

# **Phishing Detection**

## **Submitted by**

Reethu RG Thota

1RVU22CSE128

School of Computer Science

RV University

## **Submitted to**

Professor Chandramouleeswaran Sankaran

Course Lead

School of Computer Science

RV University

13/11/24

# IoT and Edge Computing (CS3100)

## Project Report

### Abstract

This project leverages IoT and edge computing to detect phishing websites by deploying a machine learning model on an ESP32 microcontroller. Using a dataset from UC Irvine focused on identifying phishing websites through specific features, the project aims to create a model that classifies websites as legitimate or phishing. Key outcomes include model accuracy and performance analysis on the ESP32.

### Introduction

#### Project Background and Relevance

The rise of IoT and edge computing has brought about the need for efficient, real-time cybersecurity solutions that operate at the network's edge, reducing latency and dependence on centralized cloud resources. Phishing attacks, a common cyber threat, often evade traditional network defenses. This project explores using a lightweight machine learning model deployed on an ESP32 microcontroller to detect phishing websites in real-time, enhancing network security at the edge.

#### Objectives

The primary objective of this project is to develop a model capable of classifying websites as phishing or legitimate based on specific feature inputs. This model will then be optimized, converted to TensorFlow Lite, and deployed on an ESP32, demonstrating a practical IoT-based phishing detection system.

### System Overview

#### System Architecture

1. **Data Collection and Preprocessing:** Dataset preparation and data cleaning.
2. **Model Training:** Developing a machine learning model on the phishing dataset using Python and TensorFlow.
3. **Conversion and Deployment:** Converting the trained model to TFLite and deploying it on the ESP32 for real-time inferencing.
4. **Inference and Output:** The ESP32 performs classification on new input data and displays the output on its serial monitor.

## Details of the Dataset

### Dataset Overview

#### 1. General Information

- **Source:** This dataset was collected from various sources, including the PhishTank and MillerSmiles archives, and using Google's search operators.
- **Date Donated:** March 25, 2015.
- **Dataset Type:** Tabular data.
- **Subject Area:** Computer Science.
- **Primary Task:** Classification (Binary Classification).
- **Feature Type:** Integer (all features are represented as integers).
- **Number of Instances:** 11,055 rows.
- **Number of Features:** 30 features plus one target variable.

#### 2. Dataset Context

This dataset addresses a major challenge in phishing detection research: the lack of a reliable, publicly available training dataset. While phishing detection is widely studied, there is no universal agreement on the features that consistently characterize phishing websites. This dataset includes 30 handpicked features that are effective for predicting phishing websites, making it a valuable resource for phishing detection tasks.

#### 3. Missing Values

No missing values are present in this dataset.

### Dataset Structure

#### Feature Overview

The dataset contains 30 features describing various characteristics of URLs, web pages, and domains. Each feature helps in distinguishing phishing websites from legitimate ones. Here is a breakdown of the main features:

##### 1. Domain and URL-Based Features:

- **having\_ip\_address:** Indicates if the URL uses an IP address rather than a domain name.
- **url\_length:** Length of the URL (phishing URLs are often longer).
- **shortening\_service:** If a URL shortening service is used (common in phishing to hide the true URL).

# IoT and Edge Computing (CS3100)

## Project Report

- **having\_at\_symbol**: Presence of the "@" symbol in the URL (a sign of phishing).
- **double\_slash\_redirecting**: Occurrence of "//" in the URL path (common in redirects).
- **prefix\_suffix**: Use of hyphens in the URL (often used in phishing domains).
- **having\_sub\_domain**: Indicates if multiple subdomains are present (used to mimic legitimate websites).

### 2. SSL and Domain Features:

- **sslfinal\_state**: Checks SSL certificate validity (a reliable indicator of website legitimacy).
- **domain\_registration\_length**: Length of time a domain has been registered (phishing domains are often short-lived).
- **favicon**: Whether the favicon is loaded from the same domain.

### 3. Content-Based Features:

- **port**: Use of non-standard ports (phishing websites may use unusual ports).
- **https\_token**: Checks if "HTTPS" appears in the URL string.
- **request\_url**: Analyzes the location of links and resources.
- **url\_of\_anchor**: Ratio of anchor tags linking to different domains.
- **links\_in\_tags**: Checks if tags contain links that lead offsite.
- **sfh (Server Form Handler)**: Observes form action URLs (phishing sites may submit forms to different domains).
- **submitting\_to\_email**: Indicates if forms are submitted directly to an email (a sign of phishing).
- **abnormal\_url**: Checks if the URL is consistent with the domain.
- **redirect**: Number of redirects (phishing sites may redirect users to different domains).
- **on\_mouseover**: Checks for abnormal behaviors like changes on mouseover.
- **rightclick**: Disables right-clicking (common on phishing sites).
- **popupwindow**: Use of popup windows (often used in phishing).
- **iframe**: Checks for iframe use (phishing sites often use iframes to hide malicious content).

