that are not only effective but also practical for real-world applications.

III. METHODOLOGY

This section describes the detection pipeline and the framework used to evaluate it. The aim is a single pipeline that compares classical machine-learning and sequential deep-learning models without introducing confounding differences in data processing. Fig. 1 shows the workflow, which has three stages: dataset preparation (acquisition and preprocessing), model development (feature representation and training of both model families), and evaluation with a common set of metrics. The preprocessing block lists the cleaning operations, whose exact order is given in Section III-C: case normalization, mapping of URLs and email addresses to placeholder tokens, removal of non-alphabetic characters, stop-word and short-token removal, lemmatization, and whitespace normalization. The model-development block separates the four TF-IDF classifiers from the four sequential networks, among which the plain bidirectional LSTM and the attention model are two separately trained variants of the same bidirectional backbone. The evaluation block lists the metrics used throughout: accuracy, precision, recall, F1-score, and ROC-AUC, complemented by the false-positive rate. Beyond these held-out metrics, the evaluation includes cross-validation with significance testing, a corpus-provenance audit, an attention ablation, a perturbation benchmark, a cascade evaluation, and a latency profile, whose protocols are described with the corresponding results in Section IV.

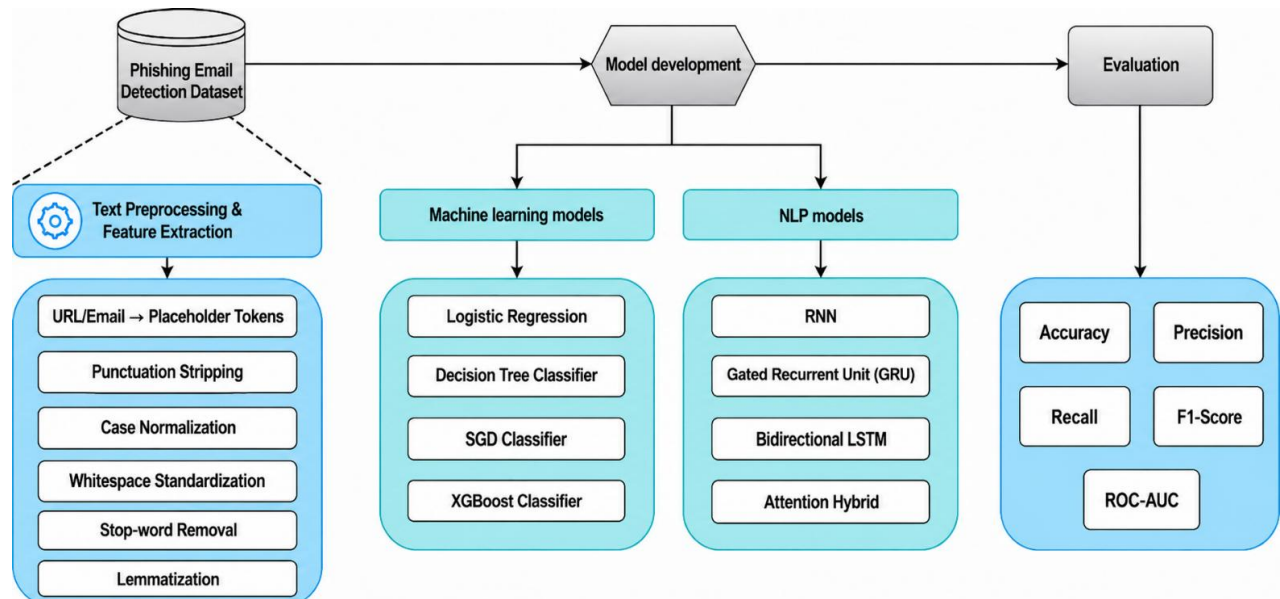


Fig. 1. Overall workflow of the proposed phishing email detection framework.

A. Overall Design

The study follows a quantitative experimental design. Raw email text is first cleaned and then transformed into two representations, a statistical one and a sequential one, so that each model family receives the input for which it is designed: classical classifiers operate on TF-IDF vectors, whereas recurrent and attention-based networks process tokenized, padded word sequences. Both families use the same cleaned corpus, the same stratified train-test split, and the same metrics. Because every pipeline stage is fixed across models, differences in the results reflect the models rather than data preparation; because a single split can still favor one model by chance, the held-out comparison is complemented by cross-validation and paired significance tests (Section IV-E).

B. Dataset Description and Acquisition

This study uses the Kaggle Phishing Email Detection dataset (subhajournal/phishingemails) [31], which consists of raw plain-text email bodies labeled "Safe Email" or "Phishing Email". The original file contains 18,650 messages. Data profiling was followed by data cleaning: records with an empty text field (16) and exact duplicate records (1,096) were removed to prevent the models from memorizing repeated text, leaving 17,538 unique samples. The file contains only the email text and its label and provides no field that identifies the source corpus of a message, so the provenance of individual emails cannot be recovered from metadata.

In the cleaned corpus, legitimate emails account for 62.4% of the messages (10,661) and phishing emails for 37.6% (6,424). Although the classes are not balanced, both are well represented, so the classifiers are less likely to be dominated by the majority class. The imbalance nevertheless means that accuracy alone can be misleading: a classifier that labeled every email as safe would reach about 62% accuracy while missing every attack.

C. Data Preparation and Preprocessing

Model quality depends strongly on the quality of the input text. The raw corpus was therefore processed by a pipeline designed to remove noise while retaining the cues that distinguish phishing from legitimate email; rather than deleting everything that is not a plain word, it deliberately preserves structural clues that phishing messages often contain. All text was first converted to lowercase, so that "URGENT" and "urgent" map to the same token. URLs and email addresses were then replaced with the URL and the email_token placeholders instead of being deleted, because the presence of a link or an address can itself be informative, and mapping every URL to a single token preserves this signal without flooding the vocabulary with unique strings; because non-alphabetic characters are removed in the next step, these placeholders appear in the vocabulary as urltoken and emailtoken. The text was then split into tokens; English stop words and tokens shorter than three characters were discarded, and the NLTK WordNet lemmatizer, applied with its default noun part of speech, reduced plural forms such as "accounts" and "passwords" to "account" and "password" [32]. Finally, repeated whitespace collapsed into a single space.

Records that became empty after cleaning were removed, as were messages whose cleaned text duplicated that of another message. This second deduplication step discarded a further 451 records; it matters because messages that differ only in punctuation or cause collapse onto the same cleaned string and could otherwise appear on both sides of the split. After it, the exact overlap between the training and test partitions was 0.00%, and the procedure left 17,085 clean samples. Only one cleaned text was shared by two raw emails with conflicting labels, and the first label was kept. Near-duplicates are not removed by this step: measured by TF-IDF cosine similarity, 4.0% of the test emails have a training email with a similarity of at least 0.99, and 24.5% have one with a similarity of at least 0.80. Overall, the pipeline removed about 57.6% of the original tokens.

D. Exploratory Data Analysis

To verify that the processed text retained discriminative content, a brief frequency analysis was performed. Fig. 2 shows the word clouds of the two classes, with words sized by frequency. Phishing messages are dominated by generic and transactional words such as "email", "address", and "company", the URL placeholder, and mailing-list vocabulary, whereas safe messages are dominated by conversational and professional terms. The visible difference supports the use of content-based features; the mailing-list vocabulary is revisited in Section IV-F.



Fig. 2. Word clouds of the most frequent terms in phishing (a) and safe (b) emails.

E. Feature Representation

After cleaning, the texts were converted into two numerical representations, one for each model family. For the classical models, TF-IDF vectors were computed with unigram features, the default scikit-learn weighting (smoothed inverse document frequency and L2 normalization), and a vocabulary limited to the 10,000 most frequent terms of the training split, on which the vectorizer was fitted. TF-IDF weighs each term by its frequency in a message relative to how common it is across the corpus, so ubiquitous terms receive low weights and rarer, more specific terms receive higher weights; it does not use the class labels. Each email is thus represented as a sparse, high-dimensional vector that records which terms are present and how informative they are, but not their order [5]. For the deep-learning models, by contrast, each email was encoded as an ordered sequence of integer word indices. This encoding preserves word order, which allows the recurrent and attention-based networks to model how deceptive phrasing unfolds across a message [3].

F. Data Partitioning

The dataset was divided into a training set (80%) and a test set (20%) with a stratified split, which preserves the class

distribution in both subsets. The split uses a fixed random seed (42), so every model is trained and tested on identical partitions: the training set contains 13,668 messages (5,139 phishing and 8,529 safe), and the test set contains 3,417 messages (1,285 phishing and 2,132 safe). The two labels were converted to integers with a label encoder, which assigns codes in alphabetical order, so "phishing" received the code 0 and "safe" the code 1. Because this ordering would make the safe class the default positive class in most toolkits, every precision, recall, F1-score, and ROC-AUC value in this study is computed with the phishing class set explicitly as the positive class, so recall always denotes the proportion of phishing messages detected. Probabilities follow the same convention: for the scikit-learn models, the phishing probability is read from the column of the phishing class, whereas the sigmoid output of each neural network estimates the probability of the encoded class 1, so the phishing probability is computed as $P(\text{phishing}) = 1 - P(\text{safe})$. Assertions in the code check the label mapping and this conversion, and every value in Table I was recomputed from the stored test-set predictions.

G. Machine Learning Models

The four classical classifiers were trained on TF-IDF features. They represent the main types of statistical text classification, from a simple linear model to a gradient-boosted ensemble, and thus show the strengths and weaknesses of feature-based learning before the sequence models are introduced.

