Blockchain espoused Recalling-Enhanced Recurrent Neural Network with Fractional Discrete Meixner Moments encryption for Cyber security in Cloud computing

```
from os import listdir
from pickle import dump
import·tensorflow·as·tf
from·numpy·import·array
from·pickle·import·load
from·keras.preprocessing.text·import·Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM as Memory_Layer
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Dropout
from keras.layers.merge import add
from tensorflow.keras.callbacks import ModelCheckpoint
import numpy as np
import cv2
import string
import os
from numpy import argmax
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu
```

```
pip install cryptography
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting cryptography
  Downloading cryptography-37.0.4-cp36-abi3-manylinux_2_24_x86_64.whl (4.1 MB)
     |████████████████████████████████| 4.1 MB 13.6 MB/s
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.7/dist-packages (from cryptography) (1.15.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from cffi>=1.12->cryptography) (2.21)
Installing collected packages: cryptography
Successfully installed cryptography-37.0.4
```

```
from scipy.stats import zscore
import pandas as pd
from cryptography.fernet import Fernet
```

adding column details

```
col_names = ["duration", "protocol_type", "service", "flag", "src_bytes",
             "dst_bytes", "land", "wrong_fragment", "urgent", "hot", "num_failed_logins",
             "logged_in", "num_compromised", "root_shell", "su_attempted", "num_root",
             "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds",
             "is_host_login", "is_guest_login", "count", "srv_count", "serror_rate",
             "srv_serror_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate",
             "diff_srv_rate", "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count",
             "dst_host_same_srv_rate", "dst_host_diff_srv_rate", "dst_host_same_src_port_rate",
             "dst_host_srv_diff_host_rate", "dst_host_serror_rate", "dst_host_srv_serror_rate",
             "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "label"]
```

```
"""loading NSL-KDD benchmark dataset"""

df = pd.read_csv("/content/sample_data/KDDTrain+_2.csv", header=None, names=col_names)
df_test = pd.read_csv("/content/sample_data/KDDTest+_2.csv", header=None, names=col_names)

df.head(5)
df.describe()
print('Label distribution Training set:')
print(df['label'].value_counts())
print()
```

```
print('Label distribution Training set:')
print(df_test[label].value_counts())
```

```
Label distribution Training set:
normal             67343
neptune            41214
satan               3633
ipsweep             3599
portsweep           2931
smurf               2646
nmap                1493
back                 956
teardrop             892
warezclient          890
pod                  201
guess_passwd          53
buffer_overflow       30
warezmaster           20
land                  18
imap                  11
rootkit               10
loadmodule             9
ftp_write              8
multihop               7
phf                    4
perl                   3
spy                    2
Name: label, dtype: int64

Label distribution Test set:
normal             9711
neptune            4657
guess_passwd       1231
mscan               996
warezmaster         944
apache2             737
satan               735
processtable        685
smurf               665
back                359
snmpguess           331
saint               319
mailbomb            293
snmpgetattack       178
portsweep           157
ipsweep             141
httptunnel          133
nmap                 73
pod                  41
buffer_overflow      20
multihop             18
named                17
ps                   15
sendmail             14
rootkit              13
xterm                13
teardrop             12
xlock                 9
land                  7
xsnoop                4
ftp_write             3
```

```
pip install Crypto
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting Crypto
  Downloading crypto-1.4.1-py2.py3-none-any.whl (18 kB)
Collecting shellescape
  Downloading shellescape-3.8.1-py2.py3-none-any.whl (3.1 kB)
Collecting Naked
  Downloading Naked-0.1.31-py2.py3-none-any.whl (590 kB)
     |████████████████████████████████| 590 kB 12.9 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from Naked->Crypto) (2.23.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (from Naked->Crypto) (3.13)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->Naked->Crypto) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->Naked->Crypto) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->Naked->Crypto) (2022.6.
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->Na
Installing collected packages: shellescape, Naked, Crypto
Successfully installed Crypto-1.4.1 Naked-0.1.31 shellescape-3.8.1
```

```
pip install pycryptodome
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pycryptodome
  Downloading pycryptodome-3.15.0-cp35-abi3-manylinux2010_x86_64.whl (2.3 MB)
     |████████████████████████████████| 2.3 MB 12.4 MB/s
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.15.0
```

```
import Crypto.Random
import time
from collections import OrderedDict
from Crypto.Hash import SHA256
from Crypto.Signature import PKCS1_v1_5
from Crypto.PublicKey import RSA
import binascii
from uuid import uuid4
import hashlib
import numpy as np
import random
import matplotlib.pyplot as plt
import pandas as pd
import networkx as nx
from networkx.drawing.nx_agraph import write_dot, graphviz_layout



DIFFICULTY = 3
WALLET_LIST=[]

def generate_wallet():

    random_gen = Crypto.Random.new().read
    private_key = RSA.generate(2048, random_gen)
    public_key = private_key.publickey()
    response = {
        'private_key': binascii.hexlify(private_key.export_key(format('PEM'))).decode('ascii'),
        'public_key': binascii.hexlify(public_key.export_key(format('PEM'))).decode('ascii')
    }
    WALLET_LIST.append((public_key, private_key))
    return response


GENESIS_ID = str(uuid4()).replace('-','')
GENESIS_KEYS = generate_wallet()




def hash_data(data):

    data= str(data).encode('utf-8')
    h = hashlib.new('sha256')
    h.update(data)
    return h.hexdigest()


def sign(sender_private_key, transaction_dict):
    private_key_obj = RSA.importKey(binascii.unhexlify(sender_private_key))
    signer_obj = PKCS1_v1_5.new(private_key_obj)
    hash_obj = SHA256.new(str(transaction_dict).encode('utf-8'))
    return binascii.hexlify(signer_obj.sign(hash_obj)).decode('ascii')


def get_previous_hashes(tips):
    previous_hashes = [tangle.transactions[tips[0]].get_hash(),
                           tangle.transactions[tips[0]].get_hash()]

    return previous_hashes


def valid_proof(previous_hashes, transaction_dict, nonce):
    guess = (str(previous_hashes)+ str(transaction_dict) + str(nonce)).encode('utf-8')
    h = hashlib.new('sha256')
    h.update(guess)
    guess_hash = h.hexdigest()
    return guess_hash[:DIFFICULTY] == '0'*DIFFICULTY


def proof_of_work(previous_hashes, transaction_dict):
    nonce = 0
    while not valid_proof(previous_hashes, transaction_dict, nonce):
        nonce = nonce + 1

    return nonce

def generate_transactions(lam=1, size=5, initial=False, initial_count=5, tip_selection_algo='weighted_random_walk'):
    if initial:
        for i in range(initial_count):
            keys = generate_wallet()
```

```
            transaction_dict = OrderedDict({
                'sender_public_key': GENESIS_KEYS['public_key'],
                'recipient_public_key': keys['public_key'],
                'data_row': 328002
                })

            tips = [GENESIS_ID, GENESIS_ID]
            previous_hashes = get_previous_hashes(tips)
            transaction = transaction_block(
                sender_public_key = transaction_dict['sender_public_key'],
                recipient_public_key = transaction_dict['recipient_public_key'],
                data= transaction_dict['data_row'],
                signature = sign(GENESIS_KEYS['private_key'], transaction_dict),
                tx_id = str(uuid4()).replace('-',''),
                approved_tx=tips,
                nonce=proof_of_work(previous_hashes, transaction_dict),
                previous_hashes=previous_hashes)

            tangle.add_transaction(transaction)


    else:
        time_length, _, _ = plt.hist(np.random.poisson(lam, size))
        plt.show(block=False)

        for t in time_length:
            for i in range(int(t)):
                keys = generate_wallet()
                transaction_dict = OrderedDict({
                'sender_public_key': GENESIS_KEYS['public_key'],
                'recipient_public_key': keys['public_key'],
                'data_row': np.random.randint(low=50,high=100)
                })

                tips = tangle.find_tips(algo=tip_selection_algo)
                previous_hashes = get_previous_hashes(tips)
                transaction = transaction_block(
                    sender_public_key = transaction_dict['sender_public_key'],
                    recipient_public_key = transaction_dict['recipient_public_key'],
                    data = transaction_dict['data_row'],
                    signature = sign(GENESIS_KEYS['private_key'], transaction_dict),
                    tx_id = str(uuid4()).replace('-',''),
                    approved_tx=tips,
                    nonce=proof_of_work(previous_hashes, transaction_dict),
                    previous_hashes=previous_hashes)

                tangle.add_transaction(transaction)
```

