Task 1: Implementing LSTM using PyTorch for detecting phishing URLs.

> Step 1: In this, we are utilizing a dataset named "phishing_site_urls.csv," which consists of URLs categorized as either legitimate (good) or malicious (bad).

```
<bound method NDFrame.head of                                                         URL Label
0       nobell.it/70ffb52d079109dca5664cce6f317373782/...   bad
1       www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...   bad
2       serviciosbys.com/paypal.cgi.bin.get-into.herf....   bad
3       mail.printakid.com/www.online.americanexpress....   bad
4       thewhiskeydregs.com/wp-content/themes/widescre...   bad
...                                                   ...   ...
549341                                   23.227.196.215/   bad
549342                                apple-checker.org/   bad
549343                                 apple-iclods.org/   bad
549344                                apple-uptoday.org/   bad
549345                                 apple-search.info   bad

[549346 rows x 2 columns]>
(549346, 2)
```

> Step 2: Initially, the dataset comprises 549,346 rows. After preprocessing the raw data, we narrowed it down to 507,111 rows, where 72% are labeled as good and 28% as bad. The data was then split into training and testing sets, with 50% allocated for this assignment. Tokenization was performed by converting characters into integers using a character-level Tokenizer, and the sequences were padded to a maximum length of 200 to address memory issues caused by the original sequence lengths.

```python
# Preprocess the data:
# Convert URLs to lowercase
data_frame ['URL'] = data_frame['URL'].apply(lambda x: x.lower ())

# remove duplicates
data_frame.drop(data_frame[data_frame.URL.duplicated() == True].index, axis=0, inplace=True)
data_frame .reset_index(drop=True)

# replace text labels with integer labels:
data_frame[ 'Label'] = data_frame[ 'Label']. replace({'bad': 0, 'good': 1})

# You can see that the number of rows is reduced since we removed duplicates:
print(data_frame.shape) # print dataset size: (# of rows, # of columns)
```

```
(507111, 2)
```

> Step 3: The URL then converted into a sequence of integers and padded to make the sequence the same length.

```
jorgensenconveyors.com/replacement.aspx
[39  2  9 20  1 10  8  1 10  7  2 10 29  1 25  2  9  8 12  7  2 11  5  9
  1 15 13  3  7  1 11  1 10  6 12  3  8 15 37  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0]
[0 1 1 1]
200
```

> Step 4: The dataset was split, data was converted into PyTorch tensors, and DataLoader instances were created.

Step 5: The LSTM model was defined.

Step 6: The LSTM model was instantiated, and the model was trained.

```
Epoch 1/10, Loss: 0.2353014051914215, Training Accuracy: 0.8949493993389969
Epoch 2/10, Loss: 0.057954415678977966, Training Accuracy: 0.9408962193457804
Epoch 3/10, Loss: 0.2591412365436554, Training Accuracy: 0.9582416368899721
Epoch 4/10, Loss: 0.09283746033906937, Training Accuracy: 0.9653880435725724
Epoch 5/10, Loss: 0.016597045585513115, Training Accuracy: 0.9712487280815921
Epoch 6/10, Loss: 0.0193292498588562, Training Accuracy: 0.9715011397966508
Epoch 7/10, Loss: 0.03154568001627922, Training Accuracy: 0.9750270159413774
Epoch 8/10, Loss: 0.02233760617673397, Training Accuracy: 0.9760287749355167
Epoch 9/10, Loss: 0.00425653625279665, Training Accuracy: 0.9779060870662659
Epoch 10/10, Loss: 0.0765443965792656, Training Accuracy: 0.9848237456320942
```

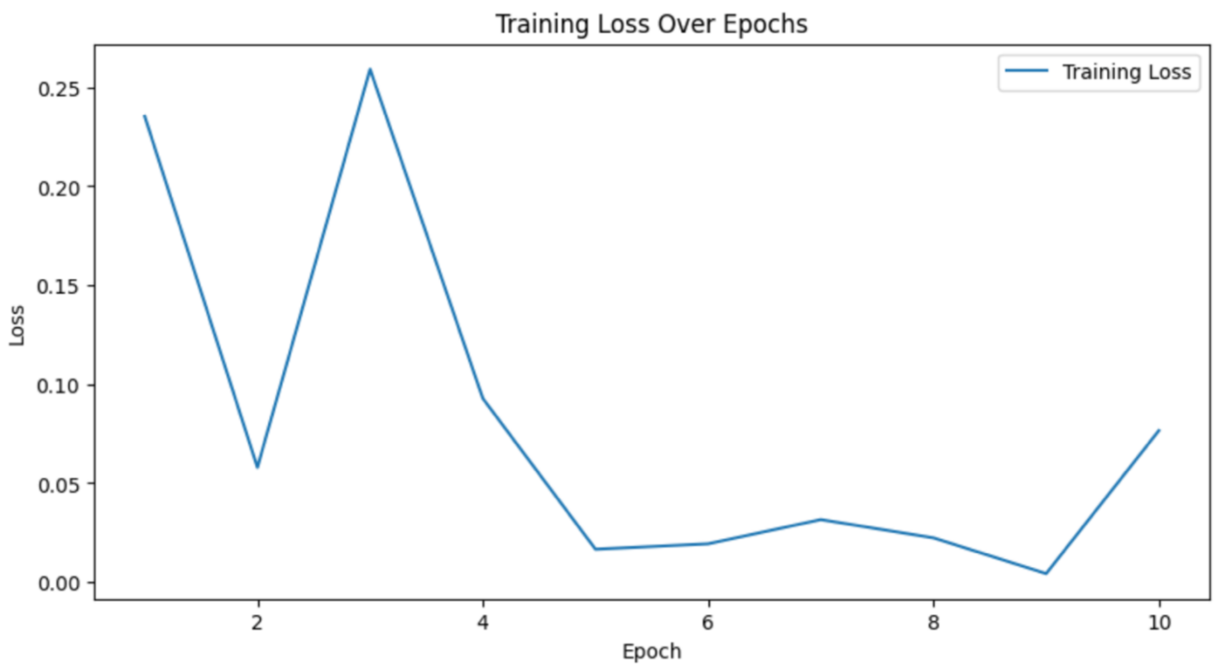
> Step 7: Evaluating the Model, obtained an accuracy of **97.07%.**

