Yash Avinash Patole

CWID: 10460520

CS-541 Artificial Intelligence
Mini Project

Title:
Environment Setup

Under the Guidance of:

Prof. Abdul Rafae Khan

Stevens Institute of Technology

<u>Title :</u> The Logical Expressiveness of Graph Neural Networks.

Authors: Pablo Barcelo´ (IMC, PUC & IMFD Chile), Egor V. Kostylev (University of Oxford), Mikael Monet ¨ (IMFD Chile), Jorge Perez ´ DCC, UChile & IMFD Chile Juan Reutter DCC, PUC & IMFD Chile Juan-Pablo Silva DCC, UChile

Name: Yash A. Patole

StevensId: ypatole@stevens.edu

CourseName: CS 541 Artificial Intelligence

<u>Abstract:</u> In this paper it states that the capacity of graph neural networks (GNNs) to differentiate nodes in graphs has recently been measured using the Weisfeiler-Lehman (WL) graph isomorphism test. However, this definition does not resolve the question of which Boolean node classifiers may be described by GNNs. In this paper it tackles the problem by using various formulas such as the logic of FOC2, a well-studied first-order logic fragment. FOC2 is linked to the WL test and, as a result, to GNNs. They have focused on explaining the class of GNNs where this class shows and updates the features of each node in the graph. But here they show that this class of GNNs is too weak to capture all FOC2 classifiers, and we give a syntactic description of the greatest subclass of FOC2 classifiers that AC-GNNs can capture. This subclass corresponds to a widely used logic in the knowledge representation community. The paper then shows demonstrate on synthetic data complying to FOC2 formulae, AC-GNNs fail to fit the training data, whereas ACR-GNNs can generalize even to graphs of sizes not encountered during training.

This means that ACGNNs aren't powerful enough to capture all FOC2 formulations. Then we look for what needs to be added to AC-GNNs so that all FOC2 can be captured. I have demonstrated that adding readout layers is sufficient for updating node features not just in terms of its neighbors, but also in terms of a global attribute vector. ACR-GNNs are GNNs with readouts. The final results experiments propose that support the findings of AC-GNNs which fail to fit in on synthetic data conforming to FOC2 but not to graded modal logic, but ACR-GNNs can generalize to graphs of sizes not observed during training.

The following is the main consideration of the paper:

1) We describe in detail the FOC2 formula fragment that can be expressed as ACGNNs. This section belongs to graded modal logic (de Rijke, 2000), or, to put it another way, the ALCQ description logic, which has sparked a lot of attention in the field of knowledge representation (Baader et al., 2003; Baader & Lutz, 2007).
2) Next, we extend the AC-GNN architecture by allowing global readouts, in which each layer computes a feature vector for the entire graph and combines it with local aggregations; these aggregate-combine-readout GNNs are called aggregate-combine-readout GNNs (ACR-GNNs). This proposal is for relational reasoning over graph representations is a subset of these networks. We show that an ACR-GNN can capture each FOC2 in this scenario.

GitHub link of the research paper: https://github.com/juanpablos/GNN-logic

Introduction:

Graph neural networks are a sort of neural network that has recently acquired popularity for a wide range of structured data applications, such as molecular categorization, knowledge graph completion, and Web page ranking. The authors' major goal in writing this study is to experiment with and comprehend the connections between neurons that aren't random but nevertheless have an impact on the structure of the incoming data.

The authors of these papers independently observe that the node labeling produced by the WL test always refines the labeling produced by any GNN. More precisely, if two nodes are labeled the same by the algorithm underlying the WL test, then the feature vectors of these nodes produced by any AC-GNN will always be the same. Answering this question, we recommend focusing on logical classifiers—that is, unary formulae expressible in first order predicate logic (FO): such a formula categorizes each node v based on whether the formula holds for v or not. This focus allows us to connect GNNs to declarative and well-understood formalisms, as well as derive inferences about GNNs based on a large body of logic research. For instance, if two GNN architectures are captured by two logics, all knowledge about the links between those logics, such as equivalence or incomparability of expressiveness, can be transferred to the GNN setting instantaneously.

The expressivity of graph neural networks is discussed in depth in this paper (GNNs). The authors show that the expressivity (aggregate and combine) of AC-GNNs can only express logical classifiers that can be articulated in graded modal logic. ACR-GNNs (aggregate, combine, and readout) can capture FOC2 (logical classifiers expressed with two variables and counting quantifiers) by adding readouts. The second theorem leaves open the possibility of ACR-GNNs capturing logical classifiers other than FOC2.

The AC-GNN architecture is simplified by enabling global readouts, in which each layer computes a feature vector for the entire graph and combines it with local aggregations; these aggregate-combine-readout GNNs, or ACR-GNNs, are called aggregate-combine-readout GNNs. These networks are a variant of Battaglia et al's networks for relational reasoning over graph representations. In this context, the paper show that an ACR-GNN can capture each FOC2 formula also show that an ACR-GNN can capture each FOC2 formula with a single readout.

Finally, in the experiment I demonstrate that when we learn from instances, the theoretical expressiveness of ACR-GNNs, as well as the differences between AC-GNNs and ACR-GNNs, can be noticed. The results that AC-GNNs fail to match the training data on synthetic graph data adhering to FOC2 formulae, whereas ACR-GNNs can generalize to graphs of sizes not observed during training.

Data:

The graphs used in the paper are in the zip-file datasets.zip. Just unzip them to src/data/datasets.

The script expects 3 folders inside src/data/datasets named p1, p2 and p3. These folders contain the datasets for the properties α_1, α_2 and α_3 described in the appendix F on data for the experiment with classifier $\alpha_i(x)$ in equation.

In the code there these datasets are accepted parsed and there is a small description of the arguments in generate_dataset.

While there are .txt in file in each p1, p2 and p3 which has test and train data respectively. Size of the data are as follows:

P1: 1) size of train data is 230087. 2) size of test data is 51377.

P2: 1) size of train data is 230232. 2) size of test data is 51198.

P3: 1) size of train data is 229788. 2) size of test data is 51239.

Link to the dataset: https://github.com/juanpablos/GNN-logic

We can find the link here and also I will attach the zip file of the dataset with this submission.

Tools & Technologies: The details of the software/library packages tqdm==4.35.0

scipy==1.3.1

numpy==1.17.1

scikit-learn==0.21.3

matplotlib==3.1.1

networkx==2.3

torch==1.2.0

torch-cluster==1.4.4

torch-scatter==1.3.1

torch-sparse==0.4.0

torch-spline-conv==1.1.0

torch-geometric==1.3.1

requests==2.22.0

For this project we will use Google Colab and it's technologies.

Following links will be required to save:

1) /content/drive/MyDrive/GNN-logic/src/logging/results/p1-0-0-acgnn-aggA-readA-combT-cl0-L1.log
2) /content/drive/MyDrive/GNN-logic/src/logging/results/p1-0-0-acgnn-aggA-readA-combT-cl0-L1.log.train
3) /content/drive/MyDrive/GNN-logic/src/logging/results/p1-0-0-acgnn-aggA-readA-combT-cl0-L1.log.test

Hardware Requirements: This project requires GPU (cuda and other torch packages). My GPU is Nvidia RTX3080 and processor is AMD Ryzen 5000.

Data:

P1	**230087**	**51377**
P2	**230232**	**51198**
P3	**229788**	**51239**

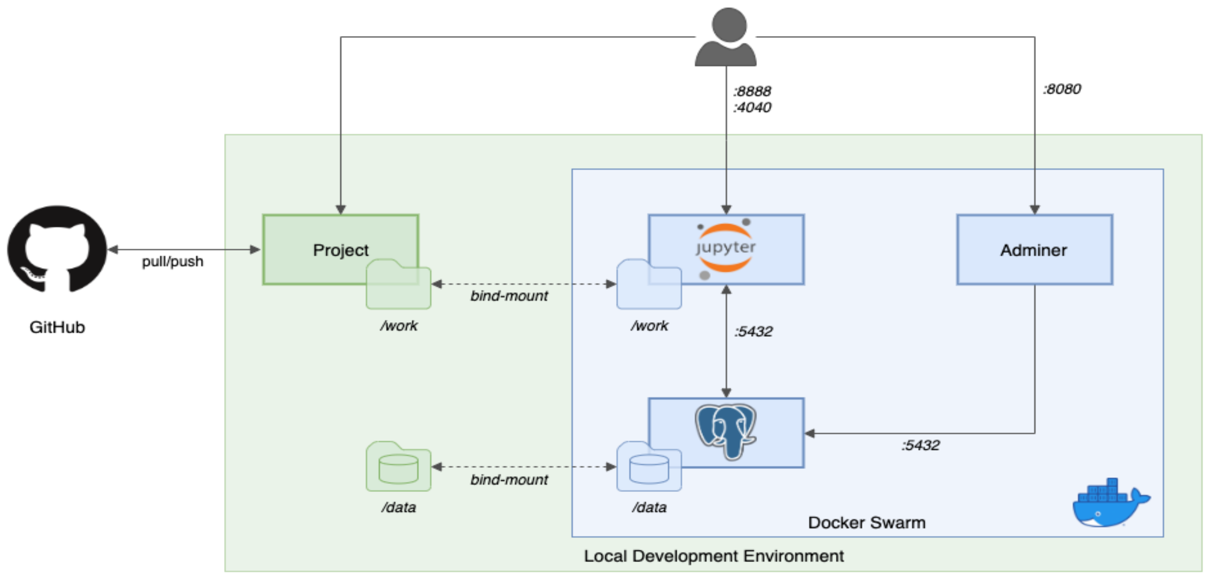
For data Pre-Processing this project uses two files named argparser_real_data.py and datasets.py to parse and get the inputs ready for processing epochs.