Split Dataset into 4 datasets for every attack category

Rename every attack label: 0=normal, 1=DoS, 2=Probe, 3=R2L and 4=U2R. Replace labels column with new labels column Make new datasets"""

```
labeldf = df['label']
labeldf_test = df_test['label']
# change the label column
newlabeldf = labeldf.replace(
    {'normal': 0, 'neptune': 1, 'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1, 'mailbomb': 1, 'apache2': 1,
     'processtable': 1, 'udpstorm': 1, 'worm': 1,
     'ipsweep': 2, 'nmap': 2, 'portsweep': 2, 'satan': 2, 'mscan': 2, 'saint': 2
        , 'ftp_write': 3, 'guess_passwd': 3, 'imap': 3, 'multihop': 3, 'phf': 3, 'spy': 3, 'warezclient': 3,
     'warezmaster': 3, 'sendmail': 3, 'named': 3, 'snmpgetattack': 3, 'snmpguess': 3, 'xlock': 3, 'xsnoop': 3,
     'httptunnel': 3,
     'buffer_overflow': 4, 'loadmodule': 4, 'perl': 4, 'rootkit': 4, 'ps': 4, 'sqlattack': 4, 'xterm': 4})
newlabeldf_test = labeldf_test.replace(
    {'normal': 0, 'neptune': 1, 'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1, 'mailbomb': 1, 'apache2': 1,
     'processtable': 1, 'udpstorm': 1, 'worm': 1,
     'ipsweep': 2, 'nmap': 2, 'portsweep': 2, 'satan': 2, 'mscan': 2, 'saint': 2
        , 'ftp_write': 3, 'guess_passwd': 3, 'imap': 3, 'multihop': 3, 'phf': 3, 'spy': 3, 'warezclient': 3,
     'warezmaster': 3, 'sendmail': 3, 'named': 3, 'snmpgetattack': 3, 'snmpguess': 3, 'xlock': 3, 'xsnoop': 3,
     'httptunnel': 3,
     'buffer_overflow': 4, 'loadmodule': 4, 'perl': 4, 'rootkit': 4, 'ps': 4, 'sqlattack': 4, 'xterm': 4})
# put the new label column back
df['label'] = newlabeldf
df_test['label'] = newlabeldf_test
print(df['label'].head())
to_drop_DoS = [2, 3, 4]
to_drop_Probe = [1, 3, 4]
to_drop_R2L = [1, 2, 4]
```

```
to_drop_U2R = [1, 2, 3]
DoS_df = df[~df['label'].isin(to_drop_DoS)];
Probe_df = df[~df['label'].isin(to_drop_Probe)];
R2L_df = df[~df['label'].isin(to_drop_R2L)];
U2R_df = df[~df['label'].isin(to_drop_U2R)];
```

```
0    0
1    0
2    1
3    0
4    0
Name: label, dtype: int64
```

```
DoS_df
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
125968	0	tcp	private	S0	0	0	0	0	0	0
125969	8	udp	private	SF	105	145	0	0	0	0
125970	0	tcp	smtp	SF	2231	384	0	0	0	0
125971	0	tcp	klogin	S0	0	0	0	0	0	0
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0

113270 rows × 42 columns

```
Probe_df['label']=Probe_df['label'].replace({1:2})
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  """Entry point for launching an IPython kernel.
```

```
Probe_df
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0
12	0	tcp	http	SF	287	2251	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
125965	0	tcp	smtp	SF	2233	365	0	0	0	0
125967	0	tcp	http	SF	359	375	0	0	0	0
125969	8	udp	private	SF	105	145	0	0	0	0
125970	0	tcp	smtp	SF	2231	384	0	0	0	0
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0

78999 rows × 42 columns

```
R2L_df['label']=R2L_df['label'].replace({1:3})
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  """Entry point for launching an IPython kernel.
```

R2L_df

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0
12	0	tcp	http	SF	287	2251	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
125965	0	tcp	smtp	SF	2233	365	0	0	0	0
125967	0	tcp	http	SF	359	375	0	0	0	0
125969	8	udp	private	SF	105	145	0	0	0	0
125970	0	tcp	smtp	SF	2231	384	0	0	0	0
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0