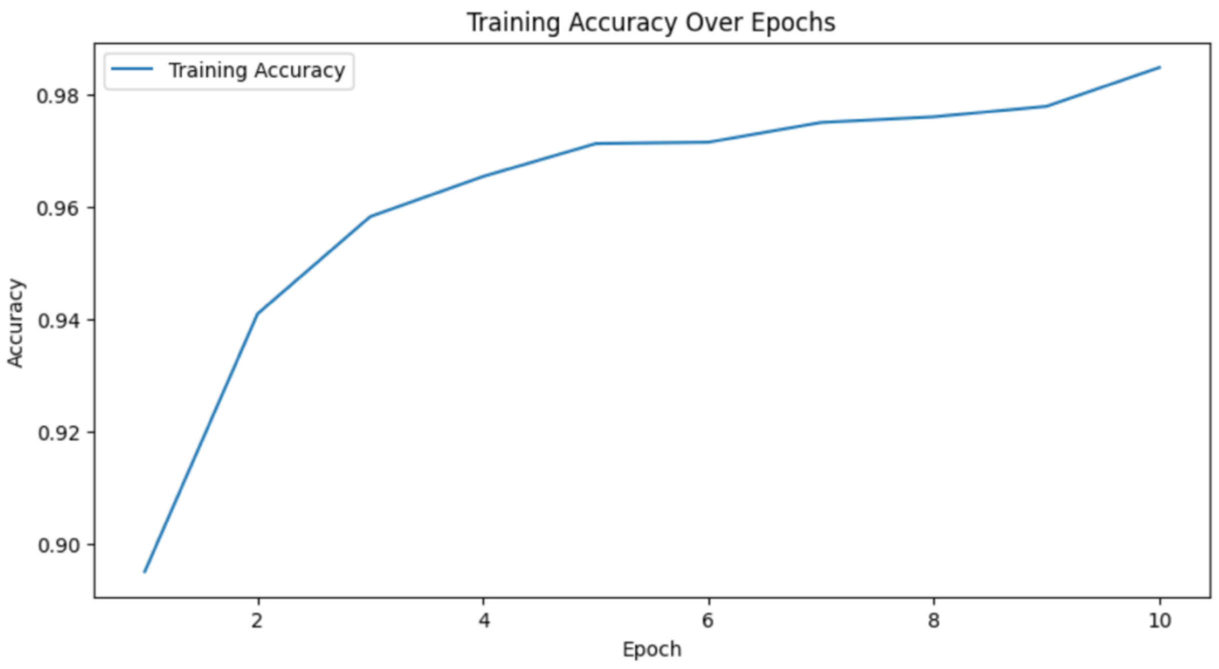
```
# Evaluate the model
model.eval()
with torch.no_grad() :
  correct = 0
  total = 0

  for data, target in test_loader:
    output = model(data)
    predicted = (output.squeeze() > 0.5).float()
    total += target.size(0)
    correct += (predicted == target).sum().item()
  print(f'Accuracy: {correct / total}')
```

```
Accuracy: 0.9707993500449604
```

> Step 8: Graphs were plotted to visualize trends and patterns.



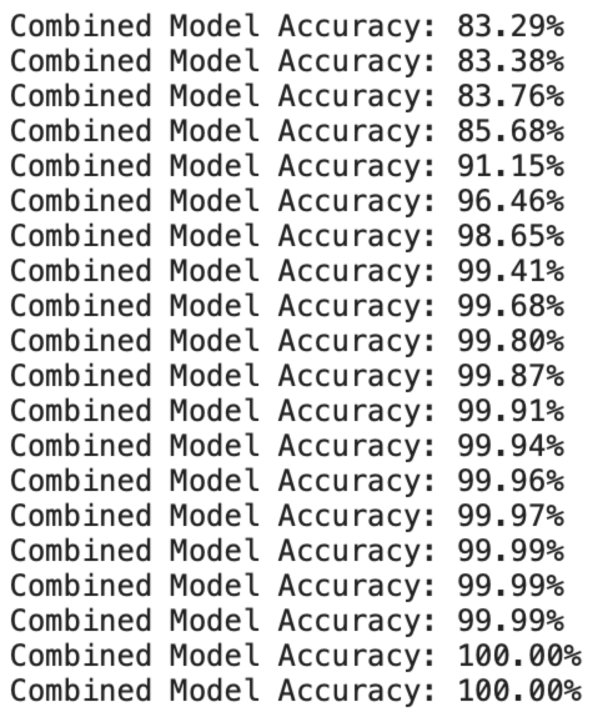


Task 2: Enhancing Model Accuracy: Integrating Last Hidden State and Feature Vectors for Improved Phishing Detection - Sample Runs

we sought to enhance it further (98%-99%) by refining the existing code. In this iteration, we combined the output of the last hidden state from the model with feature vectors extracted. The feature vectors, derived from URL length and the count of dots, were processed through a neural network with additional hidden layers.

Notably, we imposed a threshold of 70 characters for the URL length; any exceeding URLs were labeled as bad (0). Similarly, URLs with more than four dots were classified as bad.