Experiments: The code helps to get logical expressiveness of graph neural networks. In this code it considers the simple FOC2 formula: (x):= Red(x) y Every red node in a graph must satisfy the condition Blue(y) if the graph contains at least one blue node. The code first considers half of the graphs in each set (training and testing) containing no blue nodes, and 50% containing at least one blue node (about 20% of nodes are blue every set has a true class). As my initial test I will provide following training data outcomes which explains the learning rate, epochs of the our training dataset.

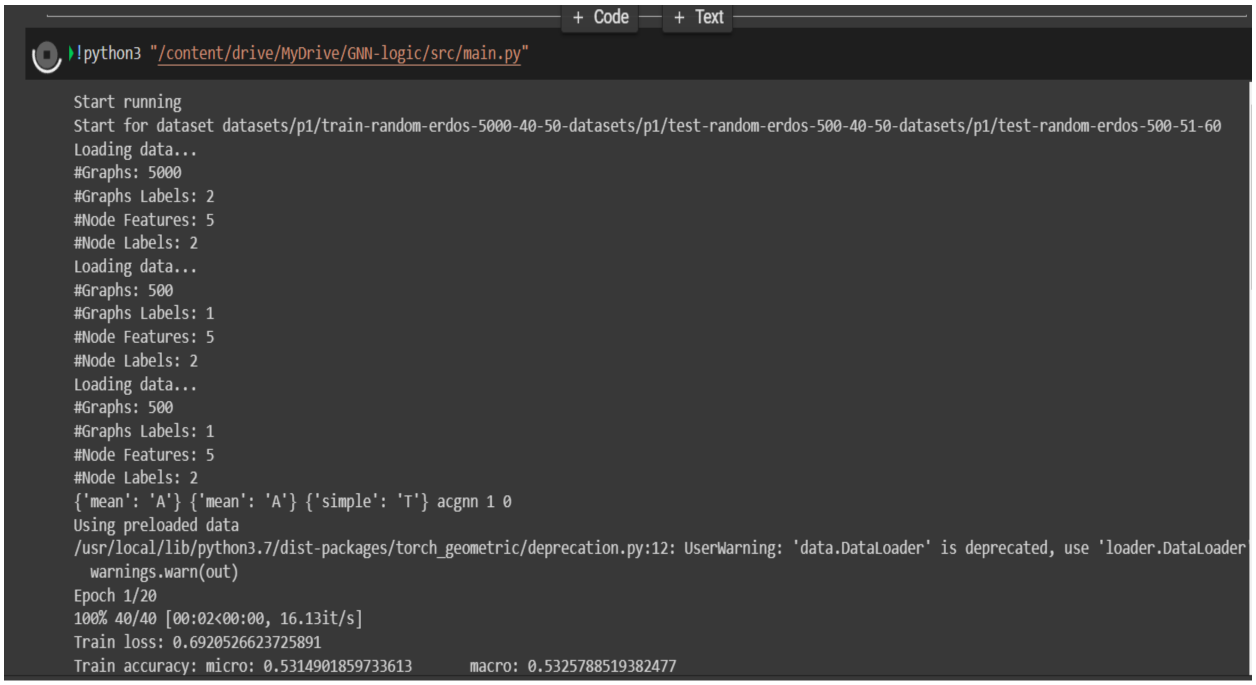
train_loss,test1_loss,test2_loss,train_micro,train_macro,test1_micro,test1_macro,test2_micro,test2_macro

0.6901471019, 0.6904880404, 0.6916720867, 0.53859858, 0.53905009, 0.53607746, 0.53637073, 0.52487661, 0.52607605

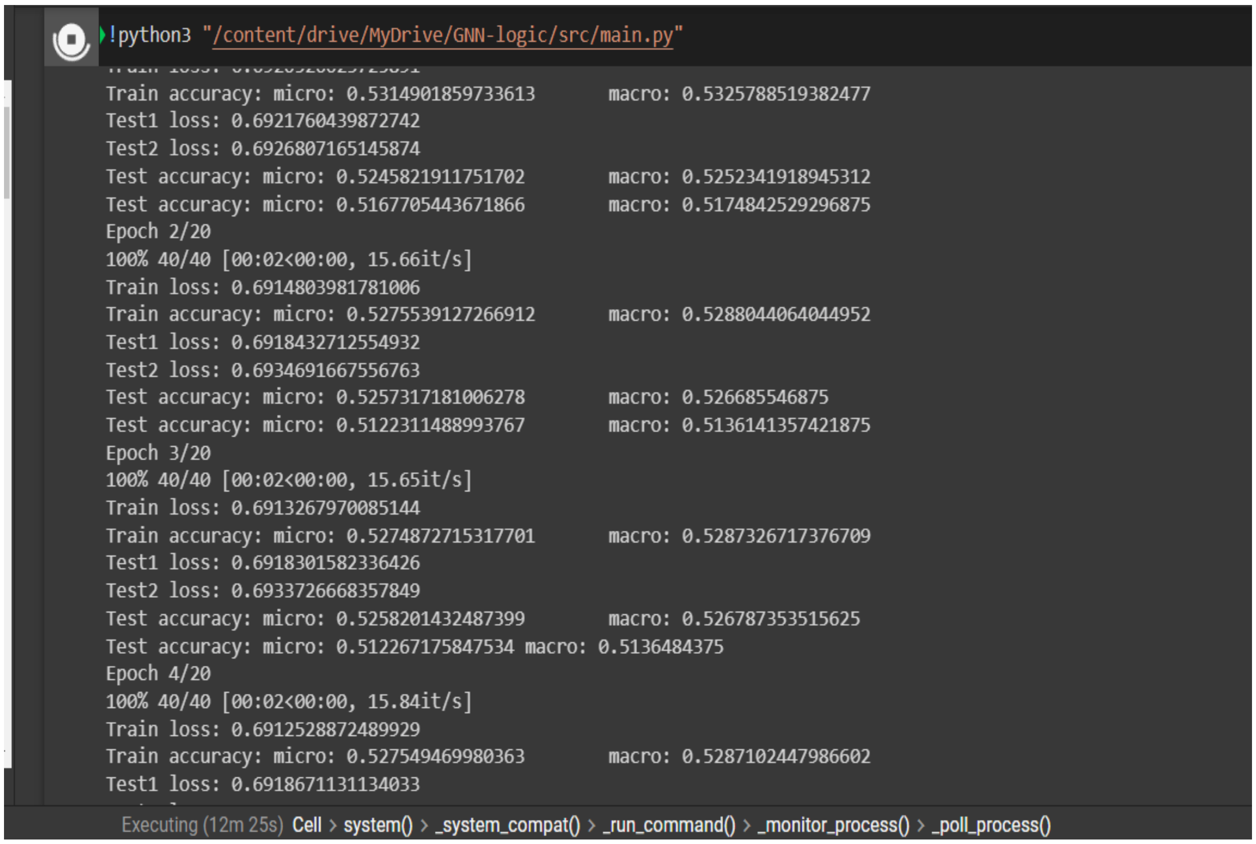
Following is similar network architecture that will be required where instead of postgres in this project I am using google colab and jupyter lab to get this project onboarded and modules are installed respectively.



Results: Following are preliminary results of this project that I managed to execute on my computer successfully. Following outputs contains 20 epochs for dataset P1. More testing, including datasets P2 and P3 are needed to be done.

