```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import hashlib
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertModel


# Load dataset
df = pd.read_csv("/content/phishing_dataset3.csv")
df.shape
```

```
(10000, 50)
```

```
# ------------------- Clean data -------------------
df['MissingTitle'] = df['MissingTitle'].fillna("Unknown")
df.drop_duplicates(inplace=True)
df['CLASS_LABEL'] = df['CLASS_LABEL'].astype(int)
print("Unique values in 'CLASS_LABEL':", df['CLASS_LABEL'].unique())
```

```
Unique values in 'CLASS_LABEL': [1 0]
```

```
# ------------------- URL Encoding -------------------
def url_to_numerical(url):
    return [ord(char) for char in str(url)]

df['numerical_url'] = df['id'].apply(url_to_numerical)



# ------------------- 2D Bloom Filter -------------------
class BloomFilter2D:
    def __init__(self, rows, cols, hash_functions):
        self.rows = rows
        self.cols = cols
        self.hash_functions = hash_functions
        self.bit_matrix = [[0] * cols for _ in range(rows)]

    def add(self, item):
        for func in self.hash_functions:
            row_idx, col_idx = func(item)
            row_idx %= self.rows
            col_idx %= self.cols
            self.bit_matrix[row_idx][col_idx] = 1

    def contains(self, item):
        for func in self.hash_functions:
            row_idx, col_idx = func(item)
            row_idx %= self.rows
            col_idx %= self.cols
            if self.bit_matrix[row_idx][col_idx] == 0:
                return False
        return True

# Example hash functions for 2D indexing
def hash_func1(item):
    h = hashlib.md5(str(item).encode()).hexdigest()
    return int(h[:8], 16), int(h[8:16], 16)

def hash_func2(item):
    h = hashlib.sha1(str(item).encode()).hexdigest()
    return int(h[:8], 16), int(h[8:16], 16)

bloom_filter_2d = BloomFilter2D(rows=100, cols=100, hash_functions=[hash_func1, hash_func2])

# Add URLs to the Bloom filter
for numerical_url in df['numerical_url']:
    bloom_filter_2d.add(tuple(numerical_url))
```

```
# Create a feature based on presence in the Bloom filter
df['bloom_feature'] = df['numerical_url'].apply(lambda x: bloom_filter_2d.contains(tuple(x))).astype(int)


# ------------------- Train-test split -------------------
X = df.drop(columns=['CLASS_LABEL'])
y = df['CLASS_LABEL']
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)


# ------------------- BERT embeddings -------------------
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embeddings(urls):
    encoded_input = tokenizer(urls, padding=True, truncation=True, return_tensors='pt')
    with torch.no_grad():
        model_output = model(**encoded_input)
    embeddings = model_output.last_hidden_state[:, 0, :].numpy()
    return embeddings

# Ensure URLs are strings
X_train['id'] = X_train['id'].astype(str)
X_val['id'] = X_val['id'].astype(str)
X_test['id'] = X_test['id'].astype(str)

# Generate embeddings
X_train_embeddings = get_bert_embeddings(X_train['id'].tolist())
X_val_embeddings = get_bert_embeddings(X_val['id'].tolist())
X_test_embeddings = get_bert_embeddings(X_test['id'].tolist())

# Combine embeddings + bloom feature
def combine_features(df_split, embeddings):
    other_features = df_split[['bloom_feature']].values
    return np.concatenate((other_features, embeddings), axis=1)

X_train_final = combine_features(X_train, X_train_embeddings)
X_val_final = combine_features(X_val, X_val_embeddings)
X_test_final = combine_features(X_test, X_test_embeddings)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 3.09kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 4.82MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 2.17MB/s]
config.json: 100%  570/570 [00:00<00:00, 21.1kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfc
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%  440M/440M [00:01<00:00, 256MB/s]
```

```
# ------------------- BiLSTM model -------------------
class PhishingBiLSTM(nn.Module):
    def __init__(self, input_size, hidden_size=64, num_layers=1, num_classes=2):
        super(PhishingBiLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, bidirectional=True)
        self.fc1 = nn.Linear(hidden_size * 2, 128)  # *2 because of bidirection
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        # Prepare input for LSTM (Batch, Time, Feature)
        x = x.unsqueeze(1)  # (Batch, Time=1, Feature)
        lstm_out, _ = self.lstm(x)
        lstm_out_last = lstm_out[:, -1, :]  # Take last output
        x = self.relu(self.fc1(lstm_out_last))
        x = self.fc2(x)
        return x
```

```
# ------------------- DataLoader -------------------
X_train_tensor = torch.tensor(X_train_final, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)
X_val_tensor = torch.tensor(X_val_final, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val.values, dtype=torch.long)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
val_dataset = TensorDataset(X_val_tensor, y_val_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)


#------------------- Training -------------------
input_size = X_train_final.shape[1]
num_classes = 2
model_bilstm = PhishingBiLSTM(input_size, hidden_size=64, num_layers=1, num_classes=num_classes)
optimizer = optim.Adam(model_bilstm.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

num_epochs = 50
train_accuracies = []
val_accuracies = []
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    # Training
    model_bilstm.train()
    correct = 0
    total = 0
    running_loss = 0.0
    for data, target in train_loader:
        optimizer.zero_grad()
        output = model_bilstm(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * data.size(0)
        pred = output.argmax(dim=1)
        correct += (pred == target).sum().item()
        total += target.size(0)
    train_acc = 100. * correct / total
    train_loss = running_loss / total

    # Validation
    model_bilstm.eval()
    val_correct = 0
    val_total = 0
    val_loss_total = 0.0
    with torch.no_grad():
        for data, target in val_loader:
            output = model_bilstm(data)
            loss = criterion(output, target)
            val_loss_total += loss.item() * data.size(0)
            pred = output.argmax(dim=1)
            val_correct += (pred == target).sum().item()
            val_total += target.size(0)
    val_acc = 100. * val_correct / val_total
    val_loss = val_loss_total / val_total

    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)
    train_losses.append(train_loss)
    val_losses.append(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Acc: {train_acc:.2f}%, Val Acc: {val_acc:.2f}%, Train Loss: {train_loss:.4f}, Val Loss: {val
```

```
Epoch 1/50, Train Acc: 94.35%, Val Acc: 97.30%, Train Loss: 0.1412, Val Loss: 0.0641
Epoch 2/50, Train Acc: 99.03%, Val Acc: 99.40%, Train Loss: 0.0303, Val Loss: 0.0155
Epoch 3/50, Train Acc: 99.33%, Val Acc: 99.50%, Train Loss: 0.0179, Val Loss: 0.0221
Epoch 4/50, Train Acc: 98.91%, Val Acc: 99.40%, Train Loss: 0.0293, Val Loss: 0.0181
Epoch 5/50, Train Acc: 99.85%, Val Acc: 99.80%, Train Loss: 0.0047, Val Loss: 0.0063
Epoch 6/50, Train Acc: 99.72%, Val Acc: 99.40%, Train Loss: 0.0071, Val Loss: 0.0194
Epoch 7/50, Train Acc: 99.83%, Val Acc: 99.40%, Train Loss: 0.0048, Val Loss: 0.0291
Epoch 8/50, Train Acc: 99.33%, Val Acc: 99.60%, Train Loss: 0.0209, Val Loss: 0.0199
Epoch 9/50, Train Acc: 99.61%, Val Acc: 98.90%, Train Loss: 0.0111, Val Loss: 0.0338
Epoch 10/50, Train Acc: 100.00%, Val Acc: 99.50%, Train Loss: 0.0008, Val Loss: 0.0259
```

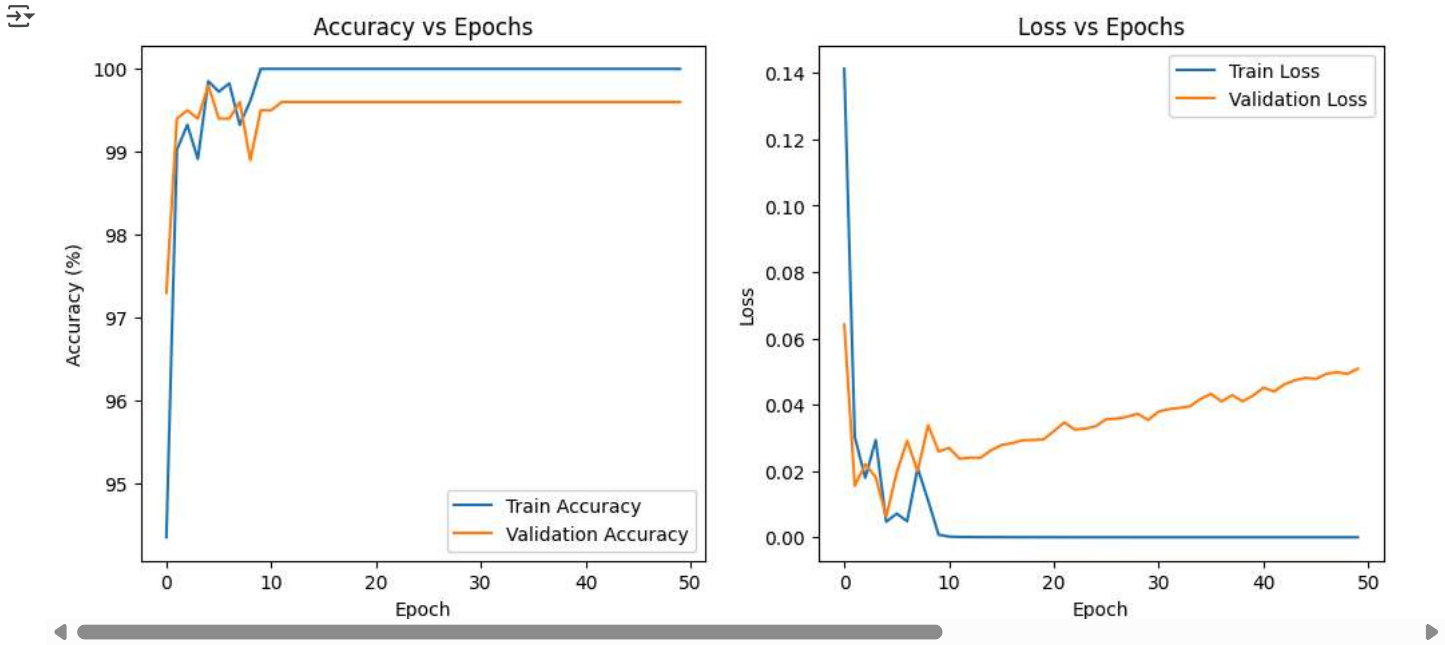
```
