National College of Ireland

MSc Research Project
Master of Science in Cybersecurity

Vatsala Narayan
Student ID: 23201126

School of Computing
National College of Ireland

Supervisor: Liam Mccabe

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Vatsala Narayan

Student ID: 23201126

Programme: MSc. In Cybersecurity **Year:** 2024-25

Module: MSc. Research Project

Lecturer: Liam Mccabe

Submission Due Date: 11/08/2025

Project Title: AI-Powered Phishing Detection: Overcoming Gaussian RBF Kernel Limitations

Word Count: 8922 **Page Count:** 42

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vatsala Narayan

Date: 11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI Acknowledgement Supplement

[Insert Module Name]
[Insert Title of your assignment]

Your Name/Student Number	Course	**Date**

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

AI Acknowledgment

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. One table should be used for each tool used.

[Insert Tool Name]	
[Insert Description of use]	
[Insert Sample prompt]	[Insert Sample response]

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:
[Place evidence here]
Additional Evidence:
[Place evidence here]

AI-Powered Phishing Detection: Overcoming Gaussian RBF Kernel Limitations

Liam Mccabe[1]

[1] National College of Ireland, Mayor Street, IFSC, Dublin 1, Ireland.
Vatsala Narayan - x23201126@student.ncirl.ie

Table of Contents

Video: Video with Presentation & Demo Artifact

Abstract

Phishing attacks are one of the most persistent and critical threats in cybersecurity, affecting nearly 86% of the organizations as seen in the statistics. They exploit vulnerabilities by imitating legitimate sources and trick the users into revealing sensitive and personal information. Nearly 90% of cybersecurity breaches are highly done by taking advantage of human vulnerability.[1] This thesis thus helps us investigate by comparing multiple machine learning methods for dynamic detection of phishing Uniform Resource Locator (URLs). It includes the traditional Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and a hybrid CNN-RNN model. The objective here is to evaluate and compare the performance of these models against the attributes such as the accuracy, generalization, and real-time deployment feasibility. All the models are trained in highly curated phishing datasets which are assessed using metrics that are used for standard classification. A real-time prototype is also built to demonstrate the prediction of the best performing and machine learning models and their feasibility. The research focuses to highlight the importance of deep learning in this complex and targeted method of phishing threats to provide deep insight on how to select one of the most effective architectures for the security of real-world applications that will save the organizations to deal with real time phishing attacks. [2]

Table of Abbreviations

Full Term	Abbreviation
Artificial Intelligence	AI
Convolutional Neural Network	CNN
Long Short-Term Memory	LSTM
Machine Learning	ML
Recurrent Neural Network	RNN
Support Vector Machine	SVM
Radial Basis Function	RBF
Term Frequency-Inverse Document Frequency	TF-IDF
Uniform Resource Locator	URL
United States Dollars	USD

Table of Figures

1. Introduction

1.1 Background and Motivation

Since 2023, the attacks caused the average data breach and the cost increased to 4.45 million United States Dollars (USD), and it has increased by 2.3% it is higher the cost bared in 2022.[1] These attacks are done in a way to deceive users that leads to revealing sensitive information's such as their usernames, passwords, bank credentials, or at times even personal data. Further by impersonating these legitimate entities such in places such as banks, social media platforms, or government services these attacks are initiated. This attack acts as an enabler to bypass the technical defenses such as firewalls and the intrusion detection systems, all by exploiting human psychology more than system vulnerabilities. Even after an increased awareness training and campaigns, the phishing techniques continue to evolve rapidly. The threat actors have started with deployment of more sophisticated, targeted, and complicated campaigns of phishing. Such that they often customized as per the target individuals and organizations. These personalized attacks make the detection more complicated and thereby increasing the chances of attackers evading the traditional defense mechanisms easily. [3]

The traditional detection methodologies such as blacklist filters, rule-based engines, and signature matching—are currently very reactive in nature. Since they are mastered on pre-identified phishing attack patterns they fail to identify to new or unknown threats.[4] Looking at the impact of increasing count of dynamic phishing attacks we urgently need intelligent phishing detection systems that are adaptive of new patterns for detection systems that are intelligent and can proactively identify these malicious content with high precision and recall on real-time. There is research that this gap can be fulfilled with the help of Artificial Intelligence (AI) and Machine Learning (ML) that can act as an enabler to develop and train systems that can learn and adapt heuristically and from vast amounts of phishing and benign data.[5]

The first attempt to detect Phishing attacks using the traditional machine learning techniques was Support Vector Machines (SVMs) with Radial Basis Function (RBF) kernels. It has been extensively explored for detection of phishing attacks. This model is known for its ability to model the non-linear relationships within feature spaces and has reflected a decent performance on a certain number of structured datasets. However, RBF-SVM models have their own set of limitations i.e. it requires a heavy dependency on manual feature engineering, which might require domain expertise to identify and extract meaningful patterns from URLs or webpage content. Also, they lack the flexibility to automatically adapt the new phishing strategi especially those involving deceptive URL embedding or adversarial input techniques that can easily bypass the static filters.[6]

To overcome these limitations the deep learning approaches are found to be a more powerful alternative. Convolutional Neural Networks (CNNs), which were designed for tasks such as image processing, have also adapted to the text and character type input classifications, which includes the URL analysis. The CNN model is capable of learning hierarchical patterns automatically that are found in data. This eliminates the need for any manual intervention. CNNs ability to detect these character-level patterns and spatial dependencies makes them the most suitable model for identifying any suspicious structures within phishing URLs.[7]

Similarly, Recurrent Neural Networks (RNNs), specifically the type of Long Short-Term Memory (LSTM), are successful in modeling the data in sequence. In phishing detection, this model provides the privilege of understanding dependencies across characters in the URL or any text that are sequential. This is helpful in situations where attackers inject the misleading subdomains, nested directories, or any type of deceptive strings into a URL to pretend that it is a legitimate source.[7]

To have an effective impact a hybrid model is built using the strengths of both the models CNNs and RNNs. The architecture of hybrid model combines the CNN layer for local pattern extraction and the RNN layers for modeling the long range of dependencies. This fusion of spatial and sequential learning boosts the defense mechanism for a wide range of phishing techniques, even for those that implement the complex obfuscation methods.[7]

1.2 Research Question

The research answers the following question:

How do deep learning models Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and the hybrid models perform in comparison to an RBF kernel-based SVM for phishing detection in terms of accuracy, generalization, and suitability for real-time deployment?"

1.3 Objectives

The objective of this thesis is to design, implement, and evaluate the AI-based phishing detection system by comparing the performance of these models. The traditional SVM with an RBF kernel, a CNN-based deep learning model, an RNN-based sequence learner, and a hybrid CNN-RNN model.[8] All the mentioned models are trained and tested on a carefully curated phishing datasets, it includes both legitimate and malicious URLs. The evaluation is carried out using the standard classification metrics such as accuracy, precision, recall, and F1-score. Additionally, the thesis explores the deployment feasibility of these trained models in real-time by assessing their performance by URL classification.

The aim of this comparative analysis is to highlight the most effective machine learning method by comparing the traditional and deep learning models. To demonstrate the adaptability and effectiveness of deep learning models to detect different patterns of obfuscated phishing attacks. The understanding gained from this research will contribute to identifying the most resilient model for an automated phishing detection system that is suitable for deployment in robust cybersecurity infrastructures.[7]

1.4 Project Specification

1.4.1 Scope

The scope of the project is to focus on the phishing detection of URLs by implementing the techniques in machine learning and deep learning. The steps include the preparation of the dataset, designing models, training the models, evaluation of metrics, followed by the result comparison of all the models. The topics such as analysis of phishing email analysis, deployment on large-scale or the browser plugin is beyond the

scope of this research. These topics are acknowledged but not included in detail to focus on research objective to evaluate the effectiveness of ML/DL models for URL-based phishing detection with a user interface implementation. The aim is to design a system that scores the phishing recall above 75–80%, as this is an important metrics that helps to decide the feasibility of the real-time deployment.

1.4.2 Performance Targets

The benchmark to evaluate the developed artifact, the performance metrics are defined, as below:

- **Recall should be ≥ 80%** – In order to reduce the count of phishing URLs that are not detected.
- **Precision should be ≥ 80%** – In order to reduce the count false positives identified and for its usability.
- **F1-Score should ≥ 80%** – In order to provide a score of recall and precision which is balanced.

1.4.3 Success Criteria

The research will be successful if the models that are developed score the metrics that id near to or exceed the above mentioned performance boundaries. These purpose of this criteria is to make sure that the research done is able detect the phishing URLs and safe to implemented in the real-time applications.

1.5 Structure of the Report

This thesis is organized into seven main chapters, to address each aspect of the research and contribute to a detailed understanding of phishing detection using by comparing the traditional and deep learning models techniques:

Chapter1: Introduction

Provides a detailed understanding of the rising impact of phishing attacks. It explains the reason, motivation, the research question and the objective.

Chapter 2: Literature Review

The Literature Reviews covers the research done on traditional machine learning approaches such as Support Vector Machines (SVM), and on deep learning models Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and hybrid. This chapter justifies the comparative approach taken in this thesis.

Chapter 3: Methodology

Details the technical implementation done which includes the dataset preparation, preprocessing techniques, feature extraction using TF-IDF, and the model development. The architecture and implementation of four models: SVM, CNN, RNN, and a hybrid CNN-RNN and their performance metrics are explained.

Chapter 4: Implementation

This chapter has the details of environment setup, execution steps, and language and tools implemented for building the phishing detection system.

Chapter 5: Model Evaluation

It displays the results of models tested to evaluate their performance on metrics of accuracy, precision, recall, F1-score, and the confusion matrices. The comparison is done after training the model using the same dataset.

Chapter 6: Real-Time Testing

This chapter describes the functionality of the application based on the trained models in real time. It can assess the models by classifying the seen and unseen URLs, to understand the real-world feasibility.

Chapter7: Conclusion and Future Work

The summarization of the research findings provides a conclusion based on the comparison of the performance of models and answers the research question. The limitations of the research and future enhancements. It includes integration with advanced architectures and more variety of datasets.

2. Literature Review

2.1 Machine Learning for Phishing Detection

The implementation of Artificial Intelligence (AI) and Machine Learning (ML) into phishing detection systems has led the models to learn attack patterns from the larger datasets. This has facilitated the identification of critical phishing attempts. Although the traditional approaches have rule-based and blacklists failed to recognize or trace the new and unseen threats as they are static in nature. In further analysis it was found that machine learning (ML) techniques also provide adaptive mechanisms, that is learning those patterns from large datasets. These patterns are identified from features extracted using the URL structures, domain characteristics, email headers, and the HTML content of the webpages. These are called supervised learning techniques and are proved to be effective by training on labeled datasets which have a combination of both the phishing and legitimate samples in the dataset. Using this process, we are able to generalize and detect the previous unseen attempts of phishing attacks with the help of these learned behavioral patterns.

The steps used for preprocessing data for the phishing detection begins with the extraction of the lexical features from URLs (e.g., length, use of special characters), host-based features (e.g., domain age, WHOIS information), and content-based features (e.g., presence of login forms, obfuscated JavaScript). These selection of features and engineering together improve the classifiers performance, especially when dealing with data that are high-dimensional or noisy in nature. [5]

2.2 Support Vector Machines (SVMs)

The use of Support Vector Machines (SVMs) is majorly for binary classification problems, even in the detection of phishing attacks. SVMs, when implemented with the Radial Basis Function (RBF) kernels, are

capable of handling features relationships that are non-linear in nature. The strength of SVM is in its ability to construct a hyperplane with maximum-margin, this helps to separate the phishing and benign samples with better accuracy in spaces that are linear and inseparable in nature.[8]

SVMs use manual feature engineering, this can sometime limit their effectiveness in identifying or adapting to the new attack vectors that are different from the data they have been trained with. As the domains are evolving so are the phishing attack methods such as, the attackers modify the patterns of URL and their content to evade any detection. In such scenarios the SVMs might struggle if it is not trained again frequently. Additionally, the SVMs interpretability for the complex feature spaces can be limited leading to be a blocker for complex attack types.[6] In comparison to them the deep learning model identifies the representations using raw URLs, which is more flexible and reliable due to its high computational ability.

2.3 Deep Learning Models

Deep learning methodologies are one of the best alternatives to the limitations that are in traditional machine learning methods, especially for tasks that have data that are unstructured or sequential. In detection phishing attacks, the **Convolutional Neural Networks (CNNs)** have shown better precision in their results by learning the spatial and structural patterns automatically all from the raw URL text or web content. This bypasses the need for extensive feature engineering. CNNs are able to capture **local dependencies** through character and word level embeddings, this leads to effective detection of lexical obfuscation and malicious token sequences. However, the CNN models can score less precision in comparison to the traditional ML models such as SVMs, because of overfitting due to imbalanced training. This reflects that the CNNs model can perform better in identifying the attacks, this may result in identification of more false positives in real-time scenarios.

Additionally, **Recurrent Neural Networks (RNNs)**, especially with **Long Short-Term Memory (LSTM)** networks, are best suited to capture the patterns that are **sequential and temporal** in URLs, domain names, and email texts. This makes CNNs and LSTMs maintain context across the data in longer sequences, which makes them ideal for detecting phishing URLs that carry malicious intent, embedded deep within the strings that are long or have deceptive prefixes or suffixes in them. The challenge is that they are resource-intensive for training and inference is slow in comparison to the CNN model, that can be a challenge for the real-time phishing detection. Some studies have found that while RNNs achieve recall comparable to CNNs, they suffer from lower precision, which reduces their reliability in deployment. [30]

The benefit is that both the CNNs and LSTMs can be trained end-to-end, using complex sequential data that allow these models to learn the best range of representations of the data without any manual intervention. The deep learning models require larger datasets that have more computational resources, and the models are prone to overfit if not regularized properly.[7]

2.4 Hybrid Architectures

To enhance the phishing detection more efficiently the advantages of both CNNs and RNNs, from the recent research it is explored that it has more precision in **hybrid CNN-RNN architectures** for phishing detection. These models together combine the **extraction capabilities** of CNNs from **local pattern** with the **sequence modeling** that are **context-aware** in RNNs. For example, a CNN layer can be implemented to detect the frequent input patterns in n-gram, with addition of the features of LSTM layer that helps to capture the dependencies across the identified patterns over time.