68338 rows × 42 columns

```
U2R_df['label']=U2R_df['label'].replace({1:4})
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  """Entry point for launching an IPython kernel.
```

U2R_df

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0
12	0	tcp	http	SF	287	2251	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
125965	0	tcp	smtp	SF	2233	365	0	0	0	0
125967	0	tcp	http	SF	359	375	0	0	0	0
125969	8	udp	private	SF	105	145	0	0	0	0
125970	0	tcp	smtp	SF	2231	384	0	0	0	0
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0

67395 rows × 42 columns

add four datasets into a newtrain dataset

```
frames = [DoS_df,Probe_df,R2L_df,U2R_df]

newdata_train = pd.concat(frames)


newdata_train
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
125965	0	tcp	smtp	SF	2233	365	0	0	0	0
125967	0	tcp	http	SF	359	375	0	0	0	0
125969	8	udp	private	SF	105	145	0	0	0	0
125970	0	tcp	smtp	SF	2231	384	0	0	0	0
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0

328002 rows × 42 columns

```
DoS_df_test =df_test[~df_test['label'].isin(to_drop_DoS)];
Probe_df_test = df_test[~df_test['label'].isin(to_drop_Probe)];
R2L_df_test = df_test[~df_test['label'].isin(to_drop_R2L)];
U2R_df_test = df_test[~df_test['label'].isin(to_drop_U2R)];
```

```
DoS_df_test
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	private	REJ	0	0	0	0	0	0
1	0	tcp	private	REJ	0	0	0	0	0	0
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0
5	0	tcp	http	SF	267	14515	0	0	0	0
6	0	tcp	smtp	SF	1022	387	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
22538	0	icmp	ecr_i	SF	1032	0	0	0	0	0
22539	0	tcp	smtp	SF	794	333	0	0	0	0
22540	0	tcp	http	SF	317	938	0	0	0	0
22541	0	tcp	http	SF	54540	8314	0	0	0	2
22542	0	udp	domain_u	SF	42	42	0	0	0	0

17171 rows × 42 columns

```
Probe_df_test['label']=Probe_df_test['label'].replace({1:2})
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  """Entry point for launching an IPython kernel.
```

```
R2L_df_test['label']=R2L_df_test['label'].replace({1:3})
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  """Entry point for launching an IPython kernel.
```

```
U2R_df_test['label']=U2R_df_test['label'].replace({1:4})
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  """Entry point for launching an IPython kernel.
```

add four datasets into a newtest dataset

```
frames1 = [DoS_df_test,Probe_df_test,R2L_df_test,U2R_df_test]

newdata_test = pd.concat(frames1)
```

```
newdata_test
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	0	tcp	private	REJ	0	0	0	0	0	0
1	0	tcp	private	REJ	0	0	0	0	0	0
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0
5	0	tcp	http	SF	267	14515	0	0	0	0
6	0	tcp	smtp	SF	1022	387	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
22533	0	tcp	http	SF	274	1623	0	0	0	0
22535	0	tcp	http	SF	280	6087	0	0	0	0
22539	0	tcp	smtp	SF	794	333	0	0	0	0
22540	0	tcp	http	SF	317	938	0	0	0	0
22542	0	udp	domain_u	SF	42	42	0	0	0	0

51677 rows × 42 columns

```
print('Train:')
print('Dimensions of DoS:', DoS_df.shape)
print('Dimensions of Probe:', Probe_df.shape)
print('Dimensions of R2L:', R2L_df.shape)
print('Dimensions of U2R:', U2R_df.shape)
print('Test:')
print('Dimensions of DoS:', DoS_df_test.shape)
print('Dimensions of Probe:', Probe_df_test.shape)
print('Dimensions of R2L:', R2L_df_test.shape)
print('Dimensions of U2R:', U2R_df_test.shape)
```

```
Train:
Dimensions of DoS: (113270, 42)
Dimensions of Probe: (78999, 42)
Dimensions of R2L: (68338, 42)
Dimensions of U2R: (67395, 42)
Test:
Dimensions of DoS: (17171, 42)
Dimensions of Probe: (12132, 42)
Dimensions of R2L: (12596, 42)
Dimensions of U2R: (9778, 42)
```

```
def plot_graph():

    edge_list = tangle.edges
    frm = []
    to = []

    for i in edge_list.keys():
        for j in edge_list[i]:
            frm.append(i)
            to.append(j)

    df = pd.DataFrame({ 'from':frm, 'to':to})

    # Build the tangle graph
    G=nx.from_pandas_edgelist(df, 'from', 'to', create_using=nx.DiGraph)

    j=25
    mapping = {}
    cols = []
```

```
    size = []

    for i in G:
        wt = tangle.transactions[i].cumulative_weight
        mapping[i]=(chr(j), wt)
        j=j+1
        size.append(1000+500*G.in_degree[i])
        if G.in_degree[i] > 0:
            cols.append('skyblue')
        else:
            cols.append('palegreen')

    nx.draw(G, labels = mapping, node_color = cols, node_size=size, pos=nx.fruchterman_reingold_layout(G))
```

pre-processing stage-, Z-score normalization

```
df1=newdata_train.iloc[:,4:41]
df1.head()
```

	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromis
0	491	0	0	0	0	0	0	0	
1	146	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	
3	232	8153	0	0	0	0	0	1	
4	199	420	0	0	0	0	0	1	

5 rows × 37 columns

```
df_z_scaled = df1.copy()

# apply normalization techniques
for column in df_z_scaled.columns:
    df_z_scaled[column] = (df_z_scaled[column] -
                           df_z_scaled[column].mean()) / df_z_scaled[column].std()