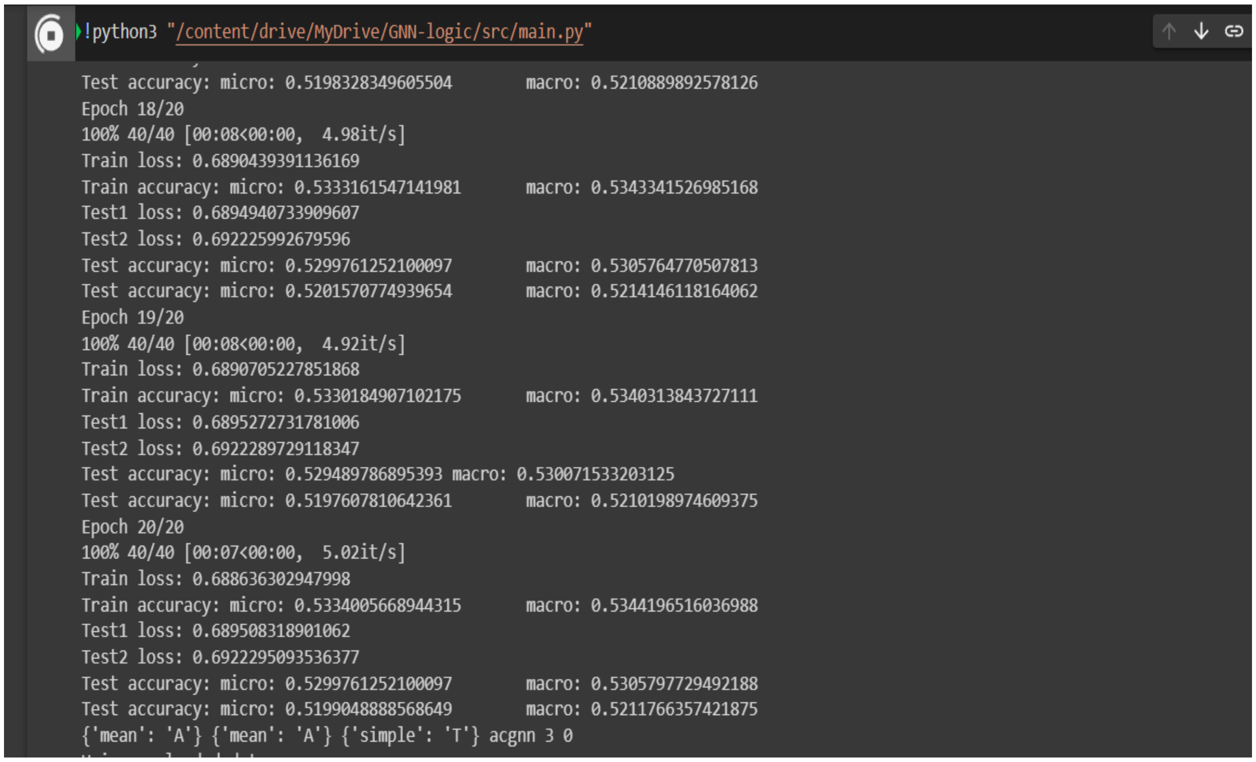


```
!python3 "/content/drive/MyDrive/GNN-logic/src/main.py"

Start running
Start for dataset datasets/p1/train-random-erdos-5000-40-50-datasets/p1/test-random-erdos-500-40-50-datasets/p1/test-random-erdos-500-51-60
Loading data...
#Graphs: 5000
#Graphs Labels: 2
#Node Features: 5
#Node Labels: 2
Loading data...
#Graphs: 500
#Graphs Labels: 1
#Node Features: 5
#Node Labels: 2
Loading data...
#Graphs: 500
#Graphs Labels: 1
#Node Features: 5
#Node Labels: 2
{'mean': 'A'} {'mean': 'A'} {'simple': 'T'} acgnn 1 0
Using preloaded data
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use 'loader.DataLoader'
  warnings.warn(out)
Epoch 1/20
100% 40/40 [00:02<00:00, 16.13it/s]
Train loss: 0.6920526623725891
Train accuracy: micro: 0.5314901859733613       macro: 0.5325788519382477
```



```
!python3 "/content/drive/MyDrive/GNN-logic/src/main.py"

Train accuracy: micro: 0.5314901859733613       macro: 0.5325788519382477
Test1 loss: 0.6921760439872742
Test2 loss: 0.6926807165145874
Test accuracy: micro: 0.5245821911751702        macro: 0.5252341918945312
Test accuracy: micro: 0.5167705443671866        macro: 0.5174842529296875
Epoch 2/20
100% 40/40 [00:02<00:00, 15.66it/s]
Train loss: 0.6914803981781006
Train accuracy: micro: 0.5275539127266912       macro: 0.5288044064044952
Test1 loss: 0.6918432712554932
Test2 loss: 0.6934691667556763
Test accuracy: micro: 0.5257317181006278        macro: 0.526685546875
Test accuracy: micro: 0.5122311488993767        macro: 0.5136141357421875
Epoch 3/20
100% 40/40 [00:02<00:00, 15.65it/s]
Train loss: 0.6913267970085144
Train accuracy: micro: 0.5274872715317701       macro: 0.5287326717376709
Test1 loss: 0.6918301582336426
Test2 loss: 0.6933726668357849
Test accuracy: micro: 0.5258201432487399        macro: 0.526787353515625
Test accuracy: micro: 0.512267175847534 macro: 0.5136484375
Epoch 4/20
100% 40/40 [00:02<00:00, 15.84it/s]
Train loss: 0.6912528872489929
Train accuracy: micro: 0.527549469980363        macro: 0.5287102447986602
Test1 loss: 0.6918671131134033
```

Executing (12m 25s) Cell > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()



```
!python3 "/content/drive/MyDrive/GNN-logic/src/main.py"
Test accuracy: micro: 0.5198328349605504       macro: 0.5210889892578126
Epoch 18/20
100% 40/40 [00:08<00:00,  4.98it/s]
Train loss: 0.6890439391136169
Train accuracy: micro: 0.5333161547141981      macro: 0.5343341526985168
Test1 loss: 0.6894940733909607
Test2 loss: 0.692225992679596
Test accuracy: micro: 0.5299761252100097       macro: 0.5305764770507813
Test accuracy: micro: 0.5201570774939654       macro: 0.5214146118164062
Epoch 19/20
100% 40/40 [00:08<00:00,  4.92it/s]
Train loss: 0.6890705227851868
Train accuracy: micro: 0.5330184907102175      macro: 0.5340313843727111
Test1 loss: 0.6895272731781006
Test2 loss: 0.6922289729118347
Test accuracy: micro: 0.529489786895393 macro: 0.530071533203125
Test accuracy: micro: 0.5197607810642361       macro: 0.5210198974609375
Epoch 20/20
100% 40/40 [00:07<00:00,  5.02it/s]
Train loss: 0.688636302947998
Train accuracy: micro: 0.5334005668944315      macro: 0.5344196516036988
Test1 loss: 0.689508318901062
Test2 loss: 0.6922295093536377
Test accuracy: micro: 0.5299761252100097       macro: 0.5305797729492188
Test accuracy: micro: 0.5199048888568649       macro: 0.5211766357421875
{'mean': 'A'} {'mean': 'A'} {'simple': 'T'} acgnn 3 0
```

Experiments & Results

Method:

The method used in this paper is encodings node colors in the graph, from a finite set of colors. While using neighborhood NG(v) of a node where v ∈ V is the set {u | {v, u} ∈ E}. GNNs can be classified using a variety of aggregate, combination, and classification functions (Hamilton et al., 2017; Kipf & Welling, 2017; Morris et al., 2019; Xu et al., 2019). Consider the sum of the feature vectors as the aggregation, which is a basic yet frequent choice.

function, as well as a combined function

f = (x1, x2)

x1C

x2A(i) + b = I + x2A(i)

(2) where C stands for (i) as well as (i) are parameter matrices, If is a non-linearity function, such as relu or sigmoid, and is a bias vector. Using these functions, we call it an AC-GNN. In addition, if all AGG are present in an AC-GNN, it is said to be homogenous (i) all identical, and they're all COM (i) are the same (common) the same settings are used in each tier.

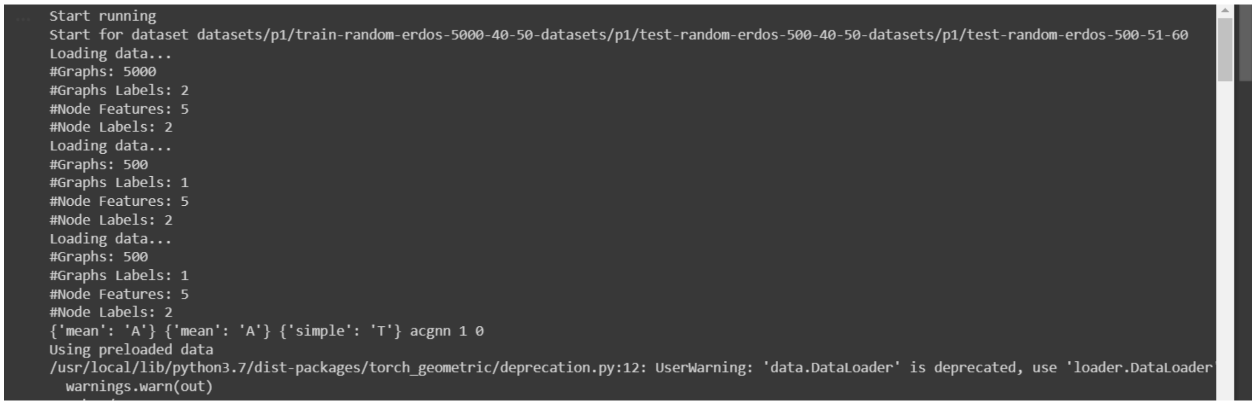
This formula has two quantified variables, y and z, and one free variable, x, that is not constrained by any quantifier of the form or. Formulas with one free variable are evaluated over the nodes of a network in general. For example, in nodes v with the hue Red with both a Blue and a Green neighbor, the above algorithm evaluates to true exactly. In this scenario, we state that G's node v satisfies and mark this with (G, v) |=α.

