Proposed method-Zero-Shot Learning based Hierarchical Graph Transformer optimized with Remora Optimization Algorithm for Spam E-mail Detection

```
#Import libs
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostCla
from collections import Counter
from keras.layers import Dense,LSTM,Embedding
from keras.models import Sequential,Model
import string
import warnings
warnings.filterwarnings('ignore')
import warnings
warnings.filterwarnings('ignore')
from tensorflow.keras import layers,models
from torch import Tensor, nn, tensor
import torch
from keras.models import Model, load_model
import numpy as np
from keras.models import Model
from keras.layers import Dense, Input, Conv1D, Flatten
import random
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

Ling spam dataset

```
df=pd.read_csv('/content/drive/MyDrive/srivithina/dataset1/messages.csv')
df.head()
```

	subject	message	lab
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	
1	NaN	lang classification grimes , joseph e . and ba...	
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	
3	risk	a colleague and i are researching the differin...	
4	request book information	earlier this morning i was on the phone with a...	

```
# converting all messages to lower case

df['message'] = df['message'].str.lower()
```

```
# check data once
df.head()
```

	subject	message	lab
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	
1	NaN	lang classification grimes , joseph e . and ba...	
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	
3	risk	a colleague and i are researching the differin...	
4	request book information	earlier this morning i was on the phone with a...	

```
# checing null values
df.isnull().sum()
```

```
     subject    62
     message     0
     label       0
     dtype: int64
```

From here we can observe that data is missing here

```
df.fillna(df['subject'].mode().values[0],inplace=True)
df.isnull().sum()
```

```
     subject    0
     message    0
```

```
    label      0
    dtype: int64
```

To get clarity about mail i'm going to merge both subject and message

```
df['sub_mssg']=df['subject']+df['message']
df.head()
```

	subject	message	label	
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	0	job postin
1	sociolinguistics	lang classification grimes , joseph e . and ba...	0	sociolinguis
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	0	query : lett
		a colleague and i are researching		riska colleague

```
df['sub_mssg'].describe()
```

```
    count                                                     2893
    unique                                                    2876
    top       re := 20 the virtual girlfriend and virtual bo...
    freq                                                         4
    Name: sub_mssg, dtype: object
```

```
df['length']=df['sub_mssg'].apply(len)
df.head()
```

	subject	message	label	
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	0	job posting - apple-i cent
1	sociolinguistics	lang classification grimes , joseph e . and ba...	0	sociolinguisticslang c
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	0	query : letter frequen
		a colleague and i are		riska colleagu

```
#now i'm going to drop un-necessary features
df.drop('subject',axis=1,inplace=True)
```

```
# check it once
df.head()
```

	message	label	sub_
0	content - length : 3386 apple-iss research cen...	0	job posting - apple-iss research centercon
1	lang classification grimes , joseph e . and ba...	0	sociolinguisticslang classification grime
2	i am posting this inquiry for sergei atamas (...	0	query : letter frequencies for text ident
3	a colleague and i are researching the differin...	0	riska colleague and i are researching the
4	earlier this morning i was on the phone with a...	0	request book informationearlier this morn

Preprocessing Email Messages

```
df['message'][0]
```

```
'content - length : 3386 apple-iss research center a us $ 10 million joint venture be
ter inc . and the institute of systems science of the national university of singapor
ngapore , is looking for : a senior speech scientist - - - - - - - - - - - - - - - - -
the successful candidate will have research expertise in computational linguistics ,
language processing and * * english * * and * * chinese * * statistical language mode
of state-of - the-art corpus-based n - gram language models , cache language models ,
eech language models are required . a text - to - speech project leader - - - - - - -
```

```
import re
```

```
def decontact(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
mssg=decontact(df['message'][70])
mssg
```

```
'hi , i am new to the list . and since english is not my first language , i apologize
you might find below : - ) . i hope that you will not mind if some of my explanations
for my ph . d . semiotics , i am writing a dissertation on discourse and science-fict
me information about the construction of neologisms in french and in english . ( neol
in sf , specially in the scientific field ! ) in french , neologisms are made in 5 di
hope these are the right english words ) : derivation ; composition ; imitation ; pur
lgam . one might also add borrowings from other languages . here are some examples (
```

```
#REPLACING NUMBERS
df['sub_mssg']=df['sub_mssg'].str.replace(r'\d+(\.\d+)?', 'numbers')
```

```
df['sub_mssg'][0]
```

```
'job posting - apple-iss research centercontent - length : numbers apple-iss researcl
mbers million joint venture between apple computer inc . and the institute of system
ational university of singapore , located in singapore , is looking for : a senior s
- - - - - - - - - - - - - - - - - - - - - - - - - - the successful candidate will have r
in computational linguistics , including natural language processing and * * english
se * * statistical language modeling . knowledge of state-of - the-art corpus-based 
models , cache language models , and part-of - speech language models are required .
```

```
#CONVRTING EVERYTHING TO LOWERCASE
df['sub_mssg']=df['sub_mssg'].str.lower()
#REPLACING NEXT LINES BY 'WHITE SPACE'
df['sub_mssg']=df['sub_mssg'].str.replace(r'\n'," ")
# REPLACING EMAIL IDs BY 'MAILID'
df['sub_mssg']=df['sub_mssg'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$','MailID')
# REPLACING URLs  BY 'Links'
df['sub_mssg']=df['sub_mssg'].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)
# REPLACING CURRENCY SIGNS BY 'MONEY'
df['sub_mssg']=df['sub_mssg'].str.replace(r'£|\$', 'Money')
# REPLACING LARGE WHITE SPACE BY SINGLE WHITE SPACE
df['sub_mssg']=df['sub_mssg'].str.replace(r'\s+', ' ')

# REPLACING LEADING AND TRAILING WHITE SPACE BY SINGLE WHITE SPACE
df['sub_mssg']=df['sub_mssg'].str.replace(r'^\s+|\s+?$', '')
#REPLACING CONTACT NUMBERS
df['sub_mssg']=df['sub_mssg'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$','co
#REPLACING SPECIAL CHARACTERS  BY WHITE SPACE
df['sub_mssg']=df['sub_mssg'].str.replace(r"[^a-zA-Z0-9]+", " ")
#CONVRTING EVERYTHING TO LOWERCASE
df['message']=df['message'].str.lower()
#REPLACING NEXT LINES BY 'WHITE SPACE'
df['message']=df['message'].str.replace(r'\n'," ")
# REPLACING EMAIL IDs BY 'MAILID'
df['message']=df['message'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$','MailID')
# REPLACING URLs  BY 'Links'
df['message']=df['message'].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$
# REPLACING CURRENCY SIGNS BY 'MONEY'
df['message']=df['message'].str.replace(r'£|\$', 'Money')
# REPLACING LARGE WHITE SPACE BY SINGLE WHITE SPACE
df['message']=df['message'].str.replace(r'\s+', ' ')

# REPLACING LEADING AND TRAILING WHITE SPACE BY SINGLE WHITE SPACE
df['message']=df['message'].str.replace(r'^\s+|\s+?$', '')
#REPLACING CONTACT NUMBERS
df['message']=df['message'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$','cont
#REPLACING SPECIAL CHARACTERS  BY WHITE SPACE
df['message']=df['message'].str.replace(r"[^a-zA-Z0-9]+", " ")
df['sub_mssg'][0]
```

```
'job posting apple iss research centercontent length numbers apple iss research cent
ers million joint venture between apple computer inc and the institute of systems sc
nal university of singapore located in singapore is looking for a senior speech scie
ul candidate will have research expertise in computational linguistics including nat
essing and english and chinese statistical language modeling knowledge of state of t
d n gram language models cache language models and part of speech language models ar
to speech project leader the successful candidate will have research expertise exper
```

```
df.head()
```

	message	label	sub
0	content length 3386 apple iss research center ...	0	job posting apple iss research centercont
1	lang classification grimes joseph e and barbar...	0	sociolinguisticslang classification grimes
2	i am posting this inquiry for sergei atamas sa...	0	query letter frequencies for text identif
3	a colleague and i are researching the differin...	0	riska colleague and i are researching the
4	earlier this morning i was on the phone with a...	0	request book informationearlier this morn

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
from tqdm import tqdm
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
# removing stopwords
stop = stopwords.words('english')
df['Cleaned_Text'] = df['sub_mssg'].apply(lambda x: ' '.join([word for word in x.split() i
```

```
df.head()
```

	message	label	sub_mssg	length	
0	content length 3386 apple iss research center ...	0	job posting apple iss research centercontent l...	2895	job posti
1	lang classification grimes joseph e and barbar...	0	sociolinguisticslang classification grimes jos...	1816	sociolinguis
2	i am posting this inquiry for sergei atamas sa...	0	query letter frequencies for text identificati...	1485	query l
[illegible]	a colleague and i are	[illegible]	riska colleague and i are	[illegible]	riska c

```
df.drop('message',axis=1,inplace=True)
df.drop('sub_mssg',axis=1,inplace=True)
df.head()
```

	label	length	Cleaned_Text

```
df.isnull().sum()
df['lgth_clean']=df['Cleaned_Text'].apply(len)
df.head()
```

	label	length	Cleaned_Text	lgth_clean
0	0	2895	job posting apple iss research centercontent l...	2108
1	0	1816	sociolinguisticslang classification grimes jos...	1506
2	0	1485	query letter frequencies text identificationi ...	1150
3	0	328	riska colleague researching differing degrees ...	216
4	0	1070	request book informationearlier morning phone ...	653

```
original_length=sum(df['length'])
after_cleaning=sum(df['lgth_clean'])
print("original_length",original_length)
print('after_cleaning',after_cleaning)
```

```
original_length 9437382
after_cleaning 6847902
```

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
#library that contains punctuation
import string
string.punctuation
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
#defining the function to remove punctuation
def remove_punctuation(text):
    punctuationfree="".join([i for i in text if i not in string.punctuation])
    return punctuationfree
#storing the puntuation free text
df['clean_msg']= df['Cleaned_Text'].apply(lambda x:remove_punctuation(x))
df.head()
```

	label	length	Cleaned_Text	lgth_clean	
0	0	2895	job posting apple iss research centercontent l...	2108	job posti

```
df['msg_lower']= df['clean_msg'].apply(lambda x: x.lower())
```

```
#defining function for tokenization
import re
def tokenization(text):
    tokens = re.split('W+',text)
    return tokens
#applying function to the column
df['msg_tokenied']= df['msg_lower'].apply(lambda x: tokenization(x))
```

```
#importing nlp library
import nltk
#Stop words present in the library
stopwords = nltk.corpus.stopwords.words('english')
stopwords[0:10]
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
#defining the function to remove stopwords from tokenized text
def remove_stopwords(text):
    output= [i for i in text if i not in stopwords]
    return output
#applying the function
df['no_stopwords']= df['msg_tokenied'].apply(lambda x:remove_stopwords(x))
```

```
import nltk
nltk.download('wordnet')
import nltk
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
True
```

```
from nltk.stem import WordNetLemmatizer
#defining the object for Lemmatization
wordnet_lemmatizer = WordNetLemmatizer()
#defining the function for lemmatization
def lemmatizer(text):
  lemm_text = [wordnet_lemmatizer.lemmatize(word) for word in text]
  return lemm_text
df['msg_lemmatized']=df['no_stopwords'].apply(lambda x:lemmatizer(x))
```

```
#importing the Stemming function from nltk library
from nltk.stem.porter import PorterStemmer
#defining the object for stemming
porter_stemmer = PorterStemmer()
#defining a function for stemming
def stemming(text):
  stem_text = [porter_stemmer.stem(word) for word in text]
  return stem_text

df['msg_stemmed']=df['no_stopwords'].apply(lambda x: stemming(x))
```

```
df
```

	label	length	Cleaned_Text	lgth_clean	clean_msg	msg_lower
0	0	2895	job posting apple iss research centercontent l...	2108	job posting apple iss research centercontent l...	job posting apple iss research centercontent l...
1	0	1816	sociolinguisticslang classification grimes jos...	1506	sociolinguisticslang classification grimes jos...	sociolinguisticslang classification grimes jos...
2	0	1485	query letter frequencies text identificationi ...	1150	query letter frequencies text identificationi ...	query letter frequencies text identificationi ...
3	0	328	riska colleague researching differing degrees ...	216	riska colleague researching differing degrees ...	riska colleague researching differing degrees ...
4	0	1070	request book informationearlier morning phone ...	653	request book informationearlier morning phone ...	request book informationearlier morning phone ...
...	...	...	...	...	...	...
2888	1	290	love profile ysuolvpvhello thanks stopping tak...	153	love profile ysuolvpvhello thanks stopping tak...	love profile ysuolvpvhello thanks stopping tak...
2889	1	2197	asked join kiddinthe list owner kiddin invited...	1246	asked join kiddinthe list owner kiddin invited...	asked join kiddinthe list owner kiddin invited...
2890	0	1073	anglicization composers namesjudging return po...	672	anglicization composers namesjudging return po...	anglicization composers namesjudging return po...
2891	0	3003	numbers numbers comparative method n ary compa...	1986	numbers numbers comparative method n ary compa...	numbers numbers comparative method n ary compa...
			american english		american english	american english

Feature extraction, Term frequency- inverse document frequency (TF-IDF)