These architectures are very effective in identifying the sophisticated phishing attacks even in case of obfuscated or tokens that are misleading and are scattered throughout a URL or the message. By combining both the structural and sequential insights, the hybrid models can improve the identification by generalizing and adding the robustness, especially when dealing with adversarial input or in case of rapidly evolving phishing attack tactics.[7] This also results more latency and requirement of the resource, this can be challenging for large scale real-time phishing detection systems.

2.5 Research Gap

After making significant progress in phishing detection using machine learning, there is a gap which is due to lack of comprehensive comparison of studies that can evaluate the traditional machine learning models and the deep learning architectures using the **consistent experimental conditions**. As most of the existing research focuses on a single model or bunch of datasets, this makes it challenging to come to definitive conclusions about these models performance and application into the real-world deployments. Also, the **lack of benchmarks not being standardized**, variations in the feature sets, and the multiple differences in evaluation metrics block the model's reproducibility and comparison.

Thus, with the help of this research the aim is to address this gap by implementing a **systematic and controlled evaluation** of SVMs, CNNs, LSTMs, and hybrid CNN-RNN models using the datasets that are unified, by preprocessing the pipelines, and common evaluation criteria. The objective is to provide clarity and insight of the trade-offs between the traditional and the deep learning methods that can offer a guidance for model selection in the systems designed for phishing detection which can be implemented for deployment in the environments that are security sensitive. The reviewed literature clearly states that the traditional ML model SVMs is efficient but cannot handle large dynamic data, the CNNs and RNN models capture patterns more clearly, but they too have the performance based challenges, whereas the hybrid model has combined strength but has scalability challenges. This raises the need of an evaluation which is comparative that focuses on accuracy, recall and their real-time feasibility.

3. Methodology

The methodologies section will outline the technical approach that is taken to design the phishing detection system by using machine learning. The initial steps include the preprocessing pipeline, feature extraction techniques, and the specific modeling strategies for two categories of approach that is traditional and the deep learning models. The overall focus is to emphasize reproducibility, robustness, and the considerations of practical deployment.

3.1 Dataset and Preprocessing

The machine learning models are trained using a good dataset to build an efficient system for phishing detection. This analysis is done using the dataset segregated using two labels for the URLs that are phishing and legitimate. The dataset is taken from a reliable source online. [27] The data was first cleaned using a preprocessing pipeline to make it more consistent and ready to use for training the models. These implementations of cleaning and processing are done in the **clean_data.py** module. [10]

3.1.1 Data Cleaning Process

The data cleaning process handling is done using the function **clean_url()** as shown in Fig. 1. It has a few layers for data normalization and sanitization to process each URL in the dataset. These steps are as follows:

- **Removal of Protocol and Subdomain**: The URLs are processed where a few prefixes are deleted to identify the data in the URL uniquely without noise.
- **Converting the data to Lowercase:** To eliminate the case-based redundancy data that is left after removing the protocol and subdomain from the URL are converted in to lowercase, to make the tokenization is consistent.
- **Sanitization of the Character:** The alphanumeric characters and restricted set of structural symbols are retained. This step is to reduce noise. They are URLs that are often used for phishing, for example in the deceptive domain fragments or in the subdirectory paths.
- **Consolidation of the Dots:** The consecutive series of dots (for example ....) are consolidated as one dot in order to maintain the integrity and to avoid the skewed token patterns if any. [10]

The purpose of this standardized preprocessing of the data is to make sure that all models—whether they are statistical or neural in nature, receive inputs that are uniform.

```
def clean_url(url):
    """
    Cleans a URL string by removing common unwanted characters and patterns.

    Parameters:
        url (str): Raw URL

    Returns:
        str: Cleaned URL
    """
    url = url.lower()
    url = re.sub(r'https?://', '', url)
    url = re.sub(r'www\.', '', url)
    url = re.sub(r'[^\w./-]', '', url)
    url = re.sub(r'\.+', '.', url)
    return url.strip()
```

Fig. 1 The data cleaning process using the **clean_url()** function.

3.1.2 Normalization and Encoding of the Label

If any of the labels in the original dataset are in consistent with their label flagging then this needs to be normalized (e.g., "phishing" vs. "1"). The preprocessing function therefore helps to normalize all the labels by converting the strings to lowercase and followed by which the **LabelEncoder** are implemented from the Scikit-learn library to convert into binary form that is:

- 0: Legitimate URL
- 1: Phishing URL

To make sure that the mapping is correct (i.e., phishing = 1), there is a validation step to ensure that there is a class alignment which is consistent even if the dataset's class order might not be predictable. [11]

3.1.3 Optional Data Augmentation

To simulate the behavior and to improve the generalization across these unseen phishing tactics, synthetic URLs can be injected into the training dataset. This can be done by enabling the parameter **augment=True**. The purpose is to mimic the real-world phishing attack patterns by using synthetic examples. This is done by including the brand names that are trusted and using it as an easy to deceptive strategy. Examples include the following:

- login-recovery-info.repl.co
- secure-pay-verification.xyz
- verify-id2025.github.io

The purpose of the augmentation is to enhance the model training by exposing it to various phishing behavior types. This will help us attain resilience from the novel attacks. [12]

3.1.4 Shuffling and Reproducibility

After the cleaning and augmentation, the dataset is shuffled by using the **sklearn.utils.shuffle()** where fixed **random_state** is set to 42. This is done to make sure that the order is consistent across training the given session by mitigating the ordering bias. [13]

3.1.5 Splitting of Dataset

The data cleaned in the dataset is partitioned into two parts. That is for training and testing the subsets using a ratio of 80:20 respectively. The purpose of the stratification is to make sure that the label distribution is consistent across the splits done. This functionality is handled by one of the utility functions **split_data()** from the **helper.py**, it helps to wrap **train_test_split()** forbetter reproducibility and modularity as shown in Fig. 2. [14]

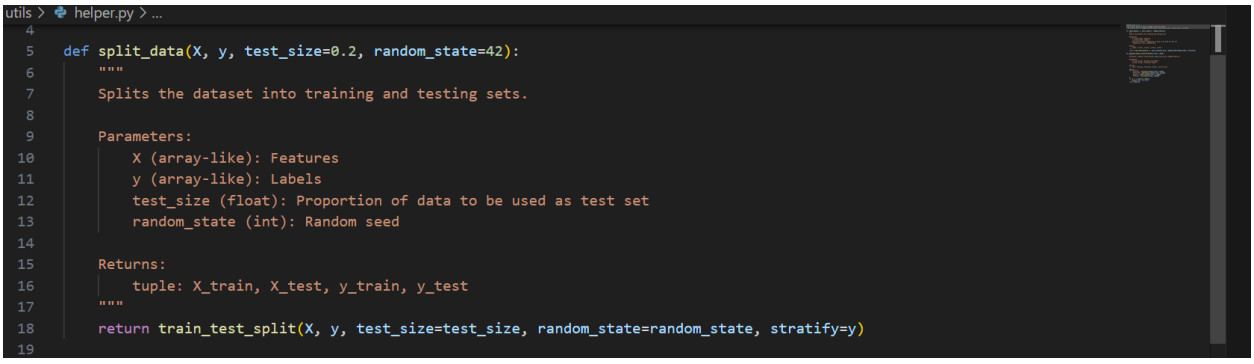


Fig. 2 Python function **split_data()** for stratified dataset partitioning

3.1.6 Ethical Consideration

No personal or sensitive information is used in this project. All testing was done locally with made-up (synthetic) messages, so we fully followed academic rules and data protection standards. Every tool and library we used was open-source and has been properly credited.

3.2 Feature Engineering with Term Frequency-Inverse Document Frequency (TF-IDF)

The step feature extraction is used to convert textual data for example raw URLs to train the machine learning models. The traditional model Support Vector Machines (SVMs) are not designed to process raw strings, thus these textual inputs are first converted into structured numerical presentations. The method **Term Frequency-Inverse Document Frequency (TF-IDF)** is implemented for the effective transformation with the help of class **URLFeatureExtractor** which is in the **feature_extraction.py** module. [15]

3.2.1 TF-IDF Encoding for Character N-Grams

The vectorizer TF-IDF operates on **character-level n-grams** that ranges from 3 to 5 characters, char_wb analyzer is implemented as shown in Fig. 3. The model captures subsequences from 3 to 5 characters **within the word boundaries**, this helps it to identify the local patterns in the URL for example the brand impersonations ("paypa", "googl", "logi"), the domain tricks (".xyz", "-secure") or any other patterns that are repetitive ("login-login"). The methodology behind this approach is the following:

- **Robustness to Tokenization**: The character n-grams capture the phishing indicators that are embedded in the obfuscated words.
- **Sparsity Control**: When the length of n-grams ranges from 3–5, it helps to attain the computational efficiency and expressiveness in the data.
- **URL Format Compatibility**: URLs are made from details such as the domain names, slashes, and tokens the word-based segmentation are not suitable for such URLs. [16]

```
def __init__(self, max_features=1000):
    self.vectorizer = TfidfVectorizer(analyzer='char_wb', ngram_range=(3, 5), max_features=max_features)
```

Fig. 3 The **TF-IDF vectorizer** operating on character-level n-grams (range 3–5) using the char_wb analyzer

3.2.2 Fit and Transform Logic

The method **fit_transform()** is implemented on to the training dataset to performs the following actions:

1. **Fit**: To learn the vocabulary. The frequent used characters n-grams in the training URLs.
2. **Transform**: The URLs are converted into a vector that represents the importance of the learned n-grams using the TF-IDF weight.

The method **transform()** is used to maintain a consistent vocabulary that was learned using the training set as shown in Fig. 4. This separation avoids leakage of data and maintains generalization. [17]

```
def fit_transform(self, X_train):
    """
    Fits the TF-IDF vectorizer and transforms the training data.

    Parameters:
        X_train (pd.Series or list): List of URLs

    Returns:
        sparse matrix: Transformed features
    """
    return self.vectorizer.fit_transform(X_train)
```

Fig. 4 The fit and transform logic for TF_IDF

3.2.3 Dimensionality Control

The value of the parameter **max_features** is set to **1000.** The informative n-grams spaces to the top 1000. Further helping with the following:

- The overfitting is minimized by eliminating the noisy feature
- The training and inference are speed up, when dealing with high-dimensional input data, for example text.

Models such as SVM require sorted data dimensions, as it impacts the efficiency of the memory and the space size impacts the convergence. [18]

3.2.4 Persistence and Reusability

The pickle module is used by the object vectorizer as it makes it more persistent. The methods **save_vectorizer()** and **load_vectorizer()** are implemented to save and load the data as shown in Fig. 5 and 6. This helps with the following:

- **Real-time prediction systems deployment.**
- **The inference requires consistency**, the URLs to be tested and the unseen data are processed with the same feature mappings method as done during the training phase.

The vectorizer is saved immediately after the training. This makes the pipeline reproducible and integration with the live environments makes the scalability easier for the phishing detection. [19]

```
def save_vectorizer(self, filepath):
    """
    Saves the TF-IDF vectorizer to disk.

    Parameters:
        filepath (str): File path to save the vectorizer
    """
    with open(filepath, 'wb') as f:
        pickle.dump(self.vectorizer, f)
```

Fig. 5 Saving the Vectorizer

```
def load_vectorizer(self, filepath):
    """
    Loads a saved TF-IDF vectorizer.

    Parameters:
        filepath (str): File path from where to load the vectorizer
    """
    with open(filepath, 'rb') as f:
        self.vectorizer = pickle.load(f)
```

Fig. 6 Loading the Vectorizer

3.3 Model Architectures

This section describes the four models that are implemented for phishing detection: a traditional Support Vector Machine (SVM), a Convolutional Neural Network (CNN), a Recurrent Neural Network (RNN), and a Hybrid CNN-RNN model. Each model has a unique capability, and each of their architecture has unique

strengths that learns from structured or sequential data which is trained using a standard setup build for fair comparison.

3.3.1 Support Vector Machine (SVM)

The linear classifier SVM model is used. It implements the **LinearSVC** in **Scikit-learn**, trained on extracted TF-IDF features using large URL datasets. The implementation is done using a linear kernel as it is more efficient and performs better when paired with character n-gram TF-IDF features.

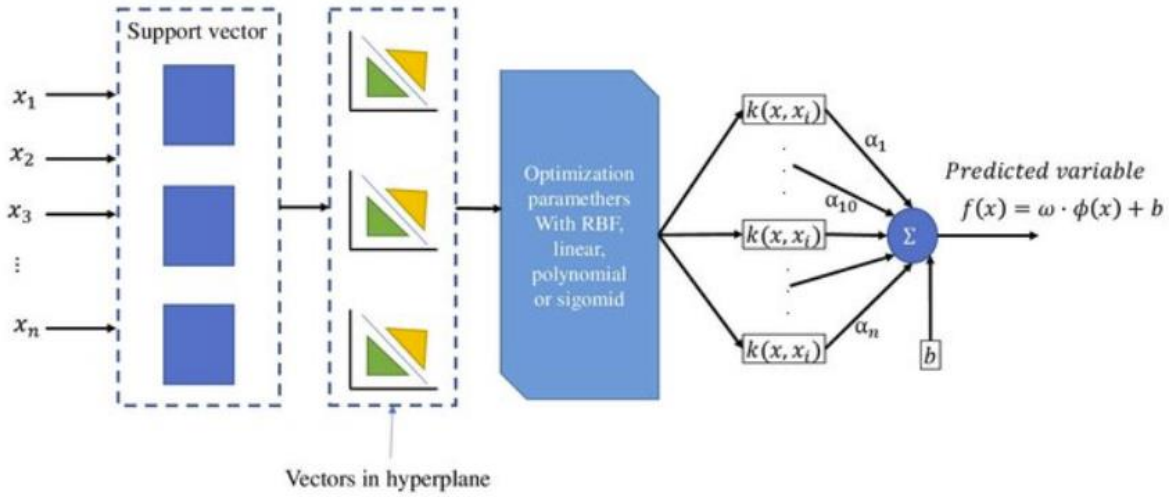


Fig. 7 Architecture of a support vector machine (SVM) model [29]

Key Characteristics: The characteristics of the SVM model are as shown in Fig. 7

- **Model Type**: Linear Support Vector Machine (LinearSVC)
- **Feature Input**: Sparse matrix from TF-IDF (char-level 3–5 n-grams)
- **Training Objective**: To maximize the margin between legitimate and phishing URL classes

The trained model is saved in the **.pkl** file using **joblib**. The performance evaluation is done using attributes such asaccuracy, precision, recall, F1-score, and a confusion matrix. [20]

3.3.2 Convolutional Neural Network (CNN)

CNNs model is good at learning from local spatial patterns that are found in data. This makes it easier for them to detect the substrings and the patterns in URLs that indicate any phishing evidence. This model recognizes each URL in the dataset as a sequence of characters, where each character is embedded into a vector space before passing through the CNN filters.