1) *Logistic regression*: Logistic regression serves as the linear baseline [6]. Each email is represented by its TF-IDF vector, and the model computes a weighted sum of the features and passes it through the logistic (sigmoid) function to obtain a class probability. Logistic regression assigns interpretable weights to terms and has a low computational cost, which makes it well suited to very large, sparse text feature spaces; it also indicates whether a single linear boundary suffices for the task or whether more complex models are needed. Eq. (1) gives the probability that the model assigns to the encoded class $y = 1$.

$$P(y = 1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}} \quad (1)$$

where, $\mathbf{x}$ is the TF-IDF feature vector, $\mathbf{w}$ is the learned weight vector, b is the bias term, and σ is the logistic function, which maps the score to a probability between zero and one. Because the label encoder assigns $y = 1$ to the safe class (Section III-F), the phishing probability is $P(\text{phishing} | \mathbf{x}) = 1 - P(y = 1 | \mathbf{x})$. The model uses the liblinear solver with L2 regularization and $C = 1.0$.

2) *Decision tree*: The Decision Tree offers a nonlinear, rule-based alternative [6]. Instead of a single global boundary, it classifies messages through a sequence of threshold rules on individual TF-IDF features, which makes its decisions easy to inspect. A drawback is that a tree grown too deep can memorize the training data, particularly with sparse text features. Each split is chosen to reduce node impurity, measured here by the entropy in Eq. (2), and the tree is grown without a depth limit.

$$H(t) = -\sum_{i=1}^C p_i \log_2 p_i \quad (2)$$

where, p_i is the proportion of samples of class i at node t , and C is the number of classes; a lower value means a purer node, and each split is chosen to maximize the resulting reduction in entropy (information gain).

3) *SGD classifier*: The Stochastic Gradient Descent (SGD) classifier is a linear model trained incrementally, updating its weights by one training example at a time rather than on the whole training set at once, which makes it fast and memory-efficient even in large, sparse feature spaces [6]. The model uses the modified Huber loss, a smooth, margin-aware objective, with L2 regularization ($\alpha = 10^{-4}$; scikit-learn defaults otherwise). With this loss, class probabilities are obtained by clipping the decision function to $[-1, 1]$ and rescaling it, so 71.5% of the test emails receive a phishing probability of exactly 0 or 1; these values are therefore not calibrated probabilities. At each step, the parameters move against the gradient of the loss, as shown in Eq. (3).

$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta; \mathbf{x}_i, y_i) \quad (3)$$

where, θ denotes the model parameters, η the learning rate, and $\nabla_{\theta} L$ the gradient of the loss evaluated on a single training example $(\mathbf{x}_i, y_i)$.

4) *XGBoost classifier*: XGBoost is a gradient-boosted ensemble that builds decision trees sequentially, each new tree correcting the errors of the previous ones [33]. It can therefore learn feature interactions, such as the joint presence of the URL placeholder and a financial trigger word, that a single tree or a linear model would miss. The model uses 100 trees, a learning rate of 0.1, a maximum depth of 6, and the log-loss objective, with the TF-IDF matrix supplied in dense form. The final score is the sum of the outputs of all trees, as shown in Eq. (4).

$$\hat{y}_i = \sum_{k=1}^K f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F} \quad (4)$$

where, the prediction for sample i is the sum over K trees, and each tree is drawn from the space of regression trees.

H. Deep Learning Models

The deep-learning models were trained on tokenized, padded sequences. A Keras tokenizer fitted on the training split kept the 10,000 most frequent words and mapped all other words to an out-of-vocabulary token, and each email was converted into a sequence of word indices. Because neural networks require fixed-length input, shorter sequences were padded at the end, and sequences longer than 150 tokens were truncated from the beginning (the Keras default), so the last 150 tokens of a long email are kept; this truncation affects 27.7% of the training emails, whose median length is 77 tokens. A global random seed (42) was fixed for Python, NumPy, and TensorFlow, with deterministic operations and a single TensorFlow thread, so that the runs are repeatable; because one seed cannot reveal run-to-run variation, multi-seed and cross-validated results are reported in Section IV. All networks were trained with the Adam optimizer (learning rate 0.001) and binary cross-entropy for five epochs with a batch

size of 32; the last 10% of the training arrays were held out for monitoring only, and no early stopping or architecture search was applied to the reported models. Processing an email as a sequence lets these networks detect patterns tied to particular word arrangements, such as urgent language combined with a request to click a link or confirm credentials. The models increase in complexity from a simple recurrent network to a bidirectional network with attention.

1) *Recurrent neural network*: The Simple RNN is the first sequence model and serves as a baseline for the family [6]. It maintains a hidden state that changes with each token, thus allowing information from previous words to affect the interpretation of subsequent words. The design employs a 32-dimensional embedding, a 64-unit SimpleRNN with tanh activation, a dropout of 0.2, and a sigmoid output. At each step, the updated hidden state is a function of the present token and the previous state, as shown in Eq. (5).

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b) \quad (5)$$

where, x_t is the token at position t , h_t the current hidden state, h_{t-1} the previous state, and W_x , W_h , b the learned parameters. In practice, the plain RNN struggles to carry information across a long message, a limitation of the next model's address.

2) *Gated recurrent unit*: The GRU adds gates to the recurrent cell that determines how much of the previous state to retain and how much to replace at each step [3]. This mitigates the vanishing-gradient problem of the plain RNN and helps the model retain relevant context in longer emails, where phishing indicators may be scattered. The implemented model uses a 128-dimensional embedding, spatial dropout (0.2), a 64-unit GRU that returns the full sequence, global max-pooling over time, a 32-unit ReLU layer with dropout (0.2), and a sigmoid output. The hidden state is a gated combination of the previous state and a candidate state, as given by Eq. (6).

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (6)$$

where, z_t is the update gate, the tilde terms the candidate state, and the circle operator denotes element-wise multiplication.

3) *Bidirectional LSTM*: A bidirectional LSTM processes the sequence in both directions, so the representation of a token can depend on the words that follow it as well as on those that precede it [3]. This helps the model infer intent when context matters: an ordinary word can become a warning sign when it appears alongside terms related to account verification, payment, or password reset. The implemented model uses a 128-dimensional embedding, spatial dropout (0.3), and a bidirectional LSTM with 64 units per direction, whose final hidden states feed a 64-unit ReLU layer with dropout (0.3) and a sigmoid output. It is reported on its own as the ablation baseline for the attention model described next, which keeps the same embedding, dropout, and recurrent settings and differs only in how the hidden states

are pooled. The vector at each position concatenates the hidden states of the forward and backward passes, as shown in Eq. (7).

$$h_t = [\vec{h}_t \parallel \tilde{h}_t] \quad (7)$$

where, the right-pointing and left-pointing states are produced by the forward and backward passes and joined into a single representation for that position.

4) *Attention hybrid model*: The final architecture adds an attention layer to the bidirectional LSTM [34]. In a conventional recurrent classifier, the decision is based mainly on a compact summary of the whole sequence, which may overlook important signals in the middle of a long message. Attention instead assigns weight to every hidden state and forms a weighted summary, allowing the network to focus on the tokens most relevant to classification. The weights are normalized over the sequence as in Eq. (8), and the context vector is their weighted sum, as in Eq. (9).

$$\alpha_t = \frac{\exp(e_t)}{\sum_{k=1}^T \exp(e_k)} \quad (8)$$

$$c = \sum_{t=1}^T \alpha_t h_t \quad (9)$$

Here, e_t denotes the score at position t , α_t the resulting attention weight, h_t the hidden state, and T the sequence length. In the implemented layer, $e_t = \tanh(h_t^T w + b_t)$, with a learned vector w and a learned bias b_t for each of the $T = 150$ positions; the softmax in Eq. (8) runs over all positions, including padded ones, because no padding mask is applied. Combined with bidirectional context, attention is intended to let the classifier focus on indicators such as urgent instructions, embedded links, and credential-related phrases wherever they occur in the email; Section IV-G tests whether it does so in practice.

I. Evaluation Metrics

To allow a direct comparison, all models were assessed with the same metrics, with the phishing class treated as the positive class throughout. Accuracy [Eq. (10)] is the proportion of correctly classified emails. Precision (Eq. (11)) is the fraction of emails classified as phishing that are actually phishing. Recall [Eq. (12)] is the fraction of actual phishing emails that the model detects; because missed attacks are usually more costly to an organization than false alarms, recall is particularly important for this task. The F1-score [Eq. (13)] is the harmonic mean of precision and recall and summarizes the trade-off between them. The false-positive rate, $FP/(FP + TN)$, is also reported where relevant, because it measures the share of legitimate emails that would be blocked.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (10)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (11)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (12)$$

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (13)$$

ROC-AUC measures how well a model ranks phishing above legitimate emails across all decision thresholds; it is reported to four decimal places in Table I so that values close to one are not rounded to an uninformative 1.00. Recall, F1-score, and the false-positive rate are the main metrics in the following tables and discussion, because the primary objective is to detect phishing without obstructing the delivery of legitimate mail. Unless stated otherwise, all metrics refer to the held-out test split.

IV. RESULTS

A. Overview

In this section, each model's performance on the held-out test set is presented. All models were evaluated using the same set of five metrics, and their results were accompanied by ROC and feature-importance analyses. The order of presentation is as follows: individual models, their combined comparison, a statistical validation with cross-validation and significance tests, a corpus-provenance analysis, an attention ablation, a perturbation-based adversarial benchmark, an end-to-end evaluation of the two-stage cascade, operational profiling of latency and throughput, and a brief description of an interactive demonstrator.

B. Machine Learning Models

The four classical models were trained on the 10,000-dimensional TF-IDF space and set a strong baseline.