# view normalized data
display(df_z_scaled)
```

	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_i
0	-0.006871	-0.004117	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	-1.198
1	-0.006966	-0.004117	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	-1.198
2	-0.007006	-0.004117	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	-1.198
3	-0.006942	-0.000847	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	0.834
4	-0.006951	-0.003949	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	0.834
...	...	...	...	...	...	...	...	
125965	-0.006394	-0.003971	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	0.834
125967	-0.006907	-0.003967	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	0.834
125969	-0.006977	-0.004059	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	-1.198
125970	-0.006395	-0.003963	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	0.834
125972	-0.006964	-0.004117	-0.011843	-0.055321	-0.008285	-0.098083	-0.027562	0.834

328002 rows × 37 columns

Entropy–Kurtosis based feature selection

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
from sklearn.feature_selection import VarianceThreshold, mutual_info_classif, mutual_info_regression
from sklearn.feature_selection import SelectKBest, SelectPercentile

from scipy.stats import kurtosis
kurtosis = df1.kurtosis()
kurtosis
```

```
src_bytes                     100829.518289
dst_bytes                     236590.116288
land                            7125.587040
wrong_fragment                   348.598758
urgent                         21371.652709
hot                              157.054714
num_failed_logins               3111.415597
logged_in                         -1.866990
num_compromised                49446.366720
root_shell                       560.531024
su_attempted                    1199.208149
num_root                       45617.023996
num_file_creations              2434.719211
num_shells                      2775.940832
num_access_files                1891.633801
num_outbound_cmds                  0.000000
is_host_login                  81996.749968
is_guest_login                    81.193238
count                              6.759983
srv_count                         19.256346
serror_rate                        3.820765
srv_serror_rate                    3.925093
rerror_rate                        8.713452
srv_rerror_rate                    8.716970
same_srv_rate                      1.549532
diff_srv_rate                     24.896813
srv_diff_host_rate                 5.450007
dst_host_count                    -1.578989
dst_host_srv_count                -1.566601
dst_host_same_srv_rate            -1.133967
dst_host_diff_srv_rate            18.154284
dst_host_same_src_port_rate        4.209579
dst_host_srv_diff_host_rate       57.794679
dst_host_serror_rate               3.802806
dst_host_srv_serror_rate           4.274094
dst_host_rerror_rate               8.893079
dst_host_srv_rerror_rate           8.790789
dtype: float64
```

```
X = df1
y=newdata_train.iloc[:,-1]


X=X.dropna()
y=y.dropna()


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)


constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(X_train)
X_train_filter = constant_filter.transform(X_train)
X_test_filter = constant_filter.transform(X_test)
X_train_T = X_train_filter.T
X_test_T = X_test_filter.T
X_train_T = pd.DataFrame(X_train_T)
X_test_T = pd.DataFrame(X_test_T)


X_train_T.duplicated().sum()
```

```
0
```

```
duplicated_features = X_train_T.duplicated()


features_to_keep = [not index for index in duplicated_features]


X_train_unique = X_train_T[features_to_keep].T
X_test_unique = X_test_T[features_to_keep].T
mi = mutual_info_classif(X_train_unique, y_train)
print((mi.shape))
```

```
(28,)
```

```
Kurtosis= np.load('/content/sample_data/features.npy')
```

Recalling-Enhanced Recurrent Neural Network

```
from pandas import read_csv
import numpy as np
from keras import Model
from keras.layers import Layer
import keras.backend as K
import tensorflow as tf
from keras.layers import Input, Dense, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import Dropout
time_steps=37
class StateLayer(Layer):
    def __init__(self,**kwargs):
        super(StateLayer,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(shape=(input_shape[-1],1),
                               initializer='random_normal', trainable=True)
        self.b=self.add_weight(shape=(input_shape[1],1),
                               initializer='zeros', trainable=True)
        super(StateLayer, self).build(input_shape)

    def call_State(self,x):
      self.add_loss(tf.abs(tf.reduce_mean(x)))


class SumLayer(Layer):
    def __init__(self,**kwargs):
        super(SumLayer,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(shape=(input_shape[-1],1),
                               initializer='random_normal', trainable=True)
        self.b=self.add_weight(shape=(input_shape[1],1),
                               initializer='zeros', trainable=True)
        super(SumLayer, self).build(input_shape)

    def call(self,x):
        e = K.tanh(K.dot(x,self.W)+self.b)
        e = K.squeeze(e, axis=-1)
        a = K.softmax(e)
        a = K.expand_dims(a, axis=-1)
        context = x * a
        return K.sum(context, axis=1)

debug_flag = int(os.environ.get('KERAS_ATTENTION_DEBUG', 0))

class MemoryLayer(object if debug_flag else Layer):
    def __init__(self,**kwargs):
        super(MemoryLayer,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(shape=(input_shape[-1],1),
                               initializer='random_normal', trainable=True)
        self.b=self.add_weight(shape=(input_shape[1],1),
                               initializer='zeros', trainable=True)
        super(MemoryLayer, self).build(input_shape)

    def __call__(self, inputs, training=None, **kwargs):
        if debug_flag:
            return self.call(inputs, training, **kwargs)
        else:
            return super(MemoryLayer, self).__call__(inputs, training, **kwargs)



def ReRNN(input_shape):
    x=Input(shape=input_shape)
    rnn_layer = SimpleRNN(32, return_sequences=True, activation='relu')(x)
    state_layer = StateLayer()(rnn_layer)
```

```
    memory_layer = MemoryLayer()(state_layer)
    sum_layer = SumLayer()(memory_layer)
    outputs=Dense(4, trainable=True, activation='softmax')(sum_layer)
    model=Model(x,outputs)
    return model

model= ReRNN(input_shape=(time_steps,1))
model.summary()
Train=Kurtosis
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=10, batch_size=20)
y_pred=np.asarray(Train)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 37, 1)]           0

 simple_rnn (SimpleRNN)      (None, 37, 32)            1088

 state_layer (StateLayer)    (None, 37, 32)            69

 memory_layer (MemoryLayer)  (None, 37, 32)            69

 sum_layer (SumLayer)        (None, 32)                69

 dense (Dense)               (None, 4)                 132

=================================================================
Total params: 1,427
Trainable params: 1,427
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
WARNING:tensorflow:Gradients do not exist for variables ['state_layer/Variable:0', 'state_layer/Variable:0', 'memory_layer/Variable
WARNING:tensorflow:Gradients do not exist for variables ['state_layer/Variable:0', 'state_layer/Variable:0', 'memory_layer/Variable
13121/13121 [==============================] - 73s 6ms/step - loss: 0.2992
Epoch 2/10
13121/13121 [==============================] - 67s 5ms/step - loss: 0.2895
Epoch 3/10
13121/13121 [==============================] - 65s 5ms/step - loss: 0.2826
Epoch 4/10
13121/13121 [==============================] - 66s 5ms/step - loss: 0.2658
Epoch 5/10
13121/13121 [==============================] - 65s 5ms/step - loss: 0.2641
Epoch 6/10
13121/13121 [==============================] - 64s 5ms/step - loss: 0.2641
Epoch 7/10
13121/13121 [==============================] - 64s 5ms/step - loss: 0.2641
Epoch 8/10
13121/13121 [==============================] - 68s 5ms/step - loss: 0.2641
Epoch 9/10
13121/13121 [==============================] - 65s 5ms/step - loss: 0.2641
Epoch 10/10
13121/13121 [==============================] - 66s 5ms/step - loss: 0.2641
```

```
X_test.shape
from sklearn.metrics import confusion_matrix


cm = confusion_matrix(np.sort(y_test),(y_pred))
cm_df = pd.DataFrame(cm,
                     index = ['No_Attack','DoS','Probe','R2L','U2R'],
                     columns = ['No_Attack','DoS','Probe','R2L','U2R'])
cm_df
```