The proof of Theorem 5.1 mentioned in the paper creates GNNs whose number of layers is determined by the formula being captured—readout functions are utilized an infinite number of times in ACR-GNNs to capture various FOC2 classifiers. Given how expensive a global calculation can be, one can wonder if it's truly necessary, or if the complexity of such classifiers can be handled with only a few readouts.

Experiments:

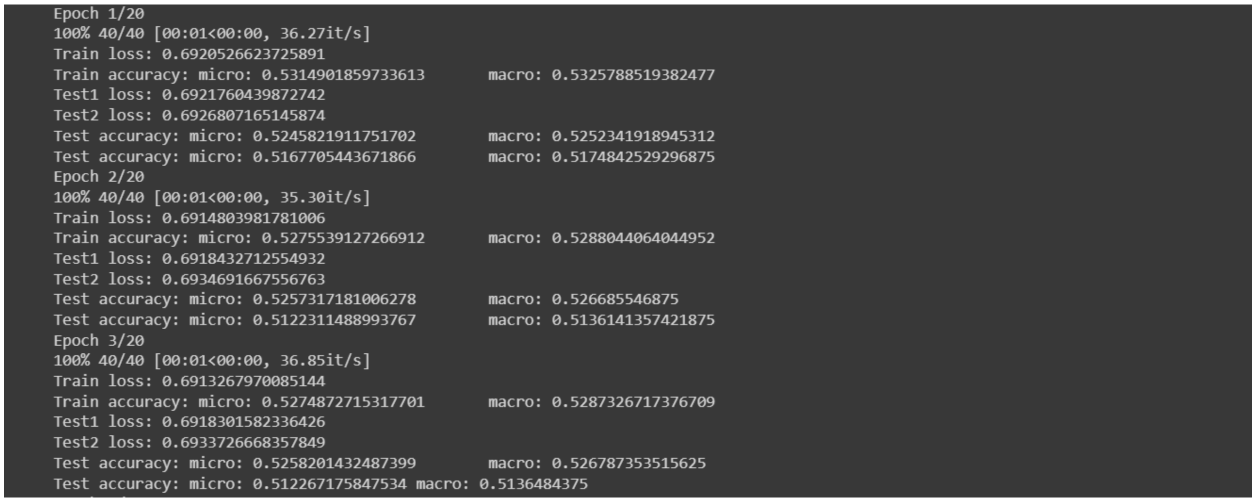
Considering the following simple FOC2 formula: (x):= Red(x) y Every red node in a graph must satisfy the condition Blue(y) if the graph contains at least one blue node. We used line-shaped graphs and Erdos-Renyi (E-R) random graphs with various connectivity to test our hypothesis. In each set (train and test), we consider 50 percent of graphs with no blue nodes and 50 percent with at least one blue node (about 20% of nodes in each set are in the true class). Single-layer ACR-GNNs have already demonstrated excellent performance for both types of graphs (ACR-1 in Table 1). Given the simplicity of the property being examined, this was to be anticipated. AC-GNNs and GINs, on the other hand (represented in Table 1 as AC-L and GINL, respectively, indicating AC-GNNs and GINs) also, GINs with L layers) have a hard time fitting the data. In the instance of the even with 7 layers, they were unable to fit the train data into a line-shaped graph. In this scenario, the performance with 7 layers was significantly better than that of random graphs. A closer examination of the experiment interprets discovered that AC-GNNs performed better for varied connectivity of E-R graphs when we use more dense graphs to educate them.

Following experiments were conducted in three different datasets provided in the repository.



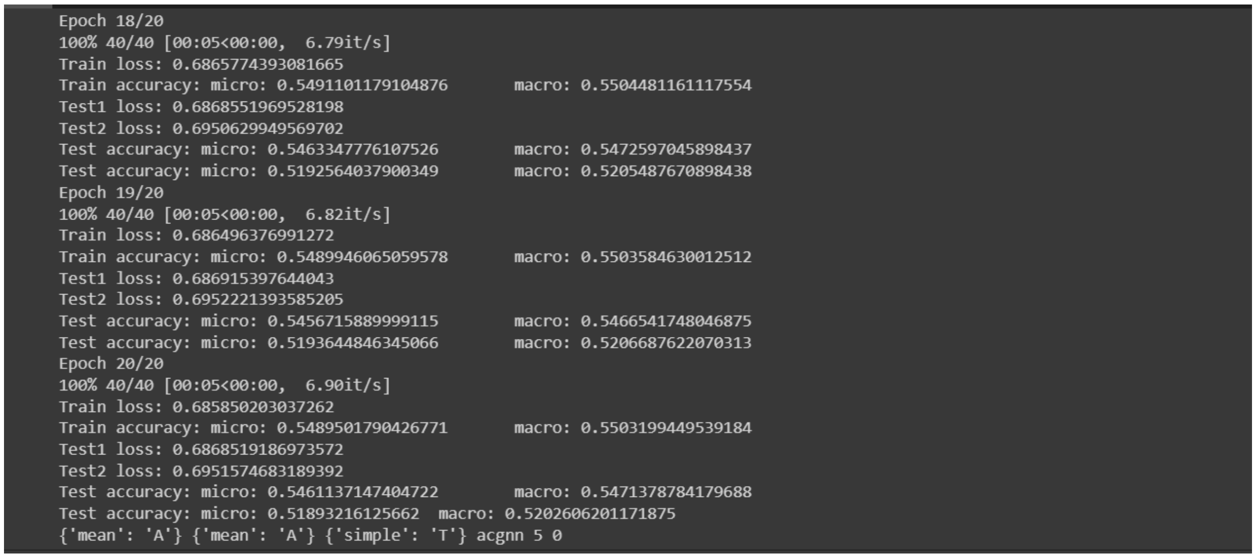
```
Start running
Start for dataset datasets/p1/train-random-erdos-5000-40-50-datasets/p1/test-random-erdos-500-40-50-datasets/p1/test-random-erdos-500-51-60
Loading data...
#Graphs: 5000
#Graphs Labels: 2
#Node Features: 5
#Node Labels: 2
Loading data...
#Graphs: 500
#Graphs Labels: 1
#Node Features: 5
#Node Labels: 2
Loading data...
#Graphs: 500
#Graphs Labels: 1
#Node Features: 5
#Node Labels: 2
{'mean': 'A'} {'mean': 'A'} {'simple': 'T'} acgnn 1 0
Using preloaded data
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use 'loader.DataLoader'
  warnings.warn(out)
```

Epoch for P1 dataset:



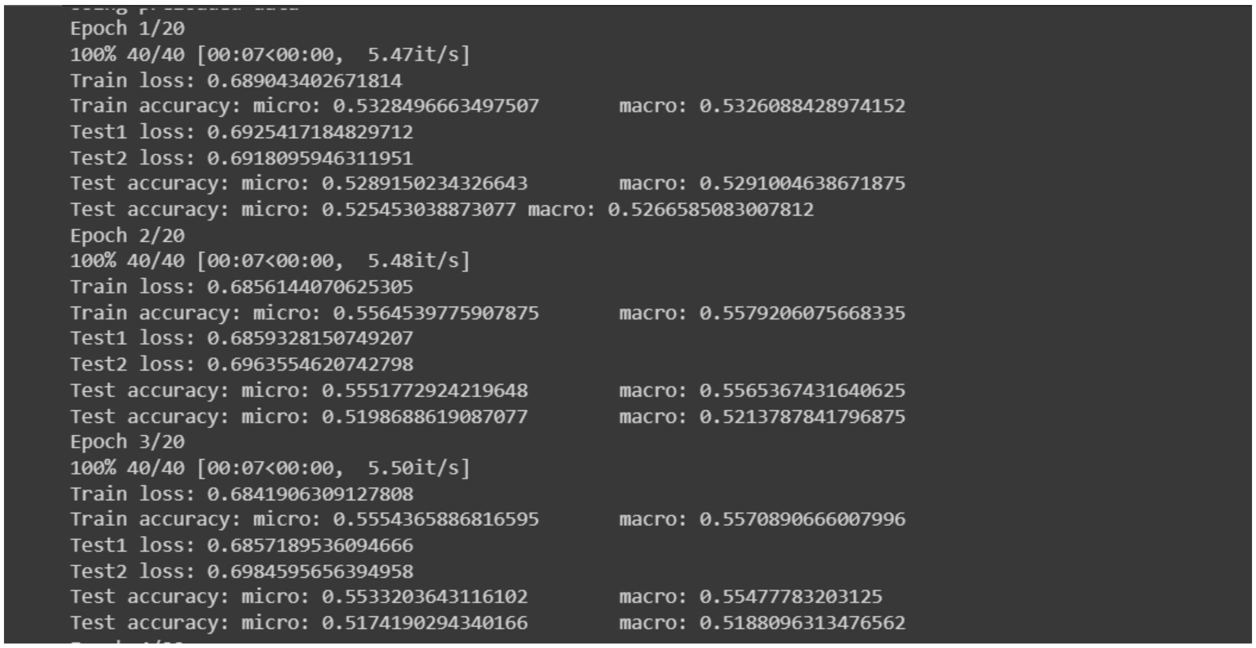
```
Epoch 1/20
100% 40/40 [00:01<00:00, 36.27it/s]
Train loss: 0.6920526623725891
Train accuracy: micro: 0.5314901859733613       macro: 0.5325788519382477
Test1 loss: 0.6921760439872742
Test2 loss: 0.6926807165145874
Test accuracy: micro: 0.5245821911751702        macro: 0.5252341918945312
Test accuracy: micro: 0.5167705443671866        macro: 0.5174842529296875
Epoch 2/20
100% 40/40 [00:01<00:00, 35.30it/s]
Train loss: 0.6914803981781006
Train accuracy: micro: 0.5275539127266912       macro: 0.5288044064044952
Test1 loss: 0.6918432712554932
Test2 loss: 0.6934691667556763
Test accuracy: micro: 0.5257317181006278        macro: 0.526685546875
Test accuracy: micro: 0.5122311488993767        macro: 0.5136141357421875
Epoch 3/20
100% 40/40 [00:01<00:00, 36.85it/s]
Train loss: 0.6913267970085144
Train accuracy: micro: 0.5274872715317701       macro: 0.5287326717376709
Test1 loss: 0.6918301582336426
Test2 loss: 0.6933726668357849
Test accuracy: micro: 0.5258201432487399        macro: 0.526787353515625
Test accuracy: micro: 0.512267175847534 macro: 0.5136484375
```