1) *Logistic regression*: Logistic Regression was both efficient and accurate, achieving 97.10% accuracy, a 96.11% F1-score, and an ROC-AUC of 0.9957. Its precision of 97.06% is the highest recorded in the study, so very few legitimate messages were wrongly flagged as phishing, although its recall of 95.18% shows that a single linear boundary still lets a small share of phishing traffic through. With the phishing class set as the positive class, the largest coefficients in the phishing direction are explicit lure vocabulary: "click", "http", "remove", "free", "money", "software", and "offer". The corpus-specific terms that dominate the opposite direction of the same boundary, such as "enron" and "university", therefore describe what makes a message look legitimate in this corpus rather than what makes it look like phishing.

2) *Decision tree*: The Decision Tree sacrificed some accuracy for transparency and ended up with 91.95% accuracy, an 89.40% F1-score, and 90.27% recall, with an ROC-AUC of 0.9201. The lower recall indicates that a single Decision Tree could not cover the full range of phishing language as well as linear or ensemble models. The most significant split features were "enron", the URL placeholder, "click", and "language", which combine organizational context, the structural cues retained after the preprocessing step, and language used to encourage users to take action.

3) *SGD classifier*: SGD obtained the highest accuracy and F1-score of all eight models on the held-out split. With the modified Huber loss, it reached 97.98% accuracy and 98.05% recall; only the Bi-LSTM with attention detected a larger

share of phishing messages (98.60%), at the cost of more false positives. Its precision of 96.63% remains high, meaning that few legitimate emails were mistakenly classified as phishing attacks, and its F1-score of 97.33% reflects the balance between the two. An ROC-AUC of 0.9979 indicates near-perfect ranking of phishing above legitimate mail on this test partition, without the implausible claim of exact separation that a rounded 1.00 would imply. The features with the largest weights in the phishing direction are lure and mailing-list vocabulary such as "click", "http", "remove", "mobile", "mortgage", and "money". The token "spamassassinsightings" also ranks highly; as Section IV-F shows, this reflects the provenance of part of the phishing corpus rather than a general phishing indicator.

4) *XGBoost classifier*: XGBoost achieved 95.58% accuracy, 94.02% precision, 94.24% recall, a 94.13% F1-score, and an ROC-AUC of 0.9901. The most influential features of the model were "language", the action token "click", "enron", and "money", followed by "price" and the URL placeholder, so the ensemble leans toward behavioral and financial lures mixed with corpus context. The model's results are solid, and it ranks below both the linear baselines and the gated sequence models.

C. Deep Learning Models

Attention now turns to the sequence models, which read each email as an ordered stream of word embeddings rather than a bag of weighted terms.

1) *Simple recurrent neural network*: The Simple RNN fell well behind every other model, reaching 65.26% accuracy, an F1-score of 24.44%, and an ROC-AUC of 0.6294. The accuracy figure on its own is misleading. The majority class (safe email) makes up about 62% of the test set, so a naive classifier that always predicted safe would already score close to that, and the Simple RNN clears the trivial baseline by only about three points. The decisive number is recall, which collapsed to 14.94%: the model missed about 85% of the phishing messages, with a precision of only 67.13% on the small subset it did flag. In operational terms, this is a failure case rather than a weak result, because a detector that passes most of the attacks it is deployed to stop has no usable security value regardless of its accuracy. This behavior is consistent with the vanishing-gradient limitation of plain recurrent networks: over 150-token sequences, gradient signals decay before reaching the earlier time steps, thus preventing the network from learning meaningful long-range dependencies, and after five epochs the model reached only 71.0% training accuracy and 66.7% validation accuracy. It indicates that, under this training budget, the ungated recurrent architecture did not learn a useful decision boundary, and it marks a performance floor that the gated and bidirectional networks clearly exceed. The highest embedding weights of the model are dominated by administrative and routing fragments such as "spamassassinsightings", "razorusers", "edu", and "exmhusers", which indicates that the model latched onto

static mailing-list markers rather than meaningful phishing language.

2) *Gated recurrent unit*: Adding gating changed the picture substantially. The GRU reached 97.42% accuracy, a 96.62% F1-score, 97.98% recall (the third highest in the study), 95.31% precision, and an ROC-AUC of 0.9970; its F1-score was the highest among the deep models, and its held-out false-positive rate was 2.91%. This high sensitivity, which matters because a missed message is costly, is consistent with the gating mechanism preserving context across the sequence. The gain over the Simple RNN, 72.18 percentage points of F1-score, is the largest in the study, but the two networks also differ in embedding size (128 versus 32 dimensions), pooling, and the dense layer, so the gain cannot be attributed to gating alone. The model's most influential tokens include "enron", "kaminski", and "click" alongside routing identifiers such as "spamd", "razorusers", and "exmh", so the network combines content cues with mailing-list markers.

3) *Plain bidirectional LSTM*: The plain bidirectional LSTM is reported here as an ablation of the attention model, since the two share the same embedding, dropout, and recurrent configuration. It reached 96.99% accuracy, 94.77% precision, 97.35% recall, a 96.05% F1-score, and an ROC-AUC of 0.9950. On this split, it scored slightly higher than the attention variant in F1-score (96.05% versus 95.91%), although the difference is within the run-to-run variation measured in Section IV-G. Its most influential embedding weights are concentrated on corpus and routing tokens such as "enron", "vince", "wrote", "kaminski", and "rpmlist", rather than on lure vocabulary.

4) *Bidirectional LSTM with attention*: The Bi-LSTM with attention obtained the highest recall of all eight models: it achieved 96.84% accuracy, 93.37% precision, 98.60% recall, a 95.91% F1-score, and an ROC-AUC of 0.9961, but its false-positive rate of 4.22% was the highest among the five leading models. Compared with the plain Bi-LSTM trained under identical conditions with the same seed, attention changed accuracy by -0.15 and F1-score by -0.14 percentage points while raising recall by 1.25 points. A single run cannot separate such small differences from seed-to-seed variation, so the contribution of the attention layer is examined with the multi-seed ablation in Section IV-G. The tokens with the largest embedding-weight magnitudes, such as "enron", "wrote", "rolex", "edu", "rpmlist", and "viagra", combine corpus context, header and mailing-list markers, and spam lure terms, so the learned representation is effective but not free of corpus-specific signal.

D. Performance Comparison

Table I compares the eight models on the held-out test set. The SGD classifier obtained the highest accuracy (97.98%), F1-score (97.33%), and ROC-AUC (0.9979), followed in F1-score by the GRU (96.62%), Logistic Regression (96.11%), the plain Bi-LSTM (96.05%), and the Bi-LSTM with attention (95.91%); a linear model on a TF-IDF representation can therefore match or exceed the recurrent networks on this

benchmark. The Bi-LSTM with attention had the highest recall (98.60%), followed by SGD (98.05%) and GRU (97.98%), whereas Logistic Regression had the highest precision (97.06%) and the lowest false-positive rate (1.74%). The five leading models lie within about 1.4 points of F1-score. The stratified-bootstrap 95% confidence interval of the SGD F1-score (0.9670–0.9795) overlaps those of the GRU (0.9588–0.9730), Logistic Regression (0.9535–0.9682), and the plain Bi-LSTM (0.9527–0.9680), but not that of the attention model (0.9513–0.9665), so the single split alone cannot rank the leading models with confidence; Section IV-E tests these differences formally. The Simple RNN reached only 65.26% accuracy and 14.94% recall, barely above the majority-class baseline of 62.4%, which shows that it did not learn a meaningful decision boundary under these training conditions. The jump from the Simple RNN (24.44% F1-score) to the gated and bidirectional networks, which all exceed 95.9%, shows that the plain recurrent network is inadequate for these long texts, although the architectural differences between the networks prevent attributing the gain to a single component. XGBoost (95.58% accuracy) and the Decision Tree (91.95%) ranked below the linear models and the gated recurrent networks. Table II places these results next to the values reported by Fares et al. [5] on the same public dataset; that comparison is discussed in Section V.

TABLE I. COMPARATIVE PERFORMANCE OF ALL EIGHT MODELS ON THE HELD-OUT TEST SET, WITH THE PHISHING CLASS AS THE POSITIVE CLASS, SORTED BY F1-SCORE (SINGLE STRATIFIED 80/20 SPLIT, SEED 42)

Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
SGD Classifier	0.9798	0.9663	0.9805	0.9733	0.9979
GRU	0.9742	0.9531	0.9798	0.9662	0.9970
Logistic Regression	0.9710	0.9706	0.9518	0.9611	0.9957
Plain Bi-LSTM	0.9699	0.9477	0.9735	0.9605	0.9950
Bi-LSTM + Attention	0.9684	0.9337	0.9860	0.9591	0.9961
XGBoost	0.9558	0.9402	0.9424	0.9413	0.9901
Decision Tree	0.9195	0.8855	0.9027	0.8940	0.9201
Simple RNN	0.6526	0.6713	0.1494	0.2444	0.6294

TABLE II. INDICATIVE COMPARISON OF MODEL-SPECIFIC RESULTS (%) WITH THE VALUES REPORTED BY FARES ET AL. [5] ON THE SAME PUBLIC DATASET; PREPROCESSING AND DATA SPLITS DIFFER, AND – DENOTES A VALUE NOT AVAILABLE FOR THAT MODEL

Study and model	Accuracy	Precision	Recall	F1-score
Fares et al. [5], SVM	97.12	–	96.91	–
Fares et al. [5], Naive Bayes	–	96.86	–	96.74
Fares et al. [5], LSTM	95.49	–	–	–
This study, SGD Classifier	97.98	96.63	98.05	97.33
This study, Logistic Regression	97.10	97.06	95.18	96.11
This study, GRU	97.42	95.31	97.98	96.62
This study, Bi-LSTM + Attention	96.84	93.37	98.60	95.91

E. Statistical Validation

To test whether the ranking in Table I is stable, the 17,085 emails were re-partitioned with stratified 3-fold cross-validation (one repetition, seed 42; about 11,390 training and 5,695 test emails per fold), with the same folds used for every model so that the results are paired. The classical models were evaluated in each fold twice: with the published settings and with nested tuning, in which a randomized search over 10 candidate settings, scored by the phishing-class F1-score on an inner stratified 3-fold split of the training fold, selected the configuration that was then refitted on the whole training fold. The GRU, the plain Bi-LSTM, and the Bi-LSTM with attention kept their published architectures; the tokenizer was refitted on each training fold, and the number of epochs was chosen by early stopping on a stratified 10% validation split of the training fold (patience of two epochs, at most 10 epochs, best weights restored), which stopped training after three to five epochs. The architecture hyperparameters of the deep models were not searched, and the Simple RNN was excluded from cross-validation because it trails the other models by more than 60 F1 points. Table III lists all model settings and the tuning search spaces. Because of the time available for this revision, a single repetition of 3-fold cross-validation was used rather than a repeated design.