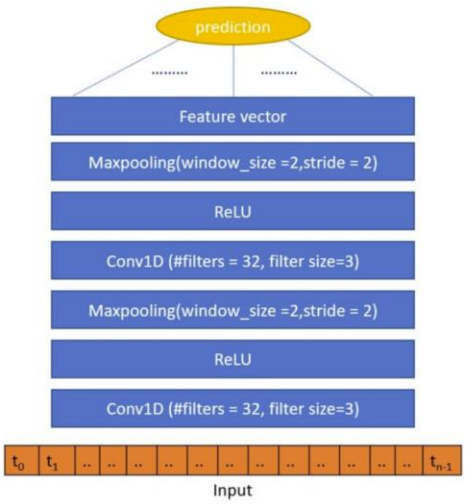


Fig. 8 Architecture od Convolutional neural network (CNN) model [28]

Model Architecture: The model architecture is described below in Fig. 8.

1. **Embedding Layer**: It maps the characters using dense vectors (embedding_dim = 64)
2. **Conv1D Layer**: 28 filters of size 5 using ReLU activation are applied
3. **GlobalMaxPooling1D**: To reduce the output to the most prominent features
4. **Dense Layers**: A layer connected layer with 64 neurons and a dropout regularization
5. **Output Layer**: A single neuron used with the sigmoid activation for binary classification

Training Features:

- **Loss Function**: Binary cross-entropy
- **Optimizer**: Adam
- **Batch Size**: 128
- **Epochs**: 5
- **Class Weights**: compute_class_weight is used to address the class imbalance
- **Threshold**: The threshold of 0.4 is used to improve phishing recall decisions[21]

3.3.3 Recurrent Neural Network (RNN)

The RNN model helps the LSTM (Long Short-Term Memory) layers to capture dependencies between URL characters sequentially. This model detects subtle and distributed phishing evidence, for example the abnormal subdomain placements or the directory paths that are disguised.

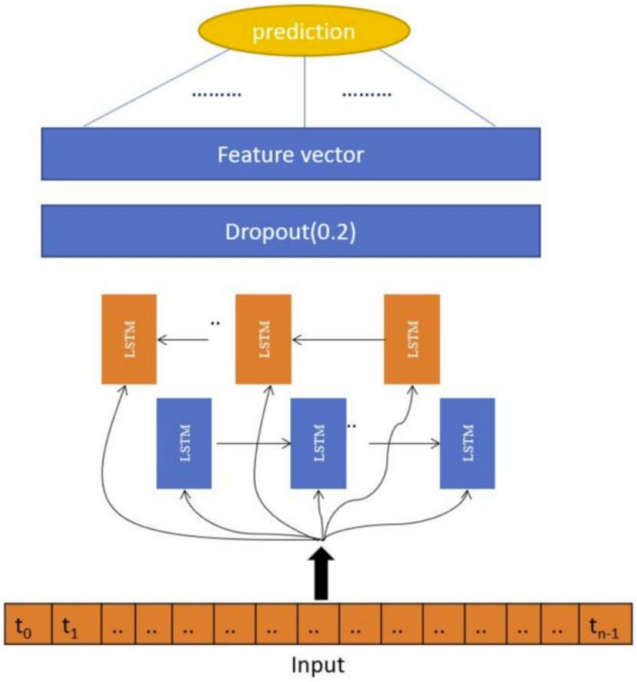


Fig. 9 Architecture of recurrent neural network (RNN) model [28]

Model Architecture: The model architecture is described below in Fig. 9.

1. **Embedding Layer**: Maps characters to a vector space, same as CNN
2. **LSTM Layer**: 128 units they process the sequences in character
3. **Dense Layers**: Has dropout and ReLU-activated layers
4. **Output Layer**: For binary output sigmoid activation is used

The RNN model is also trained line CNN using **model.fit()** and same set of metrics are evaluated. [22]

3.3.4 Hybrid CNN-RNN Model

This architecture of the **Hybrid CNN-RNN Model is made in** combination using capabilities of both CNN and the RNN model. The local features learning is taken from CNNs and the sequential modeling of that of RNN. This hybrid model begins with the pattern extraction using its convolutional layer and then models the sequential relationship using its LSTM layer. This allows the network to take note of patterns that are spatial and temporal.

Model Architecture: The model architecture is described below in Fig. 10.

1. **Embedding Layer**
2. **Conv1D Layer**: 64 filters, kernel size = 5
3. **MaxPooling1D Layer**: Reduces dimensionality
4. **LSTM Layer**: 64 units
5. **Dense + Dropout Layers**

6. **Sigmoid Output**

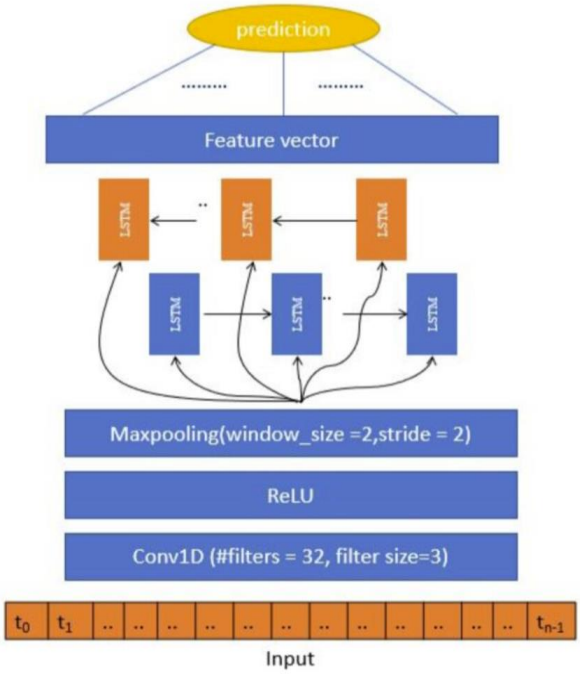


Fig. 10 Architecture of hybrid model [28]

This model reflects a high accuracy in the real-world applications because it has both feature abstraction and context retention in combination. It detects well when the phishing URLs have illegitimate tokens implemented in deceptive pattern. [23]

Model Storage:

The outputs of all the three models CNN, RNN and the Hybrid CNN-RNN are saved in .keras format using model.save(). The performance evaluation is done using attributes such asaccuracy, precision, recall, F1-score, and a confusion matrix.

Model Training Summary:

Model	Type	Feature Input	Train Time	Pros	Cons
SVM	Linear ML	TF-IDF (n-grams)	Fast	Simple, Fast	Poor generalization to obfuscation
CNN	Deep Learning	Embedded URL Sequences	Medium	Good at local patterns	Weak for long-term dependencies
RNN	Deep Learning	Embedded URL Sequences	Medium-High	Learns sequences	Slower, can overfit
Hybrid	Deep Learning	Embedded URL Sequences	High	Combines both strengths	More complex, higher resource usage

4. Implementation

4.1 Environment Setup

The implementation is done using the Python-based environment with **Visual Studio** integrated development environment (IDE). A file requirements.txt is to make sure that the environment setup is easy by making the installation of dependencies quick. The key packages used, are as listed below:

- **pandas (2.0.3)** and **numpy (≤1.24.3)** are implemented for data manipulation and its preprocessing.
- **tensorflow (2.13.0)** and **scikit-learn (1.3.2)** are implemented for model implementation, training, and evaluation.
- **matplotlib (3.7.3)** and **seaborn (0.12.2)** are implemented for visualization of data and its performance plotting.
- **streamlit (1.27.2)** is implemented for development of the user interface.
- **joblib (1.3.2)** is implemented for persistence and reusability of the model.
- **tqdm (4.66.1)** is implemented for monitoring the progress while training.
- **nltk (3.8.1)** implemented for text preprocessing and tokenization where applicable.

Visual Studio IDE is used as it has an integrated debugging with version control, helpful for the iterative experimentation on every model. The requirements file is to make sure that the development and deployment environments is consistent, to validate the real-time feasibility of the models.

4.2 Execution Steps

Below are steps to set up the environment, train models, and run real-time phishing URL detection:

Step 1: Setup Python Virtual Environment
Create a virtual environment: python -m venv phishing-env
Activate the environment: phishing-env\Scripts\activate

Step 2: Install Required Dependencies

Install all required libraries from the requirements.txt file: pip install –r requirements.txt

Step 3: Train Models
The file main.py is for model training and evaluation **python main.py**

Step 4: Set the environment path using the command **$env:PYTHONPATH = "."**
Step 5: Launch the Real-Time Detection App **streamlit run ui/streamlit_app.py**

The structured workflow of the implementation, is as follows:
(1) Collection and preprocessing of the dataset
(2) Design and training of the model
(3) Evaluation using the classification metrics
(4) Local deployment of the prototype.

This approach was used to make sure that validation is done at each step separately before integration. For example, the preprocessing validated first to confirm the balanced representation of URLs with proper ration phishing and legitimate, an important step to avoid any bias in the model training. The stepwise process is chosen in the end-to-end pipeline to have flexibility while comparing the individual models (SVM, CNN, RNN, Hybrid) under any conditions.

4.3 Languages and Tools Used

Category	Tool / Version Used
Programming Language	Python 3.10
Core Libraries	pandas=2.0.3, numpy<=1.24.3
Deep Learning	tensorflow=2.13.0, keras (bundled with TensorFlow)
Machine Learning	scikit-learn=1.3.2, joblib=1.3.2
Feature Extraction	TF-IDF (Scikit-learn), Tokenizer (Keras)
Data Handling & Preprocessing	pandas, numpy, re (Regular Expressions), nltk=3.8.1
Visualization	matplotlib=3.7.3, seaborn=0.12.2
Utilities	tqdm=4.66.1
Deployment	streamlit=1.27.2 (UI)

Python is the programming language as it has strong foundation for implementation of machine learning (ML) and deep learning (DL) as it has wide range of libraries. The flexibility and the community support make it best suitable for the research.

The development environment is done using the **Visual Studio**, as it has integrated debugging, code management, and supportive features for Python projects. The dependency is managed using a requirements.txt file to ensure reproducibility and portability across systems.

To implement the machine and deep learnings **TensorFlow (2.13.0)** and **Scikit-learn (1.3.2)** are used. TensorFlow is used for deep learning as it helps implement a GPU acceleration and flexible architecture design, the Scikit-learn provides utilities for SVM model. For the data manipulation and preprocessing

pandas (2.0.3) and **numpy (≤1.24.3 were used)**, this helped handling of large datasets in a structured manner.

For visualization matplotlib **(3.7.3)** and **seaborn (0.12.2)**, to provide details of the dataset characteristics and the performance of the models. **tqdm (4.66.1)** is to monitor the progress of training, while **nltk (3.8.1)** is for the preprocessing for example the tokenization.

For the deployment and demonstration purpose, **Streamlit (1.27.2)** is used to create a lightweight user interface. Streamlit is chosen as it is simple, easy to integrate with Python. **joblib (1.3.2)** is used to serialize the trained models to be reused in the interface.

This combination of languages and tools is chosen to balance research requirements along with the real-time feasibility, to make sure that the artifact developed can be tested both in controlled situations using known test data and in a real-time environment.