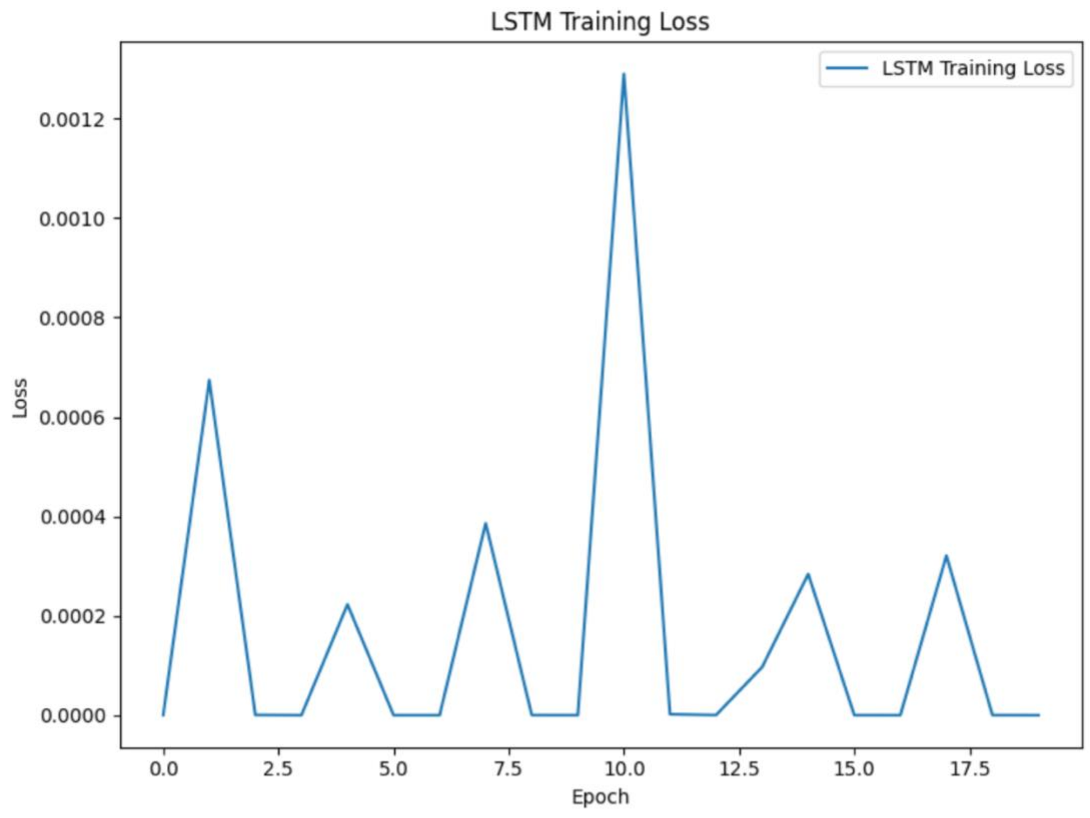
The refined model demonstrates improved accuracy from **83.29%** to **100%** over **20 epochs**, showcasing sample runs.



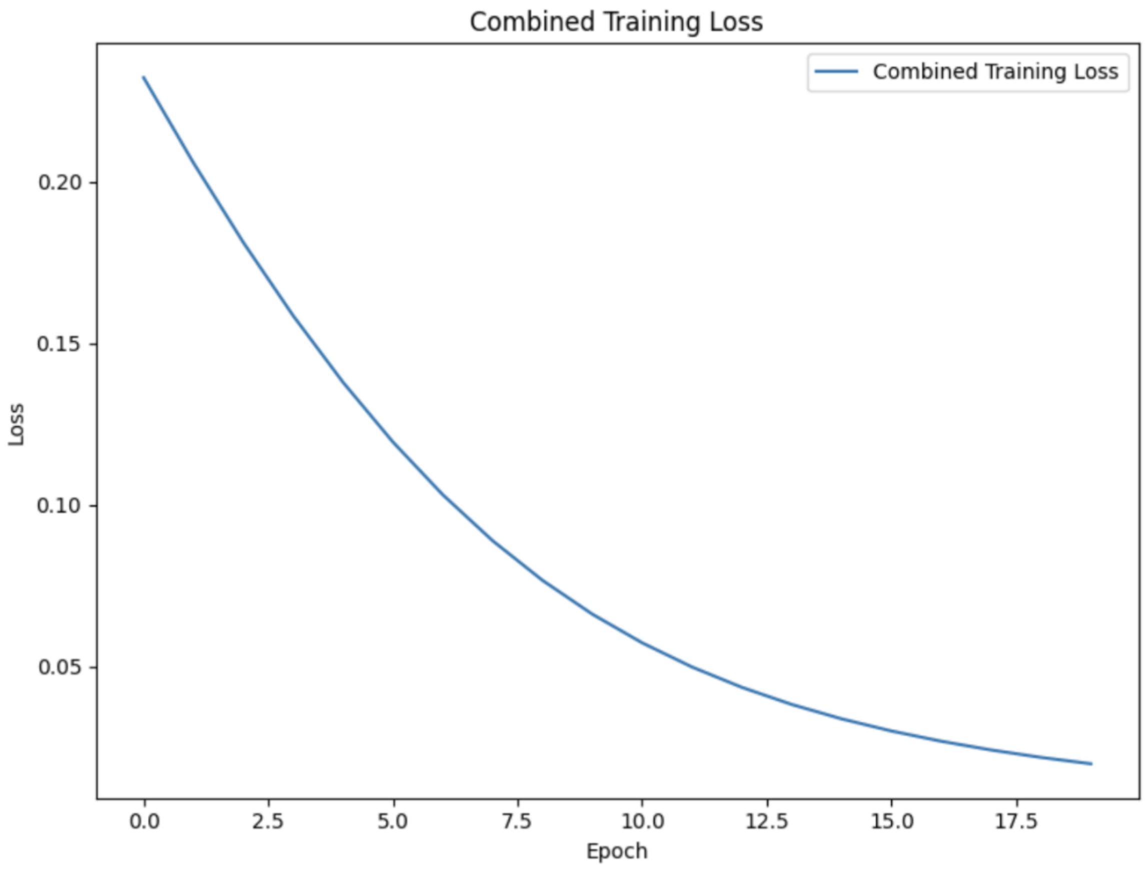
```
Combined Model Accuracy: 83.29%
Combined Model Accuracy: 83.38%
Combined Model Accuracy: 83.76%
Combined Model Accuracy: 85.68%
Combined Model Accuracy: 91.15%
Combined Model Accuracy: 96.46%
Combined Model Accuracy: 98.65%
Combined Model Accuracy: 99.41%
Combined Model Accuracy: 99.68%
Combined Model Accuracy: 99.80%
Combined Model Accuracy: 99.87%
Combined Model Accuracy: 99.91%
Combined Model Accuracy: 99.94%
Combined Model Accuracy: 99.96%
Combined Model Accuracy: 99.97%
Combined Model Accuracy: 99.99%
Combined Model Accuracy: 99.99%
Combined Model Accuracy: 99.99%
Combined Model Accuracy: 100.00%
Combined Model Accuracy: 100.00%
```