TABLE III. MODEL SETTINGS, TRAINING PROTOCOL, AND NESTED-TUNING SEARCH SPACES

Component	Settings
TF-IDF (classical models)	Unigrams; 10,000 most frequent training terms; smoothed IDF; L2 normalization; fitted on the training split
Logistic Regression	liblinear solver; L2 penalty; $C = 1.0$
Decision Tree	Entropy criterion; no depth limit; scikit-learn defaults otherwise
SGD Classifier	Modified Huber loss; L2 penalty; $\alpha = 10^{-4}$; scikit-learn defaults otherwise
XGBoost	100 trees; learning rate 0.1; maximum depth 6; log-loss; dense input
Sequence input	Tokenizer with 10,000 words and an OOV token; length 150; post-padding; truncation from the beginning
Simple RNN	Embedding 32; SimpleRNN 64 (tanh); dropout 0.2; sigmoid output
GRU	Embedding 128; spatial dropout 0.2; GRU 64 (all states); global max pooling; dense 32 (ReLU); dropout 0.2; sigmoid output
Plain Bi-LSTM	Embedding 128; spatial dropout 0.3; Bi-LSTM 64 per direction (last state); dense 64 (ReLU); dropout 0.3; sigmoid output
Bi-LSTM + Attention	As the plain Bi-LSTM, with all hidden states pooled by the attention layer of Eq. (8)–(9) (no padding mask)
Deep-model training	Adam (learning rate 0.001); binary cross-entropy; 5 epochs; batch size 32; last 10% of the training arrays used for monitoring only; seed 42
Nested tuning (CV)	Randomized search; 10 candidates; inner stratified 3-fold split; phishing-class F1-score
Search space: LR	$C \sim \log\text{-uniform}[10^{-2}, 10^2]$; TF-IDF size {5k, 10k, 20k}; sublinear TF {no, yes}
Search space: SGD	$\alpha \sim \log\text{-uniform}[10^{-6}, 10^{-3}]$; TF-IDF size {5k, 10k, 20k}; sublinear TF {no, yes}
Search space: DT	Criterion {Gini, entropy}; maximum depth {none, 20, 50, 100}; minimum leaf size {1, 2, 5, 10}; TF-IDF size {5k, 10k, 20k}

Component	Settings
Search space: XGBoost	Trees {100, 200, 400}; maximum depth {4, 6, 8}; learning rate {0.05, 0.1, 0.2}; subsample {0.8, 1.0}
Deep models (CV)	Published architectures; tokenizer refitted per fold; early stopping on a 10% validation split (patience 2, at most 10 epochs, best weights restored)

Three kinds of uncertainty are reported and kept separate in Table IV. For the single held-out split, 95% confidence intervals were obtained with a stratified bootstrap of the test set (2,000 resamples). For cross-validation, the mean over the three folds is reported with a 95% interval that applies the Nadeau–Bengio correction for the overlap of training sets between folds [35]. Pairwise differences in fold-level F1-scores were tested with the corrected resampled t-test [35] and the Wilcoxon signed-rank test, and the errors of the published models on the held-out split were compared with the exact McNemar test [36]; each family of pairwise tests was adjusted with the Holm procedure [37].

TABLE IV. CROSS-VALIDATED

Model	Held-out F1 (bootstrap 95% CI)	Held-out FPR	3-fold CV F1, mean (corrected 95% CI)
SGD Classifier	0.973 (0.967–0.980)	0.021	0.969 (0.966–0.971)
SGD, tuned	–	–	0.972 (0.959–0.985)
LR	0.961 (0.954–0.968)	0.017	0.959 (0.953–0.965)
LR, tuned	–	–	0.972 (0.960–0.983)
XGBoost	0.941 (0.932–0.950)	0.036	0.941 (0.933–0.950)
XGBoost, tuned	–	–	0.959 (0.956–0.962)
DT	0.894 (0.882–0.906)	0.070	0.893 (0.877–0.910)
DT, tuned	–	–	0.887 (0.867–0.908)
GRU	0.966 (0.959–0.973)	0.029	0.961 (0.949–0.973)
Plain Bi-LSTM	0.961 (0.953–0.968)	0.032	0.954 (0.916–0.992)
Bi-LSTM + Attention	0.959 (0.951–0.967)	0.042	0.963 (0.945–0.980)
Simple RNN	0.244 (0.217–0.273)	0.044	not run

Table IV shows that the cross-validated means agree with the held-out values within about one F1 point for every model. With nested tuning, Logistic Regression and SGD both reached a mean F1-score of 0.972, at least as high as the best deep model, the Bi-LSTM with attention (0.963), whereas tuning did not help the Decision Tree. Among the five leading models, none of the ten pairwise differences in cross-validated F1-score was significant after Holm correction (smallest adjusted $p = 0.12$, Logistic Regression versus SGD with the published settings; GRU versus SGD, $p = 0.97$); with three folds, the Wilcoxon test cannot fall below $p = 0.25$ and therefore has no power at this scale. On the held-out split, however, the exact McNemar test with Holm correction over all 28 model pairs showed that SGD made significantly fewer errors than Logistic Regression (adjusted $p = 0.004$), the plain Bi-LSTM ($p = 0.011$), and the Bi-LSTM with attention ($p = 0.003$); its advantage over the GRU was not significant ($p = 0.33$), and no other pair among the five leading models differed significantly. The evidence therefore consistently favors the linear SGD classifier.

F. Corpus-Provenance Sensitivity

Because the CSV file carries no source metadata (Section III-B), the influence of corpus provenance was estimated with keyword proxies. A fixed list of corpus markers was organized into three tiers: 1) corpus, mailing-list, and organization identifiers such as "enron", "kaminski", "spamassassinsightings", and "razorusers"; 2) header remnants and list boilerplate such as "wrote", "forwarded", and "unsubscribe"; and 3) suspected topical markers such as "language", "university", and "houston". The markers are strongly class-specific: "enron" occurs in 2,213 safe emails and in one phishing email, whereas "spamassassinsightings" occurs in 122 phishing and 2 safe emails. Tier-A markers appear in 40.9% of safe emails but only in 4.1% of phishing emails, and at least one marker of any tier appears in 75.9% of safe and 33.7% of the phishing emails. A logistic-regression probe that sees only the presence of these markers, evaluated with 5-fold cross-validation, reached accuracies of 0.634, 0.687, and 0.769 (ROC-AUC 0.705, 0.779, and 0.851) for tiers A, A+B, and A+B+C, against a majority-class baseline of 0.624; on the emails that contain a tier-A marker, its accuracy was 0.980 against a baseline of 0.943. Corpus identifiers therefore carry class information that a model can exploit without reading the phishing content.

Table V reports the F1-scores of models retrained on the identical split after the markers were deleted from both the training and the test texts. The classical models lost 0.2 to 2.3 F1 points at tier A and 0.3 to 3.5 points at the most aggressive tier; XGBoost and the Decision Tree were the most affected (XGBoost dropped from 0.941 to 0.906), whereas the two linear models changed little (SGD from 0.973 to 0.968). When the published models were applied to masked test texts without retraining, the losses were larger, up to 9.3 F1 points for the Decision Tree, whose false-positive rate rose to 21.0% at tier A+B+C. The three recurrent models retrained once on text masked at tier A+B changed by -0.31 (GRU), -0.06 (plain Bi-LSTM), and +0.79 (Bi-LSTM with attention) F1 points, differences that lie within the seed-to-seed variation reported in Section IV-G. The masked results are the more conservative estimates of the achievable performance, and they suggest that the linear models rely less on provenance markers than the three-based models.