5. Model Evaluation

5.1 Confusion Matrix

5.1.1 Support Vector Machine (SVM)

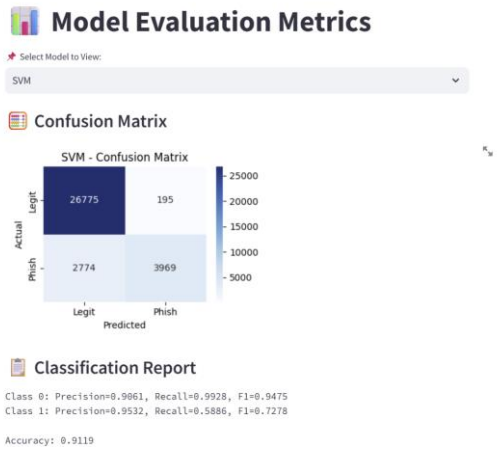


Fig. 11 SVM Model Evaluation Metrics

- **True Positives (TP)** = 3,969
- **True Negatives (TN)** = 26,775
- **False Positives (FP)** = 195
- **False Negatives (FN)** = 2,774

5.1.2 Convolutional Neural Network (CNN)

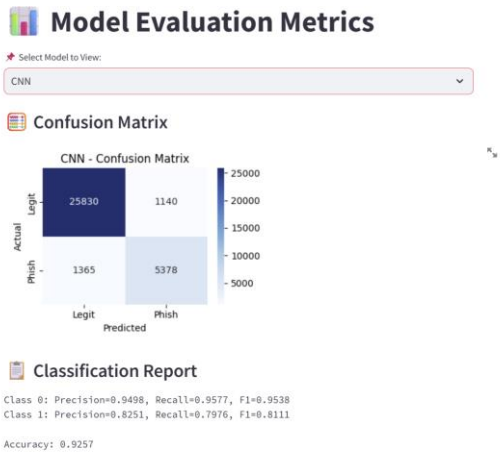


Fig. 12 CNN Model Evaluation Metrics

- **True Positives (TP)** = 5,378
- **True Negatives (TN)** = 25,830
- **False Positives (FP)** = 1,140
- **False Negatives (FN)** = 1,365

5.1.3 Recurrent Neural Network (RNN)

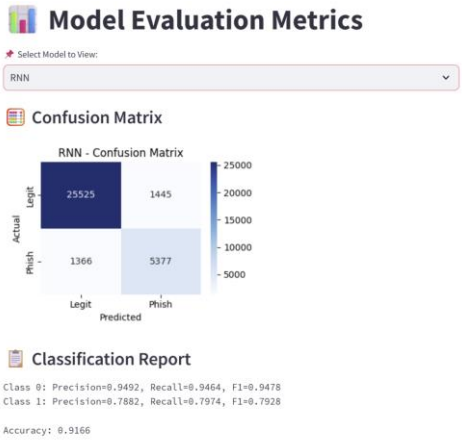


Fig. 13 RNN Model Evaluation Metrics

- **True Positives (TP)** = 5,377
- **True Negatives (TN)** = 25,525
- **False Positives (FP)** = 1,445
- **False Negatives (FN)** = 1,366

5.1.4 Hybrid CNN-RNN Model

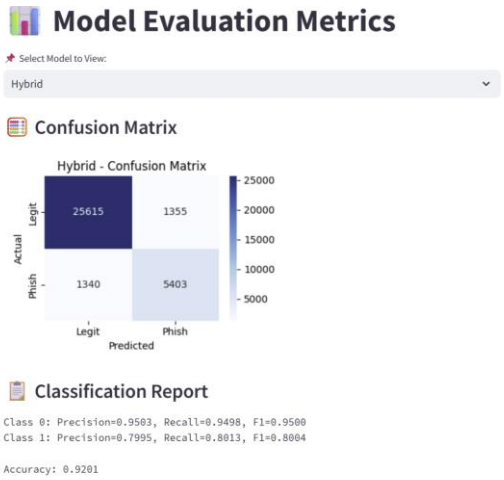
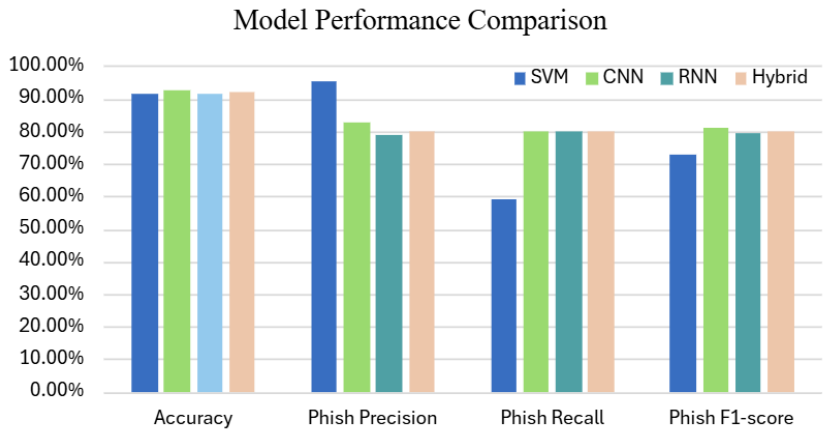


Fig. 14 Hybrid CNN-RNN Model Evaluation Metrics

- **True Positives (TP)** = 5,403
- **True Negatives (TN)** = 25,615
- **False Positives (FP)** = 1,355
- **False Negatives (FN)** = 1,340

5.2 Comparison of the Classification Report



Metric / Model	SVM	CNN	RNN	Hybrid
Accuracy	91.19%	**92.57%**	91.66%	92.01%
Phish Precision	**95.32%**	82.51%	78.82%	79.95%
Phish Recall	58.86%	**79.76%**	79.74%	80.13%
Phish F1-score	72.78%	**81.11%**	79.28%	80.04%

- **CNN** has the overall balance for phishing detection as it has the highest phishing F1-score.
- **Hybrids** performance has a better recall than CNN, but the phishing precision is lower.
- **RNN** is close to Hybrid in terms of performance but has less precision.
- **SVM** has good precision but is weak in recall for phishing, it may miss many phishing URLs.

5.3 Critical Interpretation of Results

While the CNN model has achieved the highest phishing F1-score (81.11%), the result can be interpreted in relation to its lower precision score (82.51%) in comparison with the SVM that has precision score (95.32%). These scores showcase that CNNs are more efficient in capturing the phishing attempts made, but it detects more false positives. The Hybrid of CNN-RNN models had an improved recall of (80.13%) in comparison to CNN but the precision is lower (79.95%), this reflects the trade-off of maximising detection and at the same time avoiding unnecessary alerts. The RNN, has scored balanced but the overall performance is lower in comparison to the CNN and Hybrid models. The comparison of scored metrics shows that the deep learning models have a better generalisation in detection of the phishing URLs. But a careful consideration should be given to the metrics such as precision-recall trade-offs while selecting a model for real-time implementation.

5.4 Statistical Perspective

The performance metrics have a huge impact on the size of the dataset implemented for training. The dataset contains 168,560 samples, which is split in the ratio of 80:20. The evaluation metrics are calculated on the 20% test set separated from the original dataset split (80:20) which is uploaded for training the models. This results in approximately 26,970 legitimate URLs and 6,742 phishing URLs in the evaluation set created. Which means that even a 1% change in the recall or precision will result in tens or hundreds of URLs being classified wrongly. For example, the recall of CNN's 79.76% corresponds to 1,365 phishing URLs being missed, whereas the Hybrid models has a slightly higher recall of 80.13% this can reduce the number to 1,340. Although these differences are small in when calculated in percentage, the actual number of phishing URLs being missed or legitimate URLs being wrongly correctly flagged has a huge impact on real-time deployment. These types of observations flag the need to balance recall and precision metrics to calculated very carefully, rather than just focusing on one particular metric.

5.5 False Positive Analysis

The detection of the false positives is one of the critical considerations in the detection of phishing attacks, as excess of warnings can result in the "alert fatigue" and this might reduce the confidence of the users in the system. In the evaluation performed, the false postive rate of recorded for SVM is lowest, this can lead to misclassification of only 195 legitimate URLs (≈0.72% of the legitimate test set). However, its recall score is 58.86%, which means it failed to identify a large proportion of phishing URLs. In contrast the CNN and Hybrid models, have higher false positive rates which is of about 4.23% and 5.02% respectively, whereas the RNN model produced the highest score of about 5.36%. Although these false positive rates are higher, they have much stronger recall values (≈80%) scored by the deep learning models. From the perspective of the real-world security, in such instances tolerating slightly higher false positive rate is preferable rather than allowing phishing attempts to go completely undetected. Thus, the deep learning model CNN and Hybrid have a safer balance for deployment despite they have higher false positive counts.

5.6 Evaluation Summary

The evaluation of the trained models done using the 20% test set demonstrate that the deep learning models, especially the CNN and Hybrid architectures, have performed in alignment with the predefined thresholds (≥80% for recall, precision, and F1-score). The precision of SVM scored is the highest (95.32%) with the lowest false positive rate (0.72%), but since it has a weak recall (58.86%) it is unsuitable for real-time phishing detection, as many phishing URLs will remain undetected. Whereas in contrast, the CNN and Hybrid models consistently scored the recall and F1-scores around 80%, thus validating the effectiveness in detection of most phishing attempts. Although the deep learning models have the highest false positive rates of (4–5%), this trade-off is still acceptable in context of security as missed attacks have a greater risk than the occasional false alarms. Overall, CNN and Hybrid models showcase the strongest alignment with the objective of the project for a reliable, safe and practical real-time phishing URL detection.

6. Real-Time Prediction Testing

All the models are validated in real-time scenarios with test data that are both known and unknown to the model. The unknow data are legitimate and phishing URLs and the known data is the dataset which was utilized to train the models. The real time testing represents the following result:

1. The SVM model failed to recognize both the unknow phishing and the legitimate URLs.
2. The CNN, RNN and Hybrid models were able to detect the unknow phishing and legitimate URLs successfully.
3. The SV, CNN, RNN and Hybrid models recognized both the known phishing and the legitimate URLs successfully.
4. Test Data Used
   - **Unknown Phishing URL** - https://tweurp.kcrjdrxxs.es/KVdw9@SuEPo2g
   - **Unknown Legitimate URL** - https://www.google.com/
   - **Known Phishing URL** - http://www.kueronekayacntncojp.kueronekayacoto.rzyddn.top/ai/authenticated=true&openid%2fgp%2fsignin%2fx%26i%3da%26oauth%3dm%26i%3fie%3dutf8%26ref_%3drhf_custrec_signin127763c3de8b2310c4b3bb96ddcd6822ac65e2ab
   - **Known Legitimate URL** - https://www.stardog.com

Note: The screenshot of the result is in the Appendices.

Scenarios	Observation
Validate the **SVM** model using an **unknown Phishing** URL.	The SVM model **failed to recognize** the unknown Phishing URL
Validate the **CNN** model using an **unknown Phishing** URL.	The CNN model **successfully recognized** the unknown Phishing URL
Validate the **RNN** model using **unknown Phishing** URL.	The RNN model **successfully recognized** the unknown Phishing URL
Validate the **Hybrid** model using an **unknown Phishing** URL.	The Hybrid model **successfully recognized** the unknown Phishing URL
Validate the **SVM** model using **unknown Legitimate** URL.	The SVM model **failed to recognize** the unknown Legitimate URL
Validate the **CNN** model using **unknown Legitimate** URL.	The CNN model **successfully recognized** the unknown Legitimate URL

Validate the **RNN** model using **unknown Legitimate** URL.	The RNN model **successfully recognized** the unknown Legitimate URL
Validate the **Hybrid** model using **unknown Legitimate** URL.	The Hybrid model **successfully recognized** the unknown Legitimate URL
Validate the **SVM** model using a **known Phishing URL**.	The SVM model **successfully recognized** the known Phishing URL
Validate the **CNN** model using a **known Phishing URL.**	The CNN model **successfully recognized** the known Phishing URL
Validate the **RNN** model using a **known Phishing URL.**	The RNN model **successfully recognized** the known Phishing URL
Validate the **Hybrid** model using a **known Phishing URL**.	The Hybrid model **successfully recognized** the known Phishing URL
Validate the **SVM** model using a **known Legitimate URL**.	The SVM model **failed to recognize** the known Legitimate URL
Validate the **CNN** model using a **known Legitimate URL.**	The CNN model **successfully recognized** the known Legitimate URL
Validate the **RNN** model using a **known Legitimate URL.**	The RNN model **successfully recognized** the known Legitimate URL
Validate the **Hybrid** model using a **known Legitimate URL.**	The Hybrid model **successfully recognized** the known Legitimate URL

7. Conclusion and Future Work

7.1 Conclusion

This research compares four approaches for the URL-based phishing detection: RBF-style SVM is implemented with TF-IDF char n-grams, a Convolutional Neural Network (CNN), a Recurrent Neural Network (RNN, LSTM-based), and a hybrid CNN–RNN Model. All models are evaluated with identical steps such as preprocessing, feature pipelines, and test data for a fair comparison.

Key findings:

- **Deep learning models have outperformed the traditional model SVM in generalization of the unseen phishing URL patterns.** The CNN, RNN, and Hybrid models detect more phishing samples. This indicates a higher recall in comparison to the SVM, this shows high **precision** of 95.3% but a low **recall** of 58.9%. Although the SVM's precision is high and it means few false alarms, it still misses many phishing URL's — which cannot be missed in security applications as it's costly.
- **CNN is an all-round performer.** CNN has the highest accuracy of 92.57% and a good phishing F1-score of 81.11%, with a strong balance of precision of about 82.51% and recall of 79.76%. CNNs can learn local characters and the patterns in substring from the URLs, this reduces the efforts of manual feature engineering.
- **The hybrid model has a small precision drop for a higher recall.** The Hybrid model attains 92.01% of accuracy with phishing precision of 79.95% and recalls of 80.13% (F1 = 80.04%). The results reflect that hybrid model has effectively combined the CNN models local-pattern extraction and RNN models sequence modeling, that captures the distributed or obfuscated phishing tokens better.

- **RNN model provides a good recall and lower precision as compared to the CNN and Hybrid models.** The recall for **RNN model is approximately 79.7%. This is close to CNN and Hybrid but has** a precision of 78.82%. They catch many phishing instances but give results that are more false positives.
- **Practical implication:** To maximize the recall, **Hybrid** and **CNN models** are preferred. As in the case of **SVM** model, the caveat is that many phishing instances will be missed.
- Answer to the research question: **Deep learning models (CNN, RNN, Hybrid) have showcased superior accuracy and generalization as compared to the RBF-kernel SVM for the detection of phishing URL.** The CNN model has the best balance of metrics, whereas the Hybrid model has slightly improved recall with a small difference in precision. The SVM is computational and efficient but lacks robustness against novel phishing patterns.

7.2 Limitations

1. **Dataset Biasness:** The quality and diversity of the dataset define the efficiency of the results. If the dataset has a few phishing techniques missing in it, the performance estimates may not be optimistic.
2. **Class imbalance effects:** The class weighing imbalance data can impact the metrics and have higher accuracy after missed minority-class samples.
3. **Adversarial robustness:** The models were not evaluated against complex URLs or poisoning attacks.
4. **Resource & latency constraints:** Th computation of deep models require enhanced memory, inference latency. The model size was not trained exhaustively in more advanced computation.
5. **Threshold sensitivity:** The threshold chosen is 0.4, it can affect the recall/precision decision making; the thresholds can vary based on the operational needs.
6. **Explainability and interpretability:** The interpretability of the Deep models is less than SVMs, it can affect the performance in high and environments.