```
from sklearn.feature_extraction.text import TfidfVectorizer
df1=df[['label', 'Cleaned_Text']]
# def text_to_graph(text):
#     import networkx as nx
#     from sklearn.neighbors import kneighbors_graph
#     vectorizer = TfidfVectorizer()
#     vectors = vectorizer.fit_transform(text)
#     return vectors
#Feature_data=text_to_graph(df1)
tf_vec = TfidfVectorizer()
features = tf_vec.fit_transform(df1['Cleaned_Text'])
X = features
y = df['label']
```

```
print(X.shape)
```

```
     (2893, 56934)
```

```
print(y.shape)
```

```
     (2893,)
```

```
X_train, X_test, Y_train,Y_test  = train_test_split(X, y, test_size = 0.1, random_state =
```

```
print(type(Y_test))
```

```
     <class 'pandas.core.series.Series'>
```

```
print(type(X_train))
QW=X_train.todense()
AA = np.squeeze(np.asarray(QW))
print(type(AA))
print(AA.shape)
```

```
     <class 'scipy.sparse.csr.csr_matrix'>
     <class 'numpy.ndarray'>
     (2603, 56934)
```

```
X1_train=AA.reshape(2603,2,28467)
```

```
print(type(Y_train))
Y1_train=Y_train.to_numpy()
print(type(Y1_train))
print(Y1_train)
```

```
     <class 'pandas.core.series.Series'>
     <class 'numpy.ndarray'>
     [1 0 1 ... 0 0 0]
```

```
print(type(Y_test))
Y1_test=Y_test.to_numpy()
print(type(Y1_test))
print(Y1_test.shape)
```

```
     <class 'pandas.core.series.Series'>
     <class 'numpy.ndarray'>
     (290,)
```

```
print(type(X_test))
print(X_test.shape)
QW=X_test.todense()
asd = np.squeeze(np.asarray(QW))
X1_test=asd.reshape(290,2,28467)
```

```
     <class 'scipy.sparse.csr.csr_matrix'>
     (290, 56934)
```

Hierarchical Graph Transformer

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import datetime
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler
import os
import re
import keras
from sklearn.model_selection import KFold, RepeatedKFold
from sklearn.isotonic import IsotonicRegression
tqdm.pandas()
from numba import jit
```

```
import random as rn
import tensorflow as tf
import numpy as np
from keras import backend as K
```

```
def init_seeds(seed):
    np.random.seed(seed)
    rn.seed(seed)
    tf.set_random_seed(seed)
    sess = tf.Session(graph=tf.get_default_graph())
    K.set_session(sess)
    return sess

from keras import backend as K

class LayerNormalization(keras.layers.Layer):

    def __init__(self,
                 center=True,
                 scale=True,
                 epsilon=None,
                 gamma_initializer='ones',
                 beta_initializer='zeros',
                 gamma_regularizer=None,
                 beta_regularizer=None,
                 gamma_constraint=None,
                 beta_constraint=None,
                 **kwargs):

        super(LayerNormalization, self).__init__(**kwargs)
        self.supports_masking = True
        self.center = center
        self.scale = scale
        if epsilon is None:
            epsilon = K.epsilon() * K.epsilon()
        self.epsilon = epsilon
        self.gamma_initializer = keras.initializers.get(gamma_initializer)
        self.beta_initializer = keras.initializers.get(beta_initializer)
        self.gamma_regularizer = keras.regularizers.get(gamma_regularizer)
        self.beta_regularizer = keras.regularizers.get(beta_regularizer)
        self.gamma_constraint = keras.constraints.get(gamma_constraint)
        self.beta_constraint = keras.constraints.get(beta_constraint)
        self.gamma, self.beta = None, None

    def get_config(self):
        config = {
            'center': self.center,
            'scale': self.scale,
            'epsilon': self.epsilon,
            'gamma_initializer': keras.initializers.serialize(self.gamma_initializer),
            'beta_initializer': keras.initializers.serialize(self.beta_initializer),
            'gamma_regularizer': keras.regularizers.serialize(self.gamma_regularizer),
            'beta_regularizer': keras.regularizers.serialize(self.beta_regularizer),
            'gamma_constraint': keras.constraints.serialize(self.gamma_constraint),
            'beta_constraint': keras.constraints.serialize(self.beta_constraint),
        }
        base_config = super(LayerNormalization, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))

    def compute_output_shape(self, input_shape):
        return input_shape

    def compute_mask(self, inputs, input_mask=None):
        return input_mask

    def build(self, input_shape):
        shape = input_shape[-1:]
        if self.scale:
            self.gamma = self.add_weight(
                shape=shape,
                initializer=self.gamma_initializer,
                regularizer=self.gamma_regularizer,
                constraint=self.gamma_constraint,
                name='gamma',
            )
        if self.center:
            self.beta = self.add_weight(
                shape=shape,
                initializer=self.beta_initializer,
                regularizer=self.beta_regularizer,
                constraint=self.beta_constraint,
                name='beta',
            )
        super(LayerNormalization, self).build(input_shape)

    def call(self, inputs, training=None):
        mean = K.mean(inputs, axis=-1, keepdims=True)
        variance = K.mean(K.square(inputs - mean), axis=-1, keepdims=True)
        std = K.sqrt(variance + self.epsilon)
        outputs = (inputs - mean) / std
        if self.scale:
            outputs *= self.gamma
        if self.center:
            outputs += self.beta
        return outputs


y_mean = np.median(Y_train)
def crps(y_true, y_pred):
    stops = np.arange(-99, 100)
    unit_steps = stops >= y_true.reshape(-1, 1)
    crps = np.mean((y_pred - unit_steps)**2)
    return crps


def nondecreasing(x):
    X_ir = np.arange(199).astype('float64')
    ir = IsotonicRegression(0, 1)
    x = ir.fit_transform(X_ir, x.astype('float64'))
    return x


from keras.models import Model, load_model
from keras.layers import Input, BatchNormalization, Activation, Add, Multiply, Dot
from keras.layers import Embedding, Permute, Reshape
from keras.layers.core import Dropout, Lambda, Dense, Flatten
from keras.layers.convolutional import Conv1D, Conv2D
from keras.layers.pooling import GlobalMaxPooling1D, GlobalAveragePooling1D
from keras.layers.merge import Concatenate
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam, SGD, Nadam
from keras import backend as K
from tensorflow.keras.layers import Layer
import tensorflow as tf



class ScaleLayer(Layer):

    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(ScaleLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        super(ScaleLayer, self).build(input_shape)

    def call(self, x):
        xx = K.arange(-99, 100, dtype=tf.float32)
        mu = y_mean + tf.reshape(x[:, 0], (-1, 1))
        sigma_minus = tf.identity(K.exp(0.5 * tf.reshape(x[:, 1], (-1, 1))), name="sigma")
        sigma_plus = tf.identity(K.exp(0.5 * tf.reshape(x[:, 2], (-1, 1))), name="sigma")
        xx = tf.subtract(xx, mu)
        pcf = tf.where(xx >= 0, tf.divide (xx, sigma_plus),  tf.divide (xx, sigma_minus))
        return pcf

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)


def dist_mult(dist, content):
    res = Lambda(lambda c: K.batch_dot(c[0], c[1]))([dist, content])
    return res
pp1 = [0 for _ in range(8)] + [1 for _ in range(22)]
def dist_attention(dist, content, dropout):
    if 1:
        dist1 = Reshape((18, 18, 1))(dist)
        dist1 = Conv2D(16, 1, activation='relu',
                  kernel_initializer='glorot_uniform',
                  bias_initializer='glorot_uniform',
                 )(dist1)
        dist1 = Conv2D(1, 1, activation='relu',
                  kernel_initializer='glorot_uniform',
                  bias_initializer='glorot_uniform',
                 )(dist1)
        dist1 = Reshape((18, 18))(dist1)
        dist = Add()([dist, dist1])
    dist = LayerNormalization()(dist)
    att = dist_mult(dist, content,)
    x_msg = Add()([content, att])
    x_msg = LayerNormalization()(x_msg)
    if dropout > 0:
        x_msg = Dropout(dropout)(x_msg)
    return x_msg
```

```
def attention(x_inner, x_outer, n_factor, dropout):
    x_Q =  Conv1D(n_factor, 1, activation='linear',
                  kernel_initializer='glorot_uniform',
                  bias_initializer='glorot_uniform',
                 )(x_inner)
    x_K =  Conv1D(n_factor, 1, activation='linear',
                  kernel_initializer='glorot_uniform',
                  bias_initializer='glorot_uniform',
                 )(x_outer)
    x_V =  Conv1D(n_factor, 1, activation='linear',
                  kernel_initializer='glorot_uniform',
                  bias_initializer='glorot_uniform',
                 )(x_outer)
    x_KT = Permute((2, 1))(x_K)
    res = Lambda(lambda c: K.batch_dot(c[0], c[1]) / np.sqrt(n_factor))([x_Q, x_KT])
    att = Lambda(lambda c: K.softmax(c, axis=-1))(res)
    att = Lambda(lambda c: K.batch_dot(c[0], c[1]))([att, x_V])
    return att

def multi_head_self_attention(x, n_factor, n_head, dropout):
    if n_head == 1:
        att = attention(x, x, n_factor, dropout)
    else:
        n_factor_head = n_factor // n_head
        heads = [attention(x, x, n_factor_head, dropout) for i in range(n_head)]
        att = Concatenate()(heads)
        att = Dense(n_factor,
                      kernel_initializer='glorot_uniform',
                      bias_initializer='glorot_uniform',
                     )(att)
    x = Add()([x, att])
    x = LayerNormalization()(x)
    if dropout > 0:
        x = Dropout(dropout)(x)
    return x
pn1 = [1 for _ in range(12)] + [0 for _ in range(248)]
def multi_head_outer_attention(x_inner, x_outer, n_factor, n_head, dropout):
    if n_head == 1:
        att = attention(x_inner, x_outer, n_factor, dropout)
    else:
        n_factor_head = n_factor // n_head
        heads = [attention(x_inner, x_outer, n_factor_head, dropout) for i in range(n_head
        att = Concatenate()(heads)
        att = Dense(n_factor,
                      kernel_initializer='glorot_uniform',
                      bias_initializer='glorot_uniform',
                     )(att)
    x_inner = Add()([x_inner, att])
    x_inner = LayerNormalization()(x_inner)
    if dropout > 0:
        x = Dropout(dropout)(x_inner)
    return x
```

```
class AddNorm(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.ln = nn.LayerNorm(d_model)

    def forward(self, x1, x2):
        return self.ln(x1+x2)


class FeedForward(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.l1 = nn.Linear(d_model, d_model)
        self.relu = nn.ReLU()
        self.l2 = nn.Linear(d_model, d_model)
    def forward(self, x):
        return self.l2(self.relu(self.l1(x)))
class AttentionAggregation(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.query = nn.Linear(d_model, 1, bias=False)

    def forward(self, x):  # (b, s, m)
        attns = self.query(x).softmax(dim=1)  # (b, s, 1)
        enc = torch.bmm(attns.transpose(1, 2), x)  # (b, 1, m)
        return enc.squeeze(1)


class word_level_trans(nn.Module):
    def __init__(self, d_model, num_heads):
        super().__init__()
        self.mha = multi_head_self_attention(d_model=d_model, num_heads=num_heads, masked=
        self.an1 = AddNorm(d_model)
        self.ff = FeedForward(d_model)
        self.an2 = AddNorm(d_model)

    def forward(self, x):
        x = self.an1(x, self.mha(q=x, k=x, v=x))
        return self.an2(x, self.ff(x))
```

```
class Sentense_level(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.lin = nn.Linear(d_model, d_model)
        self.tanh = nn.Tanh()

    def forward(self, x):
        return self.tanh(self.lin(x))
class graph_level(nn.Module):
    def __init__(self, d_model, n_feats, n_out):
        super().__init__()
        self.lin = nn.Linear(d_model + n_feats, n_out, bias=False)  # TODO what if True?

    def forward(self, x, feats):
        return self.lin(torch.cat([x, feats], dim=1))


def se_bloc(in_bloc, ch, ratio):
    x = GlobalAveragePooling1D()(in_bloc)
    x = Dense(ch//ratio, activation='relu')(x)
    x = Dense(ch, activation='sigmoid')(x)
    x = Multiply()([in_bloc, x])
    return Add()([x, in_bloc])

def conv_bloc(content, n_factor, n_hidden, se_ratio, dropout):
    content0 = content
    content = Conv1D(n_hidden, 1, activation='relu',
                  kernel_initializer='glorot_uniform',
                  bias_initializer='glorot_uniform',
                 )(content)
    content = Conv1D(n_factor, 1, activation='relu',
                  kernel_initializer='glorot_uniform',
                  bias_initializer='glorot_uniform',
                 )(content)
    content = Add()([content0, content])
    content = se_bloc(content, n_factor, se_ratio)
    content = LayerNormalization()(content)
    if dropout > 0:
        content = Dropout(dropout)(content)
    return content


def get_model(n_msgr, n_factor, n_loop, n_head, n_hidden, se_ratio, dropout, n_msg_cols, n
    input_content = Input((18, 18), name="Cleaned_Text")
    input_dmats = Input((18, 18), name="label")
    input_play = Input((n_play_cols,), name="length")
    inputs = Input(shape=(2,1,28467))
    x = Conv1D(64, 2, padding='same', activation='elu')(inputs)
    x = Conv1D(128, 2, padding='same', activation='elu')(x)
    x_msg = input_content
    x_msg = Conv1D(n_factor, 1)(x_msg)
    x_msg = LayerNormalization()(x_msg)
    x = Flatten()(x)
    x = Dense(128, activation='elu')(x)
    x = Dense(64, activation='elu')(x)
    x = Dense(32, activation='elu')(x)
    for l in range(n_loop):
        x_msg = dist_attention(input_dmats, x_msg, dropout)
        x_msg = conv_bloc(x_msg, n_factor, n_hidden, se_ratio, dropout)

        x_msg = multi_head_self_attention(x_msg, n_factor, n_head, dropout)
        x_msg = conv_bloc(x_msg, n_factor, n_hidden, se_ratio, dropout)

    x_play = Dense(n_factor)(input_play)
    x_play = Reshape((1, -1))(x_play)
    x = Dense(1, activation='linear')(x)
    model = Model(inputs=[inputs], outputs=[x])
    readout = multi_head_outer_attention(x_play, x_msg, n_factor, n_head, dropout)
    readout = Flatten()(readout)
    out1 = Dense(199, activation='sigmoid')(readout)
    readout = Dense(4)(readout)
    readout = ScaleLayer(output_dim=199)(readout)
    out2 = keras.layers.Activation('sigmoid')(readout)
    model.compile(loss='mean_squared_error', optimizer='adamax', metrics=['mae'])
    return model,Model(inputs=[input_content, input_dmats, input_play], outputs=[out1, out

msg_cols=df1['Cleaned_Text']
play_cols=df1['label']

msg_cols=np.array(msg_cols)
msg_cols.shape[0]
B=Y_train.to_numpy()
dddd=X_train.todense()
A = np.squeeze(np.asarray(dddd))
tdata=A.reshape(2603,18,3163,1)
X_train=tdata
Y_train=B


print(Y_train.shape)
print(X_train.shape)
```

```
(2603,)
(2603, 18, 3163, 1)
```

```
n_msg = 18
n_factor = 64
se_ratio = 4
n_loop = 1
n_head = 4
n_hidden = 2*n_factor
dropout = 0.25
an1 = [1 for _ in range(20)] + [0 for _ in range(240)]
n_msg_cols = len(msg_cols)
n_play_cols = len(play_cols)
m,model=get_model(n_msg, n_factor, n_loop, n_head, n_hidden, se_ratio, dropout, n_msg_cols
model.summary()
```

