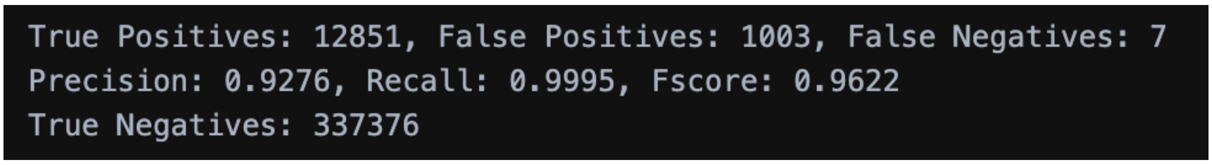
DepImpact: Installed VirtualBox and Vagrant. Still have to look at the VM image. Having issues importing the .ova file into VirtualBox. Will look mainly into DepImpact, this week.

FLASH:

- Pretrained models that were provided by authors were run and verified last week.
- This week, I retrained the models from scratch.
- I was able to complete the reproducibility process for 6 datasets, namely:

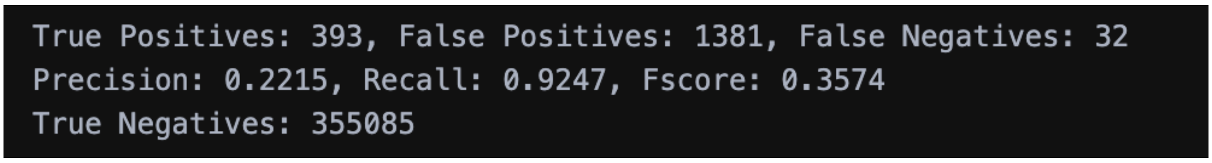
Cadets: Model from scratch provided the same results as the pre-trained model:



```
True Positives: 12851, False Positives: 1003, False Negatives: 7
Precision: 0.9276, Recall: 0.9995, Fscore: 0.9622
True Negatives: 337376
```

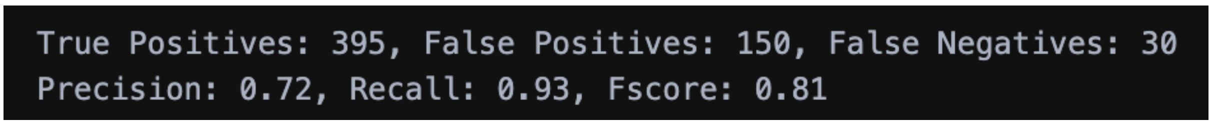
Fivedirections: Model from scratch did not provide same results as the pre-trained model. In fact on multiple runs, the performance kept getting worse.

This is the dataset that the authors had very poor precision on, but the results after training the model from scratch have even lower precision values. I made sure to make multiple runs, all with fresh files, this was the result of the last run I made:



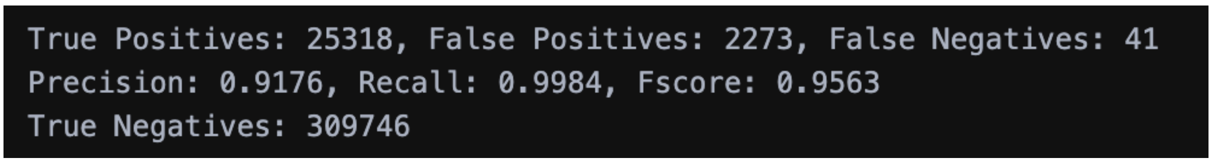
```
True Positives: 393, False Positives: 1381, False Negatives: 32
Precision: 0.2215, Recall: 0.9247, Fscore: 0.3574
True Negatives: 355085
```

While these are the results on the pre-trained model:



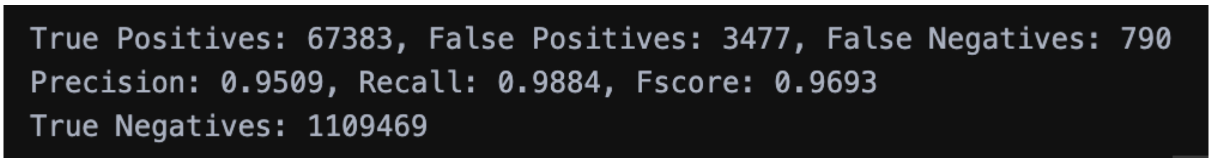
```
True Positives: 395, False Positives: 150, False Negatives: 30
Precision: 0.72, Recall: 0.93, Fscore: 0.81
```

Theia: Model from scratch provided the same results as the pre-trained model:



```
True Positives: 25318, False Positives: 2273, False Negatives: 41
Precision: 0.9176, Recall: 0.9984, Fscore: 0.9563
True Negatives: 309746
```

Trace: Model from scratch provided the same results as the pre-trained model:

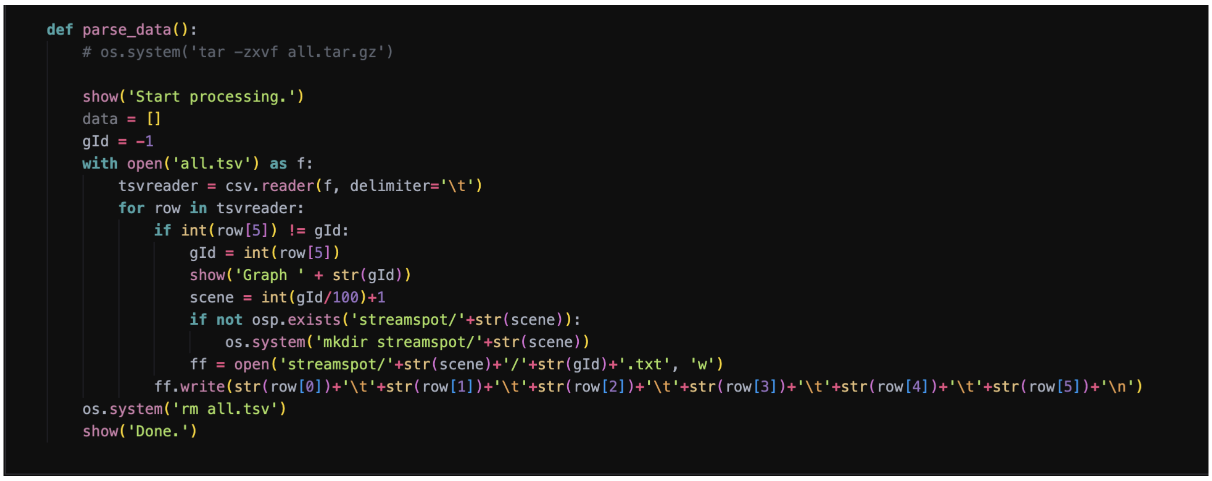


```
True Positives: 67383, False Positives: 3477, False Negatives: 790
Precision: 0.9509, Recall: 0.9884, Fscore: 0.9693
True Negatives: 1109469
```

Streamspot: The code provided to run this was erroneous;

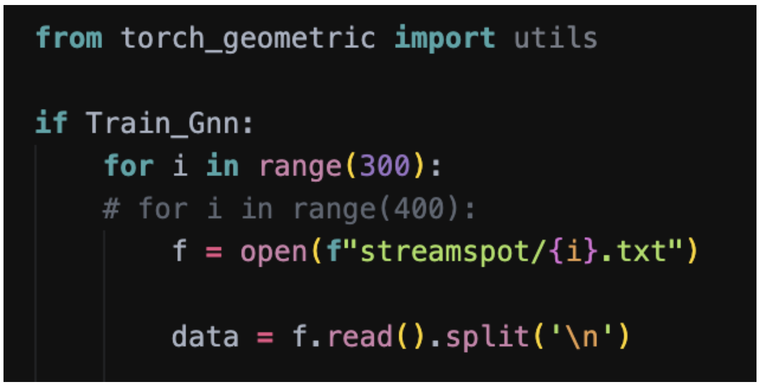
1. The dataset had to be downloaded manually from a link, but then the parser provided for the files was functioning incorrectly. Particularly, it divided the files into separate directories, but then the code after would not be accessing the directories. I had to modify the parser so that the parsed files would have correct structure and access.

This code used a variable ‘scene’ that would create ‘streamspot/0/1.txt’ for example, which would be incorrect for use in later code.



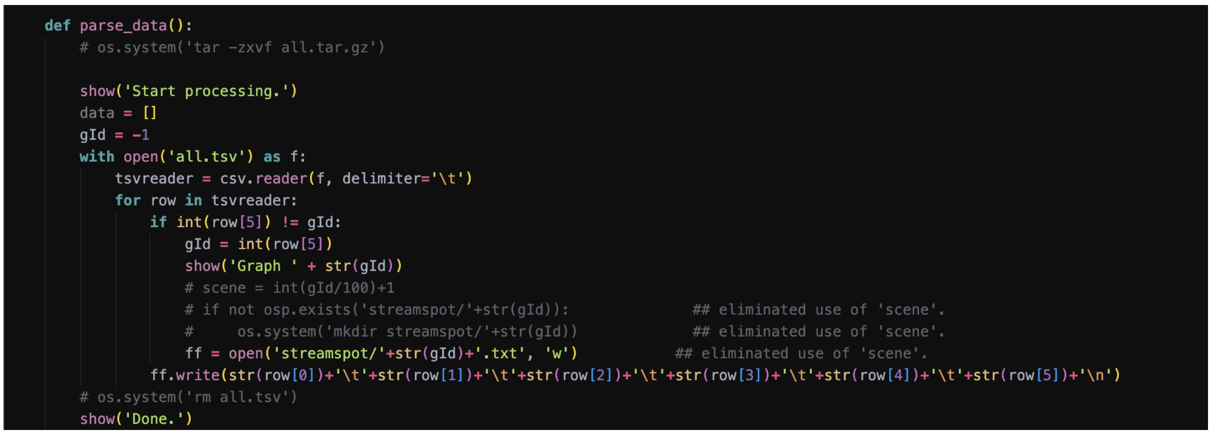
```
def parse_data():
    # os.system('tar -zxvf all.tar.gz')

    show('Start processing.')
    data = []
    gId = -1
    with open('all.tsv') as f:
        tsvreader = csv.reader(f, delimiter='\t')
        for row in tsvreader:
            if int(row[5]) != gId:
                gId = int(row[5])
                show('Graph ' + str(gId))
                scene = int(gId/100)+1
                if not osp.exists('streamspot/'+str(scene)):
                    os.system('mkdir streamspot/'+str(scene))
                ff = open('streamspot/'+str(scene)+'/'+str(gId)+'.txt', 'w')
            ff.write(str(row[0])+'\t'+str(row[1])+'\t'+str(row[2])+'\t'+str(row[3])+'\t'+str(row[4])+'\t'+str(row[5])+'\n')
    os.system('rm all.tsv')
    show('Done.')
```

Use in later code:



```
from torch_geometric import utils

if Train_Gnn:
    for i in range(300):
    # for i in range(400):
        f = open(f"streamspot/{i}.txt")

        data = f.read().split('\n')
```