Final output after 20 iterations:



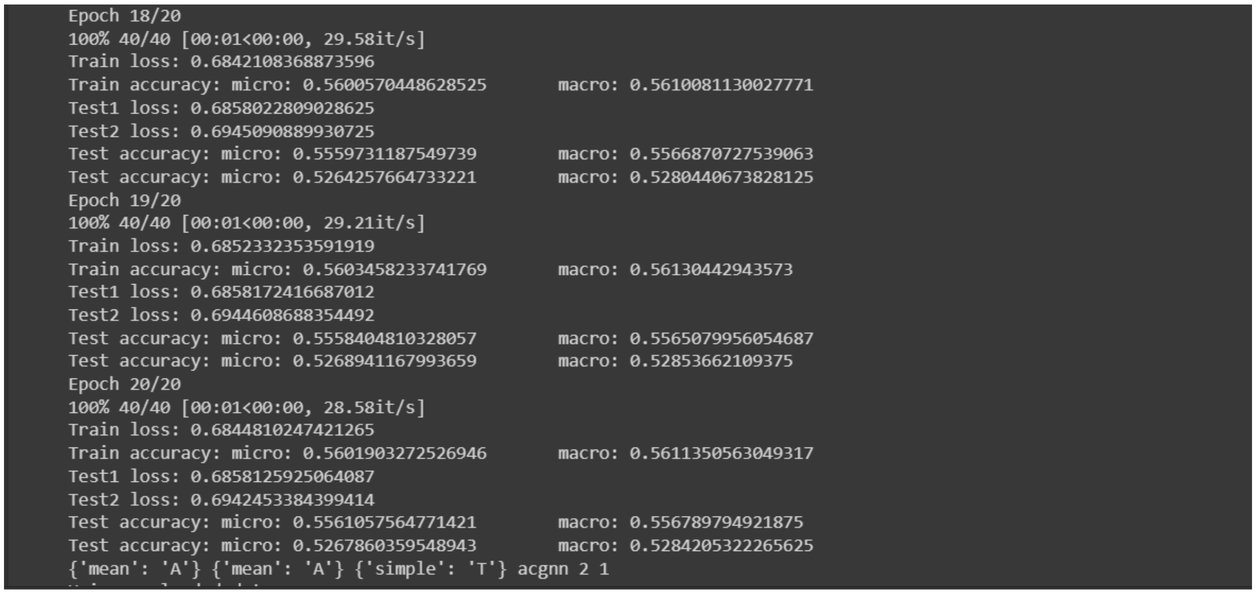
```
Epoch 18/20
100% 40/40 [00:05<00:00,  6.79it/s]
Train loss: 0.6865774393081665
Train accuracy: micro: 0.5491101179104876       macro: 0.5504481161117554
Test1 loss: 0.6868551969528198
Test2 loss: 0.6950629949569702
Test accuracy: micro: 0.5463347776107526        macro: 0.5472597045898437
Test accuracy: micro: 0.5192564037900349        macro: 0.5205487670898438
Epoch 19/20
100% 40/40 [00:05<00:00,  6.82it/s]
Train loss: 0.686496376991272
Train accuracy: micro: 0.5489946065059578       macro: 0.5503584630012512
Test1 loss: 0.686915397644043
Test2 loss: 0.6952221393585205
Test accuracy: micro: 0.5456715889999115        macro: 0.5466541748046875
Test accuracy: micro: 0.5193644846345066        macro: 0.5206687622070313
Epoch 20/20
100% 40/40 [00:05<00:00,  6.90it/s]
Train loss: 0.685850203037262
Train accuracy: micro: 0.5489501790426771       macro: 0.5503199449539184
Test1 loss: 0.6868519186973572
Test2 loss: 0.6951574683189392
Test accuracy: micro: 0.5461137147404722        macro: 0.5471378784179688
Test accuracy: micro: 0.51893216125662  macro: 0.5202606201171875
{'mean': 'A'} {'mean': 'A'} {'simple': 'T'} acgnn 5 0
```

Epoch for P2 dataset:



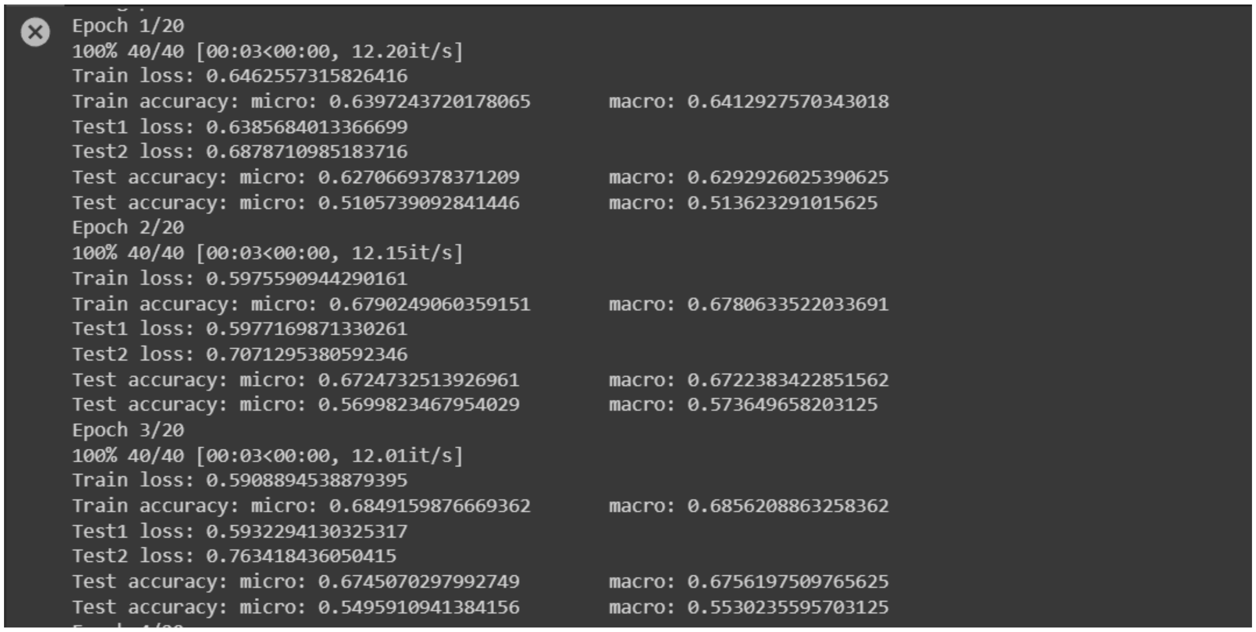
```
Epoch 1/20
100% 40/40 [00:07<00:00,  5.47it/s]
Train loss: 0.689043402671814
Train accuracy: micro: 0.5328496663497507       macro: 0.5326088428974152
Test1 loss: 0.6925417184829712
Test2 loss: 0.6918095946311951
Test accuracy: micro: 0.5289150234326643        macro: 0.5291004638671875
Test accuracy: micro: 0.525453038873077 macro: 0.5266585083007812
Epoch 2/20
100% 40/40 [00:07<00:00,  5.48it/s]
Train loss: 0.6856144070625305
Train accuracy: micro: 0.5564539775907875       macro: 0.5579206075668335
Test1 loss: 0.6859328150749207
Test2 loss: 0.6963554620742798
Test accuracy: micro: 0.5551772924219648        macro: 0.5565367431640625
Test accuracy: micro: 0.5198688619087077        macro: 0.5213787841796875
Epoch 3/20
100% 40/40 [00:07<00:00,  5.50it/s]
Train loss: 0.6841906309127808
Train accuracy: micro: 0.5554365886816595       macro: 0.5570890666007996
Test1 loss: 0.6857189536094666
Test2 loss: 0.6984595656394958
Test accuracy: micro: 0.5533203643116102        macro: 0.55477783203125
Test accuracy: micro: 0.5174190294340166        macro: 0.5188096313476562
```