TABLE V. F1-SCORES OF MODELS RETRAINED AFTER DELETING THE CORPUS MARKERS OF TIERS A, A+B, AND A+B+C (IDENTICAL SPLIT; - = NOT RUN)

Model	No mask	Tier A	Tiers A+B	Tiers A+B+C
Logistic Regression	0.9611	0.9593	0.9578	0.9577
Decision Tree	0.8940	0.8816	0.8758	0.8750
SGD Classifier	0.9733	0.9712	0.9696	0.9678
XGBoost	0.9413	0.9185	0.9160	0.9060
GRU	0.9662	-	0.9631	-
Plain Bi-LSTM	0.9605	-	0.9598	-
Bi-LSTM + ttention	0.9591	-	0.9670	-

As an approximation to a cross-corpus test, each email was assigned to a keyword-derived proxy source (Enron-like,

SpamAssassin-list-like, Ling-Spam-like, or unmarked), and the two linear models were trained without one proxy group and tested on it. These proxy labels are inferred from the text rather than taken from source metadata, and the held-out groups are small in phishing and almost entirely legitimate (for example, 34 phishing and 2,795 safe emails in the Enron-like group), so this is not an independent cross-corpus evaluation. Holding a group out nevertheless raised the false-positive rate on its legitimate emails, compared with out-of-fold predictions in which the group was seen during training: for SGD, from 0.5% to 3.8% on the Enron-like group and from 0.2% to 1.7% on the SpamAssassin-list-like group, and for Logistic Regression from 1.0% to 5.5% on the Enron-like group. Even after the most aggressive masking, a classifier could still identify the proxy source of legitimate emails with 98.9% accuracy (majority baseline 52.2%), so source-specific style and topics persist beyond the deleted tokens. Together, these results indicate that part of the 96–98% scores reflect corpus provenance and that performance on mail from unseen sources is likely to be lower; evaluation on independent external datasets is required to establish generalization.

G. Attention Ablation

To test whether the attention layer itself improves detection, five pooling variants of the same Bi-LSTM backbone were trained with the published protocol on the original split with three seeds (42, 7, and 123): the last hidden state (the plain Bi-LSTM), the published attention layer, a masked attention layer that gives zero weight to padded positions, global max pooling, and global mean pooling. The variants share every other setting, and the attention layer adds only 278 parameters to the 1,387,137 of the last-state networks. Table VI reports the mean over the seeds; paired t-tests and Wilcoxon tests across seeds were adjusted with the Holm procedure within each reference model.

TABLE VI. POOLING ABLATION OF THE BI-LSTM BACKBONE OVER THREE SEEDS ON THE HELD-OUT SPLIT (MEANS; F1-SCORE AS MEAN ± STANDARD DEVIATION)

Pooling variant	Accuracy	Recall	FPR	F1-score
Max pooling	0.9751	0.9795	0.0275	0.9673 ± 0.0018
Mean pooling	0.9725	0.9645	0.0227	0.9634 ± 0.0063
Attention published)	0.9713	0.9774	0.0324	0.9625 ± 0.0045
Masked attention	0.9716	0.9658	0.0249	0.9624 ± 0.0026
Last state (plain Bi-LSTM)	0.9704	0.9681	0.0281	0.9610 ± 0.0083

No variant differed significantly in F1-score from either the last-state network or the attention network; the smallest Holm-adjusted p-value was 0.89 (max pooling versus attention). Max pooling had the highest mean F1-score and accuracy and matched or exceeded attention in every metric, including ROC-AUC (0.9965 versus 0.9953), while mean pooling, masked attention, and attention were within 0.1 F1 points of one another. The spread across seeds is as large as the differences between variants: with seed 42 the last-state network scored above the attention network (0.9605 versus 0.9591), but with seed 123 it scored below it (0.9529 versus 0.9677). In the published attention model, 21.9% of the attention mass falls on padding positions on average (24.4% for phishing and 20.4%

for safe emails), because the softmax runs over all 150 positions and 74.6% of the test emails are shorter than that, whereas only 2.0% falls on the provenance markers of tiers A and B; masking the padding did not improve the F1-score. These results do not support the claim that attention improves detection on this dataset.

H. Adversarial Robustness

The adversarial evaluation of the original manuscript relied on five hand-written messages and on a context-aware boost that was applied to every model. Re-scoring those messages with the training preprocessing and without the boost showed that they could not discriminate between models: every model except the Simple RNN made the same three correct decisions out of five, detecting the direct credential lure and the character-substitution message and passing the legitimate control message, while missing the LLM-style and the good-word messages. The five-message test was therefore replaced by the larger benchmark described below, and the boost is evaluated as a separate component.

1) *Perturbation benchmark*: A random sample of 1,000 phishing and 1,000 safe emails was drawn from the held-out test set (seed 2026), and nine perturbations were applied programmatically to the raw text of each email before the preprocessing pipeline: leet substitution and homoglyph substitution in 30% of the eligible words, random character insertions, deletions, and swaps in 15% of the words, WordNet synonym replacement in 20% of the words, defanging of URLs (for example, "hxxp://" and "[.]"), appending a fixed benign institutional paragraph (template padding), appending the first 60 words of a randomly chosen legitimate training email (training-ham padding), wrapping the message in a polite greeting and signature (template framing), and leet substitution combined with template padding. The perturbations are synthetic and non-adaptive: they do not query the models and do not represent an attacker who optimizes against a particular detector. Robustness is measured by the attack success rate, the share of phishing emails detected before the perturbation that are no longer detected after it, with Wilson 95% confidence intervals [38]; URL defanging is evaluated only on the phishing emails that contain a URL.

2) *Context-aware scoring*: The original manuscript also applied a lightweight context-aware scoring layer after the base models to capture sentence-level phishing intent beyond single-word signals. The layer is a heuristic post-processing step computed on the raw text with fixed keyword lists and is not part of any trained architecture. It computes a weighted count of co-occurring signals, namely action verbs, urgency terms, and urgency terms in adjacent sentences, and converts this count into a context weight using Eq. (14).

$$w_{\text{context}} = \min(C_{\text{indicators}} \times 0.25, 0.75) \quad (14)$$

The indicator count is a weighted tally of the number of action verbs, urgency terms, and urgency signals in adjacent sentences. The resulting weight indicates how much the base

model's predicted probability is adjusted upward, as specified in Eq. (15),

$$p_{\text{final}} = p_{\text{orig}} + (1 - p_{\text{orig}}) \times w_{\text{context}} \quad (15)$$

Eq. (15) shows that the resulting probability comes from an initial phishing probability, denoted by p_{orig} , which is then adjusted to produce p_{final} based on the context score. Such a post-processing technique provides a straightforward way of detecting structural signs of phishing, for instance, under obfuscation attacks, where deceptive intent is not concentrated in a single token but is distributed across sentences. Because Eq. (15) can only raise the phishing probability, the boost can add detections but also false positives, and when the weight exceeds 0.5, the boosted score crosses the decision threshold regardless of the output of the model; this is why all models scored about 75% on the LLM-style message of the original five-message test. In this study, the boost is therefore evaluated as a separate component: every model is scored with and without it, using the same weights for all models, so that its effect is not attributed to any architecture.

3) *Adversarial results*: Fig. 3 reports the attack success rate of every model under each perturbation without (a) and with (b) the context-aware boost. Character-level and lexical perturbations were largely ineffective against the stronger models: homoglyph substitution, typographical edits, synonym swaps, and URL defanging changed at most 2% of the detections of SGD, Logistic Regression, XGBoost, the GRU, and both Bi-LSTM variants; template framing changed at most 7% (XGBoost); and leet substitution at most 9% (Logistic Regression).

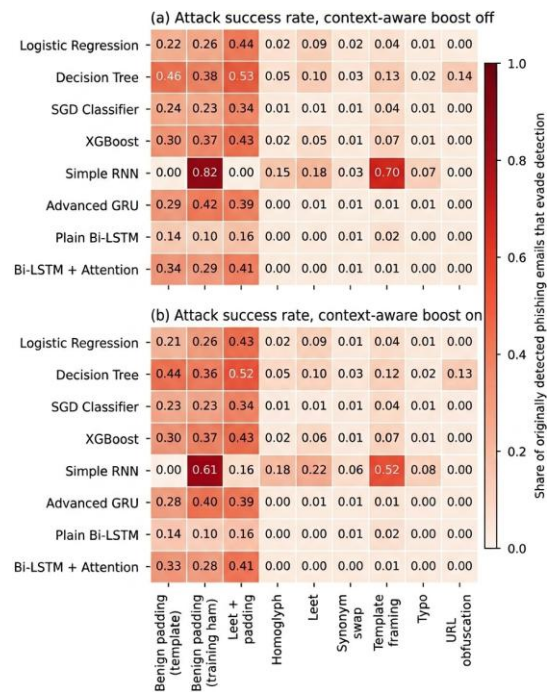


Fig. 3. Attack success rate (share of originally detected phishing emails that are missed after the perturbation) of each model under nine synthetic perturbations of 1,000 held-out phishing emails, (a) without and (b) with the context-aware boost of Eq. (14)–(15).

The Decision Tree was more fragile (14% under URL defanging and 13% under template framing), and the Simple RNN lost many of its few detections (70% under template framing). Appending benign text was the most effective attack for every model except the Simple RNN: template padding evaded 14% (plain Bi-LSTM) to 46% (Decision Tree) of the previous detections, training-ham padding 10% to 42%, and leet substitution combined with padding 16% to 53%. The Bi-LSTM with attention was not more robust than the other leading models: its success rates under the three padding attacks (34%, 29%, and 41%) were similar to or higher than those of SGD (24%, 23%, and 34%) and Logistic Regression (22%, 26%, and 44%), and the plain Bi-LSTM was the least affected of the seven other models in all three. With about 900 to 1,000 originally detected phishing emails per model (far fewer for the Simple RNN), the Wilson 95% interval of each rate is no wider than about ± 3.5 percentage points, so differences of this size are well outside sampling noise. The context-aware boost changed the success rates by about two percentage points or less for every model except the Simple RNN, so it did not protect any model against padding, and none of its effect can be attributed to the model architecture. Since the perturbations are synthetic and non-adaptive, these results describe sensitivity to simple, generic edits; they do not establish robustness against an adaptive attacker or against LLM-generated phishing.