7.3 Future Work & Recommendations

1. **Expand datasets & benchmarks:** Have a collection of diverse and time-stamped phishing datasets that has information fo example different regions, industries, languages.
2. **Adversarial testing:** The model should be trained against **hostile training** and build **resistance from data poisoning. [24]**
3. **Transformer-based models & multimodal features:** Experiment with lightweight transformer encoders (BERT variants) or fuse URL structure with WHOIS, hosting, and page-content features for richer representations. [25]
4. **Operational deployment & monitoring:** Implement a continuous learning pipeline, and for quick detection in production to maintain the effectiveness of the model as phishing tactics change over time.
5. **Combine URLs and Contextual Features:** Train the model on the combined data such as the structural URL features with contextual text embeddings (such as BERT) to build a more enhanced phishing detection system.[26]
6. **Evaluation of thresholds:** Tune thresholds for targeted points by maximizing recall at acceptable false-positive rate.

8. References

[1] U. Oner, O. Cetin, and E. Savas, “Human factors in phishing: Understanding susceptibility and resilience,” *Computer Standards & Interfaces*, vol. 94, Art. no. 104014, Apr. 2025, doi: 10.1016/j.csi.2025.104014.

[2] Y. Lai, “A comparison of traditional machine learning and deep learning in image recognition,” *Journal of Physics: Conference Series*, vol. 1314, Oct. 2019, doi: 10.1088/1742-6596/1314/1/012148.

[3] M. S. Kheruddin, M. Zuber, “Phishing attacks: Unraveling tactics, threats, and defenses in the cybersecurity landscape,” Jan. 2024, doi: 10.22541/au.170534654.48067877/v1.

[4] R. B. Basnet, A. H. Sung, and Q. Liu, “Rule-based phishing attack detection,” Jul. 2011.

[5] K. S. Araneta and N. A. Julasbi, “The role of artificial intelligence detecting and preventing phishing email,” Dec. 12, 2024. [Online]. Available: https://www.researchgate.net/

[6] E. S. Shombot, G. Dusserre, R. Bestak, and N. B. Ahmed, “An application for predicting phishing attacks: A case of implementing a support vector machine learning model,” *Cyber Security and Applications*, vol. 2, Art. no. 100036, Jan. 2024, doi: 10.1016/j.csa.2024.100036.

[7] Er. K., “A comprehensive literature review on phishing URL detection using deep learning techniques,” *Journal of Cyber Security Technology*, pp. 1–29, Jul. 2024, doi: 10.1080/23742917.2024.2378552.

[8] F. Salahdine, Z. El Mrabet, and N. Kaabouch, “Phishing attacks detection: A machine learning-based approach,” in *Proc. IEEE Int. Conf. Cybersecurity and Privacy*, Dec. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9666627

[9] P. Chinnasamy, P. Krishnamoorthy, K. Alankruthi, T. Mohanraj, B. Santhosh Kumar, and L. Chandran, “AI enhanced phishing detection system,” in *Proc. IEEE Int. Conf. INCoS*, Mar. 2024, doi: 10.1109/incos59338.2024.10527485.

[10] GeeksforGeeks, “Machine learning tutorial,” GeeksforGeeks, Jun. 11, 2024. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/machine-learning/

[11] Scikit-learn, “sklearn.preprocessing.LabelEncoder — scikit-learn 0.22.1 documentation,” Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[12] TensorFlow, “Data augmentation | TensorFlow Core,” TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/images/data_augmentation

[13] Z. Zhang, “Enhancing distributed machine learning through data shuffling: Techniques, challenges, and implications,” *ITM Web of Conferences*, vol. 73, Art. no. 03018, Jan. 2025, doi: 10.1051/itmconf/20257303018.

[14] scikit-learn, “sklearn.model_selection.train_test_split — scikit-learn 0.20.3 documentation,” Scikit-learn.org, 2018. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[15] “6.2. Feature extraction — scikit-learn 0.22.2 documentation,” scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html

[16] Scikit-learn, “TfidfVectorizer,” Scikit-learn.org, 2018. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[17] GeeksforGeeks, “What is the difference between ‘transform’ and ‘fit_transform’ in sklearnPython?,” GeeksforGeeks, Jun. 21, 2022. [Online]. Available: https://www.geeksforgeeks.org/python/what-is-the-difference-between-transform-and-fit_transform-in-sklearn-python/

[18] GeeksforGeeks, “Introduction to dimensionality reduction,” GeeksforGeeks, Jun. 2017. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/dimensionality-reduction/

[19] K. Shivashankar, G. Al Hajj, and A. Martini, “Maintainability and scalability in machine learning: Challenges and solutions,” *ACM Computing Surveys*, May 2025, doi: 10.1145/3736751.

[20] D. Anguita, A. Boni, and S. Ridella, “A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation,” *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 993–1009, Sep. 2003, doi: 10.1109/tnn.2003.816033.

[21] S. Albelwi and A. Mahmood, “A framework for designing the architectures of deep convolutional neural networks,” *Entropy*, vol. 19, no. 6, Art. no. 242, May 2017, doi: 10.3390/e19060242.

[22] H. Sak, A. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling,” in *Proc. Interspeech*, Sep. 2014, doi: 10.21437/interspeech.2014-80.

[23] M. Ashraf et al., “A hybrid CNN and RNN variant model for music classification,” *Applied Sciences*, vol. 13, no. 3, Art. no. 1476, Jan. 2023, doi: 10.3390/app13031476.

[24] M. Maabreh, A. Maabreh, B. Qolomany, and A. Al-Fuqaha, “The robustness of popular multiclass machine learning models against poisoning attacks: Lessons and insights,” *Int. J. Distributed Sensor Networks*, vol. 18, no. 7, Art. no. 15501329221105159, Jul. 2022, doi: 10.1177/15501329221105159.

[25] S. F. N. Azizah, H. D. Cahyono, S. W. Sihwi, and W. Widiarto, “Performance analysis of transformer based models (BERT, ALBERT and RoBERTa) in fake news detection,” *arXiv*, Aug. 09, 2023. [Online]. Available: https://arxiv.org/abs/2308.04950

[26] S. Afzal, M. Asim, M. O. Beg, T. Baker, A. I. Awad, and N. Shamim, “Context-aware embeddings for robust multiclass fraudulent URL detection in online social platforms,” *Computers & Electrical Engineering*, vol. 119, Art. no. 109494, Oct. 2024, doi: 10.1016/j.compeleceng.2024.109494.

[27] UCI Machine Learning Repository, “PHIUSIIL Phishing URL Dataset.” [Online]. Available: https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset.

[28] T. Kang, D. Y. Lim, H. Tayara, and K. T. Chong, “Forecasting of power demands using deep learning,” *Applied Sciences*, vol. 10, no. 20, p. 7241, Oct. 2020, doi: 10.3390/app10207241.

[29] J. M. Álvarez-Alvarado, J. G. Ríos-Moreno, S. A. Obregón-Biosca, G. Ronquillo-Lomeli, E. Ventura-Ramos, and M. Trejo-Perea, “Hybrid techniques to predict solar radiation using support vector machine and search optimization algorithms: A review,” *Applied Sciences*, vol. 11, no. 3, pp. 1044–1059, Jan. 2021, doi: 10.3390/app11031044.

[30] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," ACM Computing Surveys, vol. 56, no. 6, pp. 1–39, Dec. 2023, doi: 10.1145/3638242.

9.Appendices

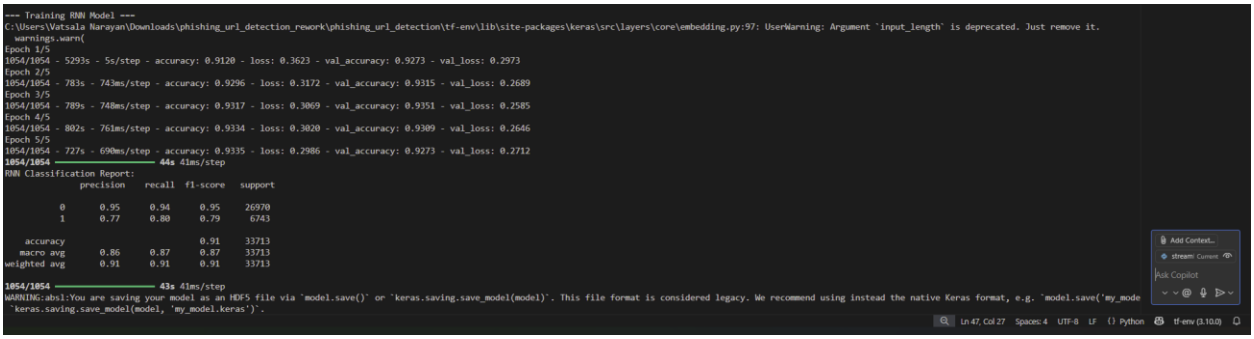
Appendix A – Training Output of SVM Model

```
(tf-env) PS C:\Users\Vatsala Narayan\Downloads\phishing_url_detection_rework\phishing_url_detection> python main.py
2025-07-28 17:41:19.552838: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results
due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`
.
2025-07-28 17:41:22.397881: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results
due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`
.

=== Training SVM Model ===
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     26970
           1       0.95      0.59      0.73      6743

    accuracy                           0.91     33713
   macro avg       0.93      0.79      0.84     33713
weighted avg       0.92      0.91      0.90     33713

SVM Accuracy: 0.9119330821938125
```

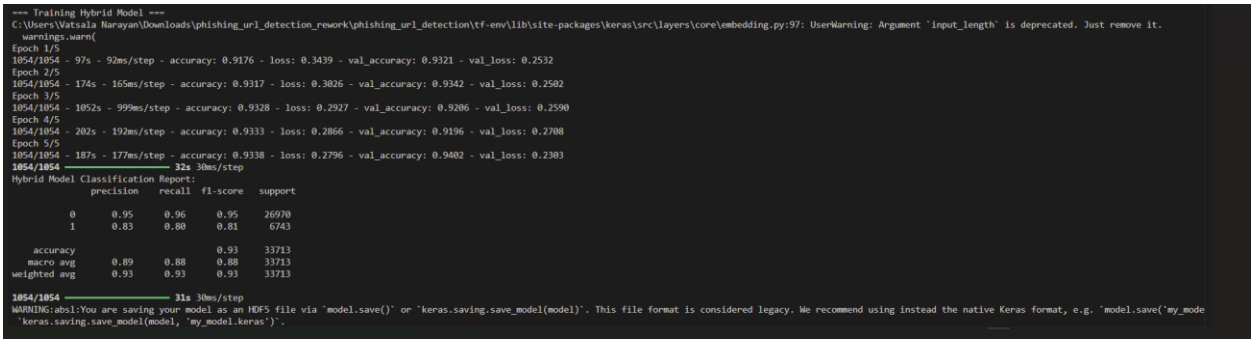
Appendix B – Training Output of CNN Model

```
=== Training CNN Model ===
C:\Users\Vatsala Narayan\Downloads\phishing_url_detection_rework\phishing_url_detection\tf-env\lib\site-packages\keras\src\layers\core\embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
2025-07-28 17:41:59.348243: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/5
1054/1054 - 68s - 64ms/step - accuracy: 0.9064 - loss: 0.3415 - val_accuracy: 0.9274 - val_loss: 0.2775
Epoch 2/5
1054/1054 - 72s - 68ms/step - accuracy: 0.9276 - loss: 0.3064 - val_accuracy: 0.9288 - val_loss: 0.2632
Epoch 3/5
1054/1054 - 71s - 67ms/step - accuracy: 0.9290 - loss: 0.2956 - val_accuracy: 0.9327 - val_loss: 0.2536
Epoch 4/5
1054/1054 - 72s - 69ms/step - accuracy: 0.9302 - loss: 0.2881 - val_accuracy: 0.9373 - val_loss: 0.2305
Epoch 5/5
1054/1054 - 72s - 68ms/step - accuracy: 0.9310 - loss: 0.2831 - val_accuracy: 0.9323 - val_loss: 0.2486
1054/1054 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step
CNN Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.95     26970
           1       0.78      0.81      0.79      6743

    accuracy                           0.92     33713
   macro avg       0.86      0.87      0.87     33713
weighted avg       0.92      0.92      0.92     33713

1054/1054 ━━━━━━━━━━━━━━━━━━━━ 8s 7ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save(`
```

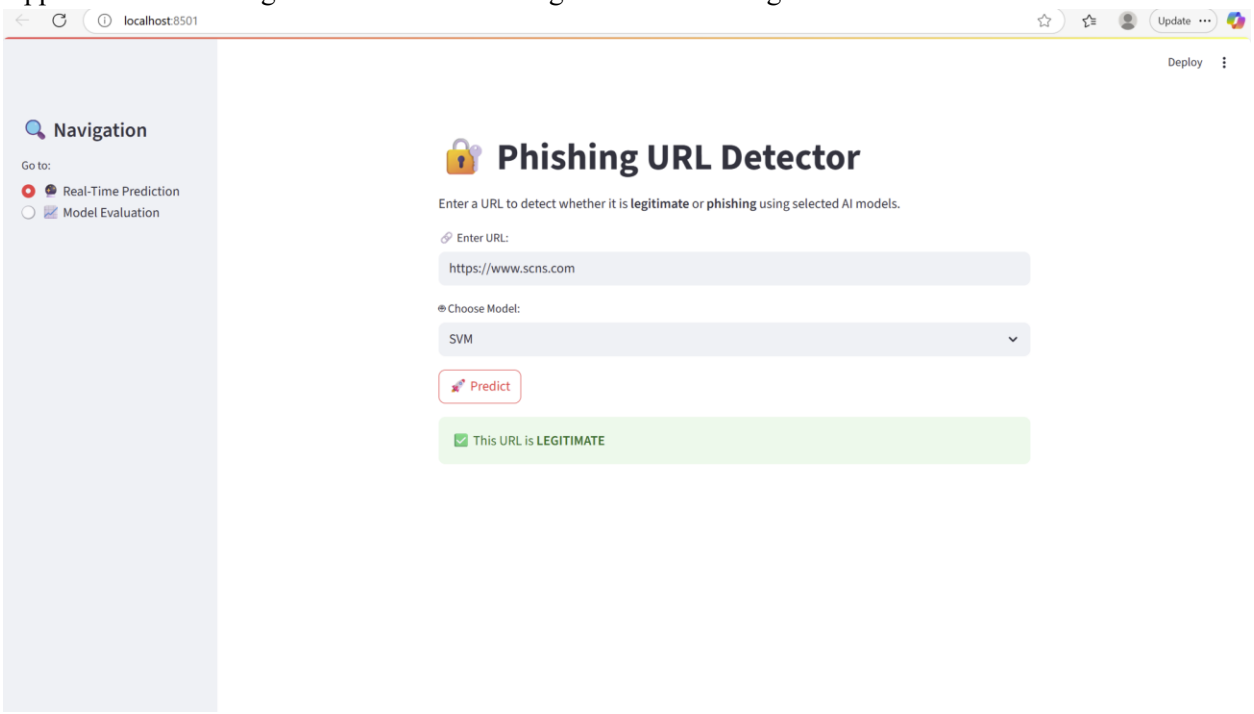
Appendix C – Training Output of RNN Model