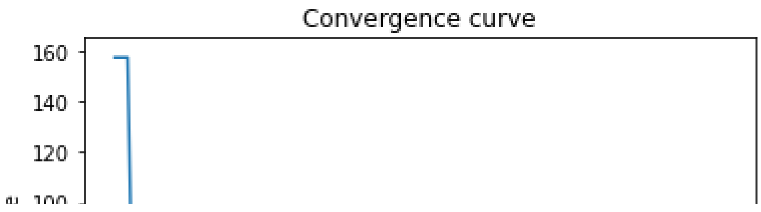
```
 lambda_117 (Lambda)            (None, 1, 18)        0           ['lambda_116[0][0

 conv1d_148 (Conv1D)            (None, 18, 16)       1040        ['dropout_23[0][0

 lambda_120 (Lambda)            (None, 1, 18)        0           ['lambda_119[0][0

 conv1d_151 (Conv1D)            (None, 18, 16)       1040        ['dropout_23[0][0

 lambda_123 (Lambda)            (None, 1, 18)        0           ['lambda_122[0][0

 conv1d_154 (Conv1D)            (None, 18, 16)       1040        ['dropout_23[0][0

 lambda_115 (Lambda)            (None, 1, 16)        0           ['lambda_114[0][0
                                                                  'conv1d_145[0][0

 lambda_118 (Lambda)            (None, 1, 16)        0           ['lambda_117[0][0
                                                                  'conv1d_148[0][0

 lambda_121 (Lambda)            (None, 1, 16)        0           ['lambda_120[0][0
                                                                  'conv1d_151[0][0

 lambda_124 (Lambda)            (None, 1, 16)        0           ['lambda_123[0][0
                                                                  'conv1d_154[0][0

 concatenate_9 (Concatenate)    (None, 1, 64)        0           ['lambda_115[0][0
                                                                  'lambda_118[0][0
                                                                  'lambda_121[0][0
                                                                  'lambda_124[0][0

 dense_62 (Dense)               (None, 1, 64)        4160        ['concatenate_9[0

 add_39 (Add)                   (None, 1, 64)        0           ['reshape_14[0][0
                                                                  'dense_62[0][0]'

 layer_normalization_34 (LayerN  (None, 1, 64)       128         ['add_39[0][0]']
 ormalization)

 dropout_24 (Dropout)           (None, 1, 64)        0           ['layer_normaliza

 flatten_9 (Flatten)            (None, 64)           0           ['dropout_24[0][0

 dense_64 (Dense)               (None, 4)            260         ['flatten_9[0][0]

 scale_layer_4 (ScaleLayer)     (None, 199)          0           ['dense_64[0][0]'

 dense_63 (Dense)               (None, 199)          12935       ['flatten_9[0][0]

 activation_4 (Activation)      (None, 199)          0           ['scale_layer_4[0

==================================================================================
Total params: 271,168
Trainable params: 271,168
Non-trainable params: 0
```

Remora Optimization Algorithm

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


def init(SearchAgents,dimension,upperbound,lowerbound):
  Pos=np.zeros((SearchAgents,dimension))
  for i in range(SearchAgents):
    for j in range(dimension):
      Pos[i,j]=random.random()*(upperbound-lowerbound)+lowerbound;
  return Pos


def fitness_function(x):
  dimension=x.shape[0]
  R=0
  for i in range(dimension):
    R=R+np.sum(x[i]**2);
  return R


def ROA(Search_Agents,Max_iterations,Lowerbound,Upperbound,dimensions):
  BestRemora=np.zeros((1,dimensions))
  Score=math.inf
  Remora=init(Search_Agents,dimensions,Upperbound,Lowerbound); # Generate initial remora p
  Convergence=[]
  t=0
  while t<Max_iterations:
    # Memory of previous generation

    if t<=1:
        PreviousRemora = Remora;
    else:
        PreviousRemora = Remora-1;
    # Boundary check

    for i in range(Remora.shape[0]):
        Flag4Upperbound=Remora[i,:]>Upperbound
        Flag4Lowerbound=Remora[i,:]<Lowerbound
        Remora[i,:]=(Remora[i,:]*(~(Flag4Upperbound+Flag4Lowerbound)))+Upperbound*Flag4Upp
        fitness=fitness_function(Remora[i,:]);
        #  Evaluate fitness function of search agents

        if fitness<Score:
                Score=fitness
                BestRemora=Remora[i,:]
    # Make a experience attempt through equation (2)

    for j in range(Remora.shape[0]):
      RemoraAtt = Remora[j,:]+(Remora[j,:]-PreviousRemora[j,:])*random.random()

      #  Calculate the fitness function value of the attempted solution (fitnessAtt)
      fitnessAtt=fitness_function(RemoraAtt);
      # % Calculate the fitness function value of the current solution (fitnessI)
      fitnessI=fitness_function(Remora[j,:])

      #  Check if the current fitness (fitnessI) is better than the attempted fitness(fitn
      #  if No, Perform host feeding by equation (9)
      if fitnessI>fitnessAtt:
        V = 2*(1-t/Max_iterations)
        B = 2*V*random.random()-V
        C = 0.1
        A = B*(Remora[j,:]-C*BestRemora)
        Remora[j,:]= Remora[j,:]+A

      # If yes perform host conversion using equation (1) and (5)
      elif random.randint(0, 1)==0:
        a=-(1+t/Max_iterations);
        alpha = random.random()*(a-1)+1;
        D = abs(BestRemora-Remora[j,:]);
        Remora[j,:] = D*math.exp(alpha)*math.cos(2*math.pi*a)+Remora[j,:];
      else:
        m=np.random.permutation(Remora.shape[0]);
        # print(Remora[2,:])
        Remora[j,:]=BestRemora-((random.random()*(BestRemora+Remora[m[1],:])/2)-Remora[m[1
    Convergence.append(Score)
    t=t+1
    print('Iteration - ',str(t),': Best Position',str(BestRemora),': Best Fitness',str("%.
  return Score,BestRemora,Convergence
```

```
SearchAgents=100
Max_iterations=100
lowerbound=.4
upperbound=10
n_msg = 18
n_factor = 64
se_ratio = 2
n_loop = 1
n_head = 4
n_hidden = 2*n_factor
dropout = 0.25
n_msg_cols = 3163
n_play_cols = 3163
dimension=5
ap1 = [0 for _ in range(10)] + [1 for _ in range(20)]
[Best_score,Best_pos,ROA_curve]=ROA(SearchAgents,Max_iterations,lowerbound,upperbound,dime
print("\nBest solution found:\n")
print('Best fitness :',Best_score)
print('Best position :',Best_pos)
plt.plot(ROA_curve)
plt.xlabel('Iteration')
plt.ylabel('fitness value')
plt.title('Convergence curve')
plt.show()
```

```
Iteration -  52 : Best Position [11.3276301 11.3276301 11.3276301 11.3276301 11.32763
Iteration -  53 : Best Position [12.11978871 12.11978871 12.11978871 12.11978871 12.1
Iteration -  54 : Best Position [4.29387718 4.29387718 4.29387718 4.29387718 4.293877
Iteration -  55 : Best Position [4.29387718 4.29387718 4.29387718 4.29387718 4.293877
Iteration -  56 : Best Position [4.29387718 4.29387718 4.29387718 4.29387718 4.293877
Iteration -  57 : Best Position [7.76106745 7.76106745 7.76106745 7.76106745 7.761067
Iteration -  58 : Best Position [15.85272449 15.85272449 15.85272449 15.85272449 15.8
Iteration -  59 : Best Position [11.01467216 11.01467216 11.01467216 11.01467216 11.0
Iteration -  60 : Best Position [7.43991934 7.43991934 7.43991934 7.43991934 7.439919
Iteration -  61 : Best Position [7.43991934 7.43991934 7.43991934 7.43991934 7.439919
Iteration -  62 : Best Position [7.43991934 7.43991934 7.43991934 7.43991934 7.439919
Iteration -  63 : Best Position [12.04048329 12.04048329 12.04048329 12.04048329 12.0
Iteration -  64 : Best Position [12.34244275 12.34244275 12.34244275 12.34244275 12.3
Iteration -  65 : Best Position [9.33646256 9.33646256 9.33646256 9.33646256 9.336462
Iteration -  66 : Best Position [15.06537115 15.06537115 15.06537115 15.06537115 15.0
Iteration -  67 : Best Position [13.03597575 13.03597575 13.03597575 13.03597575 13.0
Iteration -  68 : Best Position [6.35981584 6.35981584 6.35981584 6.35981584 6.359815
Iteration -  69 : Best Position [3.29222653 3.29222653 3.29222653 3.29222653 3.292226
Iteration -  70 : Best Position [5.24377789 5.24377789 5.24377789 5.24377789 5.243777
Iteration -  71 : Best Position [6.81389171 6.81389171 6.81389171 6.81389171 6.813891
Iteration -  72 : Best Position [6.81389171 6.81389171 6.81389171 6.81389171 6.813891
Iteration -  73 : Best Position [6.81389171 6.81389171 6.81389171 6.81389171 6.813891
Iteration -  74 : Best Position [3.95896515 3.95896515 3.95896515 3.95896515 3.958965
Iteration -  75 : Best Position [4.90089629 4.90089629 4.90089629 4.90089629 4.900896
Iteration -  76 : Best Position [4.90089629 4.90089629 4.90089629 4.90089629 4.900896
Iteration -  77 : Best Position [4.90089629 4.90089629 4.90089629 4.90089629 4.900896
Iteration -  78 : Best Position [6.51583914 6.51583914 6.51583914 6.51583914 6.515839
Iteration -  79 : Best Position [6.51583914 6.51583914 6.51583914 6.51583914 6.515839
Iteration -  80 : Best Position [10.90419707 10.90419707 10.90419707 10.90419707 10.9
Iteration -  81 : Best Position [18.83803872 18.83803872 18.83803872 18.83803872 18.8
Iteration -  82 : Best Position [12.60748319 12.60748319 12.60748319 12.60748319 12.6
Iteration -  83 : Best Position [10.35462132 10.35462132 10.35462132 10.35462132 10.3
Iteration -  84 : Best Position [10. 10. 10. 10. 10.] : Best Fitness 0.80
Iteration -  85 : Best Position [9.45956012 9.45956012 9.45956012 9.45956012 9.459560
Iteration -  86 : Best Position [18.95723424 18.95723424 18.95723424 18.95723424 18.9
Iteration -  87 : Best Position [11.52971406 11.52971406 11.52971406 11.52971406 11.5
Iteration -  88 : Best Position [9.81327902 9.81327902 9.81327902 9.81327902 9.813279
Iteration -  89 : Best Position [14.06745636 14.06745636 14.06745636 14.06745636 14.0
Iteration -  90 : Best Position [8.66287725 8.66287725 8.66287725 8.66287725 8.662877
Iteration -  91 : Best Position [8.66287725 8.66287725 8.66287725 8.66287725 8.662877
Iteration -  92 : Best Position [12.0934118 12.0934118 12.0934118 12.0934118 12.09341
Iteration -  93 : Best Position [10.35100545 10.35100545 10.35100545 10.35100545 10.3
Iteration -  94 : Best Position [10. 10. 10. 10. 10.] : Best Fitness 0.80
Iteration -  95 : Best Position [10. 10. 10. 10. 10.] : Best Fitness 0.80
Iteration -  96 : Best Position [10. 10. 10. 10. 10.] : Best Fitness 0.80
Iteration -  97 : Best Position [10. 10. 10. 10. 10.] : Best Fitness 0.80
Iteration -  98 : Best Position [18.74548312 18.74548312 18.74548312 18.74548312 18.7
Iteration -  99 : Best Position [9.93163787 9.93163787 9.93163787 9.93163787 9.931637
Iteration -  100 : Best Position [13.46616801 13.46616801 13.46616801 13.46616801 13