### 4. Domain Age and DNS Records:

- **age\_of\_domain**: Age of the domain (phishing domains are usually newer).
- **dnsrecord**: DNS record information (missing or invalid records may indicate phishing).

### 5. Traffic and Popularity Features:

# IoT and Edge Computing (CS3100)

## Project Report

- **web\_traffic**: Popularity of the website (phishing sites usually have low or no traffic).
- **page\_rank**: Google PageRank of the website.
- **google\_index**: If the page is indexed by Google (phishing pages may not be indexed).
- **links\_pointing\_to\_page**: Number of backlinks to the page.
- **statistical\_report**: Checks for statistical features associated with phishing.

### Target Variable

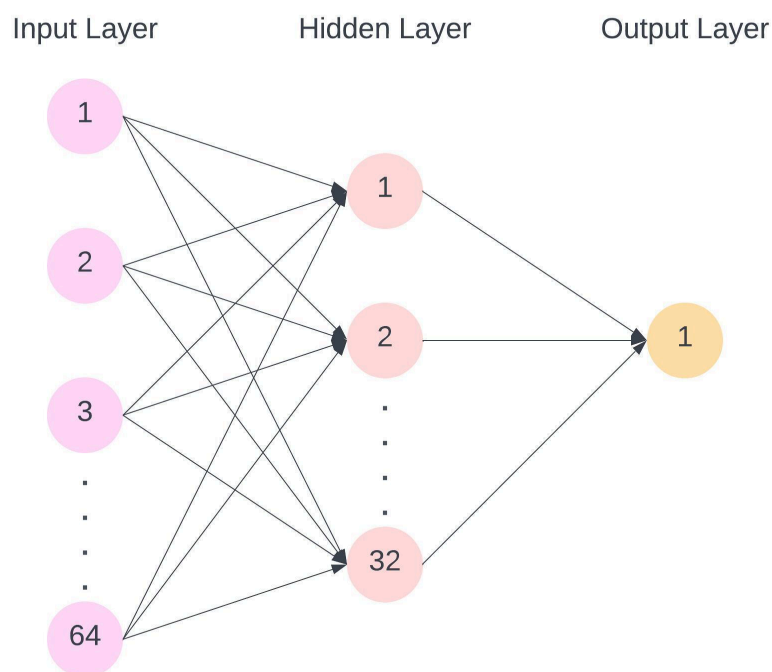
**result**: The target variable, where:

- **1** represents a phishing website.
- **0** represents a legitimate website.

### Model Architecture

The model used in this project is a neural network designed with a balance of accuracy and lightweight computation to run on an ESP32. Key components include:

- **Input Layer**: Accepts preprocessed feature inputs from the dataset.
- **Hidden Layers**: Includes dense layers with *ReLU activation* functions for learning feature representations.
- **Output Layer**: A dense layer with a *sigmoid activation* function to classify the website as either phishing or legitimate.



# IoT and Edge Computing (CS3100)

## Project Report

The model architecture includes an ESP32 deployment using **EloquentTinyML**, a library that enables running TensorFlow Lite models on microcontrollers. With this setup, the ESP32 can efficiently process a phishing detection model using limited resources.

## Requirements

### Hardware

- **ESP32 Microcontroller:** The ESP32 was chosen for its ability to run lightweight ML models on-device, providing real-time inference without relying on a cloud connection. With built-in Wi-Fi and Bluetooth, the ESP32 can also support connectivity-based IoT applications if needed in future expansions. Additionally, its low power consumption makes it suitable for battery-operated deployments.

### Software

- **Python Libraries: TensorFlow and TensorFlow Lite** were used to define, train, and optimize the phishing detection model. TensorFlow handles the primary model development and training, while TensorFlow Lite is used to convert the model into a compact format for the ESP32's limited computational resources.
- **Integrated Development Environment (IDE):** The **Arduino IDE** or is used for uploading and debugging code on the ESP32. This IDE supports the integration of the EloquentTinyML library and facilitates serial communication for monitoring ESP32 outputs.
- **EloquentTinyML Library:** This library allows ESP32 to run TensorFlow Lite models with minimal memory usage. EloquentTinyML simplifies the process of loading, executing, and obtaining predictions from TFLite models on microcontrollers.