	No_Attack	DoS	Probe	R2L	U2R
No_Attack	53605	135	40	95	0
DoS	61	9105	12	8	0
Probe	12	19	2280	20	0
R2L	2	2	1	193	1
U2R	0	0	0	1	9

```
from sklearn.metrics import classification_report

print(classification_report(np.sort(y_test),(y_pred)))
```

```
                  precision    recall  f1-score   support
```

```
                0       1.00      0.99      1.00     53875
                1       0.98      0.99      0.99      9186
                2       0.98      0.98      0.98      2331
                3       0.61      0.97      0.75       199
                4       0.90      0.90      0.90        10

         accuracy                           0.99     65601
        macro avg       0.89      0.97      0.92     65601
     weighted avg       0.99      0.99      0.99     65601
```

class-0(No attack)

```
TP=53605;
TN=11587
FP=184
FN=75;
Accuracy=(TP+TN)/(TP+TN+FP+FN)
print("Accuracy for No attack class",Accuracy)
```

```
Accuracy for No attack class 0.9960428412094544
```

class-1(Dos)

```
TP=9105;
TN=56087
FP=81
FN=156;
Accuracy=(TP+TN)/(TP+TN+FP+FN)
print("Accuracy for DoS class",Accuracy)
```

```
Accuracy for DoS class 0.9963777529841508
```

class-2(Probe)

```
TP=2280
TN=62912
FP=51
FN=53;
Accuracy=(TP+TN)/(TP+TN+FP+FN)
print("Accuracy for PROBE class",Accuracy)
```

```
Accuracy for PROBE class 0.9984072531242343
```

class-3(R2L)

```
TP=193;
TN=64999
FP=6
FN=124;
Accuracy=(TP+TN)/(TP+TN+FP+FN)
print("Accuracy for R2L class",Accuracy)
```

```
Accuracy for R2L class 0.9980098588530664
```

CLASS-4(U2R)

```
TP=9;
TN=65183
FP=1
FN=1;
Accuracy=(TP+TN)/(TP+TN+FP+FN)
print("Accuracy for U2R class",Accuracy)
```

```
Accuracy for U2R class 0.9999693223302758
```

Fractional Discrete Meixner Moments encryption

```
import hashlib
import os
import random
from collections.abc import Callable
from functools import partial
from typing import Dict, List, Tuple

import torch
from PIL import Image
from torch import nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms as tvf
import torchvision.transforms.functional as TF


key = Fernet.generate_key()
with open('mykey.key', 'wb') as filekey:
   filekey.write(key)


def data(dir: str) -> List[str]:
    files = []
    for root, _, fnames in sorted(os.walk(dir)):
        for fname in fnames:
            if files(fname):
                path = os.path.join(root, fname)
                files.append(path)
    return files

def create_jigsaw_grid(
        img_tensor: torch.Tensor,
        grid_size: Tuple[int, int],
        patch_size: Tuple[int, int],
        crop_size: Tuple[int, int] = None,
        norm_patch: bool = False,
        permutation: torch.Tensor = None
)-> Tuple[torch.Tensor, torch.Tensor]:

    h, w = img_tensor.shape[-2:]
    # check that the grid size and patch size combination are valid
    assert patch_size[0] * grid_size[0] < h,\
        f"Final  grid size on axis 0 of {patch_size[0] * grid_size[0]} too large, given: \n" \
        f"grid_size {grid_size[0]} and patch_size {patch_size[0]} and input  with axis 0 size {h}"
    assert patch_size[1] * grid_size[1] < w,\
        f"Final  grid size on axis 1 of {patch_size[1] * grid_size[1]} too large, given: \n" \
        f"grid_size {grid_size[1]} and patch_size {patch_size[1]} and input with axis 1 size {w}"

    patches = []
    # sample an anchor point to compute the jigsaw grid
    x_anchor_idx = random.randint(0, w - patch_size[1] * grid_size[1])
    y_anchor_idx = random.randint(0, h - patch_size[0] * grid_size[0])

    # check if we have crop_size and initialize crop function
    if crop_size is not None:
        crop_fn = tvf.RandomCrop(size=crop_size)
    # otherwise use identity function
    else:
        crop_fn = lambda x: x


    for xn in range(grid_size[1]):
        for yn in range(grid_size[0]):
            x_idx = x_anchor_idx + xn * patch_size[1]
            y_idx = y_anchor_idx + yn * patch_size[0]
            # extract patch
            patch = img_tensor[..., y_idx:y_idx + patch_size[0], x_idx:x_idx + patch_size[1]]
            patch = crop_fn(patch)
            if norm_patch:
                patch = patch(patch)
            patches.append(patch)
    # create patch permutation
    if permutation is None:
        permutation = torch.randperm(len(patches))
    # permute the patches
    permuted_patches = [patches[i] for i in permutation]
    # stack patches across n_patches axis
    permuted_patches = torch.stack(permuted_patches, dim=0)
    return permuted_patches, permutation
```

```
# data blocks:
class JigsawDataset(Dataset):
    def __init__(
            self,
            files_root: str,
            grid_size: Tuple[int, int],
            patch_size: Tuple[int, int],
            crop_size: Tuple[int, int] = None,
            norm_patch: bool = False,
            preprocess_transform: Callable = tvf.ToTensor(),
    ):
        super(JigsawDataset, self).__init__()
        self.transform = preprocess_transform
        self.jigsaw_transform = partial(create_jigsaw_grid, grid_size=grid_size, patch_size=patch_size, crop_size=crop_size, norm_patch=r
        self.img_paths = data(files_root)
        self.size = len(self.img_paths)

    def __len__(self):
        return self.size
    def __getitem__(self, item) -> Tuple:
        sample = Image.open(self.img_paths[item])
        sample, permutation = self.jigsaw_transform(self.transform(sample), permutation=None)
        return sample, permutation
with open('mykey.key', 'rb') as filekey:
    key = filekey.read()
fernet = Fernet(key)
with open('/content/sample_data/KDDTrain+_2.csv', 'rb') as file:
    original = file.read()
class FixedJigsawDataset(Dataset):
    def __init__(
            self,
            files_root: str,
            grid_size: Tuple[int, int],
            patch_size: Tuple[int, int],
            permutations: List[torch.Tensor],
            crop_size: Tuple[int, int] = None,
            norm_patch: bool = False,
            preprocess_transform: Callable = tvf.ToTensor(),
    ):
        super(JigsawDataset, self).__init__()
        self.transform = preprocess_transform
        self.jigsaw_transform = partial(create_jigsaw_grid, grid_size=grid_size, patch_size=patch_size, crop_size=crop_size, norm_patch=r
        self.img_paths = data(files_root)
        self.size = len(self.img_paths)

        self.permutations = {i: permutations[i] for i in range(len(permutations))}

    def __len__(self):
        return self.size

    def __getitem__(self, item) -> Tuple:
        sample = Image.open(self.img_paths[item])
        puzzle_key, puzzle_value = random.choice(list(self.permutations.items()))
        sample, _ = self.jigsaw_transform(self.transform(sample), permutation=puzzle_value)
        return sample, puzzle_key
#Perform FrDMMs on each block
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
private_key = rsa.generate_private_key(
     public_exponent=65537,
     key_size=2048,
)
private_bytes = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.BestAvailableEncryption(b'mypassword')
)
print (private_bytes)
public_key = private_key.public_key()
public_bytes = private_key.public_key().public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
print (public_bytes)
encrypted = fernet.encrypt(original)

with open('encryteddata.csv', 'wb') as encrypted_file:
    encrypted_file.write(encrypted)
```