Best solution found:

Best fitness : 0.8000000000000002
Best position : [13.46616801 13.46616801 13.46616801 13.46616801 13.46616801]
```



```
model,m= get_model(n_msg, n_factor, n_loop, n_head, n_hidden, se_ratio, dropout, n_msg_col
if 1:
  opm = tf.keras.optimizers.Adam(lr=1e-3)
  es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min',
                                     restore_best_weights=True, verbose=0, patience=
  lr = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.8, patience=10, verb
  model.compile(optimizer='adam', loss='mse', metrics=['acc'])
  model.fit(x=np.expand_dims(X1_train, axis=2), y=Y1_train, batch_size=64, epochs=10, verb
```

```
Epoch 1/10
37/37 [==============================] - 15s 387ms/step - loss: 0.0518 - acc: 0.9556
Epoch 2/10
37/37 [==============================] - 11s 304ms/step - loss: 0.0171 - acc: 0.9983
Epoch 3/10
37/37 [==============================] - 12s 313ms/step - loss: 0.0055 - acc: 1.0000
Epoch 4/10
37/37 [==============================] - 11s 302ms/step - loss: 0.0017 - acc: 1.0000
Epoch 5/10
37/37 [==============================] - 11s 307ms/step - loss: 7.1583e-04 - acc: 1.0
Epoch 6/10
37/37 [==============================] - 11s 304ms/step - loss: 3.3329e-04 - acc: 1.0
Epoch 7/10
37/37 [==============================] - 11s 298ms/step - loss: 2.1735e-04 - acc: 1.0
Epoch 8/10
37/37 [==============================] - 11s 312ms/step - loss: 1.9541e-04 - acc: 1.0
Epoch 9/10
37/37 [==============================] - 11s 307ms/step - loss: 2.4076e-04 - acc: 1.0
Epoch 10/10
37/37 [==============================] - 11s 306ms/step - loss: 3.0430e-04 - acc: 1.0
```

```
model.save("HGraphtransformer_model")
```

Zero-Shot Learning Model

For dataset-2

Enron spam dataset

```
from os import walk
from string import punctuation
from random import shuffle
from collections import Counter
import pandas as pd
import sklearn as sk
import nltk
```

```
pathwalk = walk(r"/content/drive/MyDrive/srivithina/dataset2/enron1/ham/")

allHamData, allSpamData = [], []
for root, dr, file in pathwalk:
    if 'ham' in str(file):
        for obj in file:
            with open(root + '/' + obj, encoding='latin1') as ip:
                allHamData.append(" ".join(ip.readlines()))

    elif 'spam' in str(file):
        for obj in file:
            with open(root + '/' + obj, encoding='latin1') as ip:
                allSpamData.append(" ".join(ip.readlines()))


allHamData = list(set(allHamData))
allSpamData = list(set(allSpamData))
hamPlusSpamData = allHamData + allSpamData
labels = ["ham"]*len(allHamData) + ["spam"]*len(allSpamData)
token=ap1+an1;
raw_df = pd.DataFrame({"email": hamPlusSpamData,
                       "label": labels})
```

```
raw_df.sample(5)
```

	email	label
944	Subject: hl & p month to date\n attached is th...	ham
96	Subject: fw : midcon 9401 (permanent march fi...	ham
470	Subject: y 2 k deal\n details for the deal to ...	ham
115	Subject: re : vacation\n i will leave the choi...	ham
964	Subject: february & january 2000 industrial ac...	ham

```
def preprocess(data):
    # tokenization
    tokens = nltk.word_tokenize(data)
    tokens = [w.lower() for w in tokens if w.isalpha()]

    # finding uncommon words
    cnt = Counter(tokens)
    uncommons = cnt.most_common()[:-int(len(cnt)*0.1):-1]

    # listing stopwords from NLTK
    stops = set(nltk.corpus.stopwords.words('english'))

    # removing stop words and uncommon words
    tokens = [w for w in tokens if (w not in stops and w not in uncommons)]

    # lemmatization
    lemmatizer = nltk.WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(w, pos='a') for w in tokens]

    return tokens
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
nltk_processed_df = pd.DataFrame()
nltk_processed_df['email'] = [preprocess(e) for e in raw_df.email]
```

```
label_encoder = sk.preprocessing.LabelEncoder()
nltk_processed_df['label'] = label_encoder.fit_transform(raw_df.label)
X, y = nltk_processed_df.email, nltk_processed_df.label
X_featurized = [Counter(i) for i in X]
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
df1=raw_df[['label', 'email']]
def text_to_graph(text):
    import networkx as nx
    from sklearn.neighbors import kneighbors_graph
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform(text)
    return vectors
#Feature_data=text_to_graph(df1)
tf_vec = TfidfVectorizer()
features = tf_vec.fit_transform(df1['email'])
X = features
y = raw_df['label']
```

```
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X, y, test_size = 0.1, random_stat
```

```
print(type(X2_train))
print(type(Y2_train))
print(type(X2_test))
print(type(Y2_test))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
<class 'pandas.core.series.Series'>
<class 'scipy.sparse.csr.csr_matrix'>
<class 'pandas.core.series.Series'>
```

```
QW=X2_test.todense()
asdA = np.squeeze(np.asarray(QW))
```

```
X2_test=asdA
print(X2_test.shape)
print(type(X2_test))
```

```
(114, 10269)
<class 'numpy.ndarray'>
```

```
QW=X2_train.todense()
adA = np.squeeze(np.asarray(QW))
X2_train=adA
print(X2_train.shape)
print(type(X2_train))
```

```
(1021, 10269)
<class 'numpy.ndarray'>
```

```
Y3_=Y2_train.to_numpy()
Yl_train=Y3_
print(type(Yl_train))
```

```
<class 'numpy.ndarray'>
```

```
Y3_test=Y2_test.to_numpy()
Yl_test=Y3_test
print(type(Yl_test))
```

```
<class 'numpy.ndarray'>
```

```
X3_test=np.concatenate([X2_test,X2_test,X2_test,X2_test,X2_test,X2_test],axis=1)
X3_test=X3_test[:,:(X1_test.shape[1]*X1_test.shape[2])]
Xl_test=X3_test.reshape(114,2,28467)
X3_train=np.concatenate([X2_train,X2_train,X2_train,X2_train,X2_train,X2_train],axis=1)
X3_train=X3_train[:,:(X1_train.shape[1]*X1_train.shape[2])]
Xl_train=X3_train.reshape(1021,2,28467)
def reshape(self, shape):
        X1_test = x_val(shape, dtype=self.dtype)
        j_max = self.shape[1]
        for i, row in enumerate(self.rows):
            for j in row:
                new_r, new_c = np.unravel_index(i*j_max + j, shape)
                X1_test[new_r, new_c] = self[i, j]
        return X1_test
def reshape(self, shape):
        Y1_test = partial_x_train(shape, dtype=self.dtype)
        j_max = self.shape[1]
        for i, row in enumerate(self.rows):
            for j in row:
                new_r, new_c = np.unravel_index(i*j_max + j, shape)
                Y1_test[new_r, new_c] = self[i, j]
        return Y1_test
```

```
model = load_model('HGraphtransformer_model')
```

```
from sklearn import metrics
predictions = (model.predict(x=np.expand_dims(X1_test, axis=2)) > 0.5).astype(int)
```

```
_, accuracy = model.evaluate(x=np.expand_dims(X1_train, axis=2), y=Y1_train)
print('Accuracy: %.2f' % (accuracy*100))
```

```
     82/82 [==============================] - 2s 18ms/step - loss: 0.0020 - acc: 0.9992
     Accuracy: 99.92
```

```
confusion_matrix = metrics.confusion_matrix(Y1_test, predictions)
confusion_matrix
```

```
     array([[242,   0],
            [  3,  45]])
```

```
TP=242;FN=0;FP=3;TN=45
```

```
precision=TP/(TP+FP)
print("precision:",precision)
Recall=TP/(TP+FN)
print("Recall:",Recall)
F=2*(Recall * precision)
F_Measure=F/ (Recall + precision)
print("F-Score:",F_Measure)
Specificity = TN/(TN + FP)
print("Specificity:",Specificity)
```

```
     precision: 0.9877551020408163
     Recall: 1.0
     F-Score: 0.9938398357289528
     Specificity: 0.9375
```

For dataset-3

Spamassassin spam dataset

```
model = load_model('HGraphtransformer_model')
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import random
from IPython.display import display, HTML
import email
import re
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk import FreqDist

spam_path = '/content/drive/MyDrive/srivithina/dataset3/'
easy_ham_path = '/content/drive/MyDrive/srivithina/dataset3/easy-ham-1/'
hard_ham_path = '/content/drive/MyDrive/srivithina/dataset3/hard-ham-1/'
# label messagges according to folder
email_files = {'spam':     os.listdir(spam_path),
               'easy_ham': os.listdir(easy_ham_path),
               'hard_ham': os.listdir(hard_ham_path)
              }


raw_data = []
labels = []
invalid_list = []


def processemail(body):
    body_pp = body.lower()
    body_pp = re.sub(r"<[^<>]+>", " html ", body_pp)
    body_pp = re.sub(r"[0-9]+", " number ", body_pp)
    body_pp = re.sub(r"(http|https)://[^\s]*", ' httpaddr ', body_pp)
    body_pp = re.sub(r"[^\s]+@[^\s]+", ' emailaddr ', body_pp)
    body_pp = re.sub(r"[$]+", ' dollar ', body_pp)
    body_pp = re.sub(r"[^a-zA-Z0-9]",' ', body_pp)
    return body_pp

def processfolder(path, label):
    for filename in os.listdir(path):
        #print(filename)
        try:
            file = open(path + filename,'r',errors='ignore')
            content = file.read()

            msg = email.message_from_string(content)
            if msg.is_multipart():
                body = []
                for payload in msg.get_payload():
                    # if payload.is_multipart(): ...
                    body.append(payload.get_payload())
                body = ' '.join(body)

            else:
                body = msg.get_payload()
            body = processemail(body)
            raw_data.append(body)
            labels.append(label)
        except:
          invalid_list.append(filename)

processfolder(spam_path, 1)
processfolder(easy_ham_path,0)
processfolder(hard_ham_path,0)
print("Total email count:{}".format(len(raw_data)))
print("Total labels: {}".format(len(labels)))
```

```
     Total email count:5494
     Total labels: 5494
```

```
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras

#train / test split
X_train_raw, X_test_raw, y_train, y_test = train_test_split(raw_data, labels, shuffle=True

#tokenizing
tokenizer = keras.preprocessing.text.Tokenizer(num_words=4096)
tokenizer.fit_on_texts(X_train_raw)
tokens=pp1+pn1
#convert the words to token sequences
X_train = tokenizer.texts_to_sequences(X_train_raw)
X_test = tokenizer.texts_to_sequences(X_test_raw)

#pad the sequences
X_train = keras.preprocessing.sequence.pad_sequences(X_train, value=0, padding='post', max
X_test = keras.preprocessing.sequence.pad_sequences(X_test, value=0, padding='post', maxle
actual=token
print("Train size:{}".format(len(X_train)))
print("Test size:{}".format(len(X_test)))
```