I. Two-Stage Cascade Evaluation

The two-stage design suggested in the original manuscript was evaluated end-to-end on the held-out test set. Stage 1 is an SGD classifier with the published hyperparameters, and stage 2 is the published Bi-LSTM with attention. An email is escalated to stage 2 when its stage-1 phishing probability lies strictly inside an uncertainty band (t_{low} , t_{high}); otherwise, the stage-1 decision is final. To select the band without touching the test set, stage 1 was refitted on the first 90% of the training split (12,301 emails), and the band was chosen on the remaining 10% (1,367 emails), the slice that Keras held out from the training of stage 2, so that neither stage was trained on it. A grid search with a step of 0.05 looked for the band with the lowest escalation rate whose validation recall was at least that of stage 2 and whose false-positive rate was at most that of stage 1; because no band met both constraints, the band with the highest validation F1-score (0.25, 0.60) was taken, which escalated 4.8% of the validation emails. Since the modified Huber loss gives 72.5% of the test emails a stage-1 probability of exactly 0 or 1, only the remaining emails can fall inside the band.

Table VII compares the cascade with each stage alone. On the test set, the cascade escalated 5.6% of the emails and reached 98.52% recall, 95.55% precision, a 97.01% F1-score, and a 2.77% false-positive rate. Compared with the attention model alone, it made significantly fewer errors (exact McNemar test, Holm-adjusted $p < 0.001$) and reduced the false-positive rate from 4.22% to 2.77% at almost the same recall. Compared with the published SGD model (F1-score 97.33%, recall 98.05%, false-positive rate 2.06%), the cascade gained 0.47 points of recall but added 0.71 points of false-positive rate, and the difference in errors was not significant (adjusted $p = 0.34$); it was not significantly different from its

own refitted stage 1 either (recall 97.12%, false-positive rate 1.22%, $p = 0.21$). The cascade is therefore a design option that trades a small increase in false positives for a small gain in recall, not a demonstrated improvement over SGD alone. A descriptive sweep of symmetric bands on the test set showed the same trade-off: widening the band moves the recall toward that of the attention model while the false-positive rate rises steadily.

TABLE VII. END-TO-END EVALUATION OF THE SGD $\rightarrow$ Bi-LSTM + ATTENTION CASCADE ON THE HELD-OUT TEST SET (P: HOLM-ADJUSTED EXACT MCNEMAR TEST AGAINST THE CASCADE)

System	Escalated	Recall	F1-score	FPR	p
Cascade, band (0.25, 0.60)	0.0562	0.9852	0.9701	0.0277	–
Published SGD alone	0.0000	0.9805	0.9733	0.0206	0.336
Stage 1 alone (SGD refitted on 90%)	0.0000	0.9712	0.9754	0.0122	0.211
Bi-LSTM + Attention alone	1.0000	0.9860	0.9591	0.0422	<0.001

J. Operational Profiling: Latency and Throughput

Accuracy is only half of what is required by a live email gateway; an accurate but slow model will not be able to match the pace of enterprise mail traffic. The latency of every model was therefore measured with a high-resolution counter on 1,000 randomly drawn held-out emails after 20 warm-up emails. Two modes were timed: end-to-end, which starts from the raw text and includes the shared preprocessing of Section III-C, the TF-IDF transformation or the tokenization and padding, and the model call; and inference-only, which times the model call on already vectorized input. Each mode was run at batch sizes of 1, 32, and 128, and throughput was computed from the wall-clock time as in Eq. (16), where the latency is the mean time per email. The measurements were taken on a single machine with an x86-64 processor (48 logical cores, 177 GB of memory) and one GPU, using Python 3.13, scikit-learn 1.6.1, XGBoost 3.4.1, and TensorFlow 2.21.0 with Keras 3.13.2; the classical models ran on the CPU, the neural networks were called through compiled TensorFlow functions with the GPU available, and TensorFlow was restricted to a single thread. A limit of 5 milliseconds per email was set as the criterion for direct real-time deployment in a gateway.

$$\text{Throughput (emails/sec)} = \frac{1000}{\text{Latency (ms/email)}} \quad (16)$$

Fig. 4 and Table VIII summarize the results. With a batch size of 1, the median end-to-end latency was 0.45 ms for Logistic Regression and SGD, 0.46 ms for the Decision Tree, and 0.61 ms for XGBoost, and the 95th percentiles stayed below 2 ms. For these models, the preprocessing dominated the cost: the shared cleaning step alone took 0.40 ms per email on average, whereas the inference-only time of the classifiers was 0.04 to 0.10 ms. The GRU, the plain Bi-LSTM, and the Bi-LSTM with attention had median end-to-end latencies of 1.78, 2.69, and 2.85 ms, with 95th percentiles below 4 ms, so in this

setting they also met the 5 ms budgets; only the Simple RNN (29.31 ms) did not. These values are far lower than the 88 to 146 ms reported in the original manuscript, which averaged five messages and used the generic Keras predict() call; that call alone takes 46 to 48 ms per email for the gated networks (75 ms for the Simple RNN), so the earlier figures mostly measured framework overhead. Batching raised the throughput of every model, and at a batch size of 128 all models except the Simple RNN converged to about 2,080 to 2,250 emails per second end to end, close to the 2,517 emails per second of the single-threaded preprocessing alone, which therefore becomes the bottleneck. The cascade had a median latency of 0.49 ms and a 95th percentile of 3.07 ms, because only escalated emails pay the cost of the second stage. The serialized recurrent models occupy 15.9 to 16.7 MB (3.9 MB for the Simple RNN) plus 6.0 MB for the tokenizer, compared with 81 to 222 kB for the classical classifiers plus 0.37 MB for the shared TF-IDF vectorizer. Two qualifications apply: the classical and neural models used different hardware paths (CPU versus GPU-enabled TensorFlow), so their timings are not strictly comparable, and single-machine benchmarks on stored emails do not demonstrate scalability under real mail traffic, concurrency, or network load.

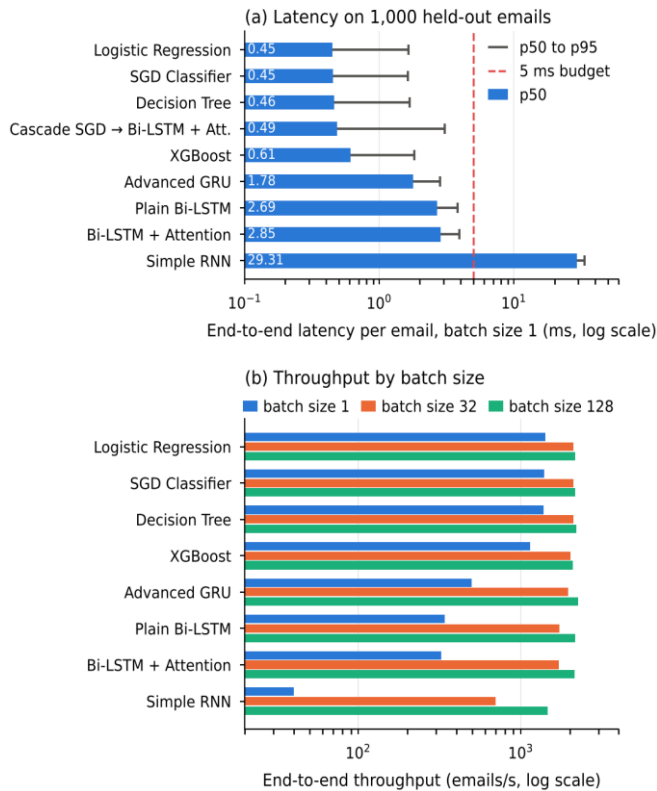


Fig. 4. End-to-end latency per email at batch size 1 (a; bars show the median and whiskers the 95th percentile) and end-to-end throughput at batch sizes 1, 32, and 128 (b), measured on 1,000 held-out emails.

TABLE VIII. LATENCY IN MS PER EMAIL AT BATCH SIZE 1 AND END-TO-END THROUGHPUT IN EMAILS/S ON 1,000 HELD-OUT EMAILS (SINGLE MACHINE; E2E = END-TO-END; MODEL CALL = MEAN INFERENCE-ONLY TIME; BS = BATCH SIZE)

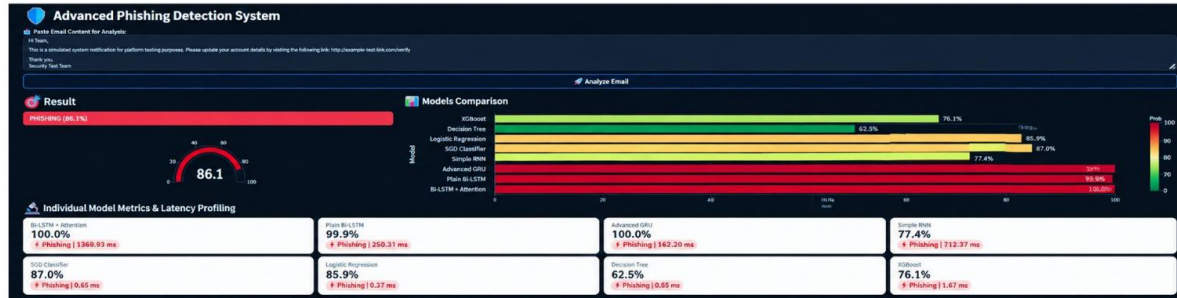
Model	E2E p50	E2E p95	Model call	Emails/s (bs 1)	Emails/s (bs 128)
Preprocessing only	0.181	1.142	–	2,517	–
Logistic Regression	0.446	1.638	0.041	1,415	2,156
Decision Tree	0.462	1.672	0.057	1,379	2,189
SGD Classifier	0.454	1.635	0.049	1,396	2,154
XGBoost	0.613	1.812	0.096	1,143	2,081
Simple RNN	29.309	33.378	27.601	40	1,463
GRU	1.776	2.837	1.387	497	2,245
Plain Bi-LSTM	2.694	3.825	2.367	340	2,154
Bi-LSTM + Attention	2.848	3.913	2.405	325	2,142
Cascade (SGD → Bi-LSTM + Attention)	0.487	3.067	–	1,146	–

K. Interactive Demonstrator

In order to demonstrate that the system was capable of functioning beyond the notebook, the trained models were packaged as an interactive phishing-analysis dashboard. The user interface is based on Streamlit, the server side runs the TensorFlow/Keras and scikit-learn models, and a Pyngrok tunnel makes the app available through a public URL. The server side registers a custom attention layer so that the Bi-LSTM with attention operates the same way it did in the experiments. In the revised version, the dashboard applies the same preprocessing, padding, truncation, and dense XGBoost input as the experiments, and the context-aware boost of Eq. (14)–(15) is an explicit option that is switched off by default. Users can input raw email text into one field, which automatically triggers inference across all eight models simultaneously. The dashboard displays an averaged phishing score on a dial, a safe or phishing decision, a comparative bar chart of models showing visual agreement or disagreement, and individual model cards reflecting the architectures' probabilities with color-coded labels and the measured latency of each call.