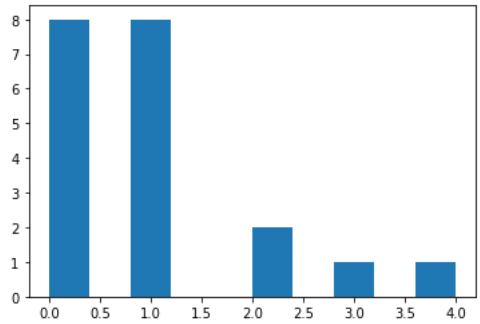
```
b'-----BEGIN RSA PRIVATE KEY-----\nProc-Type: 4,ENCRYPTED\nDEK-Info: AES-256-CBC,E7E47B810BCC0A68038C7D15CA2C8C85\n\nD1omqgaHnIEv+/
b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwtQNZrDmv9/LmFOjviH4\n6snItEc+SWN6UNoEqVsxARr2IX96VX5eif9
```

Acyclic Graph (DAG)-based blockchain

```
class transaction_block:

    def __init__(self, sender_public_key, recipient_public_key, data, signature, tx_id, approved_tx, nonce, previous_hashes):
        self.tx_id = tx_id
        self.own_weight = 1
        self.cumulative_weight = 0
        self.nonce = nonce
        self.previous_hashes = previous_hashes
        self.approved_tx = approved_tx,
        self.payload = ''
        self.timestamp = time.time(),
        self.signature = signature
        self.recipient_public_key = recipient_public_key
        self.sender_public_key = sender_public_key
        self.data = data

    def get_hash(self):
        transaction_dict = OrderedDict({
            'sender_public_key': self.sender_public_key,
            'recipient_public_key': self.recipient_public_key,
            'data': self.data
        })
        return hash_data(transaction_dict)

    def show(self):
        print("Transaction ID = ", self.tx_id)
        print("Cumulative Weight = ", self.cumulative_weight)
        print("Time Stamp = ", time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(self.timestamp[0])))
        print("Recipient Address = ", self.recipient_public_key)
        print("Sender Address = ", self.sender_public_key)
        print("data_row = ", self.data)
        print('\n')



class Tangle:

    def __init__(self):

        self.transactions = {GENESIS_ID:transaction_block(GENESIS_KEYS['public_key'], GENESIS_KEYS['public_key'], 328002,
                                sign(GENESIS_KEYS['private_key'], transaction_dict = OrderedDict({'sender_public_key': GENESIS_KEYS['publ
                                                                                                  'recipient_public_key': GENESIS_KEYS['
                                                                                                  'data_row': 328002,
                                                                                                   })),
                                GENESIS_ID, None, None, None)}

        self.edges = {GENESIS_ID:[]}
        self.reverse_edges = {GENESIS_ID:[]}


    def add_transaction(self, transaction: transaction_block):
        self.transactions[transaction.tx_id] = transaction
        self.add_edges(transaction)
        self.update_cumulative_weights(transaction)


    def add_edges(self, transaction: transaction_block):
        #Creating the forward (in time) edge dict
        approved = transaction.approved_tx[0]
        self.reverse_edges[transaction.tx_id] = []
        if transaction.tx_id not in self.edges:
            self.edges[transaction.tx_id] = approved

        if approved[0] not in self.reverse_edges:
            self.reverse_edges[approved[0]] = [transaction.tx_id]
        else:
            self.reverse_edges[approved[0]].append(transaction.tx_id)

        if approved[1] not in self.reverse_edges:
            self.reverse_edges[approved[1]] = [transaction.tx_id]
        else:
            self.reverse_edges[approved[1]].append(transaction.tx_id)


    def random_walk_weighted(self, current_node=GENESIS_ID):
        if len(self.reverse_edges[current_node]) == 0:
            return current_node
```