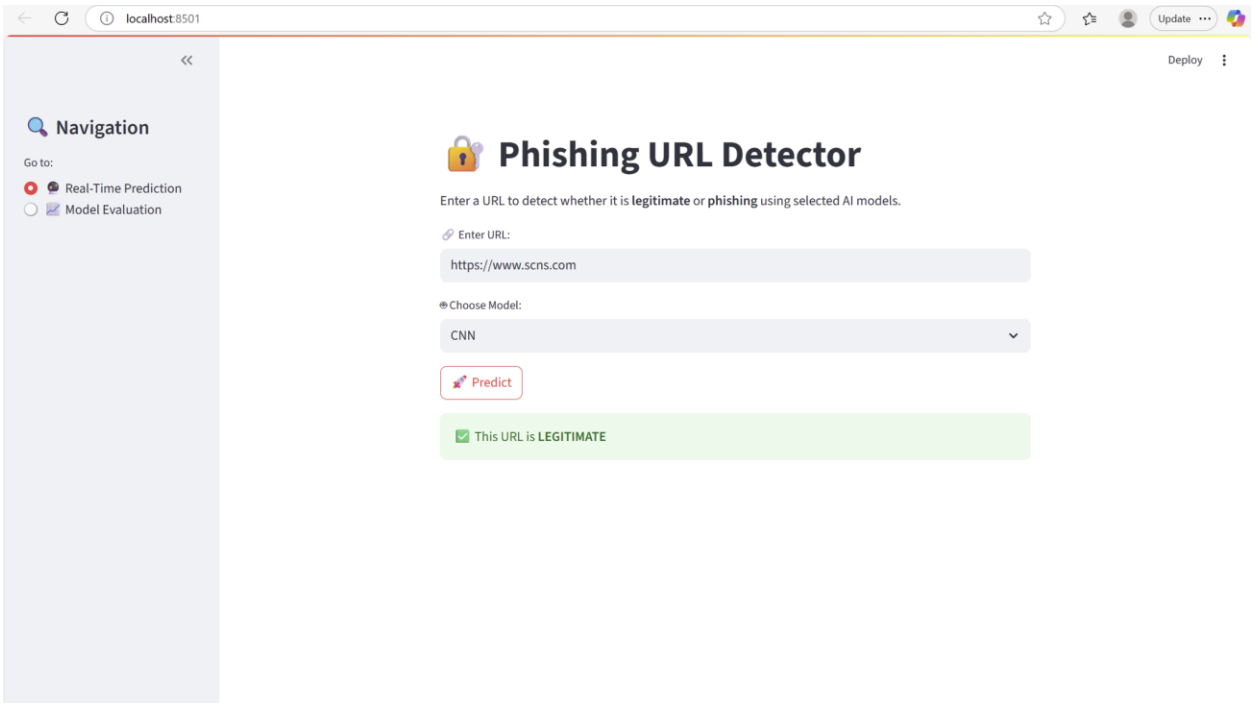
Appendix D – Training Output of Hybrid Model



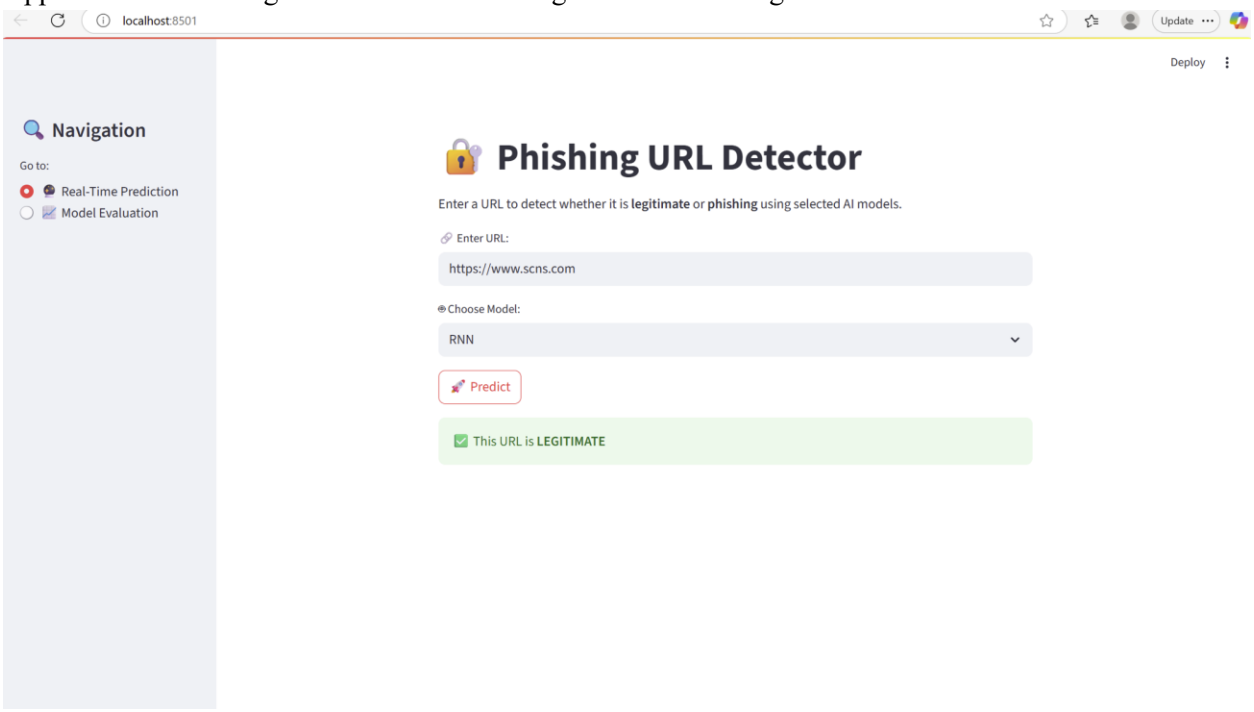
Appendix E – Phishing detection of known Legitimate URL using SVM Model



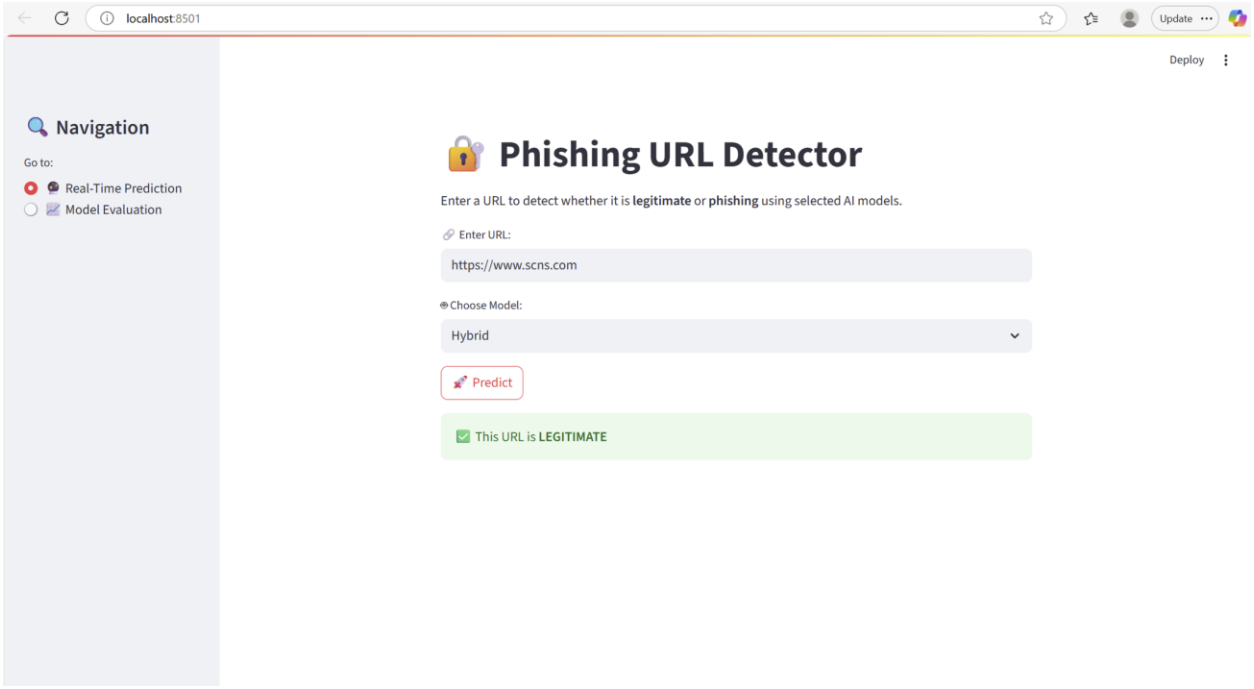
Appendix F – Phishing detection of known Legitimate URL using CNN Model



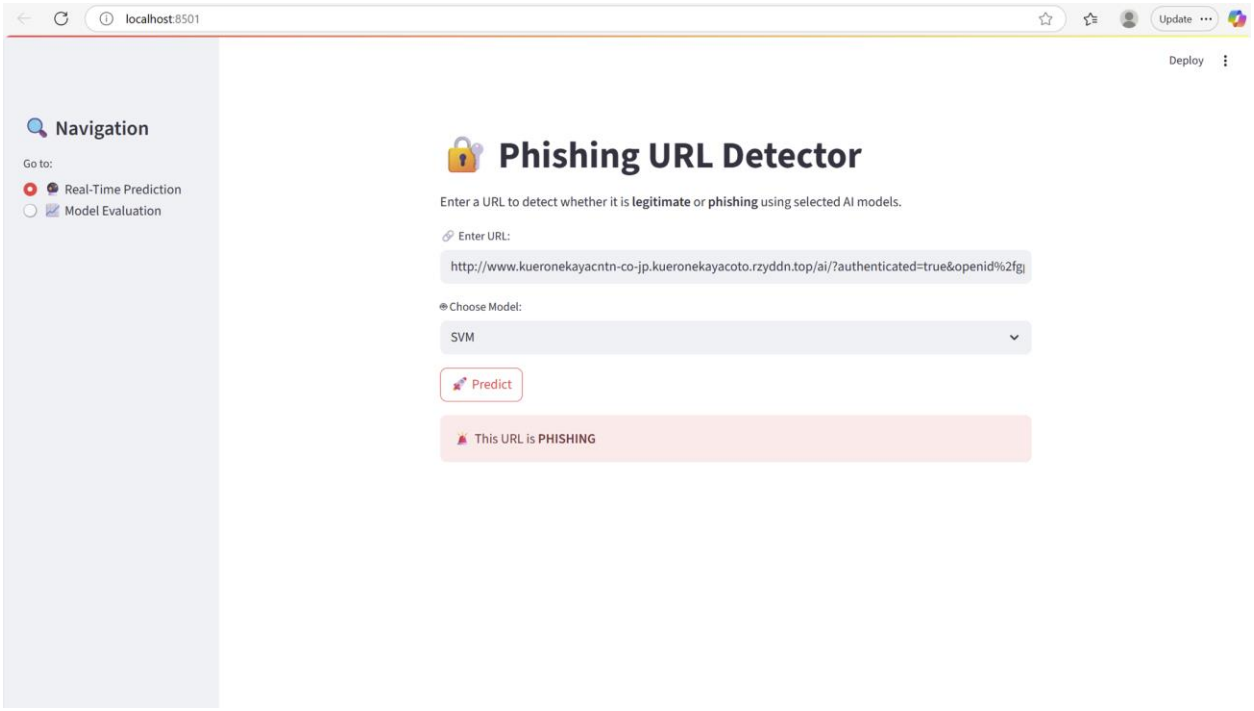
Appendix G – Phishing detection of known Legitimate URL using RNN Model



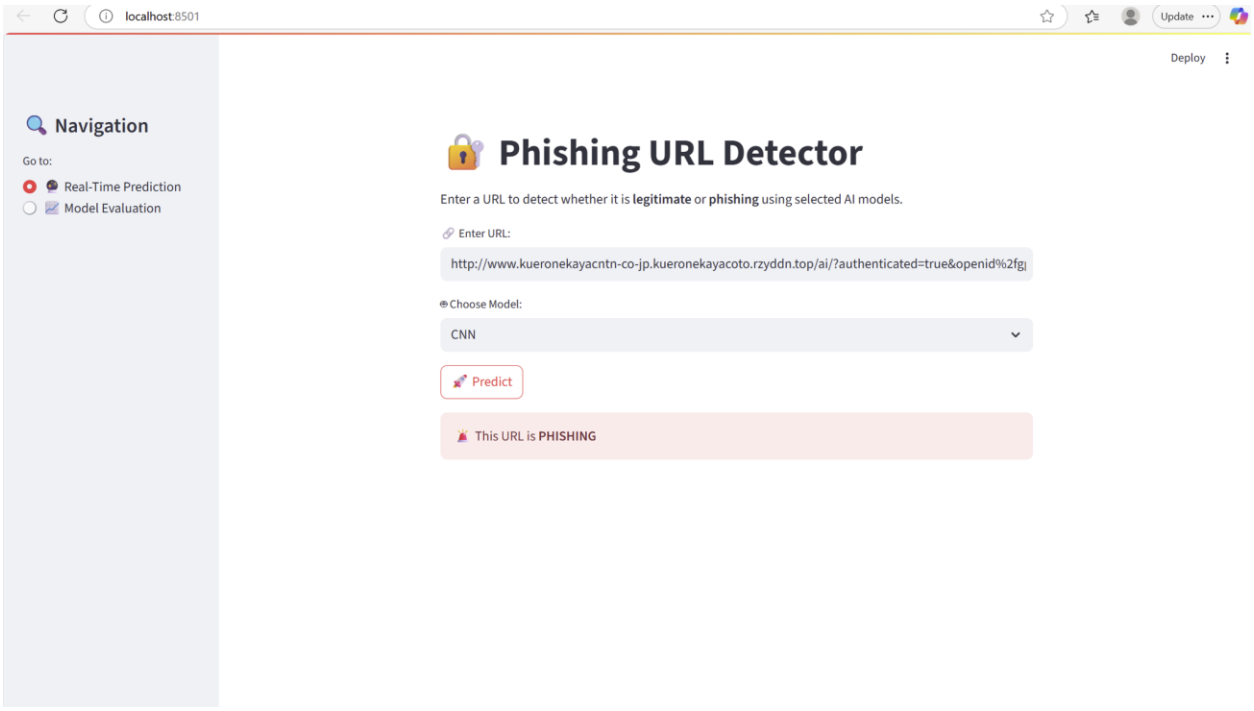
Appendix H – Phishing detection of known Legitimate URL using Hybrid Model