Fig. 5 illustrates the interface for a link-based phishing email (a) and for the notes of a weekly project meeting (b). The screenshots were taken with the initial version of the dashboard, which scored the raw text without the training preprocessing and always applied the context-aware boost; the scores displayed in them therefore illustrate the interface only and are not used as evidence in this study. The demonstrator has not been evaluated with users or on live mail traffic, and its per-call latency includes web-application overhead, so it is a proof of concept rather than a deployment.

(a) Phishing case



(b) Safe case



Fig. 5. Interface of the interactive demonstrator on a phishing case (a) and a safe case (b); screenshots from the initial version of the dashboard, with illustrative scores.

V. DISCUSSION

This study aimed to determine which models most proficiently identify phishing emails among legitimate ones under the same setup: preprocessing, dataset splitting, and assessment method. The research followed the same benchmark as recent work, namely the Phishing Email Detection dataset examined by Fares et al. [5]. Because both papers relied on the same public dataset, differences in the reported results may reflect model types, input features, preprocessing, data partitioning, and training strategies, which cannot be separated without re-running the reference setting. The fact that the two studies employed the same benchmark allows an indicative, but not a controlled, comparison.

Table II lists the values reported by Fares et al. [5] for their individual models next to the model-specific results of this study, without combining the best values of different models into a single row. The comparison is indicative only: the reference study used its own preprocessing, data split, and hyperparameters, which are not fully available; the reference setting was not re-run on the split used here; and no BERT baseline was added. With these caveats, the SGD classifier reached 97.98% accuracy and 98.05% recall, compared with 97.12% and 96.91% for the SVM of Fares et al. [5], and its F1-score of 97.33% compares with 96.74% for their Naive Bayes model; the GRU and the Bi-LSTM with attention reached 97.42% and 96.84% accuracy, compared with 95.49% for their LSTM, and Logistic Regression had a precision of 97.06%, close to the 96.86% of their Naive Bayes model. These differences of 0.2 to 1.9 percentage points are of the same order as the sampling uncertainty reported in Section IV-E and may also reflect the different preprocessing and splits, so they do not establish that the present models are better than those of the reference study.

Three practical insights follow from this work. First, a simple linear classifier trained on a high-dimensional TF-IDF representation was at least as accurate as the recurrent networks in the held-out, cross-validated, and masked comparisons, while being far smaller and easier to deploy; larger neural networks therefore did not lead to better detection on this benchmark. Second, the perturbation benchmark did not reproduce the earlier impression that the attention model is uniquely robust: all models resisted most character-level edits but were vulnerable to appended benign text, and a heuristic post-processing boost did not change this. Held-out accuracy alone would not have exposed this weakness, which supports treating adversarial and false-alarm evaluation as a complement to conventional metrics rather than as an optional extra. Third, the cascade and latency measurements show that a two-stage deployment, in which a lightweight linear classifier filters most messages and an attention-based network inspects the uncertain ones, is feasible in terms of latency, but on this dataset, it trades a small gain in recall for a small rise in false positives rather than improving on the linear classifier alone.

The study has several limitations. First, the evaluation relies on a single English-language dataset that merges several older public corpora without source metadata. The provenance analysis showed that corpus markers such as “enron” and “spamassassinsightings” carry class information and that deleting them or holding out proxy sources lowers performance, but proxy-source holdouts are not a substitute for testing on independent corpora, which was not done; the scores reported here should therefore be read as optimistic estimates for mail from unseen sources. Second, although exact duplicates were removed, 4.0% of the test emails have a near-duplicate in the training split, which may slightly inflate all held-out scores. Third, because of the time available for this revision, the statistical validation used a single repetition of 3-

fold cross-validation rather than 5-fold cross-validation; with only three paired estimates per model, the confidence intervals are wide, and the significance tests have low power, so small differences between models may go undetected. In addition, the deep architectures were not tuned beyond early stopping of the number of epochs, and the reported deep models were trained with one seed. Fourth, transformer-based models such as BERT and RoBERTa could not be fine-tuned because the required GPU resources were not available, so all conclusions apply only to the compared model families and do not describe the current state of the art. Fifth, the perturbation benchmark is synthetic and non-adaptive and does not include LLM-rewritten phishing, and the context-aware boost is a hand-crafted heuristic. Sixth, sequences longer than 150 tokens were truncated, which affects 27.7% of the training emails. Finally, the latency results come from one machine and stored emails and do not establish production-level scalability, and machine-generated phishing continues to evolve, which motivates continued investigation of detection strategies capable of handling increasingly fluent and context-aware phishing campaigns.

To support independent verification, Table III lists every model setting and search space, and the experimental notebook fixes the random seeds, stores the split indices and fold assignments, records the software versions, and exports the trained models together with all result tables used in this study.

VI. CONCLUSION

This study developed a unified, leakage-controlled pipeline for phishing email detection and used it to compare four classical machine-learning models and four recurrent deep-learning models on the same cleaned corpus of 17,085 messages, the same split, and the same metrics. The SGD classifier obtained the highest held-out accuracy (97.98%) and F1-score (97.33%) and made significantly fewer errors than three of its four closest competitors, whereas the Bi-LSTM with attention achieved the highest recall (98.60%) at a higher false-positive rate (4.22%). The five leading models were within about 1.4 F1 points of one another, and cross-validation did not establish a strict ranking among them; with nested tuning, SGD and Logistic Regression were at least as accurate as the best recurrent model. A multi-seed ablation showed no benefit of attention over simpler pooling. Compared with the values reported by Fares et al. [5] on the same dataset, the results are similar or slightly higher, although the comparison is indicative because preprocessing and splits differ.

Beyond the headline scores, three outcomes shape the recommendation. Deleting corpus markers and holding out proxy sources showed that part of the performance reflects corpus provenance, a finding relevant to every study that uses this benchmark. The perturbation benchmark showed that the models resist most character-level edits but are evaded by appended benign text in 10% to 53% of cases, and that the context-aware boost does not change this. The cascade and latency measurements showed that a fast linear first stage with an attention-based second stage operates at a median latency of 0.49 ms per email on a single machine and significantly reduces the errors of the attention model, although it does not significantly outperform the linear classifier alone. Taken

together, the findings identify a well-tuned linear TF-IDF classifier as a strong and efficient choice for this task, with sequence models as optional second-stage inspectors whose added value must be confirmed on independent data.

Future research should focus on overcoming these shortcomings. The framework's effectiveness should first be verified on various datasets and languages in order to distinguish genuine phishing indicators from corpus-induced patterns. It is also important to test it against adaptive attacks and LLM-rewritten phishing, and to repeat the statistical comparison with a repeated design such as 5×2 cross-validation. In addition, fine-tuned transformer baselines such as BERT and RoBERTa should be added under the same protocol, and adversarial training techniques can help the framework keep pace with increasingly sophisticated machine-generated phishing schemes. Putting the two-stage system into a live email stream is one of the most straightforward ways to check whether the results remain stable in practice.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA, for funding this research work through the project number “NBU-FFRGS-2026”.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest regarding the present study.

