International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 2, Mar-Apr-2026, ISSN (Online): 2395-566X

A New Website Fingerprinting Method for Tor Hidden Service

Dr Y Subba Reddy[1], A Guru Jyotshna[2], K Deepthi[3], B Paramesh[4], D.Siva Ganga Keerthi[5]
[1]HOD & Professor, Department of Computer Science and Engineering, Sai Rajeswari Institute of Technology
[2,3,4,5]UG students, Department of Computer Science and Engineering, Sai Rajeswari Institute of Technology

Abstract- Neuroplasticity, as the name suggests, refers to the brain's remarkable ability to reorganize itself by forming new connections throughout life. Neuroplasticity has been observed to be more active in early childhood, as the processes of synaptic pruning and myelination are more active during this period. Research has shown that environmental stimulation has a direct effect on the thickness of the cortex, as well as the dendritic branching patterns of the neurons. Functional magnetic resonance imaging has shown that the brains of adults have a lot of plasticity, which enables the brains to recover from injury as well as to learn new skills. The neuroplasticity framework has a lot of implications, especially in the field of educational psychology as well as rehabilitation medicine. Experimental results using crawled Tor URL datasets demonstrate that the proposed method achieves 97.50% accuracy, outperforming conventional CNN-based deep fingerprinting techniques. Further optimization is achieved by incorporating a BiGRU layer after LSTM, enabling bidirectional feature extraction and improving prediction performance to 97.86%. Performance metrics including precision, recall, F1-score, and confusion matrices confirm the enhanced effectiveness of this methodology for distinguishing normal and attack-type Tor services, providing a robust framework for secure network monitoring.

Keywords – Tor network, hidden services, web fingerprinting, BERT, LSTM, BiGRU, time-series analysis, network security.

I. INTRODUCTION

The Tor network is a widely used privacy-preserving system designed to provide anonymous communication over the Internet. By employing multi-layered encryption and onion routing, Tor conceals user identity, browsing behavior, and location, allowing users to communicate without revealing sensitive information. Each message is encrypted in multiple layers and relayed through a distributed network of volunteer nodes, which makes traffic analysis and user tracking challenging. While these features enhance privacy and protect legitimate users, they also create opportunities for malicious actors to exploit the network for illegal activities, including drug trafficking, weapon sales, and distribution of malware.

The anonymous nature of Tor hidden services complicates the monitoring and detection of such activities, presenting significant challenges in network security and law enforcement. Effective identification and classification of Tor hidden services are essential to mitigate risks, prevent illegal operations, and ensure safer online environments. Traditional approaches relying solely on network-level features, such as packet size and flow patterns, often fall short in accurately distinguishing between normal and malicious services due to the dynamic and encrypted nature of traffic. This necessitates the exploration of advanced techniques that can leverage both semantic and temporal patterns to enhance the detection and monitoring of hidden Tor services.

Objective

- To develop an advanced website fingerprinting framework for accurate detection and classification of Tor hidden services.
- To utilize the CNN-based Deep Fingerprinting model for extracting traffic-related features and evaluating baseline performance.
- To apply the BERT model for generating high-dimensional feature representations from textual content of Tor hidden service web pages.
- To use an LSTM network to capture sequential dependencies and temporal behavior patterns in the extracted features.
- To integrate a BiGRU layer for bidirectional feature extraction and improve robustness and classification efficiency in identifying normal and malicious Tor services.

II. SYSTEM DESIGN

SYSTEM ARCHITECTURE

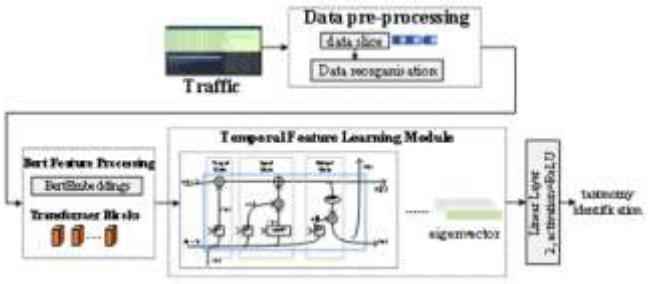


Fig. System Architecture

DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a visual representation that illustrates how information moves through a system, showing the flow of data between processes, storage points, and external entities. In the context of this project, a DFD helps explain how learner-submitted code travels through different functional components to generate meaningful feedback. It highlights how the system receives input, processes the code through validation and analysis stages, and delivers constructive responses back to the learner.By mapping these interactions, the DFD clarifies the transformation of raw code input into structured feedback that supports a smoother learning experience. It focuses on the movement and handling of information rather than internal workings, allowing clear understanding of major functional pathways. In this project, the DFD emphasizes how the system ensures efficient feedback flow, minimizes unnecessary token usage, and maintains accuracy throughout the interaction cycle. It offers a simplified overview of the feedback pipeline, showing how data enters, passes through evaluation processes, and exits as refined guidance. This understanding helps ensure transparency in system behavior and supports effective communication of how the code review mechanism operates.