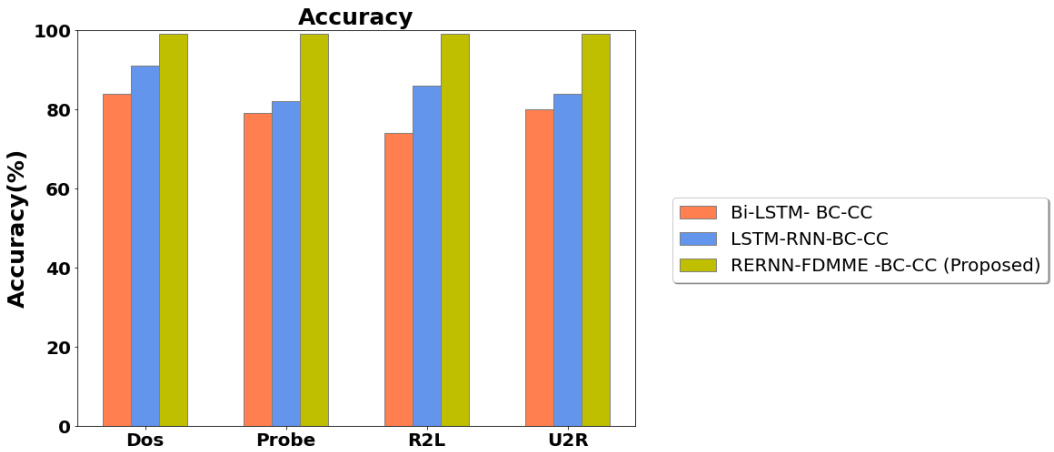
```
        elif len(self.reverse_edges[current_node]) < 3:
                option = np.random.choice(np.arange(0,2))
                if option==0:
                    return current_node

        prob = []
        for next_node in self.reverse_edges[current_node]:
            prob.append(self.transactions[next_node].cumulative_weight)

        prob = prob/np.sum(prob)

        choice = np.random.choice(np.arange(0,len(self.reverse_edges[current_node])), p=prob)
        return self.random_walk_weighted(self.reverse_edges[current_node][choice])


    def random_walk_unweighted(self, current_node=GENESIS_ID):
        if len(self.reverse_edges[current_node]) == 0:
            return current_node

        elif len(self.reverse_edges[current_node]) < 3:
                option = np.random.choice(np.arange(0,2))
                if option==0:
                    return current_node

        choice = np.random.choice(np.arange(0,len(self.reverse_edges[current_node])))
        return self.random_walk_weighted(self.reverse_edges[current_node][choice])


    def find_tips(self, algo='weighted_random_walk'):

        if algo=='recently_added':
            return list(random.sample(set(list(self.transactions.keys())[-2:]), 2))

        elif algo=='unweighted_random_walk':
            tips_list = []
            for n in range(2):
                tips_dict = {}
                for i in range(100):
                    tip = self.random_walk_unweighted()
                    if tip not in tips_dict:
                        tips_dict[tip] = 1
                    else:
                        tips_dict[tip] += 1

                max=0
                max_tip=''
                for i in tips_dict:
                    if tips_dict[i]>max:
                        max = tips_dict[i]
                        max_tip = i

                tips_list.append(max_tip)

            return tips_list


        elif algo=='weighted_random_walk':
            tips_list = []
            for n in range(2):
                tips_dict = {}
                for i in range(1):
                    tip = self.random_walk_weighted()
                    if tip not in tips_dict:
                        tips_dict[tip] = 1
                    else:
                        tips_dict[tip] += 1

                max=0
                max_tip=''
                for i in tips_dict:
                    if tips_dict[i]>max:
                        max = tips_dict[i]
                        max_tip = i

                tips_list.append(max_tip)

            return tips_list

        else:
            return list(random.sample(set(self.transactions.keys()), 2))
```

```
    def update_cumulative_weights(self, current_node):
        if current_node.tx_id is GENESIS_ID:
            current_node.cumulative_weight += 1

        else:
            current_node.cumulative_weight += 1
            a, b = current_node.approved_tx[0]
            self.update_cumulative_weights(self.transactions[a])
            self.update_cumulative_weights(self.transactions[b])



if __name__=="__main__":
    tangle = Tangle()
    generate_transactions(initial= True, size=5)
    generate_transactions(size=20, tip_selection_algo='weighted_random_walk')
    for i in tangle.transactions:
        tangle.transactions[i].show()


    plot_graph()
```




```
Transaction ID =  2e6dea810ffc4229b69532c49dc373fe
Cumulative Weight =  432
Time Stamp =  2022-07-13 08:19:42
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  328002


Transaction ID =  af00603f4f4e45f6ab57572699784599
Cumulative Weight =  144
Time Stamp =  2022-07-13 08:19:42
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  328002


Transaction ID =  0429b7bfaa064ff9950f2ffe8838533a
Cumulative Weight =  37
Time Stamp =  2022-07-13 08:19:43
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  328002


Transaction ID =  058a67136e3e4ca3919c234a1f5a14fb
Cumulative Weight =  28
Time Stamp =  2022-07-13 08:19:44
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  328002


Transaction ID =  a0622f873b2a4bf0842ea47161a3930a
Cumulative Weight =  1
Time Stamp =  2022-07-13 08:19:44
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  328002


Transaction ID =  153f094dd8f9456fb274211f266bb6fb
Cumulative Weight =  6
Time Stamp =  2022-07-13 08:19:45
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  328002


Transaction ID =  e103050e4ba04c6a94c893d3ad7d661a
Cumulative Weight =  26
Time Stamp =  2022-07-13 08:19:46
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  91


Transaction ID =  5da1c49a972d4762b9499cc8c7705043
Cumulative Weight =  33
Time Stamp =  2022-07-13 08:19:47
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  59


Transaction ID =  b718263eabd342368c581c2ec9a0f37e
Cumulative Weight =  4
Time Stamp =  2022-07-13 08:19:47
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
Sender Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b694
data_row =  89


Transaction ID =  419dff93d493464bb75dee6234d69856
Cumulative Weight =  1
Time Stamp =  2022-07-13 08:19:48
Recipient Address =  2d2d2d2d2d424547494e205055424c4943204b45592d2d2d2d2d0a4d494942496a414e42676b71686b
```