>> Plotting graph for LSTM Training Loss

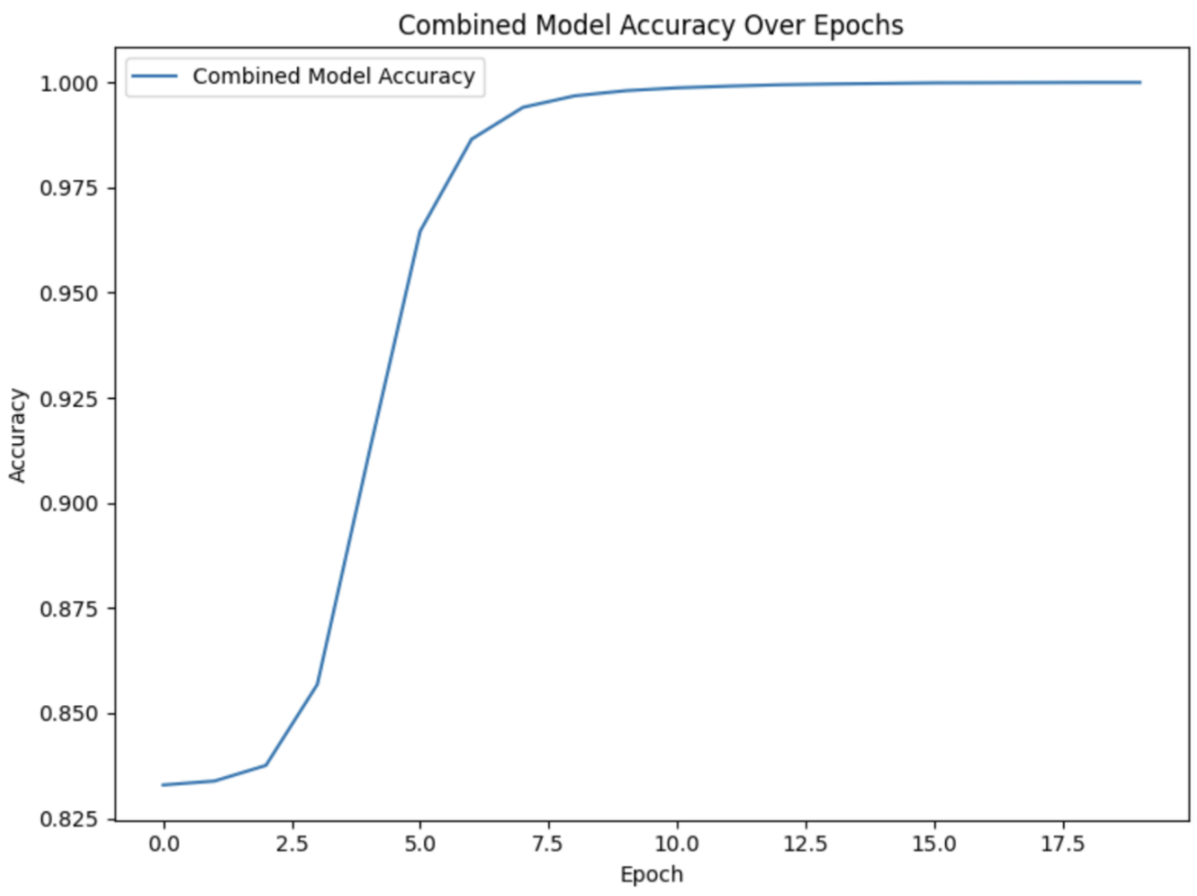
```
<matplotlib.legend.Legend at 0x79280a08c220>
```



>> Plotting for Combined Training Loss



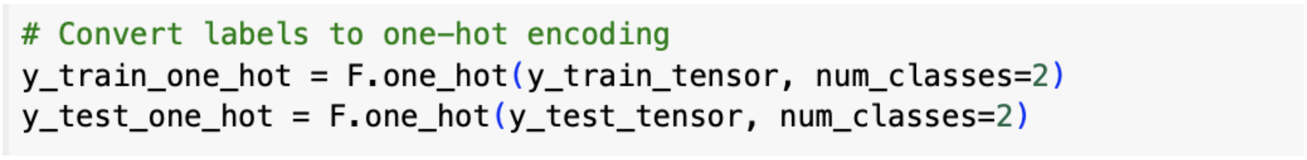
>> Plotting for Combined Model Accuracy over Epochs



Sorting One-Hot Encoding, Embedding, and LSTM Structure

> One-Hot Encoding in the Code:
The one-hot encoding step is performed using the "***F.one_hot***" function from the PyTorch library ***"import torch.nn.functional as F"***. This step converts the categorical labels (binary classification in this case) into one-hot encoded vectors. Specifically, for each sample, it converts a single integer label into a binary vector of length equal to the number of classes. For binary classification, this vector has two elements, and the element corresponding to the true class is set to 1, while the other is set to 0.