REFERENCES

- [1] S. Salloum, T. Gaber, S. Vadera, and K. Shaalan, "Phishing email detection using natural language processing techniques: a literature survey," *Procedia Comput. Sci.*, vol. 189, pp. 19–28, 2021, doi: 10.1016/j.procs.2021.05.077.
- [2] W. Han, J. Zhu, C. Zhang, Z. Zhang, Y. Mei, and L. Wang, "Phish-Master: leveraging large language models for advanced phishing email generation and detection," *Appl. Sci.*, vol. 15, no. 22, art. 12203, Nov. 2025, doi: 10.3390/app152212203.
- [3] N. Altwaijry, I. Al-Turaiki, R. Alotaibi, and F. Alakeel, "Advancing phishing email detection: a comparative study of deep learning models," *Sensors*, vol. 24, no. 7, art. 2077, Mar. 2024, doi: 10.3390/s24072077.
- [4] H. Zhang, Y. Shi, M. Liu, L. Chen, S. Wu, and Z. Xue, "A combined feature selection approach for malicious email detection based on a comprehensive email dataset," *Cybersecurity*, vol. 8, Feb. 2025, doi: 10.1186/s42400-024-00309-6.
- [5] H. Fares, N. Mouakkal, Y. Baddi, and N. Hajraoui, "Robust email phishing detection using machine learning and deep learning approach," *Int. J. Commun. Netw. Inf. Secur.*, vol. 16, pp. 370–383, 2024.
- [6] A. Alhuzali, A. Alloqmani, M. Aljabri, and F. Alharbi, "In-depth analysis of phishing email detection: evaluating the performance of machine learning and deep learning models across multiple datasets," *Appl. Sci.*, vol. 15, no. 6, art. 3396, Mar. 2025, doi: 10.3390/app15063396.
- [7] A. Gupta, A. K. Mishra, and K. Arora, "Detecting phishing emails using natural language processing," in *Proc. Int. Conf. Pervasive Comput. Technol. (ICPCT)*, 2025, pp. 234–238, doi: 10.1109/ICPCT64145.2025.10941056.
- [8] M. F. Rabbi, A. I. Champa, and M. F. Zibran, "Phishy? Detecting phishing emails using ML and NLP," in *Proc. IEEE/ACIS 21st Int. Conf. Softw. Eng. Res., Manage. Appl. (SERA)*, Orlando, FL, USA, 2023, pp. 77–83, doi: 10.1109/SERA57763.2023.10197758.
- [9] G. Alkhodhairy and K. Saleem, "Machine learning algorithm for detecting suspicious email messages using natural language processing

- (NLP)," Alexandria Eng. J., vol. 128, pp. 153–165, Sep. 2025, doi: 10.1016/j.aej.2025.04.067.
- [10] A. Al-Subaiey, M. Al-Thani, N. A. Alam, K. F. Antora, A. Khandakar, and S. A. U. Zaman, "Novel interpretable and robust web-based AI platform for phishing email detection," Comput. Electr. Eng., vol. 120, art. 109625, Dec. 2024, doi: 10.1016/j.compeleceng.2024.109625.
- [11] R. Meléndez, M. Ptaszynski, and F. Masui, "Comparative investigation of traditional machine-learning models and transformer models for phishing email detection," Electronics, vol. 13, no. 24, art. 4877, Dec. 2024, doi: 10.3390/electronics13244877.
- [12] S. Sahit, S. Vaishnavi, A. Vaibhav Reddy, and M. Chaitra, "AI sentries: evaluating machine learning models for superior phishing email detection," in Proc. 8th Int. Conf. Comput. Syst. Inf. Technol. Sustain. Solutions (CSITSS), 2024, pp. 1–5, doi: 10.1109/CSITSS64042.2024.10817065.
- [13] N. A. Mohammed, M. A. S. Al-Hitawi, A. A. Alsabhan, A. H. A. Al-Jumaili, M. A. Al-Shibly, O. D. Madeeh, Y. H. Ali, and O. S. F. Shareef, "Recognizing phishing in emails by using natural language processing and machine learning techniques," in Proc. 3rd Int. Conf. Cyber Resilience (ICCR), 2025, pp. 1–7, doi: 10.1109/ICCR67387.2025.11292212.
- [14] L. Franchina, S. Ferracci, and F. Palmaro, "Detecting phishing e-mails using text mining and features analysis," in Proc. 5th Italian Conf. Cybersecurity (ITASEC), CEUR Workshop Proc., vol. 2940, 2021, pp. 106–119.
- [15] A. F. Naswir, L. Q. Zakaria, and S. Saad, "Determining the best email and human behavior features on phishing email classification," Int. J. Adv. Comput. Sci. Appl., vol. 13, no. 8, pp. 175–184, 2022, doi: 10.14569/IJACSA.2022.0130821.
- [16] M. Somesha and A. R. Pais, "Classification of phishing email using word embedding and machine learning techniques," J. Cyber Secur. Mobil., vol. 11, no. 3, pp. 279–320, 2022, doi: 10.13052/jcsm2245-1439.1131.
- [17] A. Mittal, D. Engels, H. Kommanapalli, R. Sivaraman, and T. Chowdhury, "Phishing detection using natural language processing and machine learning," SMU Data Sci. Rev., vol. 6, no. 2, art. 14, 2022.
- [18] L. Naik, B. KS, S. SS, and S. K, "AI-driven email classification and phishing attack detection for secure communication," in Proc. 9th Int. Conf. Comput. Syst. Inf. Technol. Sustain. Solutions (CSITSS), 2025, pp. 1–5, doi: 10.1109/CSITSS67709.2025.11295172.
- [19] D. Jeevitha, P. Venkadesh, S. V. Divya, J. Lingeshwaran, S. Kavinkumar, and S. Hariharan, "Enhanced phishing detection using machine learning, NLP and emerging technologies," in Proc. 8th Int. Conf. Circuit, Power Comput. Technol. (ICCPCT), 2025, pp. 1994–1999, doi: 10.1109/ICCPCT65132.2025.11176674.
- [20] S. Atawneh and H. Aljehani, "Phishing email detection model using deep learning," Electronics, vol. 12, no. 20, art. 4261, Oct. 2023, doi: 10.3390/electronics12204261.
- [21] N. Pimpason, P. Viboonsang, and S. Kosolsombat, "Phishing email detection model using deep learning," in Proc. IEEE Int. Conf. Cyberm. Innov. (ICCI), Chonburi, Thailand, 2025, pp. 1–5, doi: 10.1109/ICCI64209.2025.10987422.
- [22] V. K. Shrivastava, P. Dubey, S. Dixith, A. K. Singh, and Y. S. Kumar, "SmartSort for Outlook: a machine learning approach to automated email classification," in Proc. IEEE Int. Conf. Adv. Comput. Res. Sci. Eng. Technol. (ACROSET), 2025, pp. 1–8, doi: 10.1109/ACROSET66531.2025.11281252.
- [23] C. O. Ugwunna, I. C. Obagbuwa, M. S. Obadina, I. G. Akawuku, and K. C. Ukaoha, "Leveraging large language model on spam email detection and classification," Appl. Comput. Intell. Soft Comput., vol. 2025, no. 1, art. 7032960, 2025, doi: 10.1155/acs/7032960.
- [24] L. Gallo, A. Maiello, A. Botta, and G. Ventre, "2 years in the anti-phishing group of a large company," Comput. Secur., vol. 105, art. 102259, Jun. 2021, doi: 10.1016/j.cose.2021.102259.
- [25] R. Kumar, K. Srjana, P. Sampath, J. C. Patni, P. Agrawal, and S. R. Mallick, "Detecting phishing emails with artificial intelligence: a hybrid machine learning approach," in Proc. Int. Conf. Emerg. Trends Netw. Comput. Commun. (ETNCC), Windhoek, Namibia, 2025, pp. 132–136, doi: 10.1109/ETNCC66224.2025.11299563.
- [26] A. Ahmed, G. Rahman, M. Ahmed, S. B. Fattah, and S. L. Shah, "A cutting-edge approach to phishing email detection using machine learning," in Proc. Int. Conf. Adv. Mach. Intell. Cybersecurity Technol. (AMICT), 2025, pp. 31–36, doi: 10.1109/AMICT65811.2025.11402746.
- [27] H. Himdi and F. Alhayan, "Optimized ensemble stacking approaches to detect Arabic phishing email," J. Eng. Res., vol. 14, no. 1, pp. 776–788, Mar. 2026, doi: 10.1016/j.jer.2025.08.013.
- [28] D. Lappas, P. Karamelas, and G. Fesakis, "Enhancement of phishing email detection with Bayesian networks: a cyber security training module," in Proc. 24th Eur. Conf. Cyber Warfare Secur. (ECCWS), vol. 24, no. 1, 2025, doi: 10.34190/eccws.24.1.3315.
- [29] C. S. Eze and L. Shamir, "Analysis and prevention of AI-based phishing email attacks," Electronics, vol. 13, no. 10, art. 1839, May 2024, doi: 10.3390/electronics13101839.
- [30] M. Weinz, N. Zannone, L. Allodi, and G. Apruzzese, "The impact of emerging phishing threats: assessing quishing and LLM-generated phishing emails against organizations," in Proc. 20th ACM Asia Conf. Comput. Commun. Secur. (ASIA CCS), Hanoi, Vietnam, 2025, pp. 1550–1566, doi: 10.1145/3708821.3736195.
- [31] S. Chakraborty, "Phishing email detection," Kaggle, 2023. [Online]. Available: <https://www.kaggle.com/datasets/subhajournal/phishingemails>
- [32] P. Bountakas, K. Koutroumpouchos, and C. Xenakis, "A comparison of natural language processing and machine learning methods for phishing email detection," in Proc. 16th Int. Conf. Availability, Rel. Secur. (ARES), Vienna, Austria, Aug. 2021, art. 127, pp. 1–12, doi: 10.1145/3465481.3469205.
- [33] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, San Francisco, CA, USA, Aug. 2016, pp. 785–794, doi: 10.1145/2939672.2939785.
- [34] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL-HLT), San Diego, CA, USA, Jun. 2016, pp. 1480–1489, doi: 10.18653/v1/N16-1174.
- [35] C. Nadeau and Y. Bengio, "Inference for the generalization error," Mach. Learn., vol. 52, no. 3, pp. 239–281, 2003, doi: 10.1023/A:1024068626366.
- [36] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," Neural Comput., vol. 10, no. 7, pp. 1895–1923, 1998, doi: 10.1162/089976698300017197.
- [37] S. Holm, "A simple sequentially rejective multiple test procedure," Scand. J. Statist., vol. 6, no. 2, pp. 65–70, 1979.
- [38] E. B. Wilson, "Probable inference, the law of succession, and statistical inference," J. Amer. Statist. Assoc., vol. 22, no. 158, pp. 209–212, 1927, doi: 10.1080/01621459.1927.10502953.