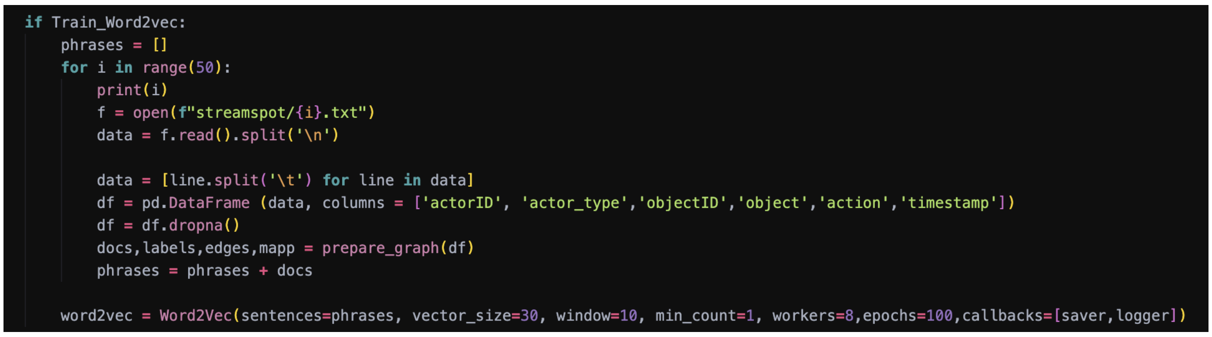
Changes made:



```python
def parse_data():
    # os.system('tar -zxvf all.tar.gz')

    show('Start processing.')
    data = []
    gId = -1
    with open('all.tsv') as f:
        tsvreader = csv.reader(f, delimiter='\t')
        for row in tsvreader:
            if int(row[5]) != gId:
                gId = int(row[5])
                show('Graph ' + str(gId))
                # scene = int(gId/100)+1
                # if not osp.exists('streamspot/'+str(gId)):              ## eliminated use of 'scene'.
                #     os.system('mkdir streamspot/'+str(gId))             ## eliminated use of 'scene'.
                ff = open('streamspot/'+str(gId)+'.txt', 'w')            ## eliminated use of 'scene'.
            ff.write(str(row[0])+'\t'+str(row[1])+'\t'+str(row[2])+'\t'+str(row[3])+'\t'+str(row[4])+'\t'+str(row[5])+'\n')
    # os.system('rm all.tsv')
    show('Done.')
```

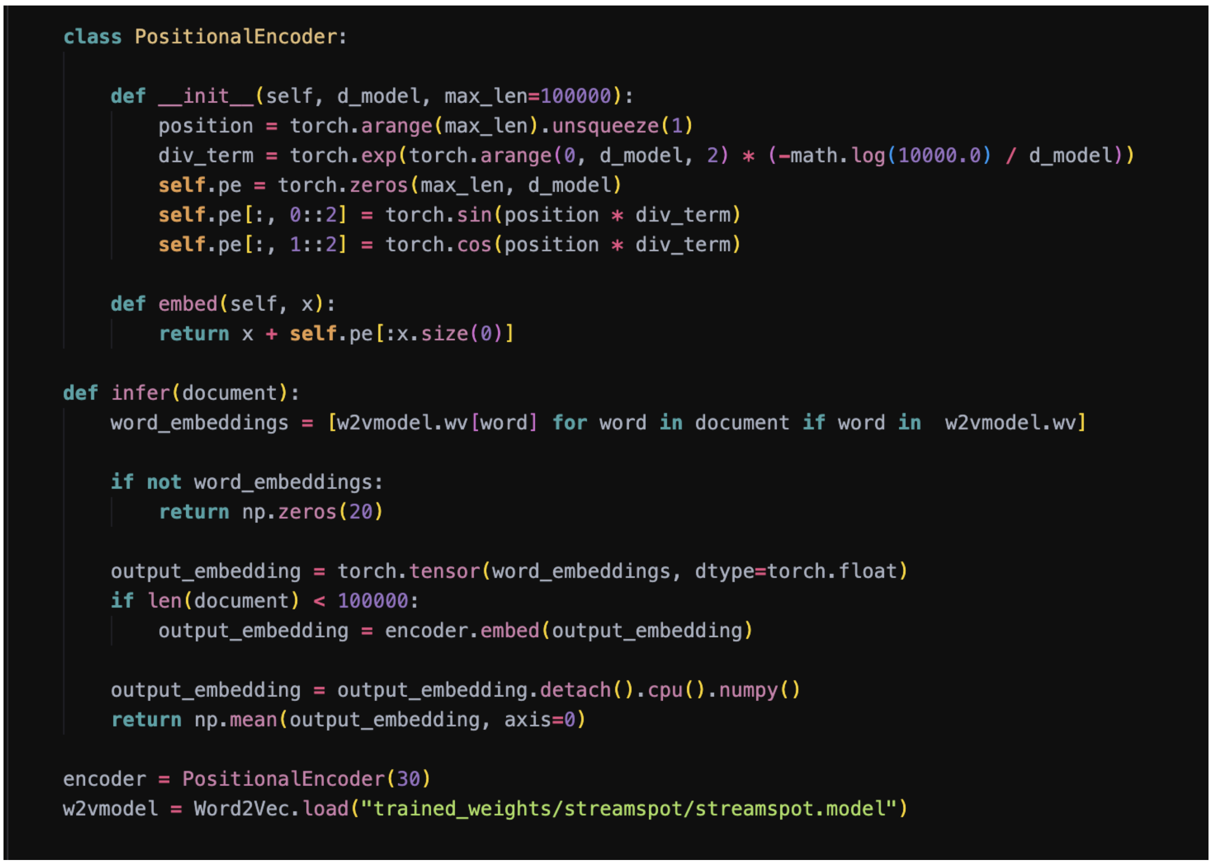
2. The word2vector model was being trained from scratch (if Train_Word2vec was True), but the model trained from this was not being used in the code anywhere. I had to include lines of code to ensure that my trained Word2vec model was the one that was to be used.