Epoch 11/50, Train Acc: 100.00%, Val Acc: 99.50%, Train Loss: 0.0002, Val Loss: 0.0269
Epoch 12/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0001, Val Loss: 0.0237
Epoch 13/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0001, Val Loss: 0.0240
Epoch 14/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0240
Epoch 15/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0262
Epoch 16/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0278
Epoch 17/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0283
Epoch 18/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0293
Epoch 19/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0293
Epoch 20/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0295
Epoch 21/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0320
Epoch 22/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0346
Epoch 23/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0325
Epoch 24/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0328
Epoch 25/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0335
Epoch 26/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0356
Epoch 27/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0358
Epoch 28/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0363
Epoch 29/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0372
Epoch 30/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0354
Epoch 31/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0379
Epoch 32/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0386
Epoch 33/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0390
Epoch 34/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0395
Epoch 35/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0417
Epoch 36/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0432
Epoch 37/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0409
Epoch 38/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0429
Epoch 39/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0410
Epoch 40/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0427
Epoch 41/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0451
Epoch 42/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0440
Epoch 43/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0461
Epoch 44/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0473
Epoch 45/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0481
Epoch 46/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0478
Epoch 47/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0493
Epoch 48/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0498
Epoch 49/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0493
Epoch 50/50, Train Acc: 100.00%, Val Acc: 99.60%, Train Loss: 0.0000, Val Loss: 0.0508
```

```
# ------------------- Evaluation -------------------
X_test_tensor = torch.tensor(X_test_final, dtype=torch.float32)
with torch.no_grad():
    outputs = model_bilstm(X_test_tensor)
    preds = outputs.argmax(dim=1).numpy()

# Accuracy
accuracy = accuracy_score(y_test, preds)
print(f'Accuracy: {accuracy:.4f}')

# Classification Report
print("\nClassification Report:\n", classification_report(y_test, preds))
```

```
Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       500
           1       1.00      1.00      1.00       500

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000
```

```
# ------------------- Accuracy & Loss Plots -------------------
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.title('Accuracy vs Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
```

```
plt.title('Loss vs Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Confusion Matrix
cm = confusion_matrix(y_test, preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, square=True)
plt.title('Confusion Matrix (4x4 Size)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```