III. IMPLEMENTATION

SAMPLE CODE

```
import os
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, ZeroPadding2D, BatchNormalization, Activation, MaxPooling2D, Flatten
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import cv2
from os import listdir
import time
%matplotlib inline
input_path = '/kaggle/input/socofing/SOCOFing/'
real_path = '/kaggle/input/socofing/SOCOFing/Real/'
altered_path = '/kaggle/input/socofing/SOCOFing/Altered/Altered-Medium/'
def split_data(x, y, test_size=0.2):
    x_train, x_test_val, y_train, y_test_val = train_test_split(x, y, test_size=test_size)
    x_test, x_val, y_test, y_val = train_test_split(x_test_val, y_test_val, test_size=0.5)
    return x_train, y_train, x_val, y_val, x_test, y_test
def load_data(dir_list, image_size):
  x = []
  y = []
  image_width, image_height = image_size
  for directory in dir_list:
    flist = listdir(directory)
    for f in flist:
      img = cv2.imread(directory + '/' + f)
      img = cv2.resize(img, (image_width, image_height), interpolation=cv2.INTER_CUBIC)
      img = img / 255.0
      x.append(img)
      if directory[-5:-1] != 'Real':
        y.append(0)
      else:
        y.append(1)
  x = np.array(x)
  y = np.array(y)

  x, y = shuffle(x, y)
  return x, y
x, y = load_data([real_path, altered_path], (96, 103))
x_train, y_train, x_val, y_val, x_test, y_test = split_data(x, y, test_size=0.2)
class_names = ['Real', 'Altered']
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(y_train[i])
plt.show()
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(103, 96, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size = 32, epochs=100,
                    validation_data=(x_val, y_val))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
val_loss, val_acc = model.evaluate(x_val, y_val, verbose=2)
y_pred = model.predict(x_test)
y_pred_argmax = np.argmax(y_pred, axis=1)
accuracy = tf.keras.metrics.Accuracy()(y_test, y_pred_argmax)
precision = tf.keras.metrics.Precision()(y_test, y_pred_argmax)
recall = tf.keras.metrics.Recall()(y_test, y_pred_argmax)
f1 = 2 * (precision * recall) / (precision + recall)
print("Accuracy: %", format(accuracy.numpy()*100))
print("Precision: %", format(precision.numpy()*100))
print("Recall: %", format(recall.numpy()*100))
print("F1-score: %", format(f1.numpy()*100))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

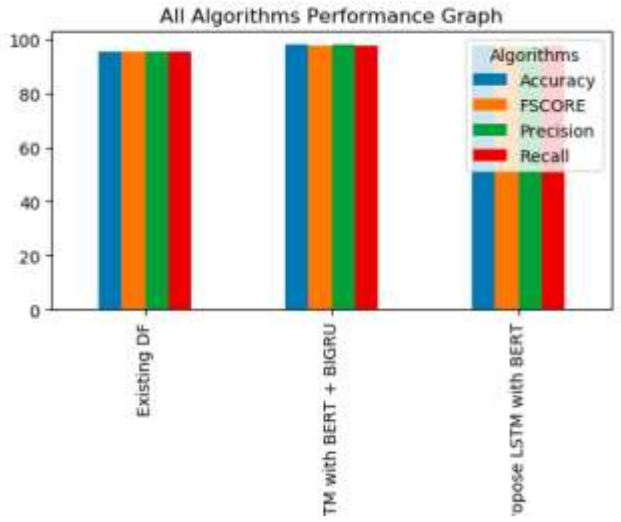
IV. RESULTS

PERFORMANCE EVALUATION TABLES

Table.1 Performance Evaluation

Algorithm Name	Accuracy	Precision	Recall	F-Score
Existing CNN Based Deep Fingerprint	95.374	95.207	95.310	95.257
Propose LSTM with BERT Fingerprint	97.509	97.393	97.502	97.446
Extension LSTM with BERT + BiGRU	97.865	97.926	97.692	97.803

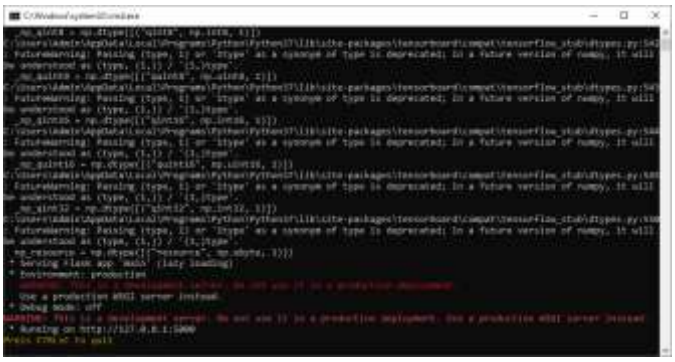
COMPARISON GRAPHS



Comparison Graph

SCREENSHOTS

To run web prediction double click on 'runFlask.bat' file to start flask server and then will get below page

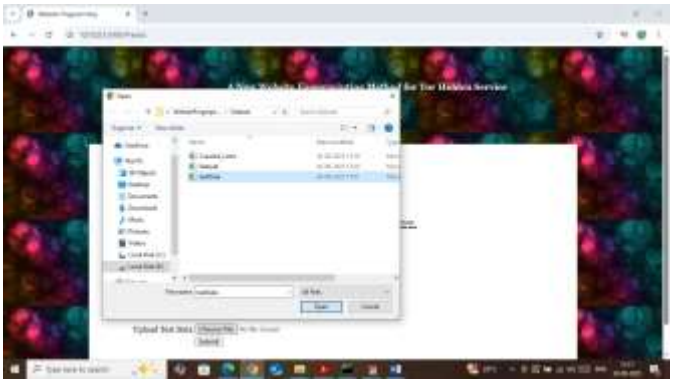


In above screen flask server started and now open browser and enter URL as http://127.0.0.1:5000/index and then press enter key to get below page

In above screen click on 'User Login' link to get below page

In above screen user is login by entering username and password as 'admin and admin'. After login will get below page

In above screen click on 'Predict Tor Service Type' link to get below page



In above screen selecting and uploading test data file and then click on buttons to get below page

In above screen in first column can see TOR URL web homepage text data and then in second column can see predicted TOR URL type as 'Normal or Attack'.

Future Scope