A ‘word2vec’ variable is created but not used in further code.



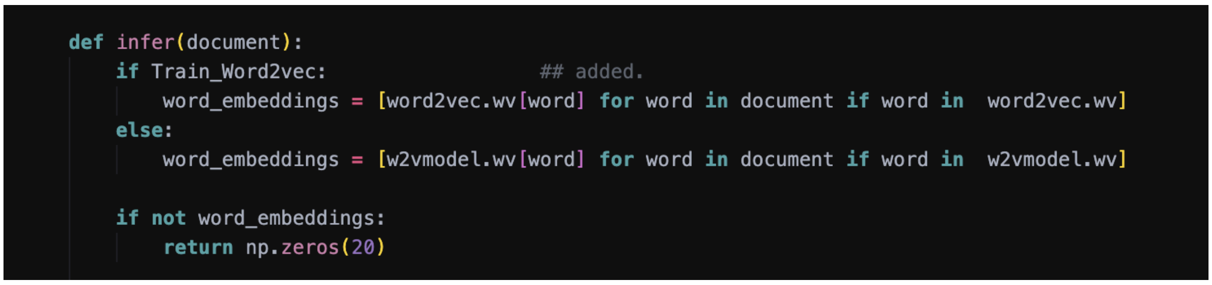
```python
if Train_Word2vec:
    phrases = []
    for i in range(50):
        print(i)
        f = open(f"streamspot/{i}.txt")
        data = f.read().split('\n')

        data = [line.split('\t') for line in data]
        df = pd.DataFrame (data, columns = ['actorID', 'actor_type','objectID','object','action','timestamp'])
        df = df.dropna()
        docs,labels,edges,mapp = prepare_graph(df)
        phrases = phrases + docs

    word2vec = Word2Vec(sentences=phrases, vector_size=30, window=10, min_count=1, workers=8,epochs=100,callbacks=[saver,logger])
```

A new ‘w2vmodel’ is initialised and used regardless of the Train_Word2vec variable value.



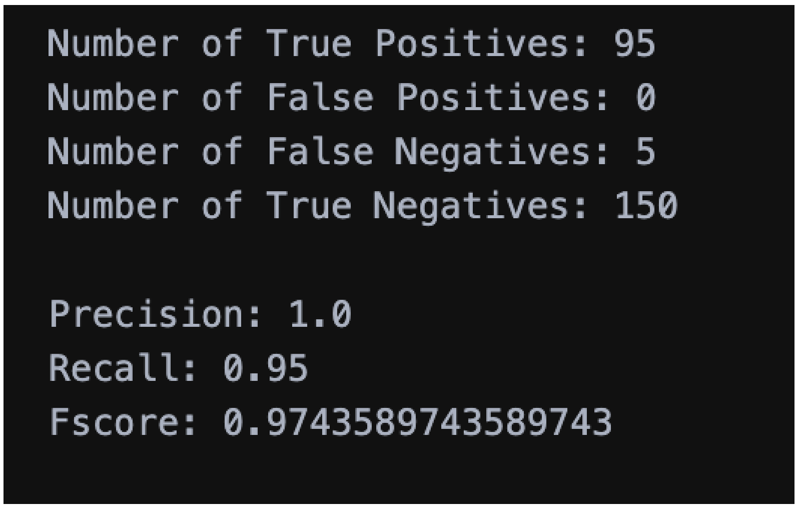
```
class PositionalEncoder:

    def __init__(self, d_model, max_len=100000):
        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log(10000.0) / d_model))
        self.pe = torch.zeros(max_len, d_model)
        self.pe[:, 0::2] = torch.sin(position * div_term)
        self.pe[:, 1::2] = torch.cos(position * div_term)

    def embed(self, x):
        return x + self.pe[:x.size(0)]

def infer(document):
    word_embeddings = [w2vmodel.wv[word] for word in document if word in  w2vmodel.wv]

    if not word_embeddings:
        return np.zeros(20)

    output_embedding = torch.tensor(word_embeddings, dtype=torch.float)
    if len(document) < 100000:
        output_embedding = encoder.embed(output_embedding)

    output_embedding = output_embedding.detach().cpu().numpy()
    return np.mean(output_embedding, axis=0)

encoder = PositionalEncoder(30)
w2vmodel = Word2Vec.load("trained_weights/streamspot/streamspot.model")
```

I made the following change to ensure that my trained model from scratch would be used:



```
def infer(document):
    if Train_Word2vec:                  ## added.
        word_embeddings = [word2vec.wv[word] for word in document if word in  word2vec.wv]
    else:
        word_embeddings = [w2vmodel.wv[word] for word in document if word in  w2vmodel.wv]

    if not word_embeddings:
        return np.zeros(20)
```