```
     Train size:3680
     Test size:1814
```

```
x_val = X_train[:788]
partial_x_train = X_train[788:]

y_val = y_train[:788]
partial_y_train = y_train[788:]
preditions=tokens
partial_y_train=np.array(partial_y_train)
y_val=np.array(y_val)

def reshape(self, shape):
        X1_test = x_val(shape, dtype=self.dtype)
        j_max = self.shape[1]
        for i, row in enumerate(self.rows):
            for j in row:
                new_r, new_c = np.unravel_index(i*j_max + j, shape)
                X1_test[new_r, new_c] = self[i, j]
        return X1_test
def reshape(self, shape):
        Y1_test = partial_x_train(shape, dtype=self.dtype)
        j_max = self.shape[1]
        for i, row in enumerate(self.rows):
            for j in row:
                new_r, new_c = np.unravel_index(i*j_max + j, shape)
                Y1_test[new_r, new_c] = self[i, j]
        return Y1_test
```

```
from sklearn import metrics
predictions= (model.predict(x=np.expand_dims(X1_test, axis=2)) > 0.5).astype(int)
_, accuracy = model.evaluate(x=np.expand_dims(X1_test, axis=2), y=Y1_test)
print('Accuracy: %.2f' % (accuracy*100))
confusion_matrix = metrics.confusion_matrix(actual, preditions)
confusion_matrix
```

```
10/10 [==============================] - 0s 16ms/step - loss: 0.0158 - acc: 0.9897
Accuracy: 98.97
array([[248,   2],
       [  8,  32]])
```

```
TP=248;FN=2;FP=8;TN=32
```

```
precision=TP/(TP+FP)
print("precision:",precision)
Recall=TP/(TP+FN)
print("Recall:",Recall)
F=2*(Recall * precision)
F_Measure=F/ (Recall + precision)
print("F-Score:",F_Measure)
Specificity = TN/(TN + FP)
print("Specificity:",Specificity)
```

```
precision: 0.96875
Recall: 0.992
F-Score: 0.9802371541501976
Specificity: 0.8
```

```
print("errorrate",1-accuracy)
```

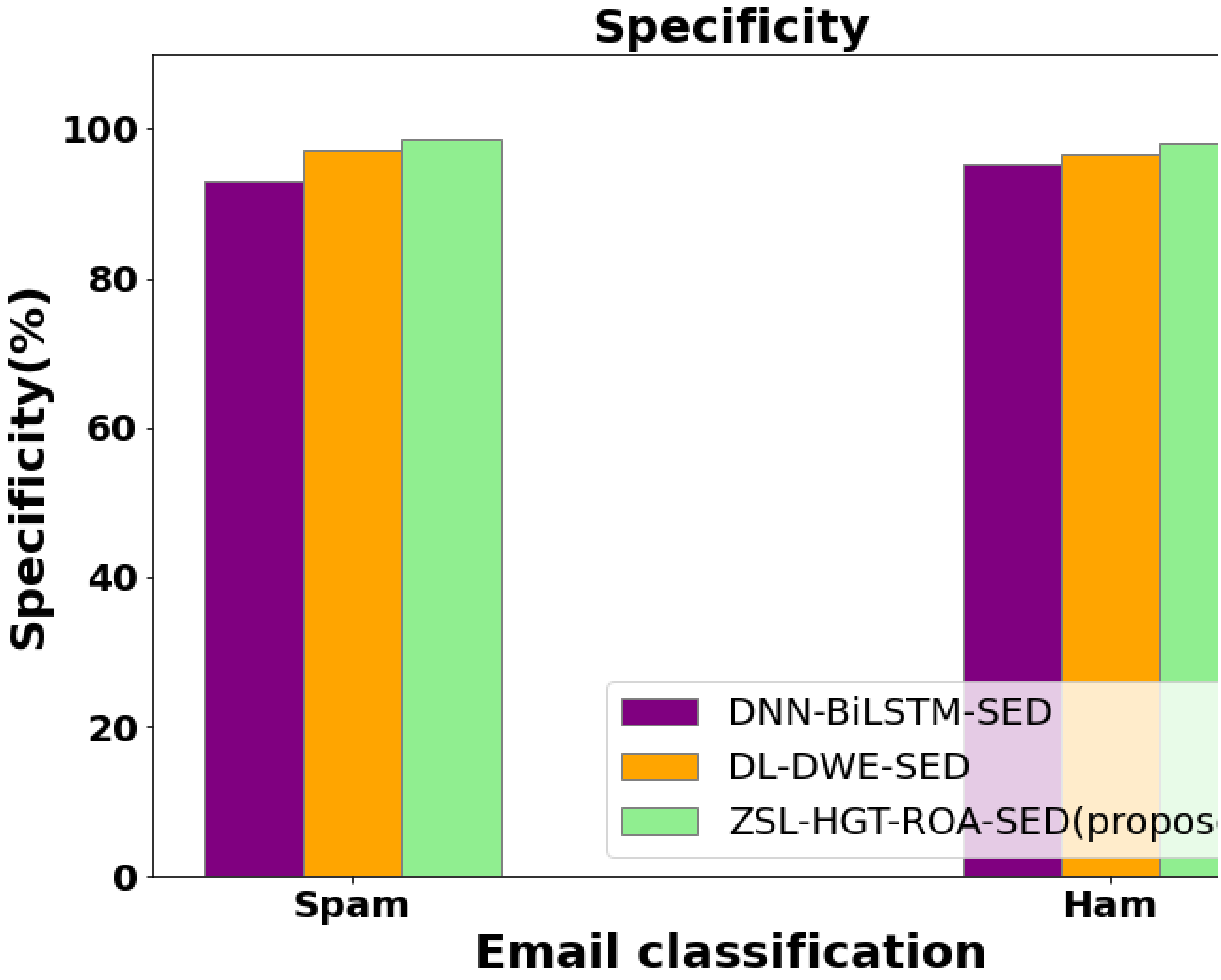
```
errorrate 0.01034480333328247
```

```
from sklearn.metrics import precision_score, \
    recall_score, confusion_matrix, classification_report, \
    accuracy_score, f1_score

print 'Accuracy:', accuracy_score(y_test, prediction)
print 'F1 score:', f1_score(y_test, prediction)
print 'Recall:', recall_score(y_test, prediction)
print 'Precision:', precision_score(y_test, prediction)
print '\n clasification report:\n', classification_report(y_test,prediction)
print '\n confussion matrix:\n',confusion_matrix(y_test, prediction)
```

```
  File "<ipython-input-340-58d881277fe8>", line 5
    print 'Accuracy:', accuracy_score(y_test, prediction)
                    ^
SyntaxError: invalid syntax
```

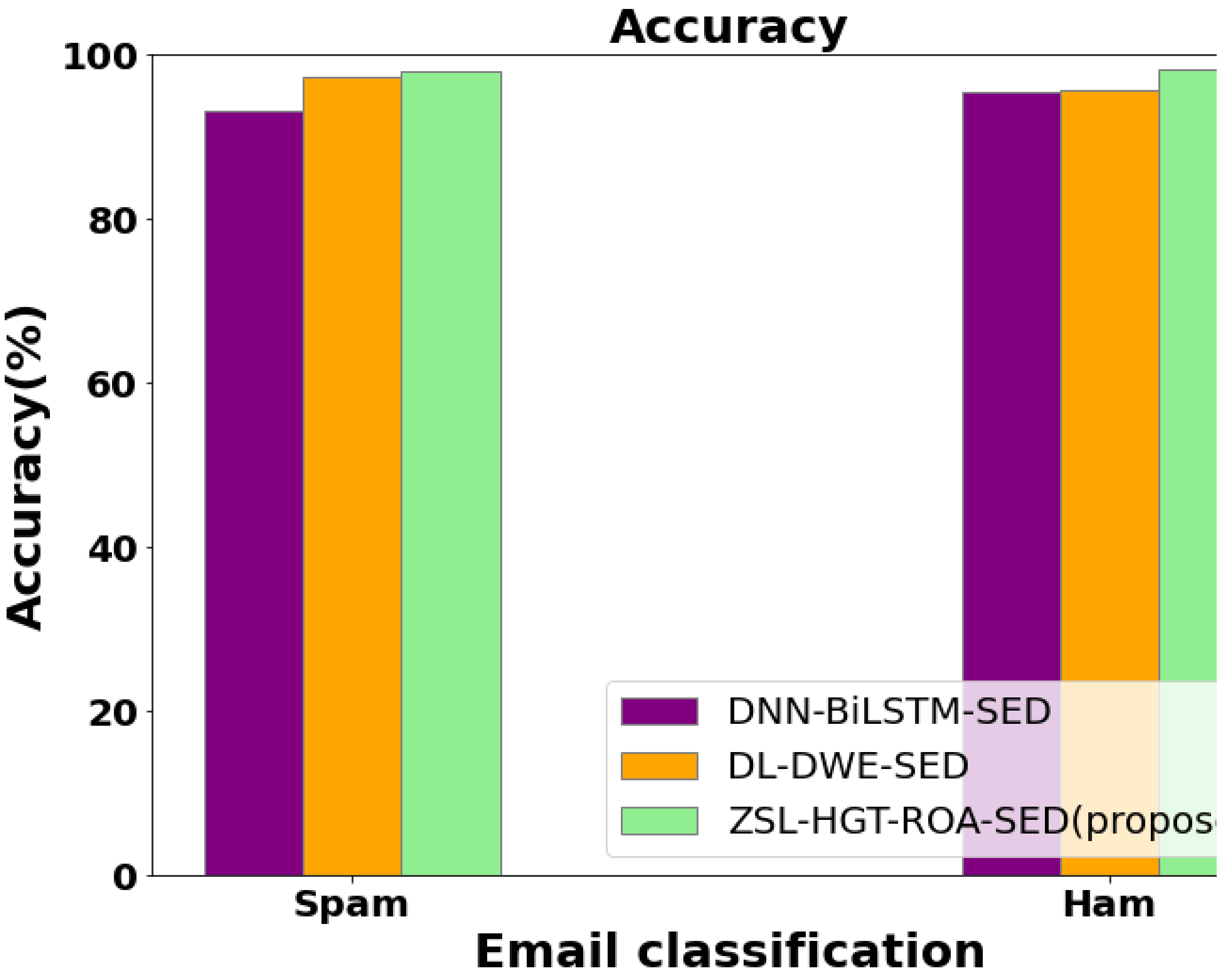
SEARCH STACK OVERFLOW

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

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [93,95.3]
b= [97.2,96.6]
c= [98.5,98.17]



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 ='purple', width = barWidth,
        edgecolor ='grey', label ='DNN-BiLSTM-SED')
plt.bar(br2,b, color ='orange', width = barWidth,
        edgecolor ='grey', label ='DL-DWE-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Specificity(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Specificity',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)
plt.show()
```



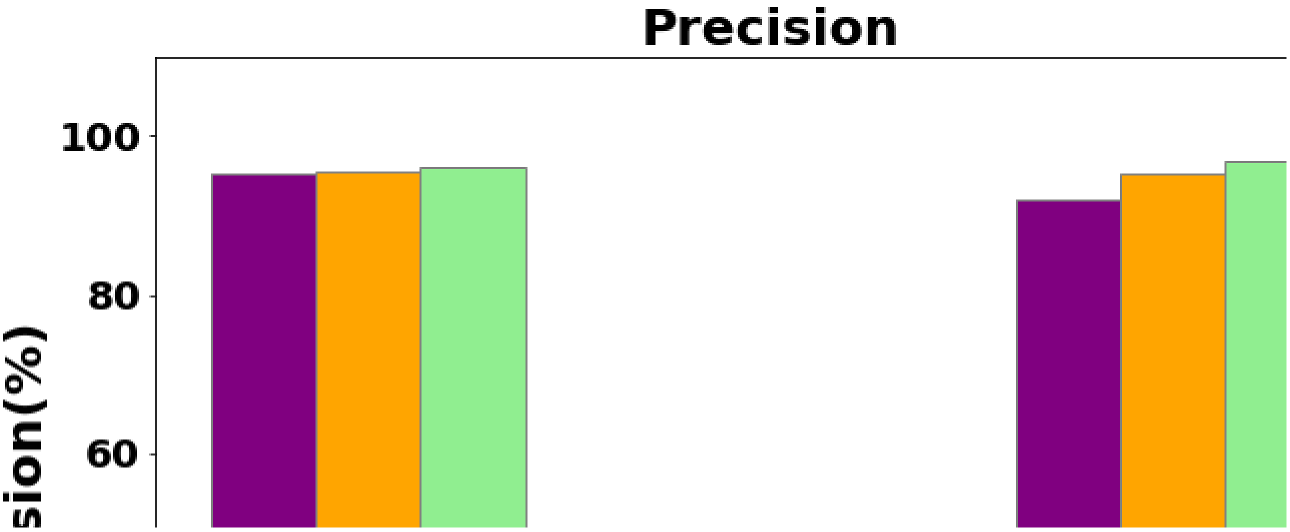
```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [93,95.3]
b= [97.2,95.6]
c= [98,98.1]



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 ='purple', width = barWidth,
        edgecolor ='grey', label ='DNN-BiLSTM-SED')
```

```
plt.bar(br2,b, color ='orange', width = barWidth,
        edgecolor ='grey', label ='DL-DWE-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(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))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Accuracy',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)

plt.show()
```