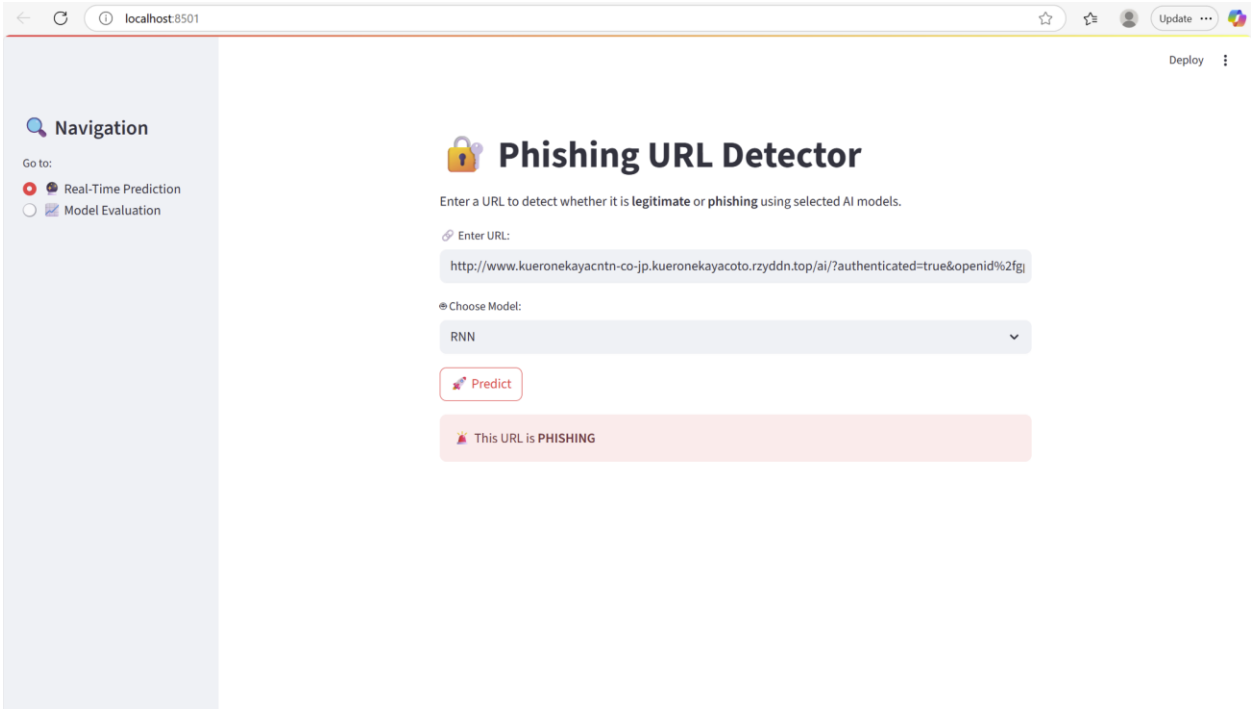
Appendix I – Detection of known Phishing URL using SVM Model



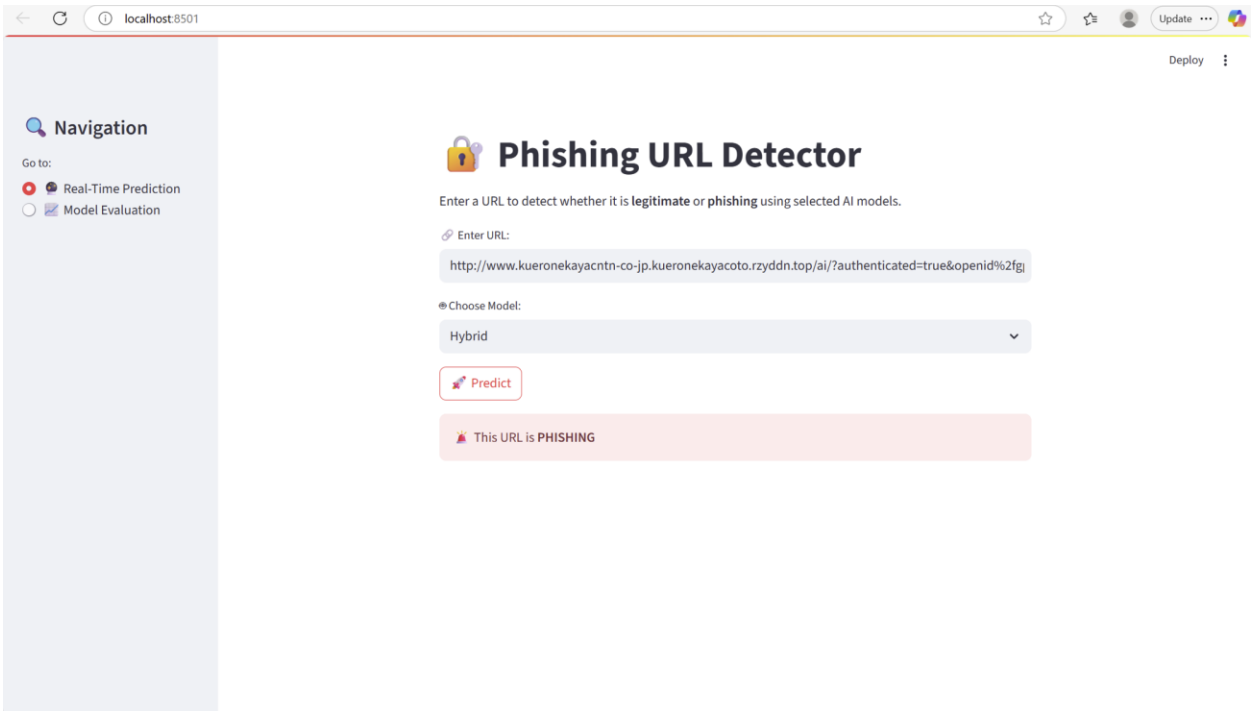
Appendix J – Detection of known Phishing URL using CNN Model



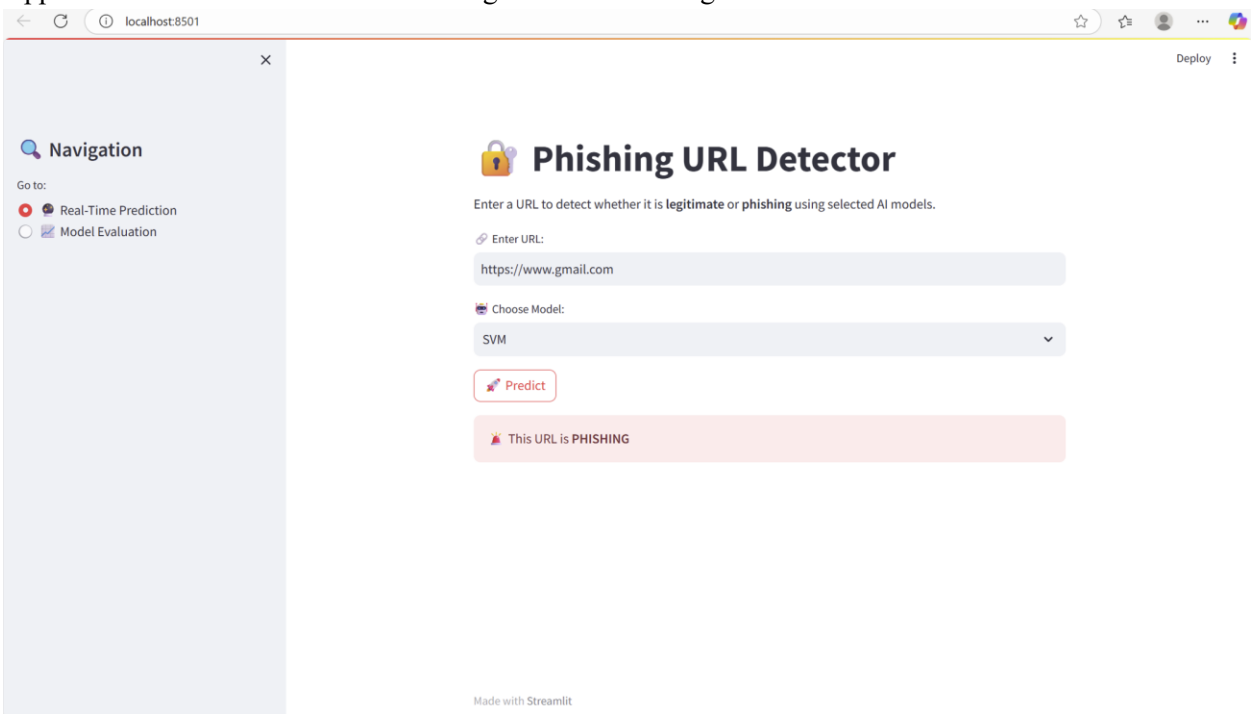
Appendix K – Detection of known Phishing URL using RNN Model



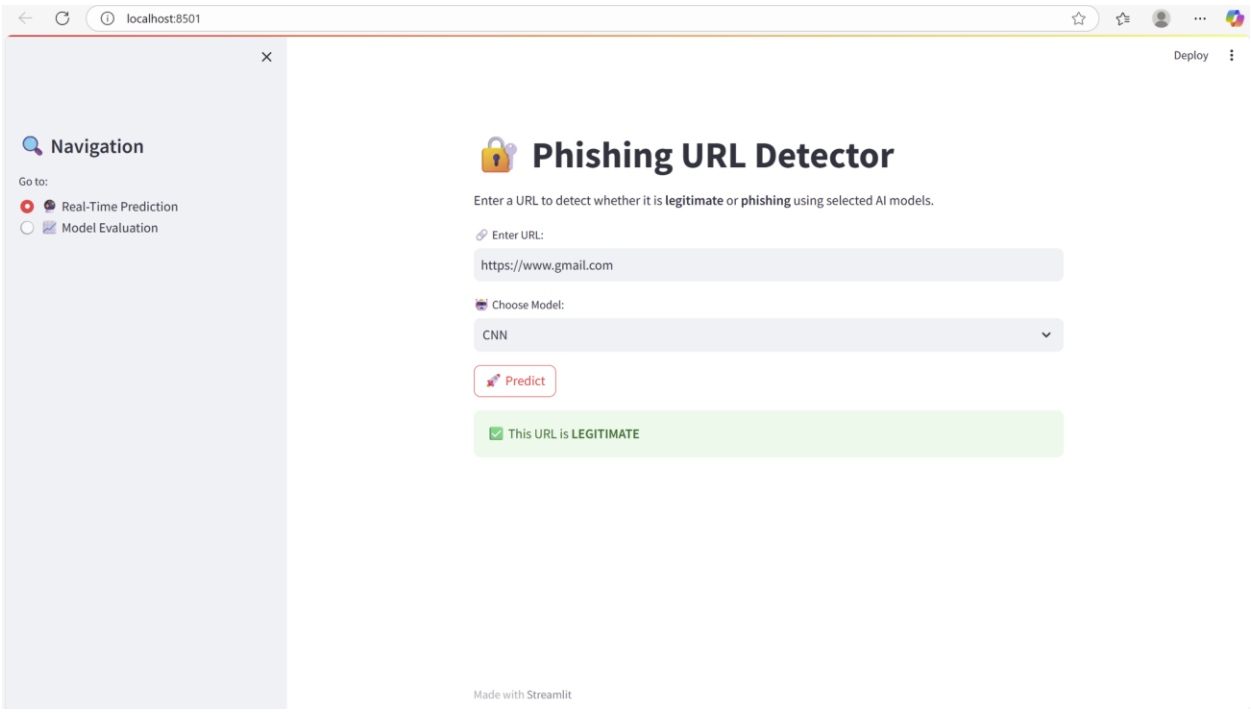
Appendix L – Detection of known Phishing URL using Hybrid Model