Here's the relevant part of the code:



```
# Convert labels to one-hot encoding
y_train_one_hot = F.one_hot(y_train_tensor, num_classes=2)
y_test_one_hot = F.one_hot(y_test_tensor, num_classes=2)
```

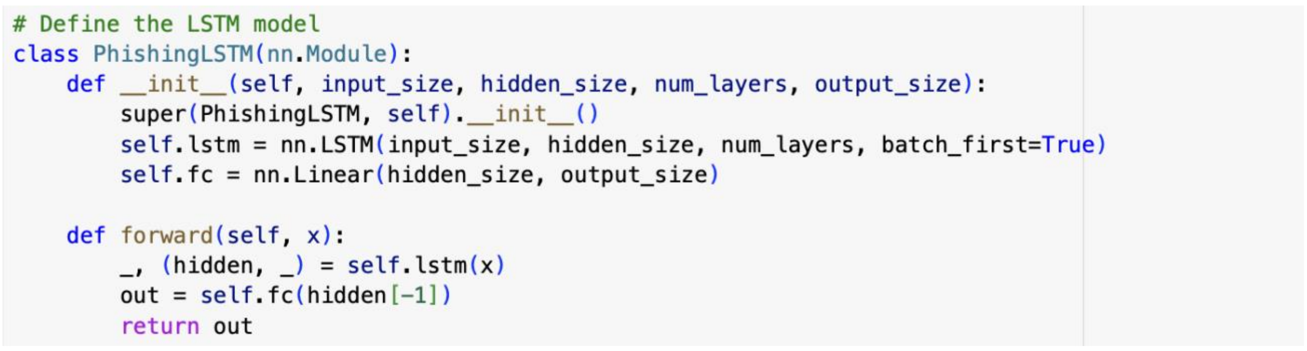
> Embedding Step Explanation:
In my code, I did not include an embedding layer because the input features, specifically `num_dots` and `url_len`, were already in numerical form. Embedding layers are typically used when dealing with categorical variables or discrete indices that need to be transformed into continuous vectors. For instance, in natural language processing (NLP), words are often represented using embeddings to capture semantic relationships.

In my case, the input features were numerical representations of the number of dots and URL length, and there was no need for an additional layer to convert categorical variables into continuous vectors. This is because embedding layers are specifically designed for scenarios where we have categorical data that requires transformation.

For example, if I were working with word indices to represent different features, I would have utilized an embedding layer to convert those indices into continuous vectors before feeding them into the LSTM. However, given the numerical nature of my input features, the inclusion of an embedding layer was not necessary for the tasks at hand.

> LSTM Structure Overview:
The LSTM structure in the code is defined by the PhishingLSTM class. It has an LSTM layer (nn.LSTM) with a specified input size, hidden size, number of layers, and batch-first set to True. The LSTM is followed by a fully connected (linear) layer (nn.Linear) that maps the hidden state to the output size. The last hidden state of the LSTM is used for making predictions.

Here's the relevant part of the LSTM model:

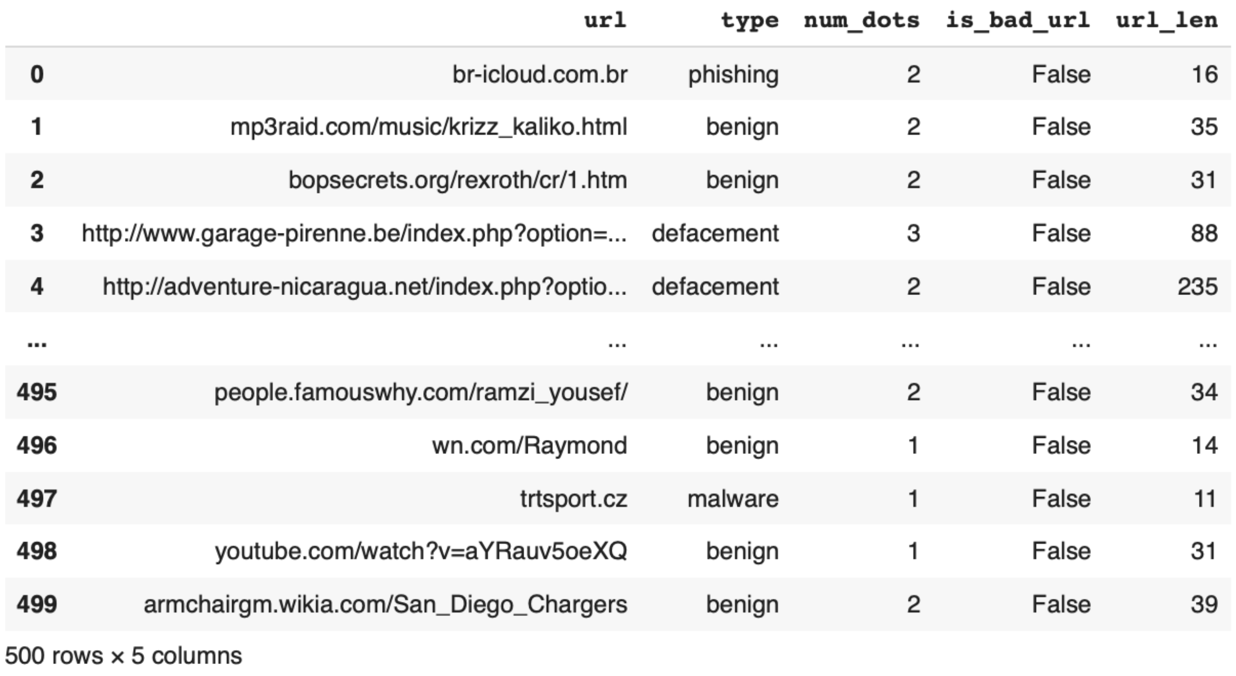


```
# Define the LSTM model
class PhishingLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(PhishingLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (hidden, _) = self.lstm(x)
        out = self.fc(hidden[-1])
        return out
```