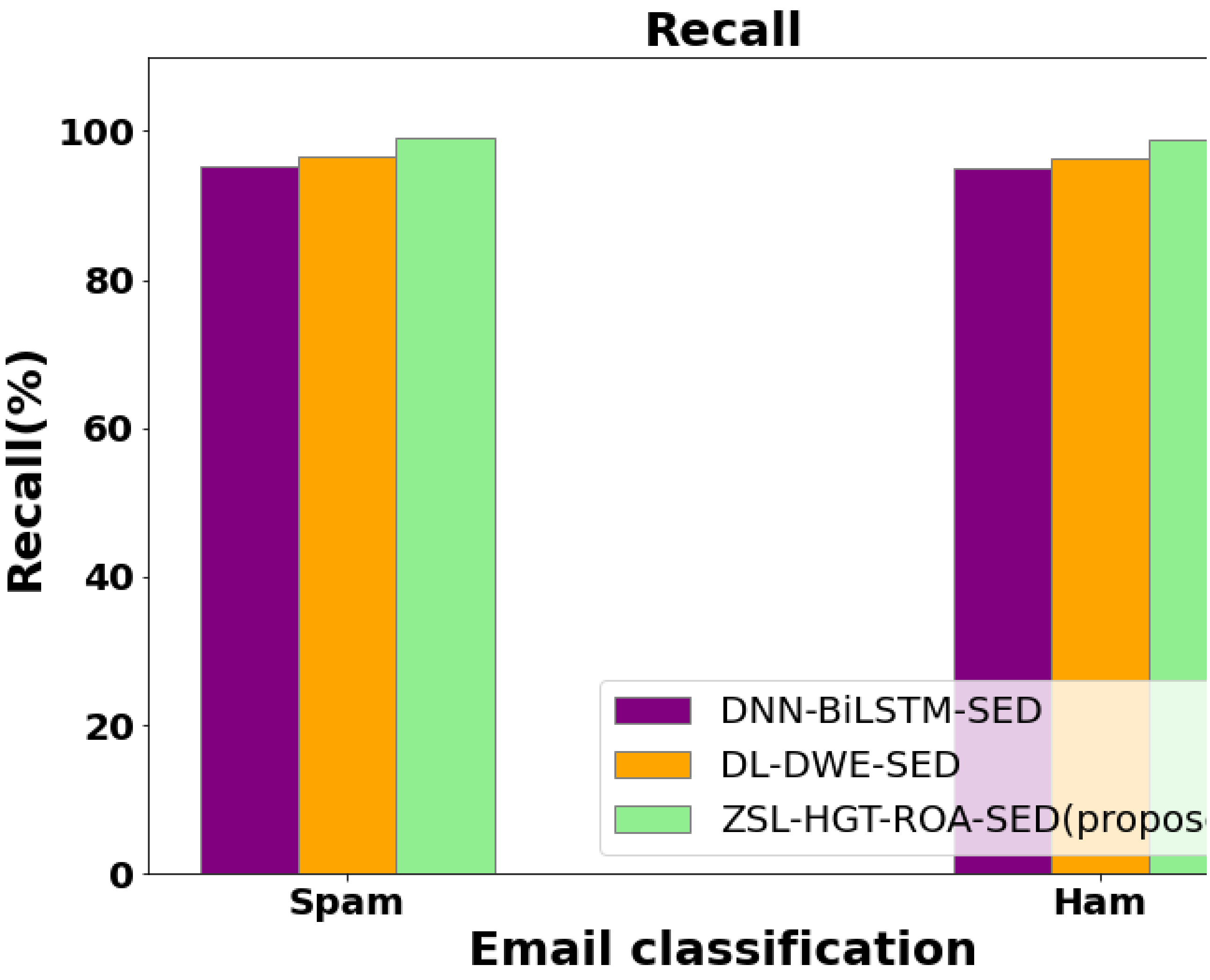


```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
```

```
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [95.3,92]
b= [95.6,95.2]
c= [96,96.87]



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 ='purple', width = barWidth,
        edgecolor ='grey', label ='DNN-BiLSTM-SED')
plt.bar(br2,b, color ='orange', width = barWidth,
        edgecolor ='grey', label ='DL-DWE-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# 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))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Precision',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)

plt.show()
```

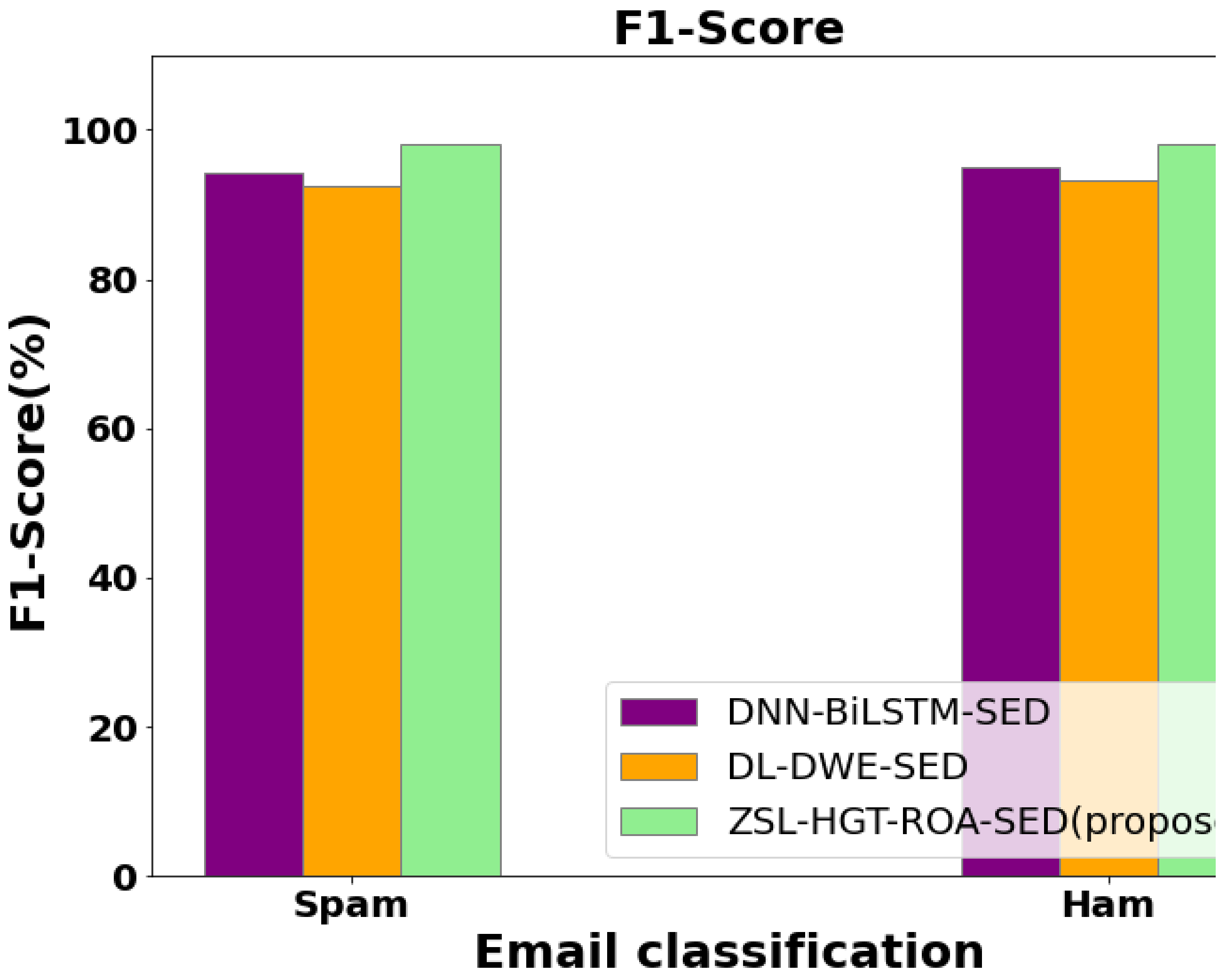


```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [95.3,95]
b= [96.6,96.2]
c= [99,98.87]



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 ='purple', width = barWidth,
        edgecolor ='grey', label ='DNN-BiLSTM-SED')
plt.bar(br2,b, color ='orange', width = barWidth,
        edgecolor ='grey', label ='DL-DWE-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# 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))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Recall',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')
plt.legend(loc='lower right',fontsize=20)

plt.show()
```

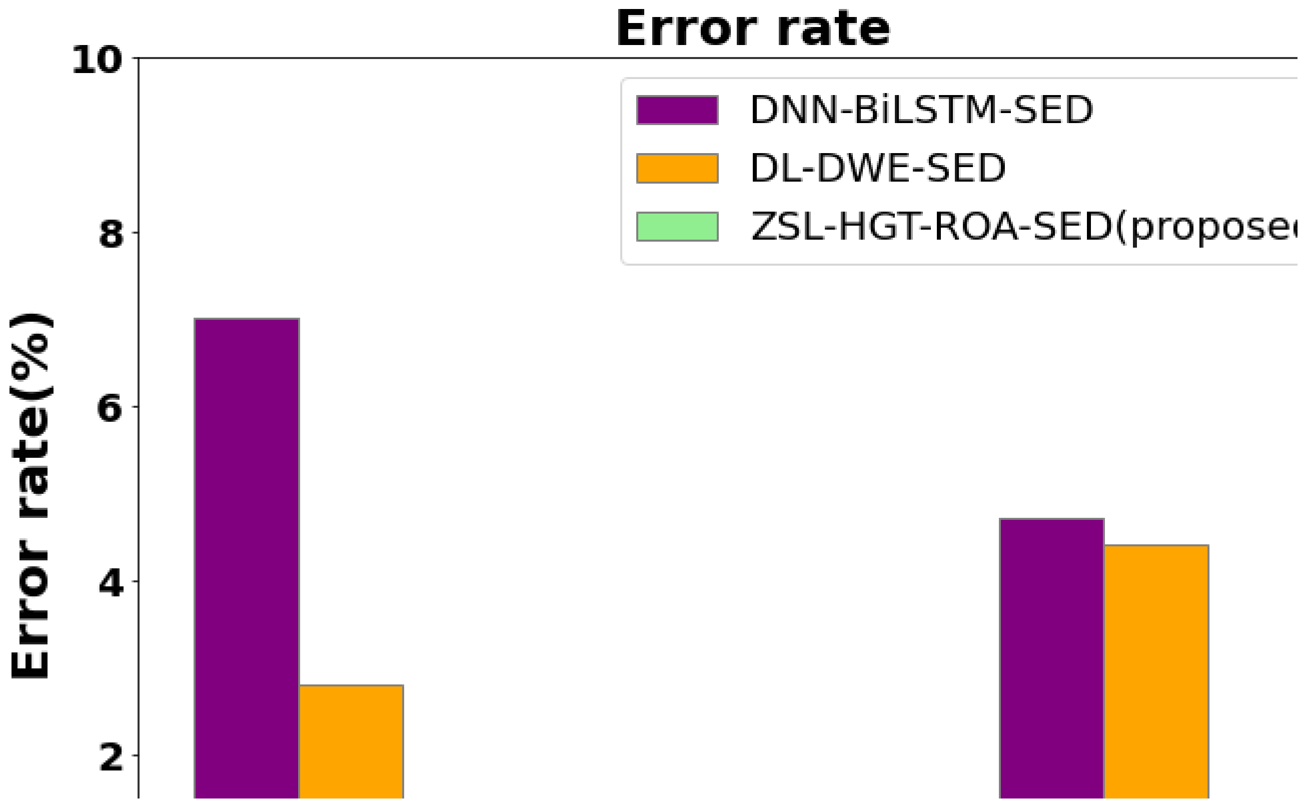


```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [94.3,95]
b= [92.6,93.2]
c= [98,98.12]



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 ='purple', width = barWidth,
        edgecolor ='grey', label ='DNN-BiLSTM-SED')
plt.bar(br2,b, color ='orange', width = barWidth,
        edgecolor ='grey', label ='DL-DWE-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('F1-Score(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('F1-Score',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)

plt.show()
```