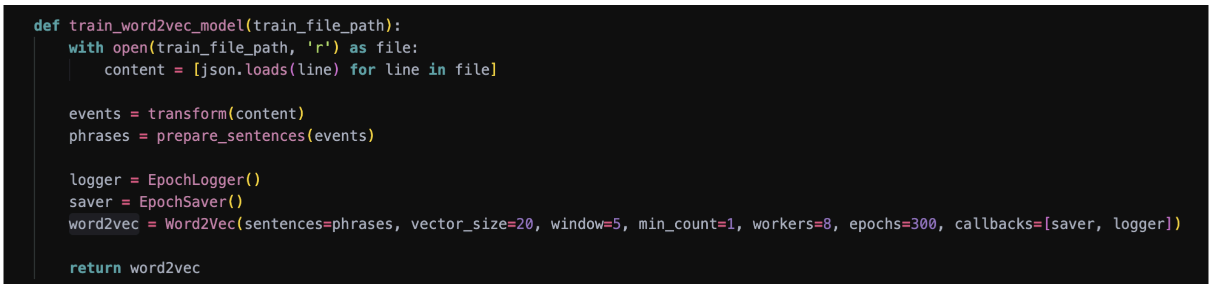
Model from scratch provided the same results as the pre-trained model:



```
Number of True Positives: 95
Number of False Positives: 0
Number of False Negatives: 5
Number of True Negatives: 150

Precision: 1.0
Recall: 0.95
Fscore: 0.9743589743589743
```

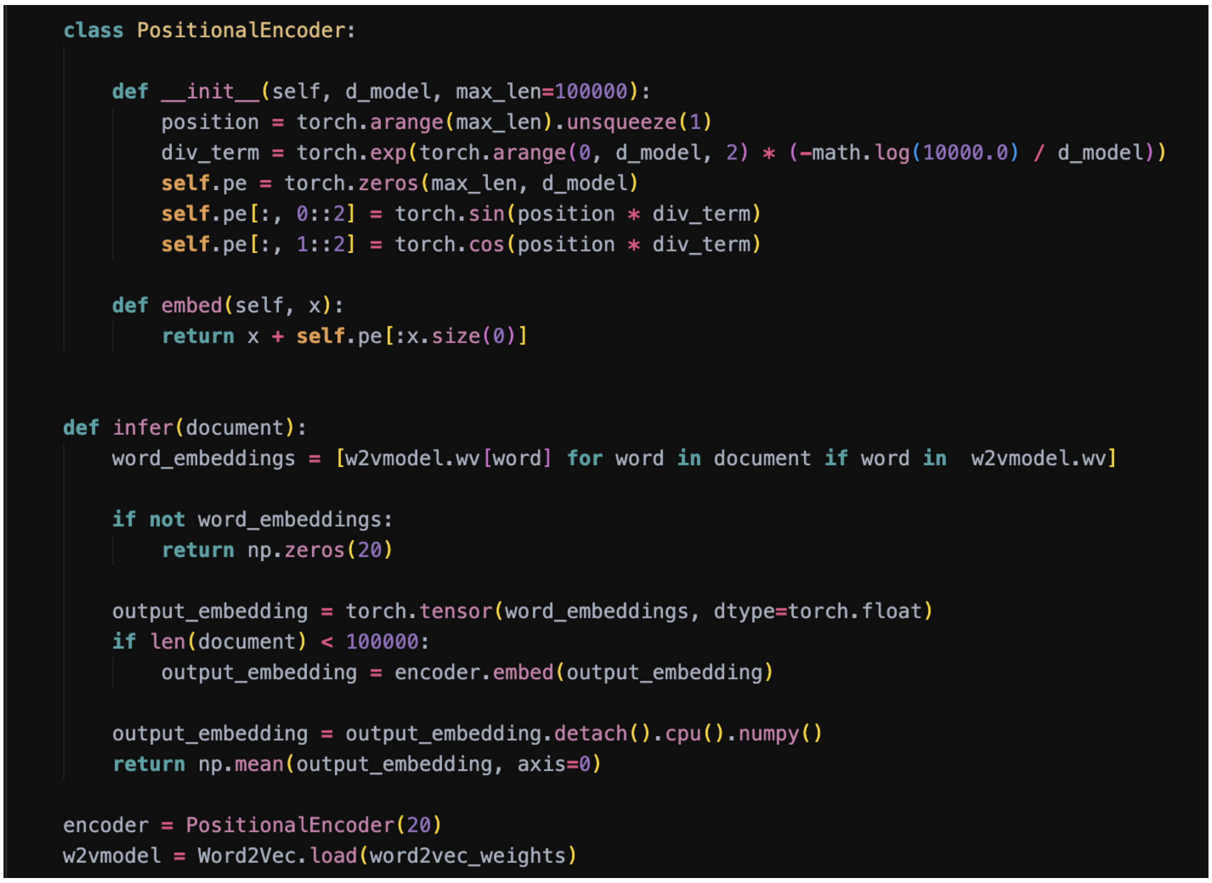
OpTC: Had an error last week with xgboost being used in my environment, but was able to fix this.

Again, a function for training word2vec is included in the script, but never run.



```python
def train_word2vec_model(train_file_path):
    with open(train_file_path, 'r') as file:
        content = [json.loads(line) for line in file]

    events = transform(content)
    phrases = prepare_sentences(events)

    logger = EpochLogger()
    saver = EpochSaver()
    word2vec = Word2Vec(sentences=phrases, vector_size=20, window=5, min_count=1, workers=8, epochs=300, callbacks=[saver, logger])

    return word2vec
```

I did not modify the code this time, as I wasn't sure if modifying it would result in correct functioning (as it is supposed to) throughout the script file.