Deep Neural Network in PyTorch:

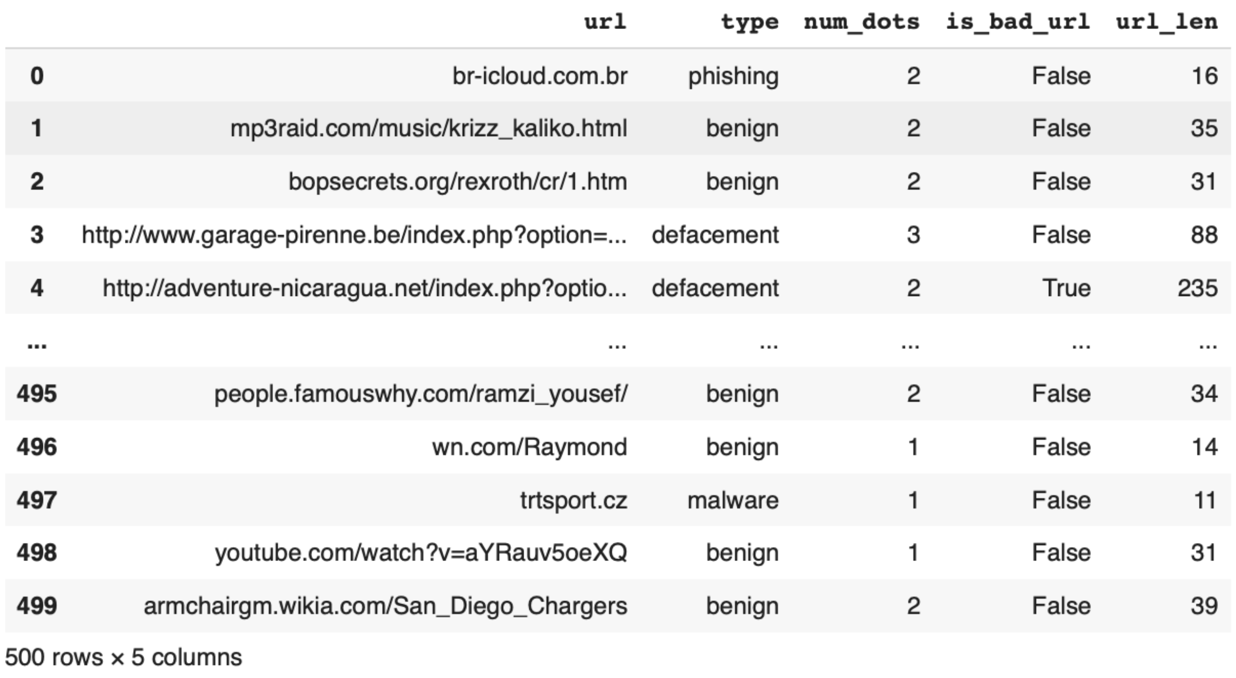
> Feature Extraction for Number of Dots



	url	type	num_dots	is_bad_url	url_len
0	br-icloud.com.br	phishing	2	False	16
1	mp3raid.com/music/krizz_kaliko.html	benign	2	False	35
2	bopsecrets.org/rexroth/cr/1.htm	benign	2	False	31
3	http://www.garage-pirenne.be/index.php?option=...	defacement	3	False	88
4	http://adventure-nicaragua.net/index.php?optio...	defacement	2	False	235
...	...	...	...	...	...
495	people.famouswhy.com/ramzi_yousef/	benign	2	False	34
496	wn.com/Raymond	benign	1	False	14
497	trtsport.cz	malware	1	False	11
498	youtube.com/watch?v=aYRauv5oeXQ	benign	1	False	31
499	armchairgm.wikia.com/San_Diego_Chargers	benign	2	False	39