## Methodology

### Setup

The setup phase included both hardware and software configurations to ensure successful deployment and operation of the phishing detection model on the ESP32.

- **Hardware Configuration**
  - The **ESP32 microcontroller** was connected to a laptop via USB for code uploading, debugging, and serial monitoring of outputs. Power was supplied via the USB connection, making it convenient to program and test.
- **Software Configuration:**
  - **Python Environment:** The Python environment was configured with **TensorFlow** and **TensorFlow Lite** to support model training and conversion. The phishing detection model was developed in TensorFlow, leveraging its

# IoT and Edge Computing (CS3100)

## Project Report

extensive capabilities for deep learning model construction, compilation, and training.

- **Model Conversion:** After training, the model was converted into the **TensorFlow Lite (TFLite)** format to create a compact, optimized model suitable for ESP32 deployment. This step reduced the model's size and optimized it for limited memory.
- **IDE for ESP32:** **Arduino IDE** was used to upload the `.ino` code onto the ESP32. The IDE also provided a serial monitor, which was instrumental in observing real-time outputs.
- **EloquentTinyML Library:** This library allowed the ESP32 to load and run the TFLite model. The library facilitated model integration with ESP32's limited resources, enabling efficient model deployment and prediction.

### Code Summary

Key functions and modules in the code were structured to accomplish model conversion, deployment, and inferencing on the ESP32.

#### 1. Model Building and Training

- **Model Architecture:** A lightweight binary classification model was built in TensorFlow, using dense layers and a sigmoid output layer to predict *phishing (1)* or *legitimate (0)* websites.
- **Training:** The model was trained on the 30-feature phishing dataset with **Standard Scaling**, optimizing hyperparameters for accuracy. Model performance was verified post-training to ensure it met accuracy requirements for phishing detection.
- **Model Evaluation:** Key metrics confirmed model accuracy before conversion to TensorFlow Lite format for deployment on the ESP32.

#### 2. Model Conversion Module

This module, written in Python, converts the trained TensorFlow model to TFLite format. Conversion techniques like quantization are used to reduce model size while retaining accuracy. The converted model is then saved as a header file (`phishing_detection_model_tflite.h`) for inclusion in the ESP32 code.

#### 3. Deployment Code

The ESP32 deployment code includes initialization of the model using **EloquentTinyML**, feeding input data, running inferences, and interpreting outputs.

# IoT and Edge Computing (CS3100)

## Project Report

No explicit normalization or scaling of input features was performed in the code, as the input arrays (e.g., `input1`, `input2`) directly use raw feature values.

## Implementation and Testing

### Core Code Excerpts

#### 1. Model Training & Evaluation

```
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build a simple feedforward neural network model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid') # Output layer for binary
classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')
```

- Used Standard Scaling to scale the features
- Building a model with 1 Hidden layer using ReLU activation function and Sigmoid Activation function for the Output Layer as it is a binary classification problem.
- Using binary cross entropy loss function and adam optimizer for 50 epochs with batch size as 32.

#### 2. Conversion to TFLite

```
# Converting a tf.Keras model to a TensorFlow Lite model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
# Save the model in TFLite format whose size is just 5 KB
# It brings down the size from 40 KB to 5 KB, 8 times reduction
with open('phishing_detection_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

#### 3. Model Initialization and Setup on Arduino IDE

This section includes the setup for serial communication and model initialization

```
#include <EloquentTinyML.h>
```



# IoT and Edge Computing (CS3100)

## Project Report

```
#include "phishing_detection_model_tflite.h" // Include your model header
file

#define NUMBER_OF_INPUTS 30 // Number of input features from your dataset
#define NUMBER_OF_OUTPUTS 1 // Single output since it's binary
classification
#define TENSOR_ARENA_SIZE 5*1024 // Adjust size as needed

Eloquent::TinyML::TfLite<NUMBER_OF_INPUTS, NUMBER_OF_OUTPUTS,
TENSOR_ARENA_SIZE> ml;

void setup() {
    Serial.begin(115200);

    ml.begin(phishing_detection_model_tflite); // Initialize the model
}
```

#### 4. Prediction Logic and Inferencing

Code to predict output based on sample inputs (**input1** and **input2**), where each input represents a set of 30 features from the phishing dataset:

```
void loop() {
    // Sample input data based on your dataset
    float input1[] = {-1.0f, 1.0f, 1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f,
-1.0f, 1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f,
0.0f, 1.0f, 1.0f, 1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 1.0f};
    // Expected Output is 0.0

    float input2[] = {1.0f, -1.0f, 1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 1.0f,
-1.0f,-1.0f, 1.0f, -1.0f, 1.0f, 0.0f, -1.0f, -1.0f, -1.0f, -1.0f, 0.0f,
-1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 0.0f, -1.0f};
    // Expected Output is 1.0

    float fResult[NUMBER_OF_OUTPUTS] = {0};
    float fRes;
    int final_output; // Declare final_output here

    // Predicting for the first input
    fRes = ml.predict(input1, fResult);
    final_output = (fResult[0] > 0.5f) ? 1 : 0;
    Serial.print("\nThe output value returned for input1 is: ");
    Serial.println(final_output);

    // Predicting for the second input
    fRes = ml.predict(input2, fResult);
    final_output = (fResult[0] > 0.5f) ? 1 : 0;
    Serial.print("\nThe output value returned for input2 is: ");
    Serial.println(final_output);

    delay(5000); // Delay between predictions
}
```

# IoT and Edge Computing (CS3100)

## Project Report

### Testing

#### Testing in Python:

- The Tensorflow model's accuracy was tested on the phishing dataset before conversion to TFLite. The model successfully classified phishing and legitimate websites based on the test accuracy, which was observed and recorded. Various performance metrics such as accuracy, precision, recall, and F1-score were calculated to assess the model's effectiveness in distinguishing between phishing and legitimate websites. These results were documented to ensure the model's performance was satisfactory before deployment.