Final output of P2 dataset after 20 iterations:



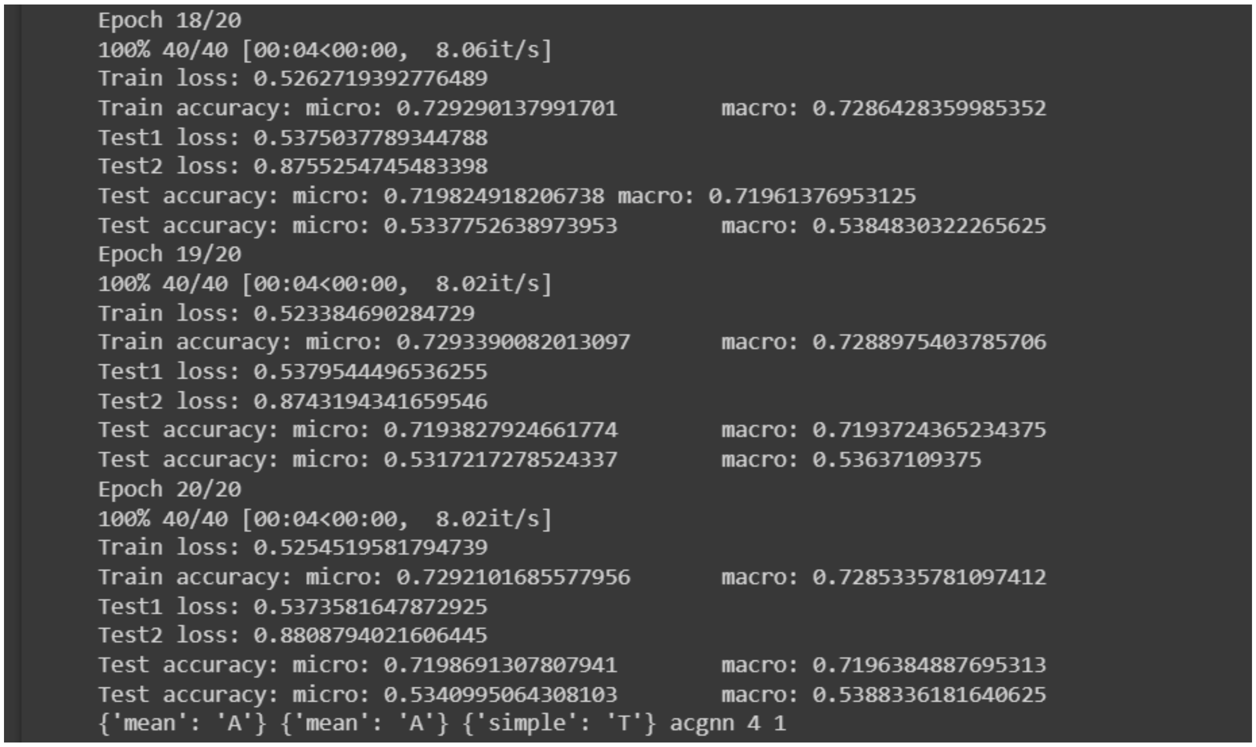
```
Epoch 18/20
100% 40/40 [00:01<00:00, 29.58it/s]
Train loss: 0.6842108368873596
Train accuracy: micro: 0.5600570448628525       macro: 0.5610081130027771
Test1 loss: 0.6858022809028625
Test2 loss: 0.6945090889930725
Test accuracy: micro: 0.5559731187549739        macro: 0.5566870727539063
Test accuracy: micro: 0.5264257664733221        macro: 0.5280440673828125
Epoch 19/20
100% 40/40 [00:01<00:00, 29.21it/s]
Train loss: 0.6852332353591919
Train accuracy: micro: 0.5603458233741769       macro: 0.56130442943573
Test1 loss: 0.6858172416687012
Test2 loss: 0.6944608688354492
Test accuracy: micro: 0.5558404810328057        macro: 0.5565079956054687
Test accuracy: micro: 0.5268941167993659        macro: 0.52853662109375
Epoch 20/20
100% 40/40 [00:01<00:00, 28.58it/s]
Train loss: 0.6844810247421265
Train accuracy: micro: 0.5601903272526946       macro: 0.5611350563049317
Test1 loss: 0.6858125925064087
Test2 loss: 0.6942453384399414
Test accuracy: micro: 0.5561057564771421        macro: 0.556789794921875
Test accuracy: micro: 0.5267860359548943        macro: 0.5284205322265625
{'mean': 'A'} {'mean': 'A'} {'simple': 'T'} acgnn 2 1
```

Epoch for P3 dataset:



```
Epoch 1/20
100% 40/40 [00:03<00:00, 12.20it/s]
Train loss: 0.6462557315826416
Train accuracy: micro: 0.6397243720178065       macro: 0.6412927570343018
Test1 loss: 0.6385684013366699
Test2 loss: 0.6878710985183716
Test accuracy: micro: 0.6270669378371209        macro: 0.6292926025390625
Test accuracy: micro: 0.5105739092841446        macro: 0.513623291015625
Epoch 2/20
100% 40/40 [00:03<00:00, 12.15it/s]
Train loss: 0.5975590944290161
Train accuracy: micro: 0.6790249060359151       macro: 0.6780633522033691
Test1 loss: 0.5977169871330261
Test2 loss: 0.7071295380592346
Test accuracy: micro: 0.6724732513926961        macro: 0.6722383422851562
Test accuracy: micro: 0.5699823467954029        macro: 0.573649658203125
Epoch 3/20
100% 40/40 [00:03<00:00, 12.01it/s]
Train loss: 0.5908894538879395
Train accuracy: micro: 0.6849159876669362       macro: 0.6856208863258362
Test1 loss: 0.5932294130325317
Test2 loss: 0.763418436050415
Test accuracy: micro: 0.6745070297992749        macro: 0.6756197509765625
Test accuracy: micro: 0.5495910941384156        macro: 0.5530235595703125
```

Final output of P3 dataset after 20 iterations:

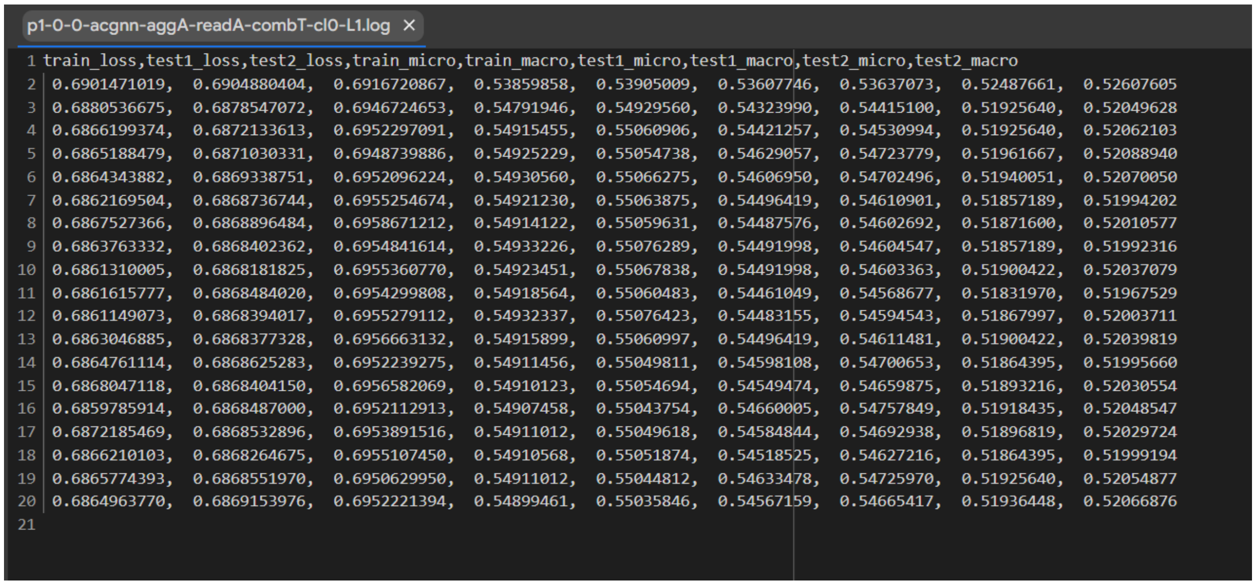


```
Epoch 18/20
100% 40/40 [00:04<00:00,  8.06it/s]
Train loss: 0.5262719392776489
Train accuracy: micro: 0.729290137991701        macro: 0.7286428359985352
Test1 loss: 0.5375037789344788
Test2 loss: 0.8755254745483398
Test accuracy: micro: 0.719824918206738 macro: 0.71961376953125
Test accuracy: micro: 0.5337752638973953        macro: 0.5384830322265625
Epoch 19/20
100% 40/40 [00:04<00:00,  8.02it/s]
Train loss: 0.523384690284729
Train accuracy: micro: 0.7293390082013097       macro: 0.7288975403785706
Test1 loss: 0.5379544496536255
Test2 loss: 0.8743194341659546
Test accuracy: micro: 0.7193827924661774        macro: 0.7193724365234375
Test accuracy: micro: 0.5317217278524337        macro: 0.53637109375
Epoch 20/20
100% 40/40 [00:04<00:00,  8.02it/s]
Train loss: 0.5254519581794739
Train accuracy: micro: 0.7292101685577956       macro: 0.7285335781097412
Test1 loss: 0.5373581647872925
Test2 loss: 0.8808794021606445
Test accuracy: micro: 0.7198691307807941        macro: 0.7196384887695313
Test accuracy: micro: 0.5340995064308103        macro: 0.5388336181640625
{'mean': 'A'} {'mean': 'A'} {'simple': 'T'} acgnn 4 1
```

Results:

After training all the datasets final result files are saved in the logs and which can be viewed as following images.