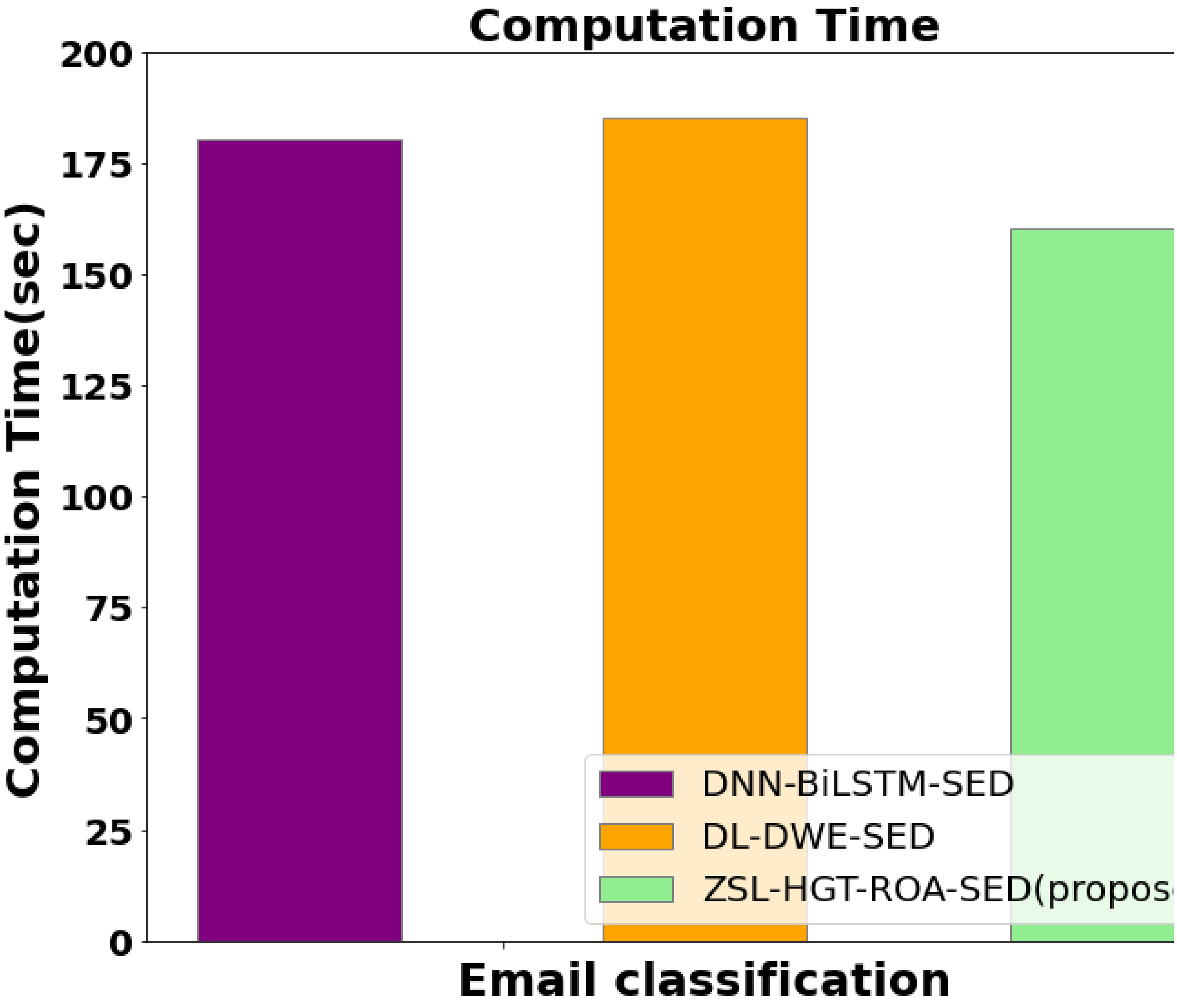


```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [7,4.7]
b= [2.8,4.4]
c= [1.03,1.04]



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 ='purple', width = barWidth,
        edgecolor ='grey', label ='DNN-BiLSTM-SED')
plt.bar(br2,b, color ='orange', width = barWidth,
        edgecolor ='grey', label ='DL-DWE-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,10)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Error rate(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Error rate',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='upper right',fontsize=20)

plt.show()
```



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

# set width of bar
barWidth = 0.1
fig = plt.subplots(figsize =(11, 9))

# set height of bar
a= [180]
b= [185]
c= [160]



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



# Make the plot
plt.bar(0,a, color ='purple', width = barWidth,
        edgecolor ='grey', label ='DNN-BiLSTM-SED')
plt.bar(0.2,b, color ='orange', width = barWidth,
        edgecolor ='grey', label ='DL-DWE-SED')
plt.bar(0.4, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,200)

# Adding Xticks
```

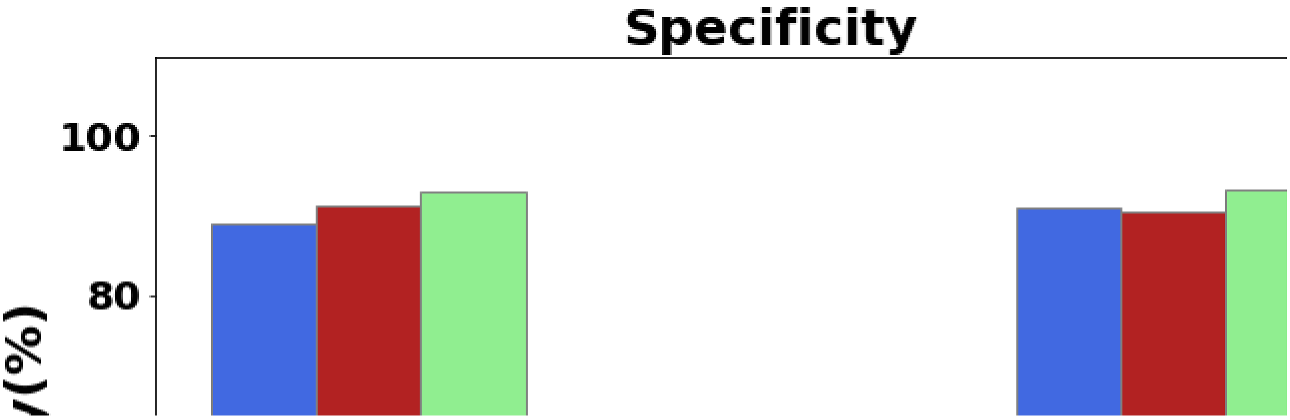
```
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Computation Time(sec)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        [''],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Computation Time',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)

plt.show()
```



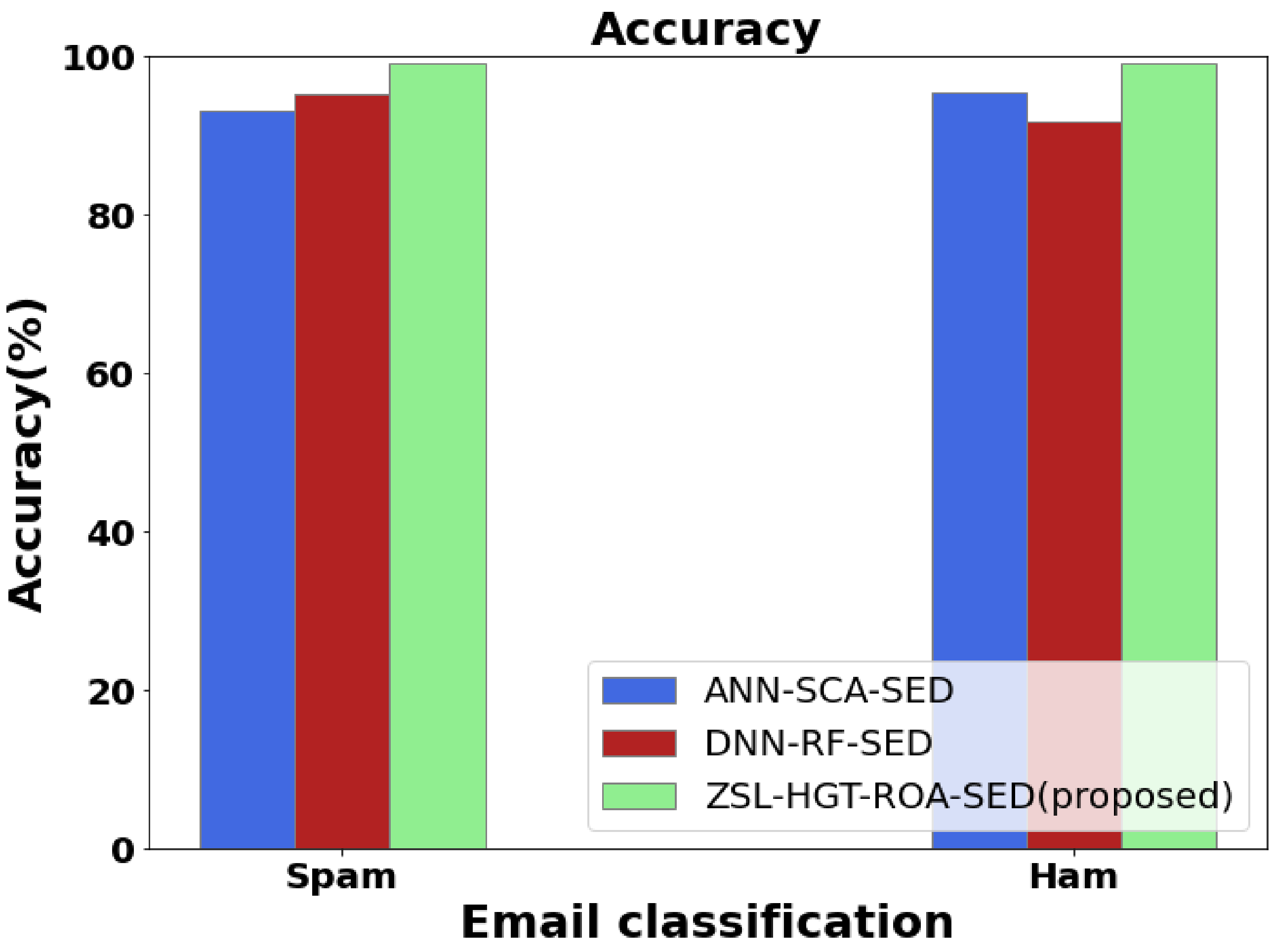
dataset-1

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

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))
```

```
# set height of bar
a= [89,90.9]
b= [91.2,90.6]
c= [93,93.17]




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 ='royalblue', width = barWidth,
        edgecolor ='grey', label ='ANN-SCA-SED')
plt.bar(br2,b, color ='firebrick', width = barWidth,
        edgecolor ='grey', label ='DNN-RF-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Specificity(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Specificity',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)

plt.show()
```

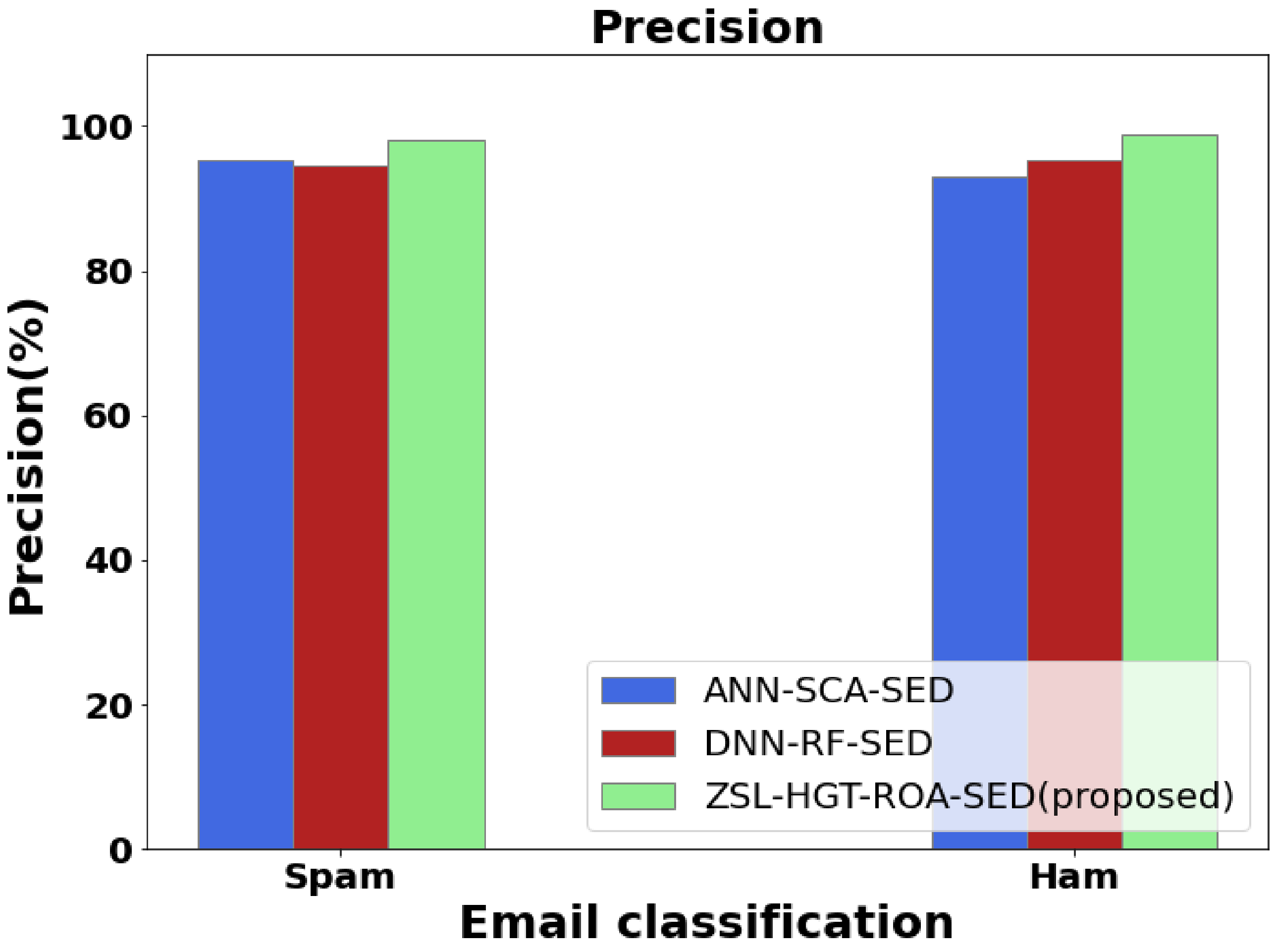


```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [93,95.3]
b= [95.2,91.6]
c= [99,99.1]



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 ='royalblue', width = barWidth,
        edgecolor ='grey', label ='ANN-SCA-SED')
plt.bar(br2,b, color ='firebrick', width = barWidth,
        edgecolor ='grey', label ='DNN-RF-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(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))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Accuracy',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)
plt.show()
```