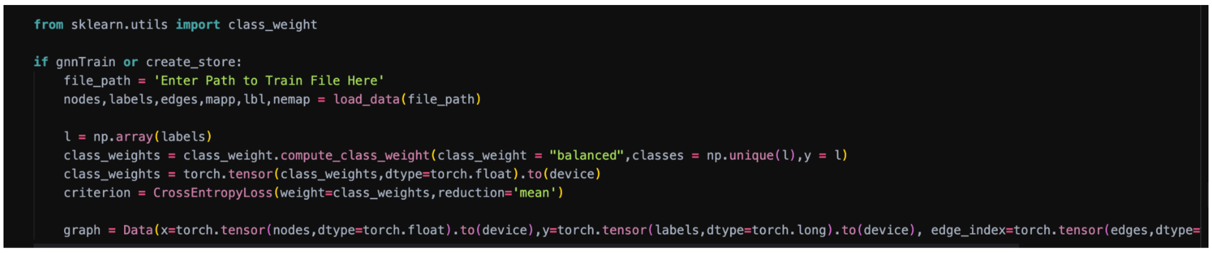


```python
class PositionalEncoder:

    def __init__(self, d_model, max_len=100000):
        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log(10000.0) / d_model))
        self.pe = torch.zeros(max_len, d_model)
        self.pe[:, 0::2] = torch.sin(position * div_term)
        self.pe[:, 1::2] = torch.cos(position * div_term)

    def embed(self, x):
        return x + self.pe[:x.size(0)]


def infer(document):
    word_embeddings = [w2vmodel.wv[word] for word in document if word in  w2vmodel.wv]

    if not word_embeddings:
        return np.zeros(20)

    output_embedding = torch.tensor(word_embeddings, dtype=torch.float)
    if len(document) < 100000:
        output_embedding = encoder.embed(output_embedding)

    output_embedding = output_embedding.detach().cpu().numpy()
    return np.mean(output_embedding, axis=0)

encoder = PositionalEncoder(20)
w2vmodel = Word2Vec.load(word2vec_weights)
```

I was then able to run the code for the provided pre-trained model.

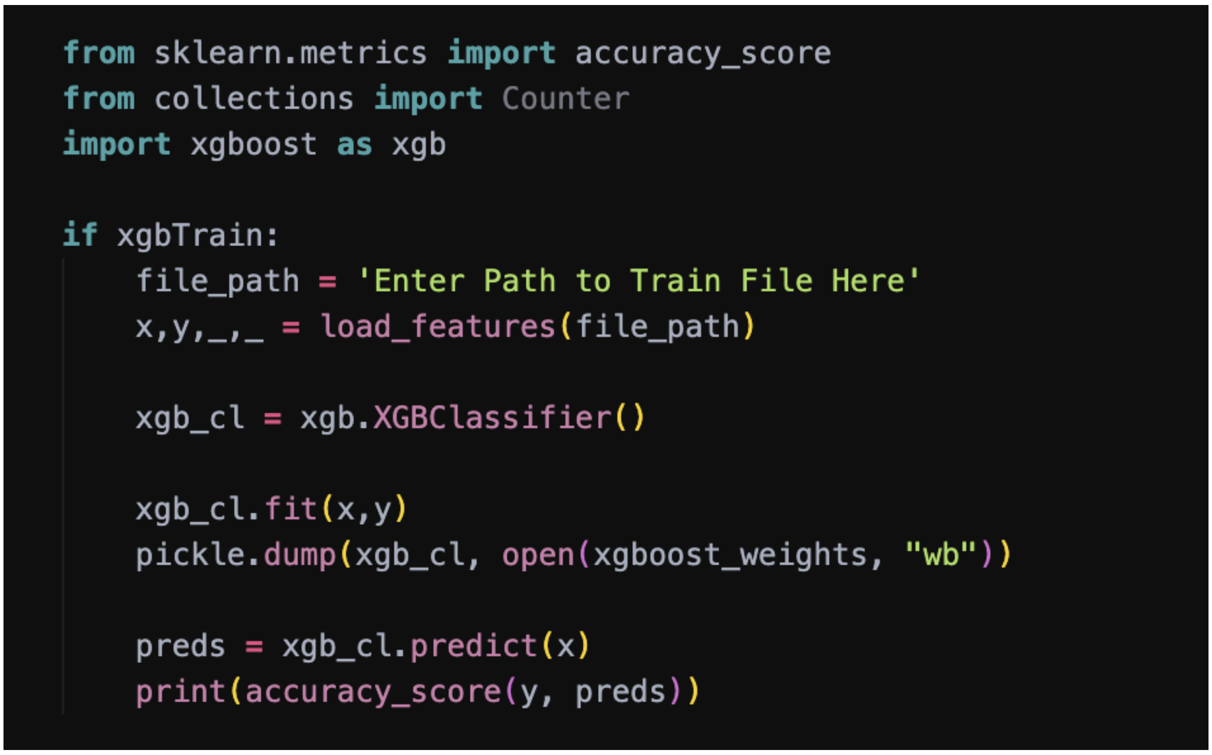
When running the code from scratch, retraining all of the models, there is this line of code that raises an error: ‘file_path = ‘Enter path to train file here’.

Understandably, I would need to add the file path to the train file, but I have no idea what ‘train file’ refers to. There aren’t any instructions regarding what this might be.



```python
from sklearn.utils import class_weight

if gnnTrain or create_store:
    file_path = 'Enter Path to Train File Here'
    nodes,labels,edges,mapp,lbl,nemap = load_data(file_path)

    l = np.array(labels)
    class_weights = class_weight.compute_class_weight(class_weight = "balanced",classes = np.unique(l),y = l)
    class_weights = torch.tensor(class_weights,dtype=torch.float).to(device)
    criterion = CrossEntropyLoss(weight=class_weights,reduction='mean')

    graph = Data(x=torch.tensor(nodes,dtype=torch.float).to(device),y=torch.tensor(labels,dtype=torch.long).to(device), edge_index=torch.tensor(edges,dtype=
```