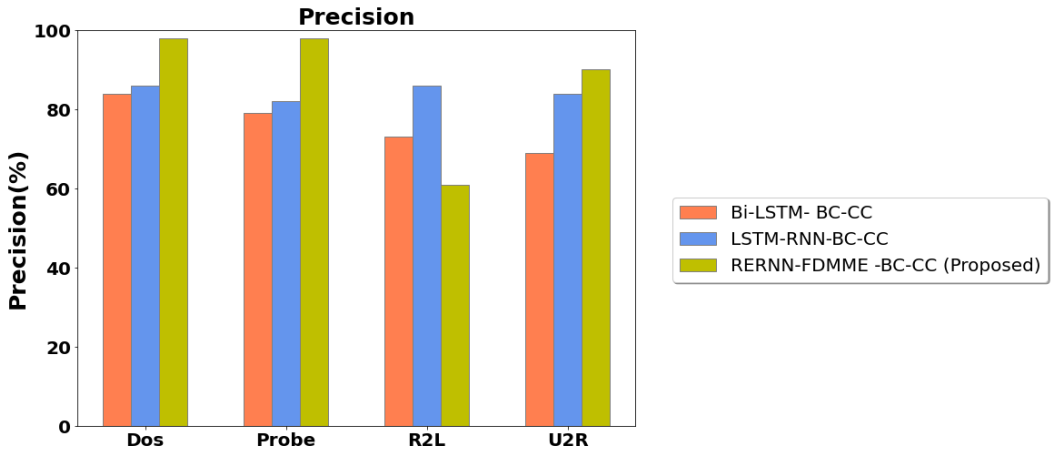
```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.2
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [84,79,74,80]
b= [91,82,86,84]
c= [99,99,99,99]



br1 = np.arange(len(a))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]



# Make the plot
plt.bar(br1,a, color ='coral', width = barWidth,
        edgecolor ='grey', label ='Bi-LSTM- BC-CC ')
plt.bar(br2,b, color ='cornflowerblue', width = barWidth,
        edgecolor ='grey', label ='LSTM-RNN-BC-CC  ')
plt.bar(br3, c, color ='y', width = barWidth,
        edgecolor ='grey', label ='RERNN-FDMME -BC-CC (Proposed)')


#plt.xlim(0,20)
plt.ylim(0,100)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Accuracy(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Dos','Probe','R2L','U2R'],fontsize = 20,fontweight="bold")
plt.title('Accuracy',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)

plt.show()
```



Double-click (or enter) to edit

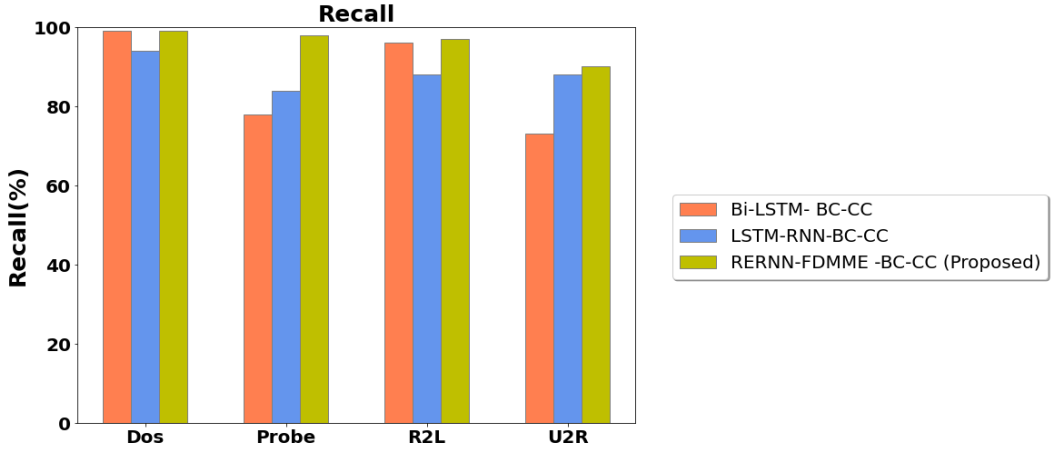
```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.2
```

```
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [84,79,73,69]
b= [86,82,86,84]
c= [98,98,61,90]



br1 = np.arange(len(a))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]



# Make the plot
plt.bar(br1,a, color ='coral', width = barWidth,
        edgecolor ='grey', label ='Bi-LSTM- BC-CC ')
plt.bar(br2,b, color ='cornflowerblue', width = barWidth,
        edgecolor ='grey', label ='LSTM-RNN-BC-CC  ')
plt.bar(br3, c, color ='y', width = barWidth,
        edgecolor ='grey', label ='RERNN-FDMME -BC-CC (Proposed)')


#plt.xlim(0,20)
plt.ylim(0,100)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Precision(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Dos','Probe','R2L','U2R'],fontsize = 20,fontweight="bold")
plt.title('Precision',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)

plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.2
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [99,78,96,73]
b= [94,84,88,88]
c= [99,98,97,90]
```

```
br1 = np.arange(len(a))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]



# Make the plot
plt.bar(br1,a, color ='coral', width = barWidth,
        edgecolor ='grey', label ='Bi-LSTM- BC-CC ')
plt.bar(br2,b, color ='cornflowerblue', width = barWidth,
        edgecolor ='grey', label ='LSTM-RNN-BC-CC  ')
plt.bar(br3, c, color ='y', width = barWidth,
        edgecolor ='grey', label ='RERNN-FDMME -BC-CC (Proposed)')


#plt.xlim(0,20)
plt.ylim(0,100)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Recall(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Dos','Probe','R2L','U2R'],fontsize = 20,fontweight="bold")
plt.title('Recall',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)

plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.2
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [84,78,73,71]
b= [91,85,86,89]
c= [99,98,75,90]



br1 = np.arange(len(a))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]



# Make the plot
plt.bar(br1,a, color ='coral', width = barWidth,
        edgecolor ='grey', label ='Bi-LSTM- BC-CC ')
plt.bar(br2,b, color ='cornflowerblue', width = barWidth,
```

```
        edgecolor ='grey', label ='LSTM-RNN-BC-CC  ')
plt.bar(br3, c, color ='y', width = barWidth,
        edgecolor ='grey', label ='RERNN-FDMME -BC-CC (Proposed)')


#plt.xlim(0,20)
plt.ylim(0,100)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('F-score(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Dos','Probe','R2L','U2R'],fontsize = 20,fontweight="bold")
plt.title('F-score',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)

plt.show()
```