500 rows × 5 columns

> Feature Extraction for Number of Dots

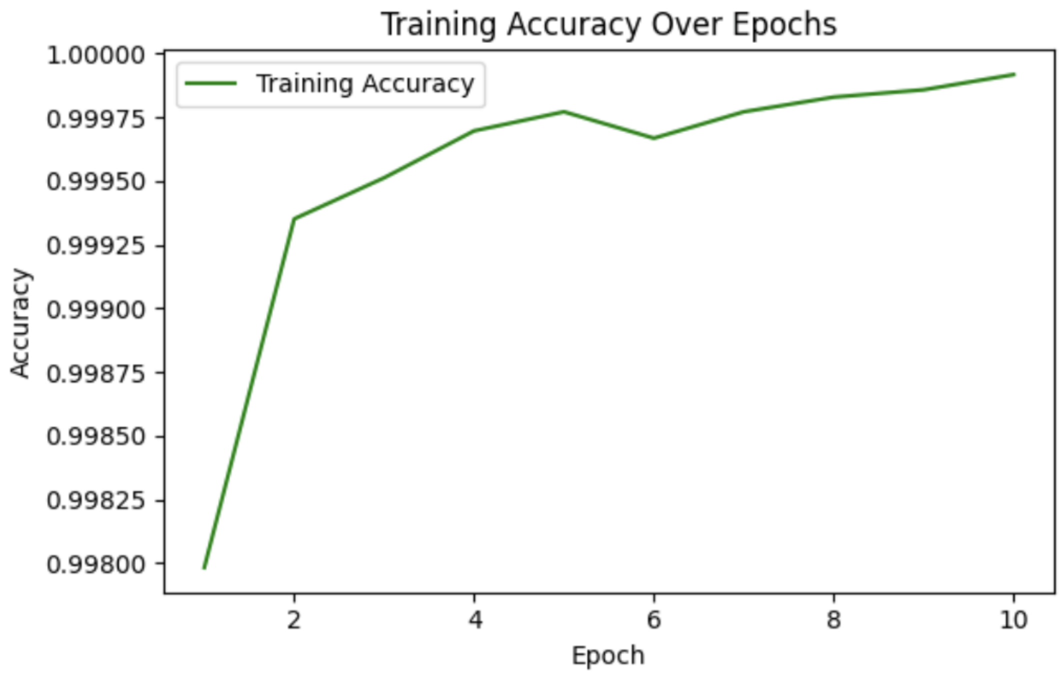


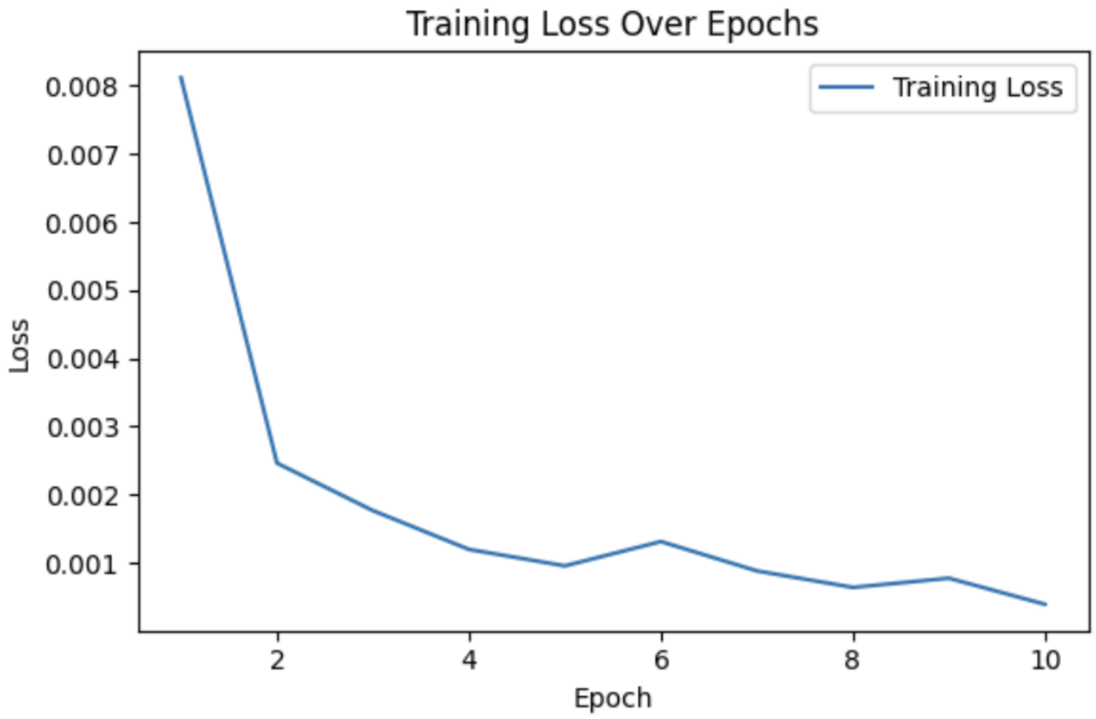
	url	type	num_dots	is_bad_url	url_len
0	br-icloud.com.br	phishing	2	False	16
1	mp3raid.com/music/krizz_kaliko.html	benign	2	False	35
2	bopsecrets.org/rexroth/cr/1.htm	benign	2	False	31
3	http://www.garage-pirenne.be/index.php?option=...	defacement	3	False	88
4	http://adventure-nicaragua.net/index.php?optio...	defacement	2	True	235
...	...	...	...	...	...
495	people.famouswhy.com/ramzi_yousef/	benign	2	False	34
496	wn.com/Raymond	benign	1	False	14
497	trtsport.cz	malware	1	False	11
498	youtube.com/watch?v=aYRauv5oeXQ	benign	1	False	31
499	armchairgm.wikia.com/San_Diego_Chargers	benign	2	False	39

500 rows × 5 columns

```
Epoch [1/10], Training Loss: 0.0081, Training Accuracy: 99.80%
Epoch [2/10], Training Loss: 0.0025, Training Accuracy: 99.94%
Epoch [3/10], Training Loss: 0.0018, Training Accuracy: 99.95%
Epoch [4/10], Training Loss: 0.0012, Training Accuracy: 99.97%
Epoch [5/10], Training Loss: 0.0010, Training Accuracy: 99.98%
Epoch [6/10], Training Loss: 0.0013, Training Accuracy: 99.97%
Epoch [7/10], Training Loss: 0.0009, Training Accuracy: 99.98%
Epoch [8/10], Training Loss: 0.0006, Training Accuracy: 99.98%
Epoch [9/10], Training Loss: 0.0008, Training Accuracy: 99.99%
Epoch [10/10], Training Loss: 0.0004, Training Accuracy: 99.99%
Testing Accuracy: 100.00%
```

> Plotting the results.