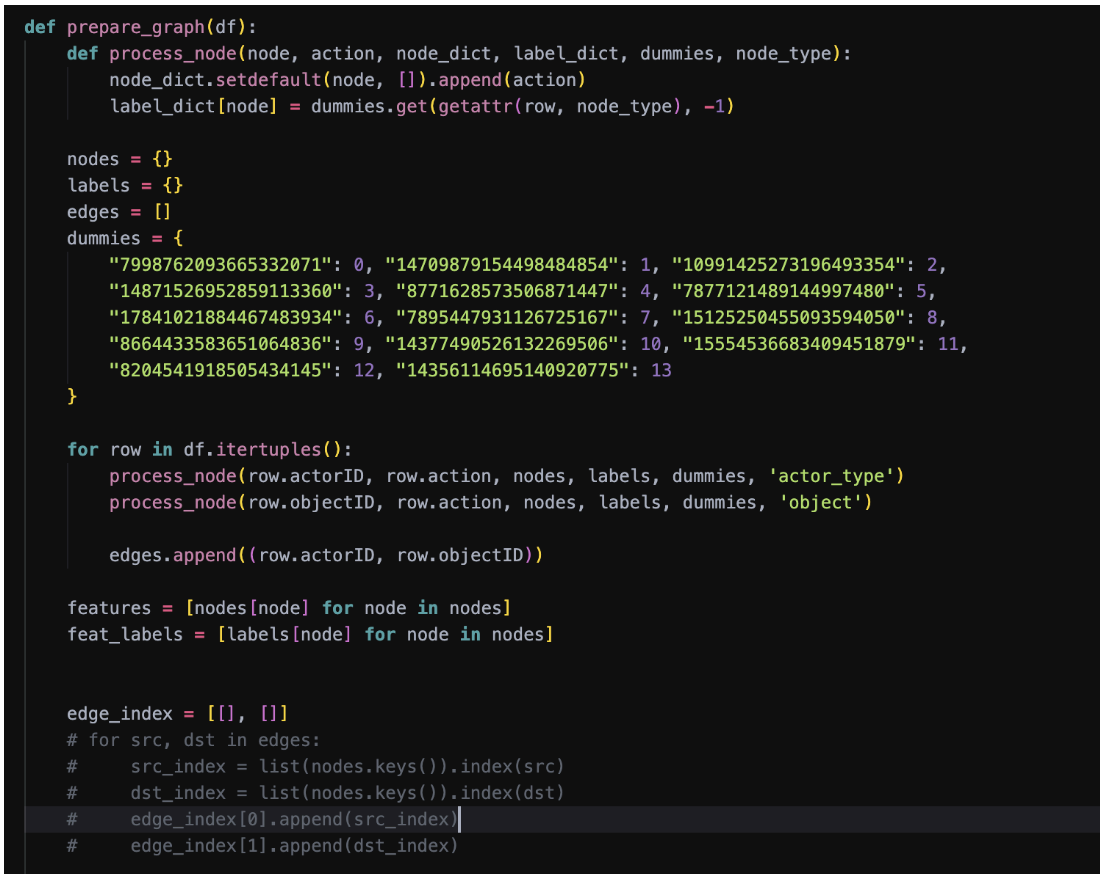
Similar issue later in the code:



```python
from sklearn.metrics import accuracy_score
from collections import Counter
import xgboost as xgb

if xgbTrain:
    file_path = 'Enter Path to Train File Here'
    x,y,_,_ = load_features(file_path)

    xgb_cl = xgb.XGBClassifier()

    xgb_cl.fit(x,y)
    pickle.dump(xgb_cl, open(xgboost_weights, "wb"))

    preds = xgb_cl.predict(x)
    print(accuracy_score(y, preds))
```

Have raised a Github issue regarding this.

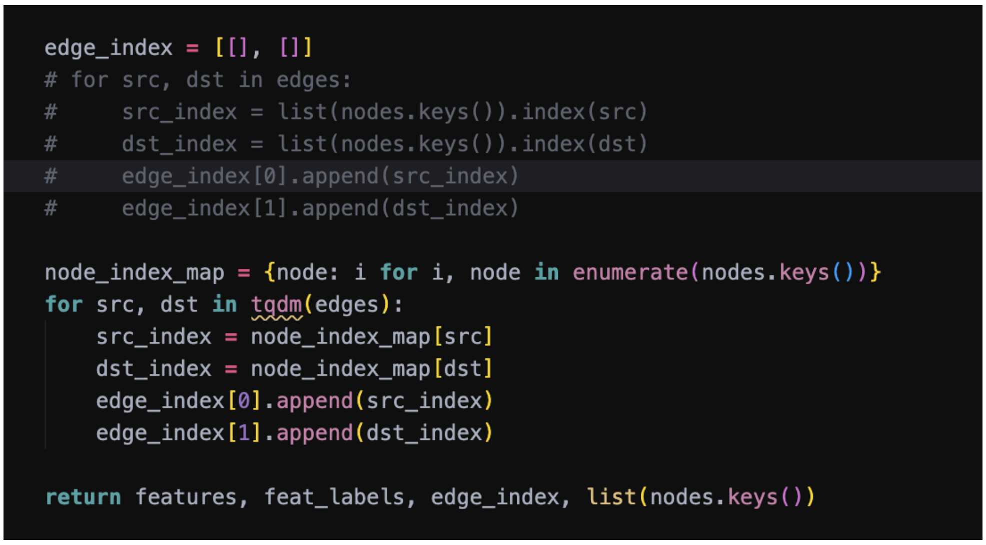
Unicorn: The (scratch) model is still being trained, I have only ran the pre-trained model as of yet. This dataset is taking unusually more time than the others, the reason for which I have pinpointed to this specific function:



```python
def prepare_graph(df):
    def process_node(node, action, node_dict, label_dict, dummies, node_type):
        node_dict.setdefault(node, []).append(action)
        label_dict[node] = dummies.get(getattr(row, node_type), -1)

    nodes = {}
    labels = {}
    edges = []
    dummies = {
        "7998762093665332071": 0, "14709879154498484854": 1, "10991425273196493354": 2,
        "14871526952859113360": 3, "8771628573506871447": 4, "7877121489144997480": 5,
        "17841021884467483934": 6, "7895447931126725167": 7, "15125250455093594050": 8,
        "8664433583651064836": 9, "14377490526132269506": 10, "15554536683409451879": 11,
        "8204541918505434145": 12, "14356114695140920775": 13
    }

    for row in df.itertuples():
        process_node(row.actorID, row.action, nodes, labels, dummies, 'actor_type')
        process_node(row.objectID, row.action, nodes, labels, dummies, 'object')

        edges.append((row.actorID, row.objectID))

    features = [nodes[node] for node in nodes]
    feat_labels = [labels[node] for node in nodes]


    edge_index = [[], []]
    # for src, dst in edges:
    #     src_index = list(nodes.keys()).index(src)
    #     dst_index = list(nodes.keys()).index(dst)
    #     edge_index[0].append(src_index)
    #     edge_index[1].append(dst_index)
```