Future work can focus on expanding the system to handle larger and more diverse Tor hidden services by incorporating richer text sources and multilingual content. Additional enhancements may include integrating adaptive sequence modelling techniques to better capture rapidly changing website structures. The system can also be improved by adopting lightweight embedding methods that reduce computation time without sacrificing performance. Further experimentation with hybrid deep learning layers may offer deeper contextual understanding of hidden service behaviour. Extending the framework into real-time monitoring environments could enhance practical usability and early threat detection. Incorporating continuous learning mechanisms may help the model automatically adapt to new patterns, ensuring long-term robustness against evolving malicious Tor activities.

V. CONCLUSION

The study demonstrates that combining BERT-based feature extraction with sequential time-series modeling significantly improves the identification of Tor hidden service types. By

converting textual content from Tor homepages and sub-pages into sequential windows, the LSTM model effectively captures temporal changes in web data, enabling more accurate differentiation between normal and attack-type services. Experimental evaluation on a curated Tor URL dataset shows that the LSTM model with BERT fingerprinting achieves 97.50% accuracy, indicating a substantial improvement over conventional CNN-based deep fingerprinting techniques. Further enhancement is realized by integrating a BiGRU layer with the LSTM-BERT framework, allowing bidirectional feature learning and filtering, which increases prediction accuracy to 97.86%. This high-performance model also demonstrates strong precision, recall, and F1-score metrics, reflecting reliable detection capabilities with minimal misclassifications. The results validate that leveraging advanced natural language processing models alongside recurrent architectures provides an effective mechanism for Tor traffic analysis and monitoring. These findings highlight the potential of sequential feature extraction from textual data as a robust approach for network security applications, supporting proactive identification of malicious hidden services while maintaining overall efficiency in computation and training.

REFERENCES

1. Chen, Y., Wang, Y., & Yang, L. (2022). SRP: A microscopic look at the composition mechanism of website fingerprinting. Applied Sciences, 12(15), 7937.
2. Bang, J., Jeong, J., & Lee, J. (2023). Fedfingerprinting: A federated learning approach to website fingerprinting attacks in tor networks. IEEE Access, 11, 78431-78444.
3. Ding, Y., & Hu, B. (2024). A Multi-Granularity Features Representation and Dimensionality Reduction Network for Website Fingerprinting. IEEE Access.
4. Sun, H., Huang, Y., Han, L., Long, X., Liu, H., & Zhou, C. (2022, December). Neural-FacTOR: Neural Representation Learning for Website Fingerprinting Attack over TOR Anonymity. In 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) (pp. 435-440). IEEE.
5. Bahramali, A., Bozorgi, A., & Houmansadr, A. (2023, November). Realistic website fingerprinting by augmenting network traces. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (pp. 1035-1049).
6. Y. Wang, R. Chen, X. Huang, and B. Wang, "Secure anonymous communication on corrupted machines with reverse firewalls," IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 6, pp. 3837–3854, 2022.
7. Q. Tan, X. Wang, W. Shi, J. Tang, and Z. Tian, "An anonymity vulnerability in Tor," IEEE/ACM Transactions on Networking, vol. 30, no. 6, pp. 2574–2587, 2022.
8. I. Karunanayake, J. Jiang, N. Ahmed, and S. K. Jha, "Exploring uncharted waters of website fingerprinting," IEEE Transactions on Information Forensics and Security, vol. 19, pp. 1840–1854, 2024.
9. S. E. Oh, N. Mathews, M. S. Rahman, M. Wright, and N. Hopper, "GANDaLF: GAN for data-limited fingerprinting," Proceedings on Privacy Enhancing Technologies, vol. 2021, no. 2, pp. 305–322, 2021.