- After converting the model to TFLite, the model was also tested on the phishing dataset. The same test data used for the TensorFlow model was fed into the TFLite model to evaluate if there were any significant discrepancies in performance after conversion. The results showed that the TFLite model maintained a similar level of accuracy and performed efficiently on the dataset, confirming that the conversion did not introduce any performance degradation.

#### ESP32 Testing:

- After deploying the model on ESP32, test predictions were observed on the serial monitor. Input feature arrays representing legitimate and phishing websites were fed into the ESP32's prediction function, and the results were printed on the serial output.
- Basic tests showed that the ESP32 could correctly classify both sample inputs, with results consistent with the model's performance in the Python environment.

### Results

#### Data Output

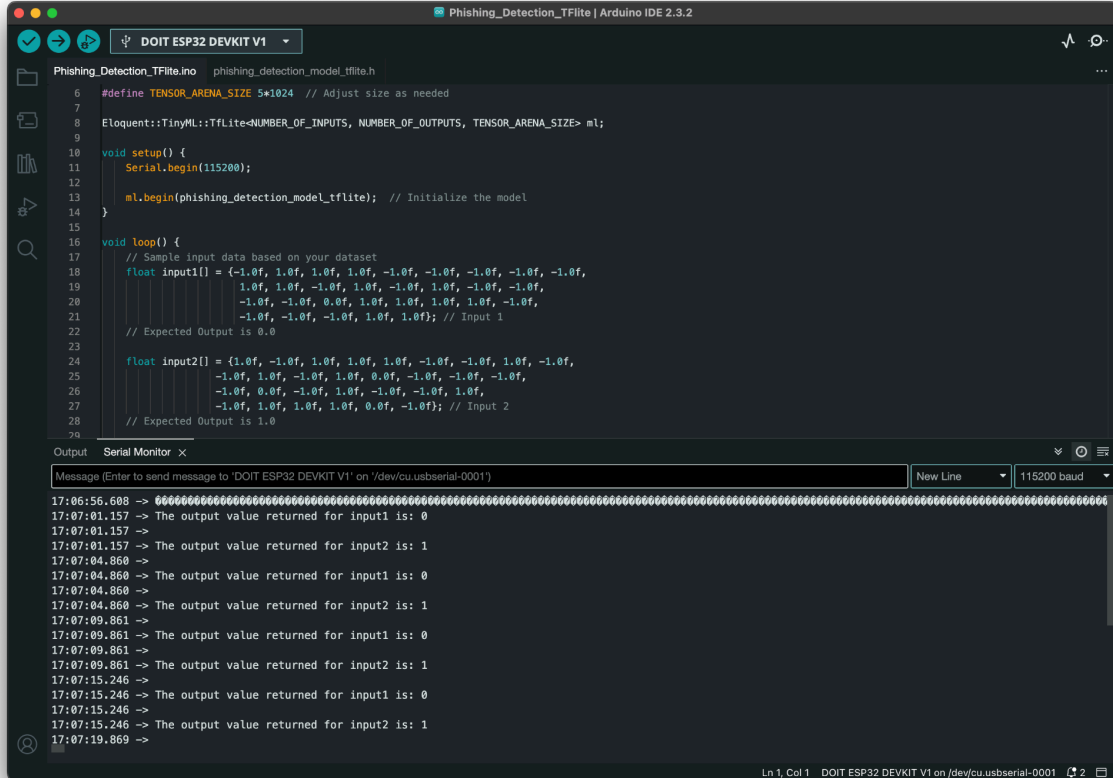
- The phishing dataset was used to train and test the model in Python. The model achieved **97%** accuracy in classifying phishing and legitimate websites. After converting to TFLite, the model maintained similar performance on the same dataset.
- On the ESP32, input feature arrays were processed, and the device successfully classified both phishing and legitimate websites, with predictions printed on the serial monitor.

# IoT and Edge Computing (CS3100)

## Project Report

### Screenshots of outputs

#### 1. Output on ESP32 (serial monitor)



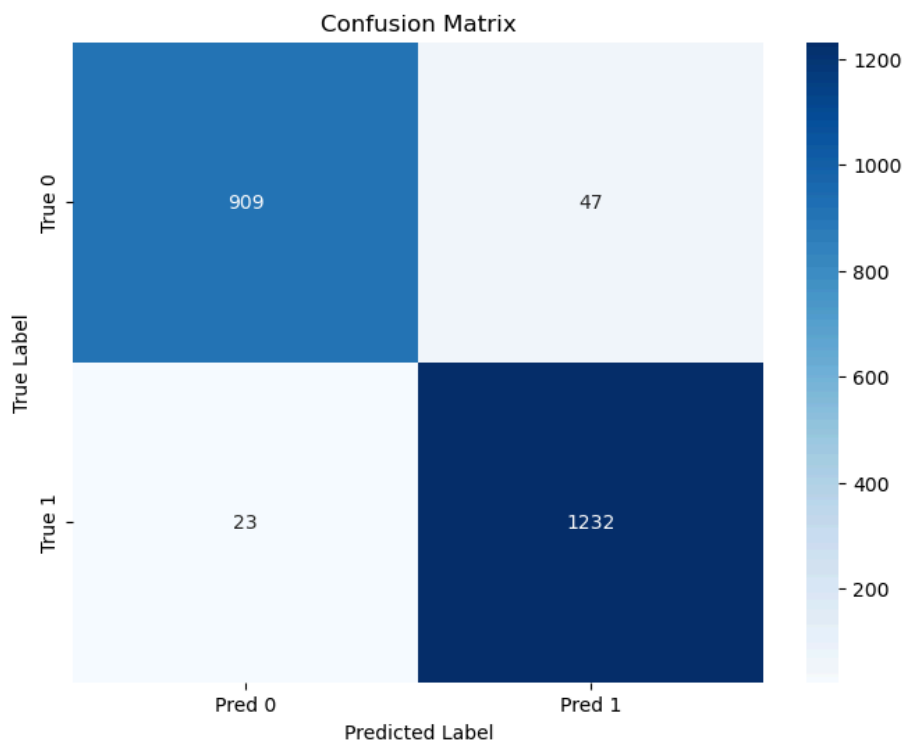
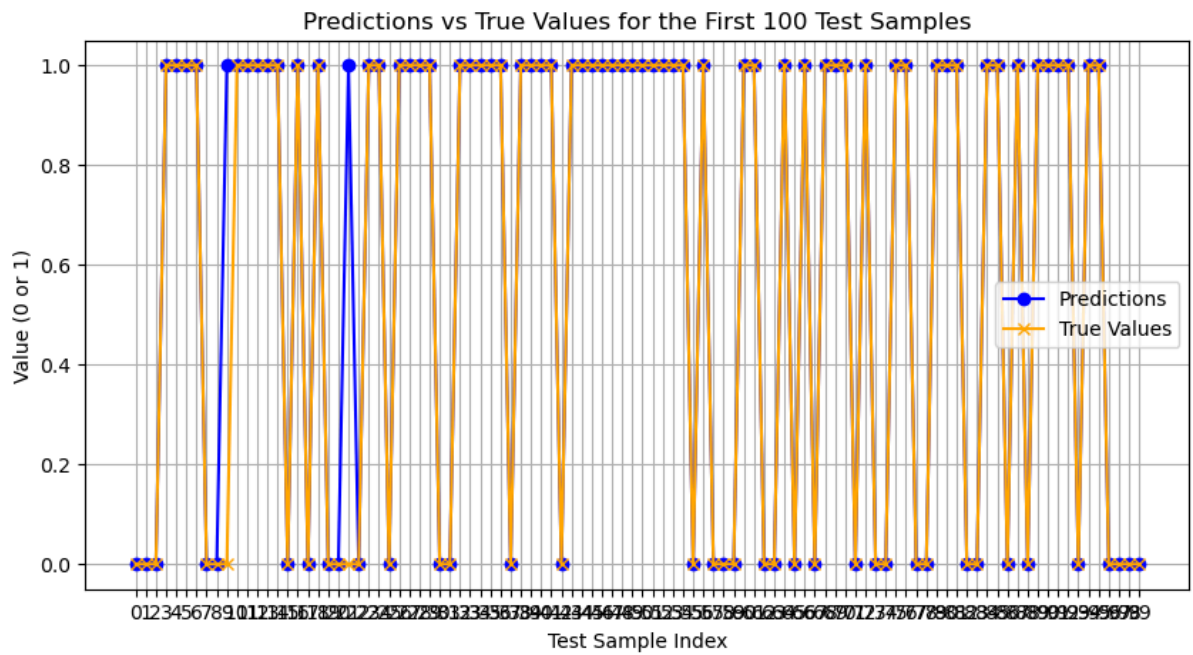
The screenshot shows the Arduino IDE interface with the file 'Phishing\_Detection\_TFlite.ino' open. The code defines a TensorFlow Lite model for phishing detection. The Serial Monitor is open, showing the output of the program. The output consists of a series of messages indicating the classification results for two input samples. The first sample is classified as 0.0, and the second sample is classified as 1.0. The output is as follows:

```
17:06:56.608 -> 0.0
17:07:01.157 -> The output value returned for input1 is: 0
17:07:01.157 -> 
17:07:01.157 -> The output value returned for input2 is: 1
17:07:04.860 -> 
17:07:04.860 -> The output value returned for input1 is: 0
17:07:04.860 -> The output value returned for input2 is: 1
17:07:09.861 -> 
17:07:09.861 -> The output value returned for input1 is: 0
17:07:09.861 -> The output value returned for input2 is: 1
17:07:15.246 -> 
17:07:15.246 -> The output value returned for input1 is: 0
17:07:15.246 -> The output value returned for input2 is: 1
17:07:19.869 ->
```

#### 2. Accuracy of TFLite model

Classification Report:					
	precision	recall	f1-score	support	
0.0	0.98	0.95	0.96	956	
1.0	0.96	0.98	0.97	1255	
accuracy			0.97	2211	
macro avg	0.97	0.97	0.97	2211	
weighted avg	0.97	0.97	0.97	2211	
Confusion Matrix:					
[[ 909  47]					
[  23 1232]]					

IoT and Edge Computing (CS3100)  
Project Report



# IoT and Edge Computing (CS3100)

## Project Report

### 3. Test Accuracy (Tensorflow model before conversion to TFLite)

```
70/70 [=====] - 0s 374us/step - loss: 0.1378 - accuracy: 0.9643
Test Accuracy: 0.96
70/70 [=====] - 0s 384us/step
Classification Report:
              precision    recall  f1-score   support

     0.0         0.96      0.95      0.96         956
     1.0         0.96      0.97      0.97        1255

 accuracy          0.96          0.96          0.96        2211
  macro avg         0.96          0.96          0.96        2211
 weighted avg         0.96          0.96          0.96        2211

Confusion Matrix:
[[ 911   45]
 [   34 1221]]
```

## Conclusion

- **Accomplishments:**
  - Successfully trained a model for phishing detection and deployed it as a TFLite model on the ESP32.
  - Achieved good classification performance in both Python and on the ESP32.
- **What Was Learned:**
  - Converting TensorFlow models to TFLite for edge devices like the ESP32 is feasible with optimizations.
  - Resource constraints on embedded devices require careful model testing and optimization, but performance trade-offs can be effectively managed.