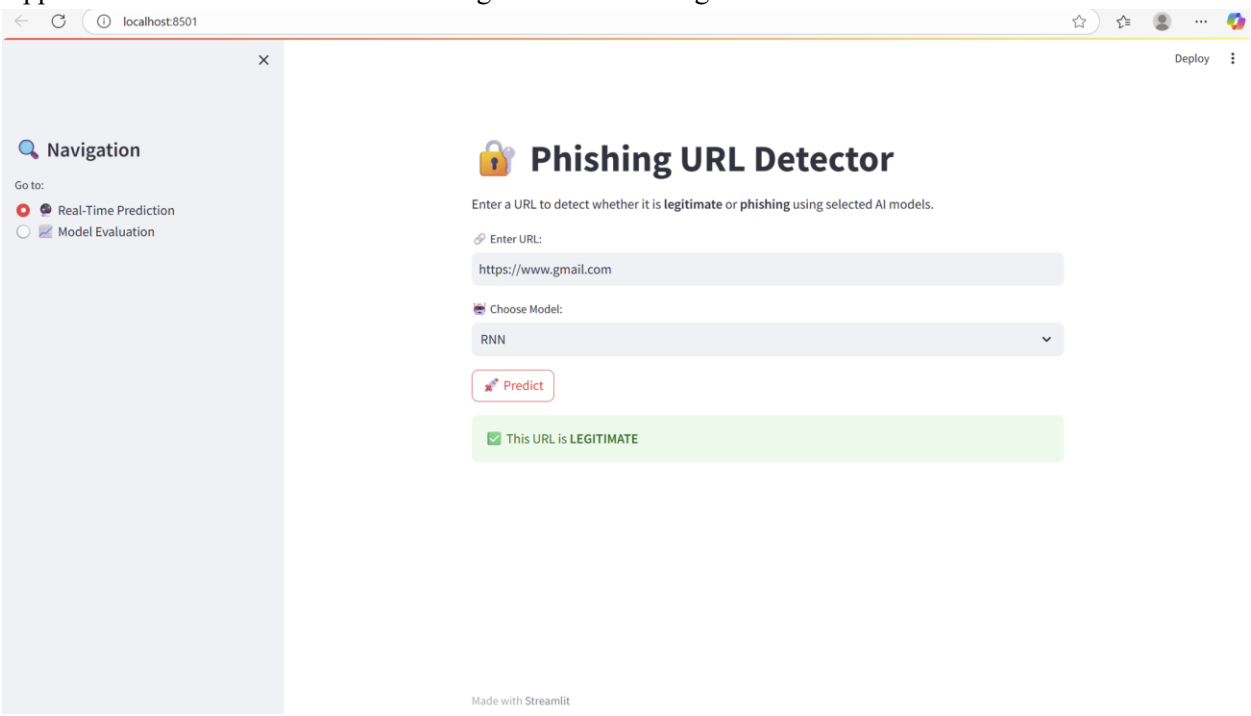
Appendix M – Detection of unknown Legitimate URL using SVM Model



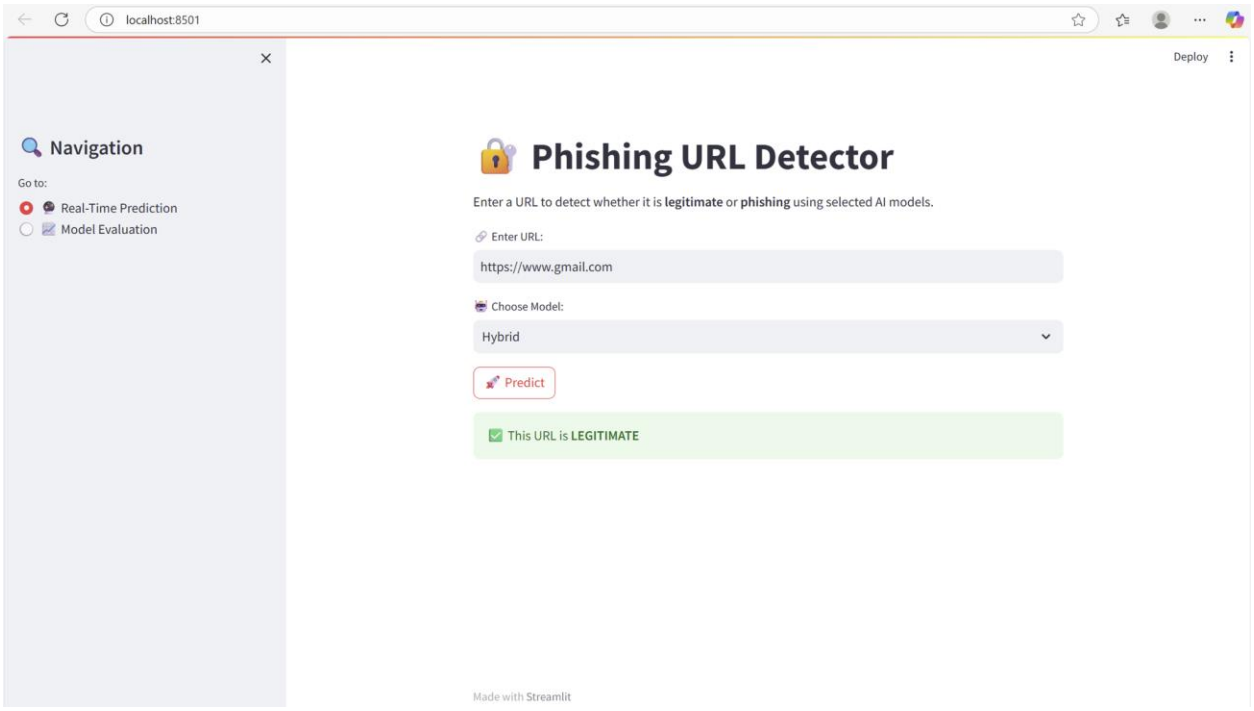
Appendix N – Detection of unknown Legitimate URL using CNN Model



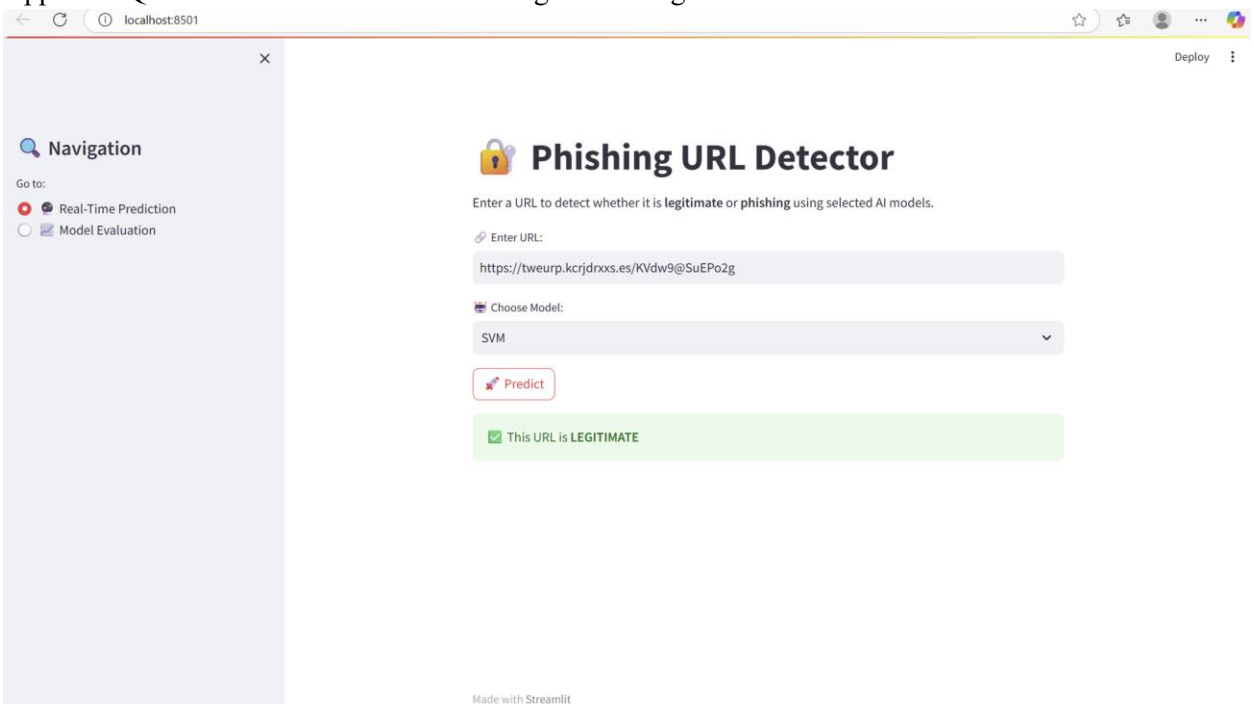
Appendix O – Detection of unknown Legitimate URL using RNN Model



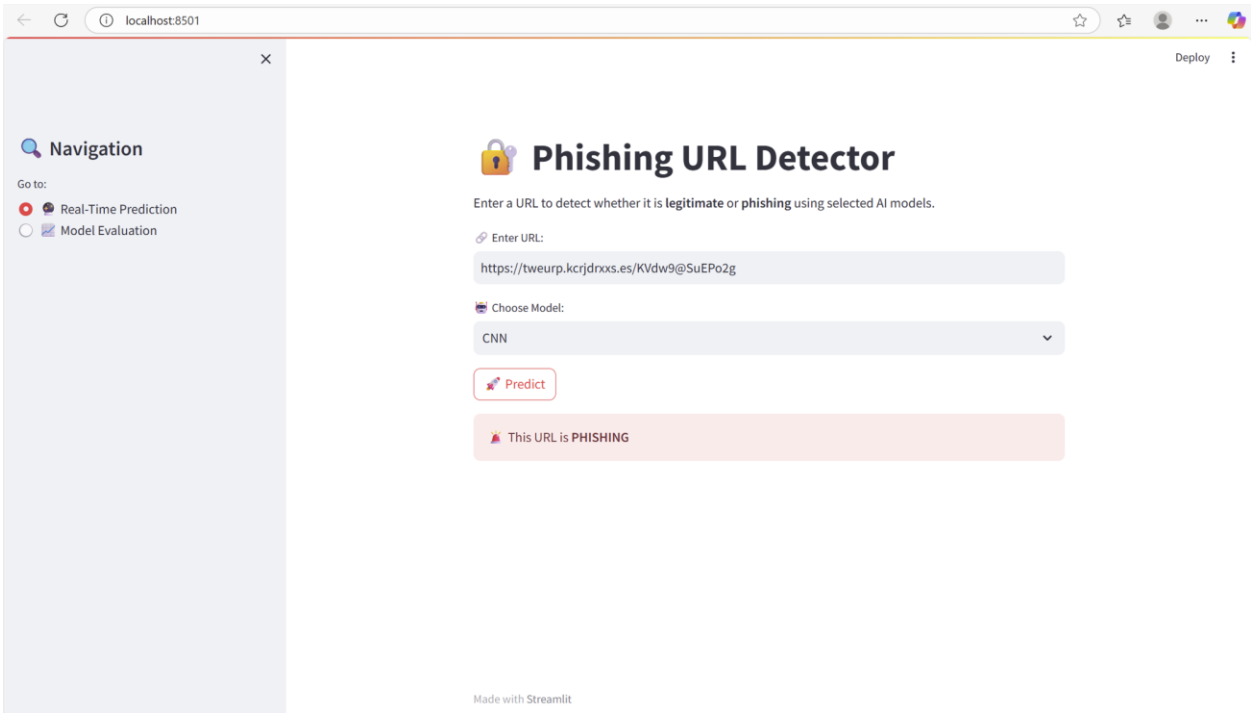
Appendix P– Detection of unknown Legitimate URL using Hybrid Model



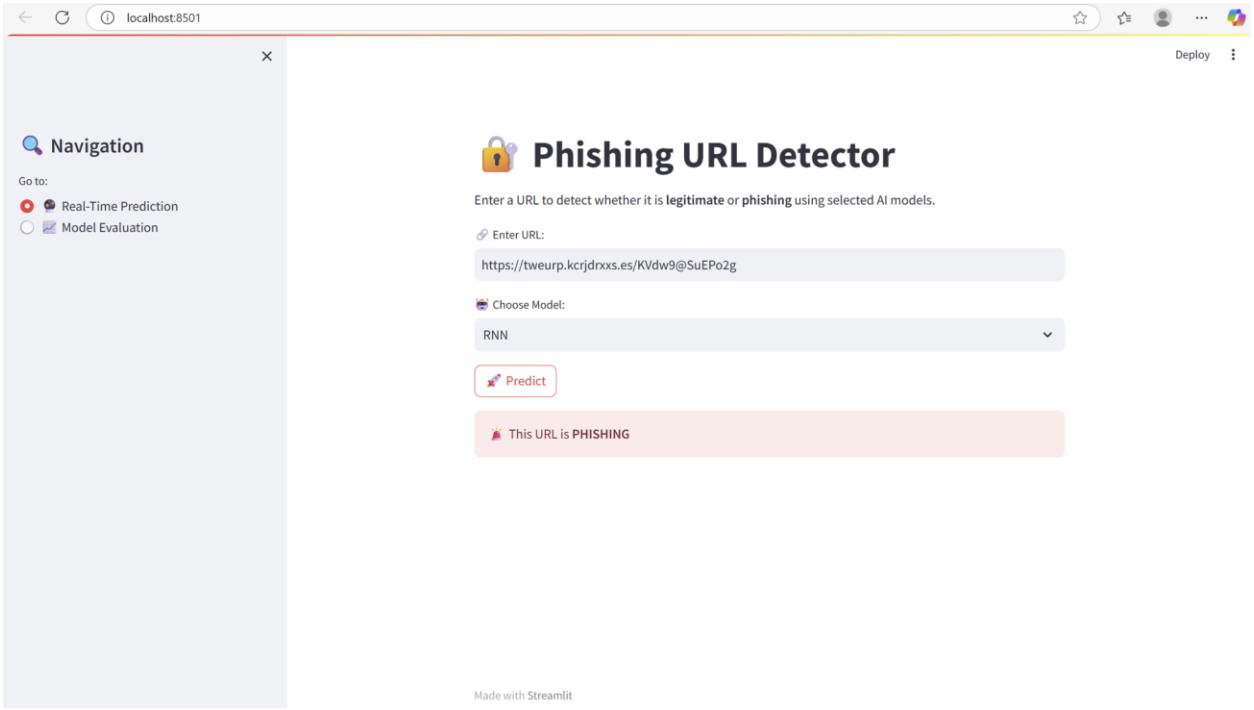
Appendix Q – Detection of unknown Phishing URL using SVM Model



Appendix R – Detection of unknown Phishing URL using CNN Model



Appendix S – Detection of unknown Phishing URL using RNN Model



Appendix T– Detection of unknown Phishing URL using Hybrid Model