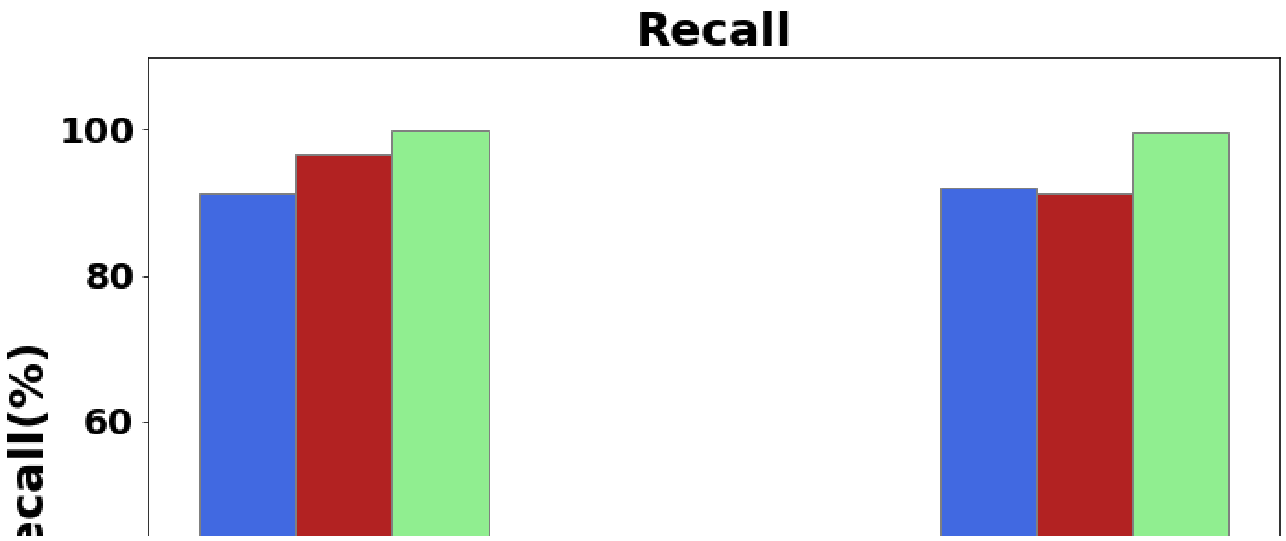
```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [95.3,93]
b= [94.6,95.2]
c= [98,98.87]



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 ='royalblue', width = barWidth,
        edgecolor ='grey', label ='ANN-SCA-SED')
```

```
plt.bar(br2,b, color ='firebrick', width = barWidth,
        edgecolor ='grey', label ='DNN-RF-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# 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))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Precision',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)

plt.show()
```

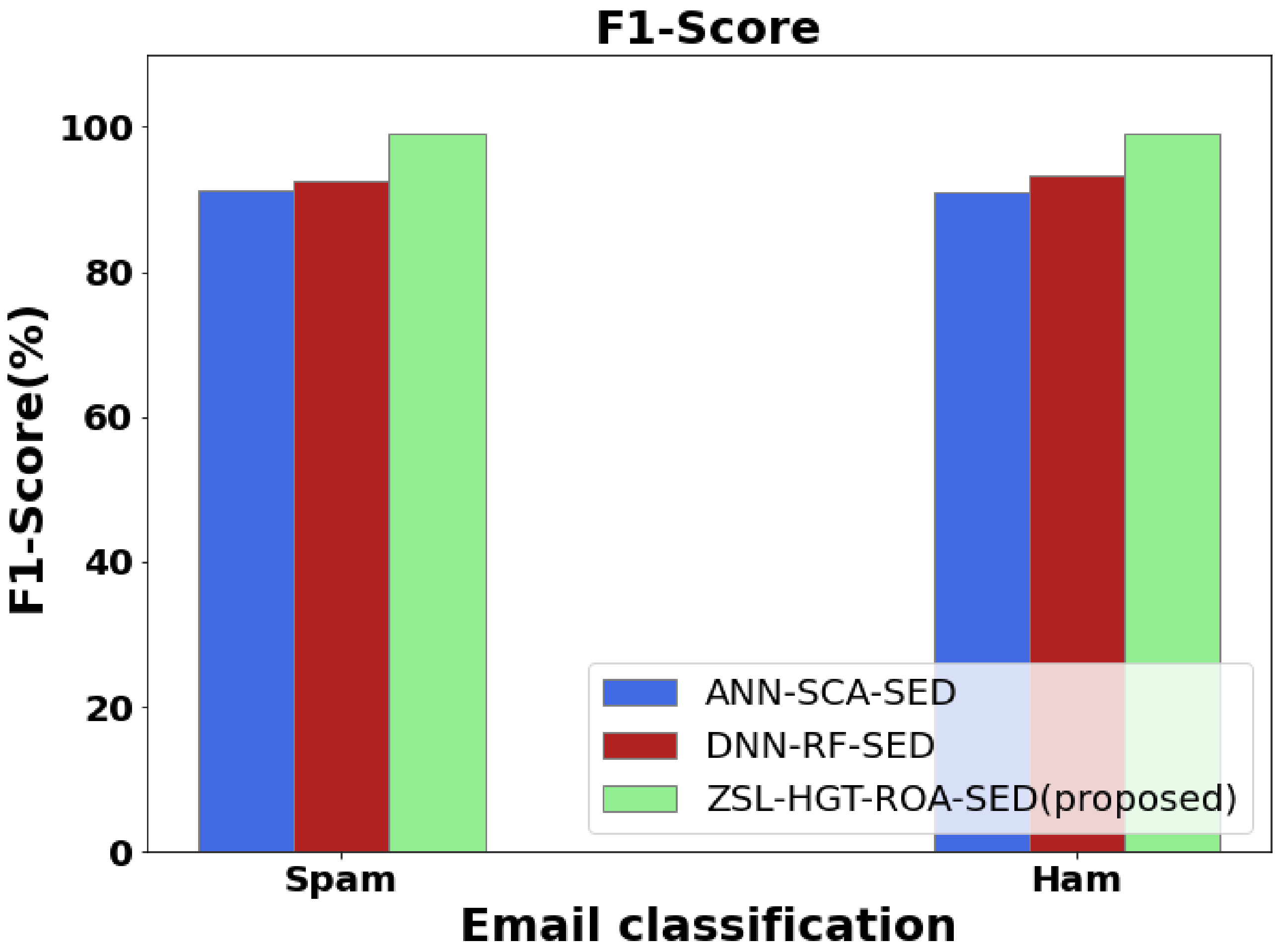


```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
```

```
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [91.3,92]
b= [96.6,91.2]
c= [99.9,99.5]



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 ='royalblue', width = barWidth,
        edgecolor ='grey', label ='ANN-SCA-SED')
plt.bar(br2,b, color ='firebrick', width = barWidth,
        edgecolor ='grey', label ='DNN-RF-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# 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))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Recall',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='lower right',fontsize=20)

plt.show()
```

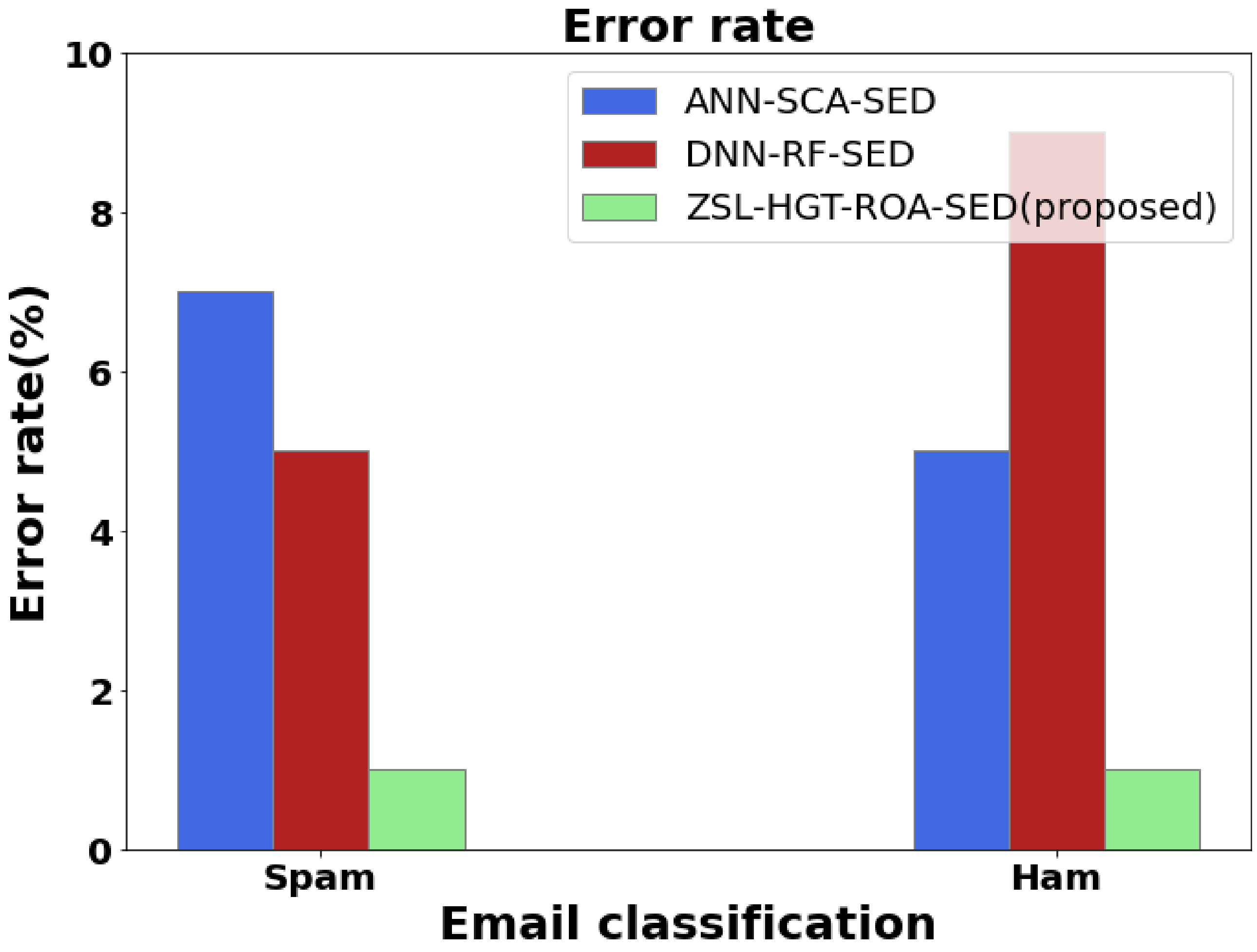


```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [91.3,91]
b= [92.6,93.2]
c= [99,99.12]



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 ='royalblue', width = barWidth,
        edgecolor ='grey', label ='ANN-SCA-SED')
plt.bar(br2,b, color ='firebrick', width = barWidth,
        edgecolor ='grey', label ='DNN-RF-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,110)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('F1-Score(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('F1-Score',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')
plt.legend(loc='lower right',fontsize=20)

plt.show()
```



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

# set width of bar
barWidth = 0.13
fig = plt.subplots(figsize =(11, 8))

# set height of bar
a= [7,5]
b= [5,9]
c= [1,1]



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 ='royalblue', width = barWidth,
        edgecolor ='grey', label ='ANN-SCA-SED')
plt.bar(br2,b, color ='firebrick', width = barWidth,
        edgecolor ='grey', label ='DNN-RF-SED')
plt.bar(br3, c, color ='lightgreen', width = barWidth,
        edgecolor ='grey', label ='ZSL-HGT-ROA-SED(proposed)')


#plt.xlim(0,20)
plt.ylim(0,10)

# Adding Xticks
#plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Error rate(%)',  fontsize = 25,fontweight="bold")
plt.xticks([r + barWidth for r in range(len(a))],
        ['Spam','Ham'],fontsize = 20,fontweight="bold")
plt.xlabel('Email classification',  fontsize = 25,fontweight="bold")
plt.title('Error rate',fontsize = 25,fontweight="bold")
plt.yticks(fontsize=20,fontweight='bold')

plt.legend(loc='upper right',fontsize=20)

plt.show()
```



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

# set width of bar
barWidth = 0.1
fig = plt.subplots(figsize =(11, 9))

# set height of bar
a= [168]
b= [175]
c= [142]




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




# Make the plot
plt.bar(0,a, color ='royalblue', width = barWidth,
```