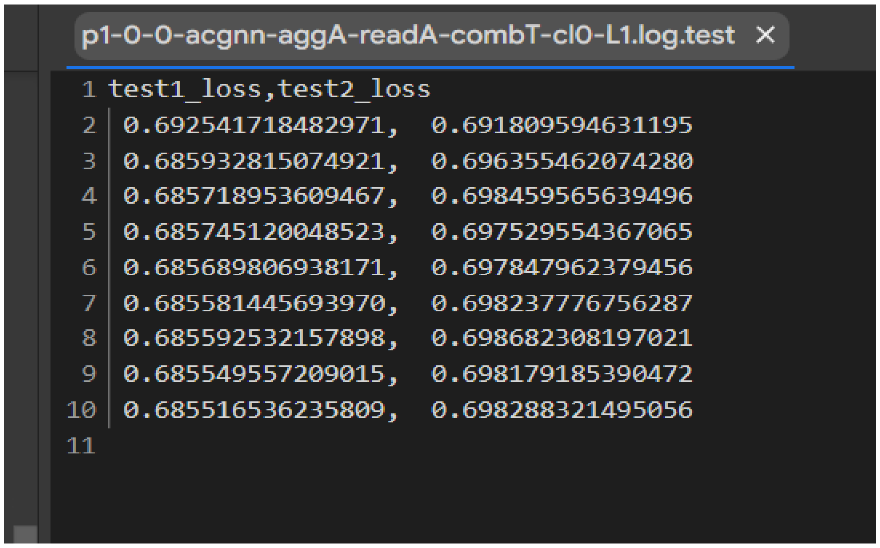
Log file for training all the graph coordinates present in P1, P2 and P3 datasets which can give a perfect overview of local aggregations to distances up number of layers. This combined with the fact that random graphs that are denser make the maximum distances between nodes shorter, may explain the boost in performance for AC-GNNs.



p1-0-0-acgnn-aggA-readA-combT-cl0-L1.log

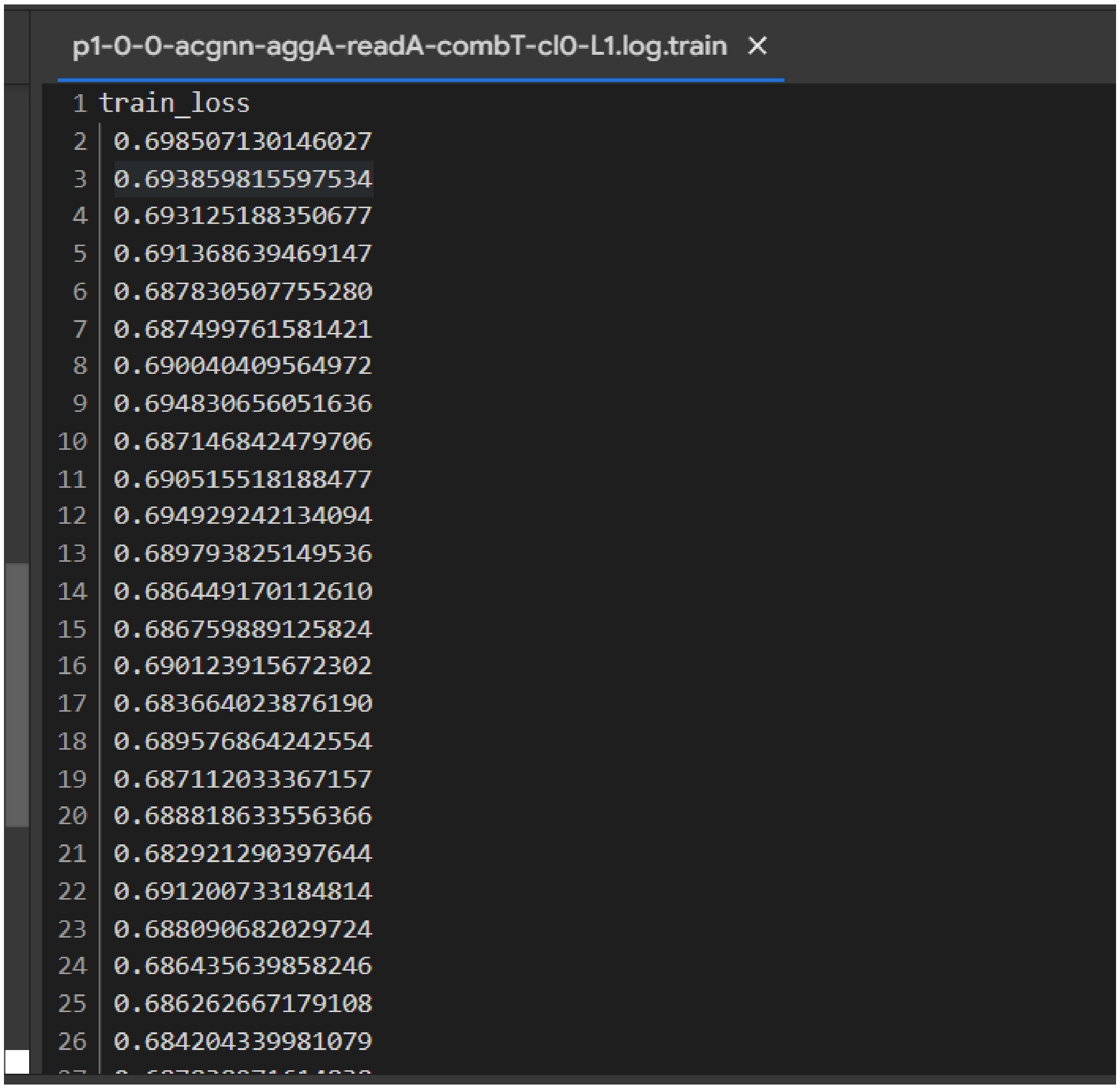
```
train_loss,test1_loss,test2_loss,train_micro,train_macro,test1_micro,test1_macro,test2_micro,test2_macro
 0.6901471019,  0.6904880404,  0.6916720867,  0.53859858,  0.53905009,  0.53607746,  0.53637073,  0.52487661,  0.52607605
 0.6880536675,  0.6878547072,  0.6946724653,  0.54791946,  0.54929560,  0.54323990,  0.54415100,  0.51925640,  0.52049628
 0.6866199374,  0.6872133613,  0.6952297091,  0.54915455,  0.55060906,  0.54421257,  0.54530994,  0.51925640,  0.52062103
 0.6865188479,  0.6871030331,  0.6948739886,  0.54925229,  0.55054738,  0.54629057,  0.54723779,  0.51961667,  0.52088940
 0.6864343882,  0.6869338751,  0.6952096224,  0.54930560,  0.55066275,  0.54606950,  0.54702496,  0.51940051,  0.52070050
 0.6862169504,  0.6868736744,  0.6955254674,  0.54921230,  0.55063875,  0.54496419,  0.54610901,  0.51857189,  0.51994202
 0.6867527366,  0.6868896484,  0.6958671212,  0.54914122,  0.55059631,  0.54487576,  0.54602692,  0.51871600,  0.52010577
 0.6863763332,  0.6868402362,  0.6954841614,  0.54933226,  0.55076289,  0.54491998,  0.54604547,  0.51857189,  0.51992316
 0.6861310005,  0.6868181825,  0.6955360770,  0.54923451,  0.55067838,  0.54491998,  0.54603363,  0.51900422,  0.52037079
 0.6861615777,  0.6868484020,  0.6954299808,  0.54918564,  0.55060483,  0.54461049,  0.54568677,  0.51831970,  0.51967529
 0.6861149073,  0.6868394017,  0.6955279112,  0.54932337,  0.55076423,  0.54483155,  0.54594543,  0.51867997,  0.52003711
 0.6863046885,  0.6868377328,  0.6956663132,  0.54915899,  0.55060997,  0.54496419,  0.54611481,  0.51900422,  0.52039819
 0.6864761114,  0.6868625283,  0.6952239275,  0.54911456,  0.55049811,  0.54598108,  0.54700653,  0.51864395,  0.51995660
 0.6868047118,  0.6868404150,  0.6956582069,  0.54910123,  0.55054694,  0.54549474,  0.54659875,  0.51893216,  0.52030554
 0.6859785914,  0.6868487000,  0.6952112913,  0.54907458,  0.55043754,  0.54660005,  0.54757849,  0.51918435,  0.52048547
 0.6872185469,  0.6868532896,  0.6953891516,  0.54911012,  0.55049618,  0.54584844,  0.54692938,  0.51896819,  0.52029724
 0.6866210103,  0.6868264675,  0.6955107450,  0.54910568,  0.55051874,  0.54518525,  0.54627216,  0.51864395,  0.51999194
 0.6865774393,  0.6868551970,  0.6950629950,  0.54911012,  0.55044812,  0.54633478,  0.54725970,  0.51925640,  0.52054877
 0.6864963770,  0.6869153976,  0.6952221394,  0.54899461,  0.55035846,  0.54567159,  0.54665417,  0.51936448,  0.52066876
```