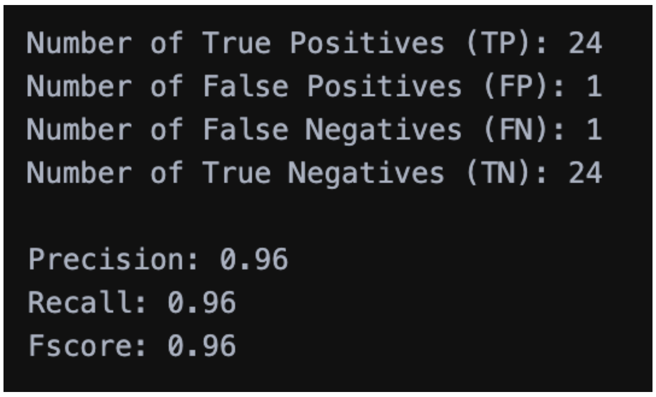
The function had been running for over 5 hours, until I interrupted it. I saw that a Github issue had been created regarding this and someone had suggested an edit that produced the same results but in much less time.

I have made the edits for now and am trying to see if I am able to reproduce the same result.

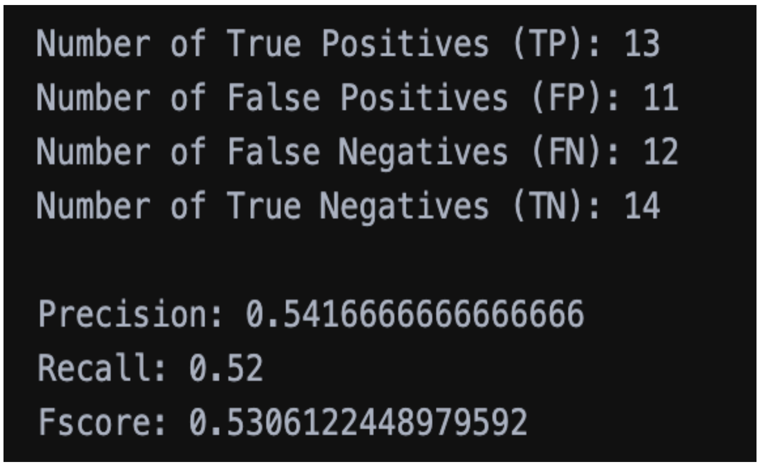


```python
edge_index = [[], []]
# for src, dst in edges:
#     src_index = list(nodes.keys()).index(src)
#     dst_index = list(nodes.keys()).index(dst)
#     edge_index[0].append(src_index)
#     edge_index[1].append(dst_index)

node_index_map = {node: i for i, node in enumerate(nodes.keys())}
for src, dst in tqdm(edges):
    src_index = node_index_map[src]
    dst_index = node_index_map[dst]
    edge_index[0].append(src_index)
    edge_index[1].append(dst_index)

return features, feat_labels, edge_index, list(nodes.keys())
```

Comparison of results: Left: Pretrained Model, Right: My run from scratch.



```
Number of True Positives (TP): 24
Number of False Positives (FP): 1
Number of False Negatives (FN): 1
Number of True Negatives (TN): 24

Precision: 0.96
Recall: 0.96
Fscore: 0.96
```



```
Number of True Positives (TP): 13
Number of False Positives (FP): 11
Number of False Negatives (FN): 12
Number of True Negatives (TN): 14

Precision: 0.5416666666666666
Recall: 0.52
Fscore: 0.5306122448979592
```

However, the results on the right are after I made the edits in the code. It might be possible that the results match if I run the original code but the run takes a lot of time. Moreover, I do not think the edits suggested *do anything different* from what the original code intends to do.

AirTag:

- SDatasets and MDatasets were already reproduced.
- This week, I had to work on figuring out if I could reproduce the results for the UDatasets and DepImpact Datasets.
- The issue was that there were no UDataset/DI Datasets output logs present to verify reproduction results against.
- Or as Talha worded it: The preprocessed, AirTag-tokenized version of UDatasets are missing which are required for the false positive filtering step in the evaluation file.
- Initial plan was to check the provided pre-trained models. If there was one for the UDatasets, I could use that to generate the ‘ground truth’ for the UDatasets.
- **However**, the pre-trained models provided did not have one for the UDatasets, or the DEPIMPACT datasets.
- Will need to raise a Github issue/email, will coordinate with Talha on this.