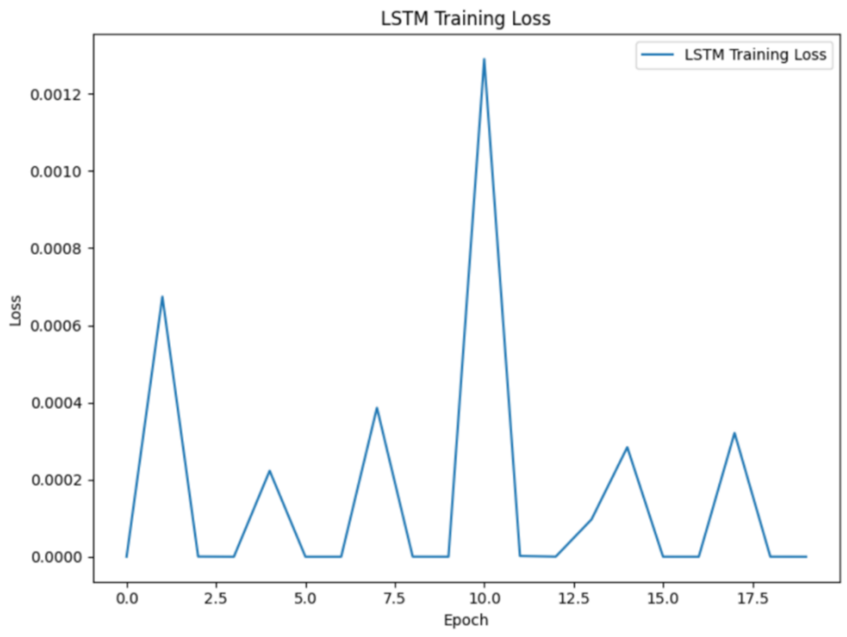
Implementing LSTM in PyTorch

The refined model demonstrates improved accuracy from **83.29%** to **100%** over **20 epochs**, showcasing sample runs.

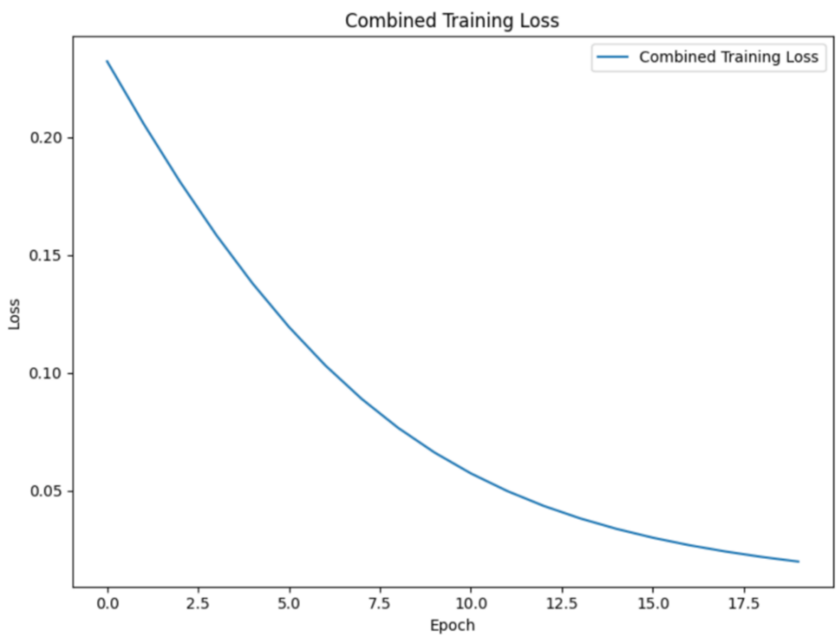
```
Combined Model Accuracy: 83.29%
Combined Model Accuracy: 83.38%
Combined Model Accuracy: 83.76%
Combined Model Accuracy: 85.68%
Combined Model Accuracy: 91.15%
Combined Model Accuracy: 96.46%
Combined Model Accuracy: 98.65%
Combined Model Accuracy: 99.41%
Combined Model Accuracy: 99.68%
Combined Model Accuracy: 99.80%
Combined Model Accuracy: 99.87%
Combined Model Accuracy: 99.91%
Combined Model Accuracy: 99.94%
Combined Model Accuracy: 99.96%
Combined Model Accuracy: 99.97%
Combined Model Accuracy: 99.99%
Combined Model Accuracy: 99.99%
Combined Model Accuracy: 99.99%
Combined Model Accuracy: 100.00%
Combined Model Accuracy: 100.00%
```

>> Plotting graph for LSTM Training Loss

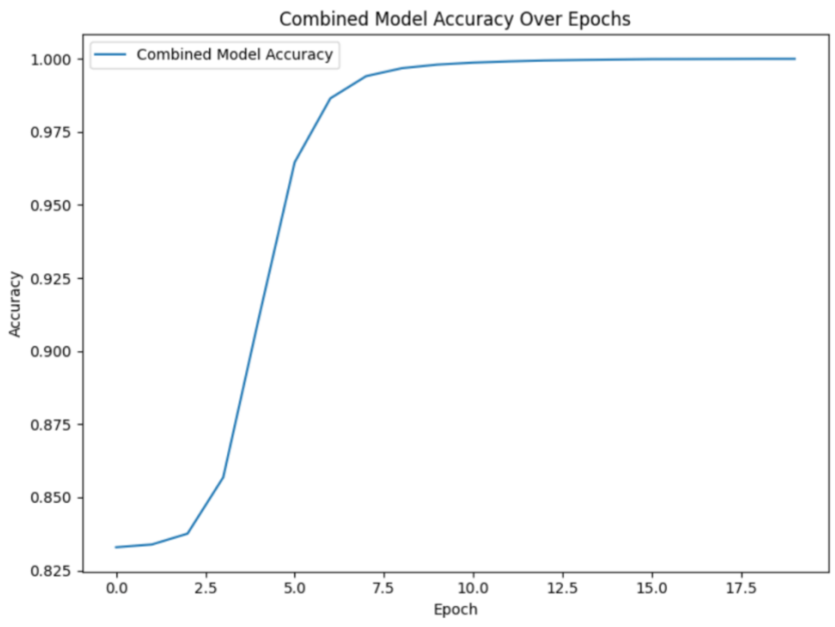
```
<matplotlib.legend.Legend at 0x79280a08c220>
```



>> Plotting for Combined Training Loss



>> Plotting for Combined Model Accuracy over Epochs


Combined Model Accuracy Over Epochs
Combined Model Accuracy
Accuracy
Epoch