Loss in test1 and test 2.



p1-0-0-acgnn-aggA-readA-combT-cl0-L1.log.test

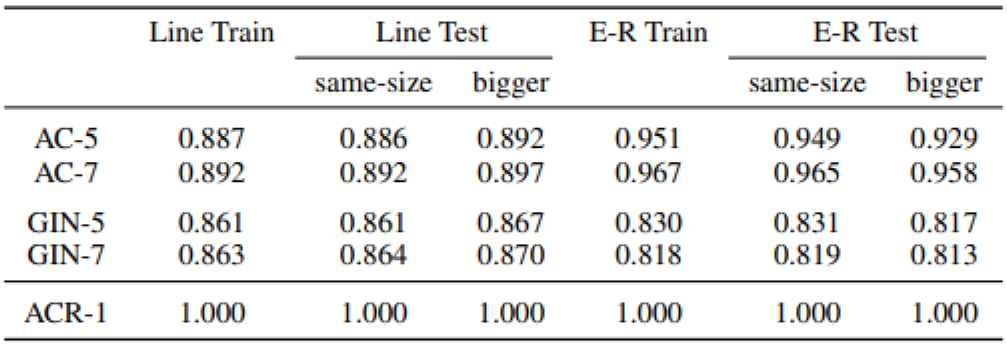
```
test1_loss,test2_loss
 0.692541718482971,  0.691809594631195
 0.685932815074921,  0.696355462074280
 0.685718953609467,  0.698459565639496
 0.685745120048523,  0.697529554367065
 0.685689806938171,  0.697847962379456
 0.685581445693970,  0.698237776756287
 0.685592532157898,  0.698682308197021
 0.685549557209015,  0.698179185390472
 0.685516536235809,  0.698288321495056
```



p1-0-0-acgnn-aggA-readA-combT-cl0-L1.log.train

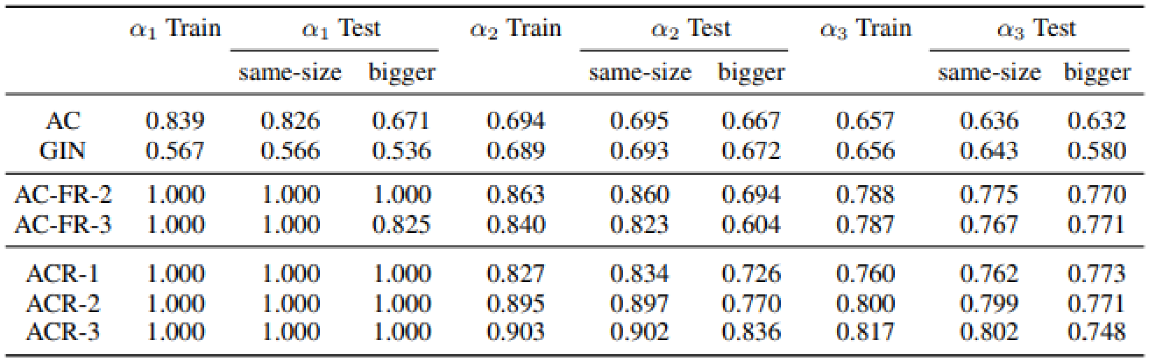
```
train_loss
 0.698507130146027
 0.693859815597534
 0.693125188350677
 0.691368639469147
 0.687830507755280
 0.687499761581421
 0.690040409564972
 0.694830656051636
 0.687146842479706
 0.690515518188477
 0.694929242134094
 0.689793825149536
 0.686449170112610
 0.686759889125824
 0.690123915672302
 0.683664023876190
 0.689576864242554
 0.687112033367157
 0.688818633556366
 0.682921290397644
 0.691200733184814
 0.688090682029724
 0.686435639858246
 0.686262667179108
 0.684204339981079
```

Results on synthetic data for nodes labeled by classifier $\alpha(x) := Red(x) \land \exists y\, Blue(y)$ are obtained by above files and can be seen as the table below:



	Line Train	Line Test		E-R Train	E-R Test	
		same-size	bigger		same-size	bigger
AC-5	0.887	0.886	0.892	0.951	0.949	0.929
AC-7	0.892	0.892	0.897	0.967	0.965	0.958
GIN-5	0.861	0.861	0.867	0.830	0.831	0.817
GIN-7	0.863	0.864	0.870	0.818	0.819	0.813
ACR-1	1.000	1.000	1.000	1.000	1.000	1.000

Similarly, as per the equation (6) in the paper Results on E-R synthetic data for nodes labeled by classifiers αi(x) can be seen in following table.



	α_1 Train	α_1 Test		α_2 Train	α_2 Test		α_3 Train	α_3 Test	
		same-size	bigger		same-size	bigger		same-size	bigger
AC	0.839	0.826	0.671	0.694	0.695	0.667	0.657	0.636	0.632
GIN	0.567	0.566	0.536	0.689	0.693	0.672	0.656	0.643	0.580
AC-FR-2	1.000	1.000	1.000	0.863	0.860	0.694	0.788	0.775	0.770
AC-FR-3	1.000	1.000	0.825	0.840	0.823	0.604	0.787	0.767	0.771
ACR-1	1.000	1.000	1.000	0.827	0.834	0.726	0.760	0.762	0.773
ACR-2	1.000	1.000	1.000	0.895	0.897	0.770	0.800	0.799	0.771
ACR-3	1.000	1.000	1.000	0.903	0.902	0.836	0.817	0.802	0.748

Problems/Issues Faced: I faced few problems while setting this up on my local machine. First of all I tried to get this on VScode which did not work due to some version error. Then I decided to get this project on Jupyter and google colab. This step helped me setting up my environment successfully. While installing few libraries like torch-cluster, torch-scatter, scipy(numpy) I faced issues due to some errors. One of which included cannot get wheel setup for torch. Main issue I faced was creating a log file in my system since they had designed it for their own systems. I had to change the code according to my system to save the logging result to my system. Every time I ran the code it failed to detect my GPU for some reason, I tried to resolve this using it on google colab. Mainly torch version used in the code and given in requirement.txt is quite old and creates many issues every time when I start training the model. All the changes had to be made every time whenever I trained the model because colab does not save the modules installed in prior testing.

Conclusion:

The main contribution of this paper is the identification of logical classifiers that can be represented within an AC-GNN, a fragment of first order logic known as graded modal logic, as well as an extension of AC-GNNs that can capture a strictly more expressive fragment of first order logic known as ACR-GNN. Both of these conclusions are backed up by formal proofs and derivations, giving the study not only theoretical importance but also intuitive insights into the reasons for the AC-

GNN limits. Furthermore, with the exception of a few datasets where no significant difference in performance between AC-GNNs and ACR-GNNs was discovered, the reported studies indicate the practical relevance of the findings. When classifying nodes in a graph, the results demonstrate the theoretical benefits of combining local and global information. Recently, Deng et al. exploited global-context aware local descriptors to categorize objects in 3D point clouds, demonstrating these benefits in practice.

On other hand, researching the theoretical features of GNNs makes them more transparent and illustrates the wide range of applications they can be applied to. On the other hand, while the author's GNN variant is a special case of an existing class of GNNs, the motivation for its creation is distinct and follows organically from the paper's discussion.

Graphs in k-GNNs are structures that connect k-tuples of nodes rather than merely pairs. We intend to investigate how our logical classifier findings relate to k-GNNs, particularly in relation to the logic FOCk, which extends FOC2 by allowing formulae with k variables for each fixed k > 1.

This study, I believe, makes a substantial contribution.

All of the above experiments can be seen and viewed in the following links:

Link for the entire repository:

https://github.com/yash171298/CS-541-Artificial-Intelligence/tree/main/GNN-logic-experiment-project

Link for log file having results saved after training the model:

https://github.com/yash171298/CS-541-Artificial-Intelligence/tree/main/GNN-logic-experiment-project/src/logging/results

Link for experiments performed before having the final output:

https://github.com/yash171298/CS-541-Artificial-Intelligence/blob/main/GNN-logic-experiment-project/final_experiment_GNN.ipynb