# Progess of the Project

Tsung-Min Pai

2023/12/01

# Outline

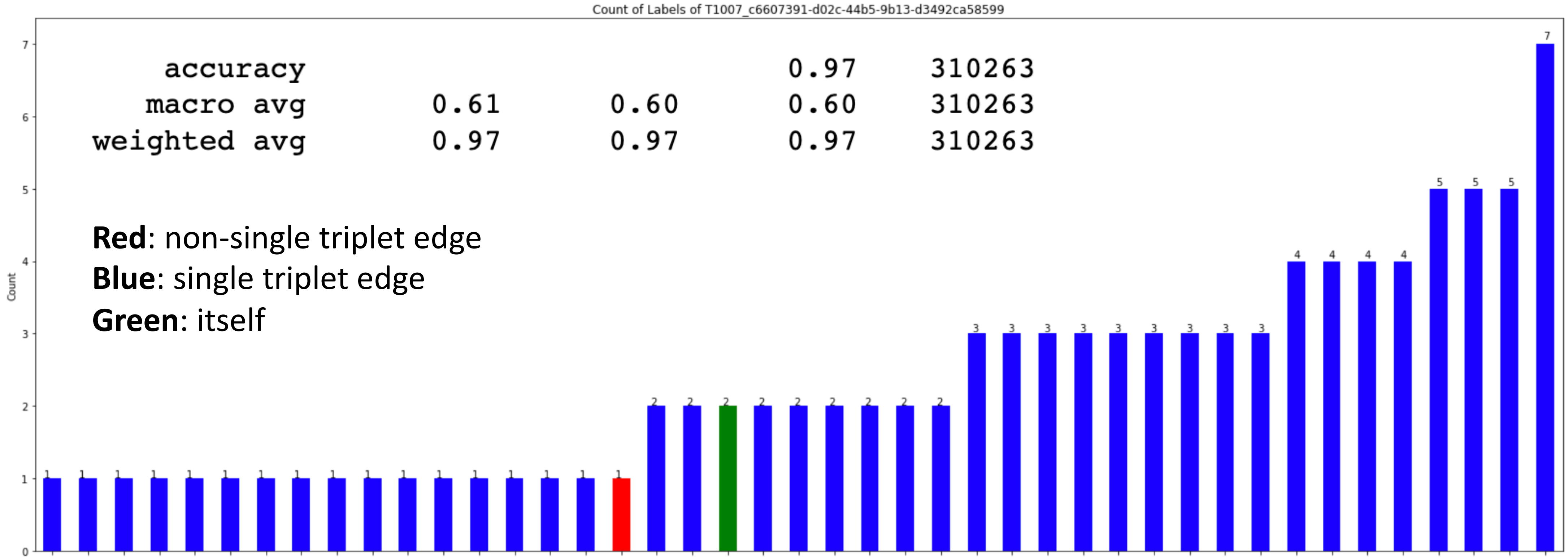
- **GNN**
  - **Recap**
  - **Experiment – Change the Dataset**
  - **Experiment – Predict More Labels**
  - **Experiment – DAPRA Format**

- **Future Work**

# Recap

# Observation of the Prediction

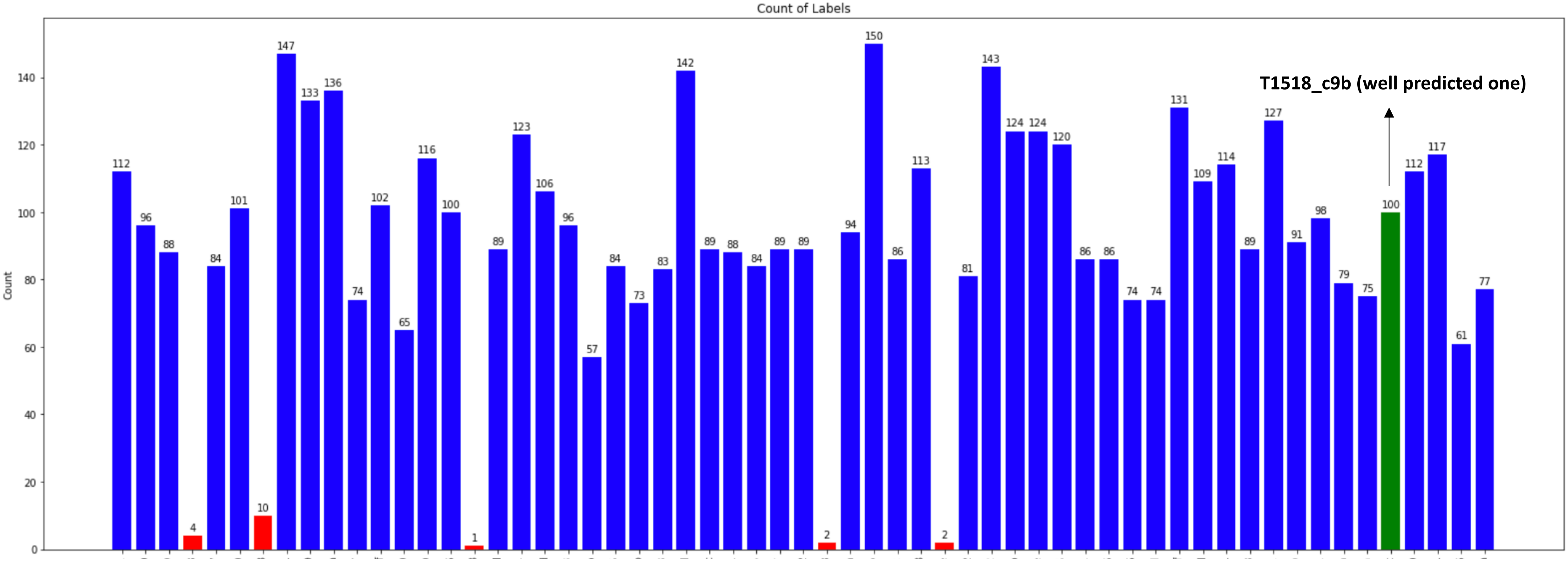
- Use the original training set
- The distribution of the prediction is so sparse
  - Most of the predictions are like this:



# Observation of the Prediction

- The # of the predicted labels

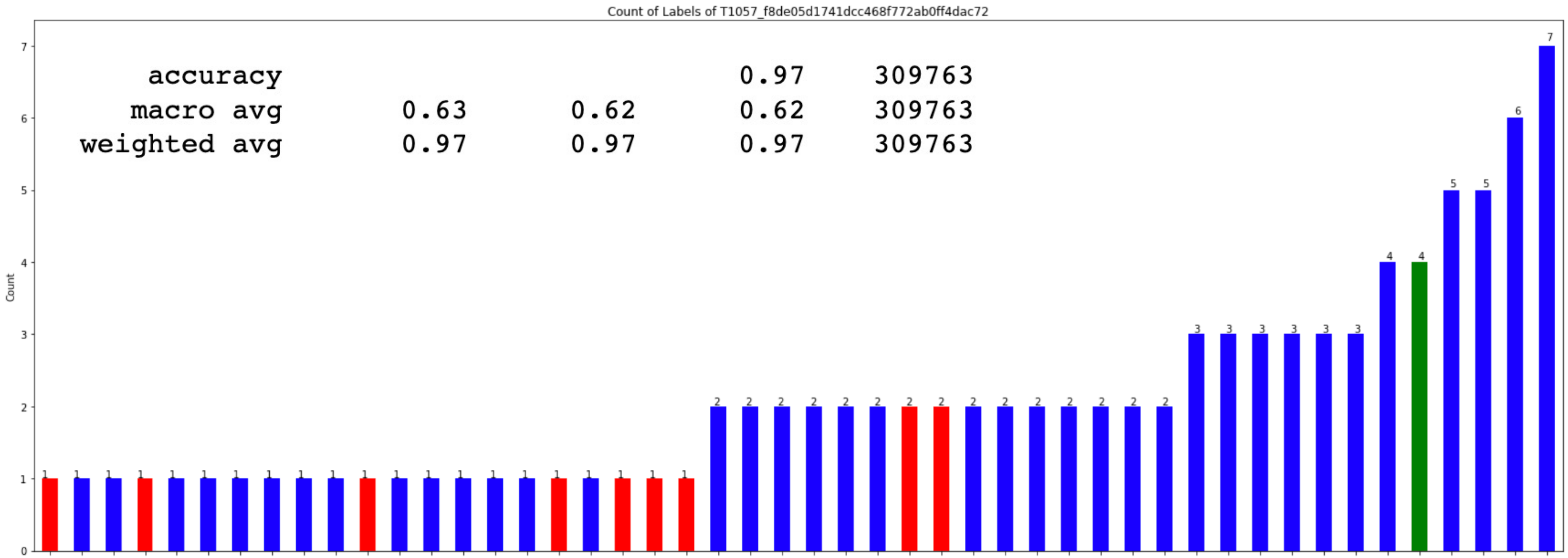
```
Top 5 Labels and Counts:
T1082_29451844-9b76-4e16-a9ee-d6feab4b24db: 150
T1016_921055f4-5970-4707-909e-62f594234d91: 147
T1124_fa6e8607-e0b1-425d-8924-9b894da5a002: 143
T1053.005_ee454be9197890de62705ce6255933fd: 142
T1016_e8017c46-acb8-400c-a4b5-b3362b5b5baa: 136
```



# Remove the Popular TTPs

```
Top 5 Labels and Counts:
T1082_29451844-9b76-4e16-a9ee-d6feab4b24db: 150
T1016_921055f4-5970-4707-909e-62f594234d91: 147
T1124_fa6e8607-e0b1-425d-8924-9b894da5a002: 143
T1053.005_ee454be9197890de62705ce6255933fd: 142
T1016_e8017c46-acb8-400c-a4b5-b3362b5b5baa: 136
```

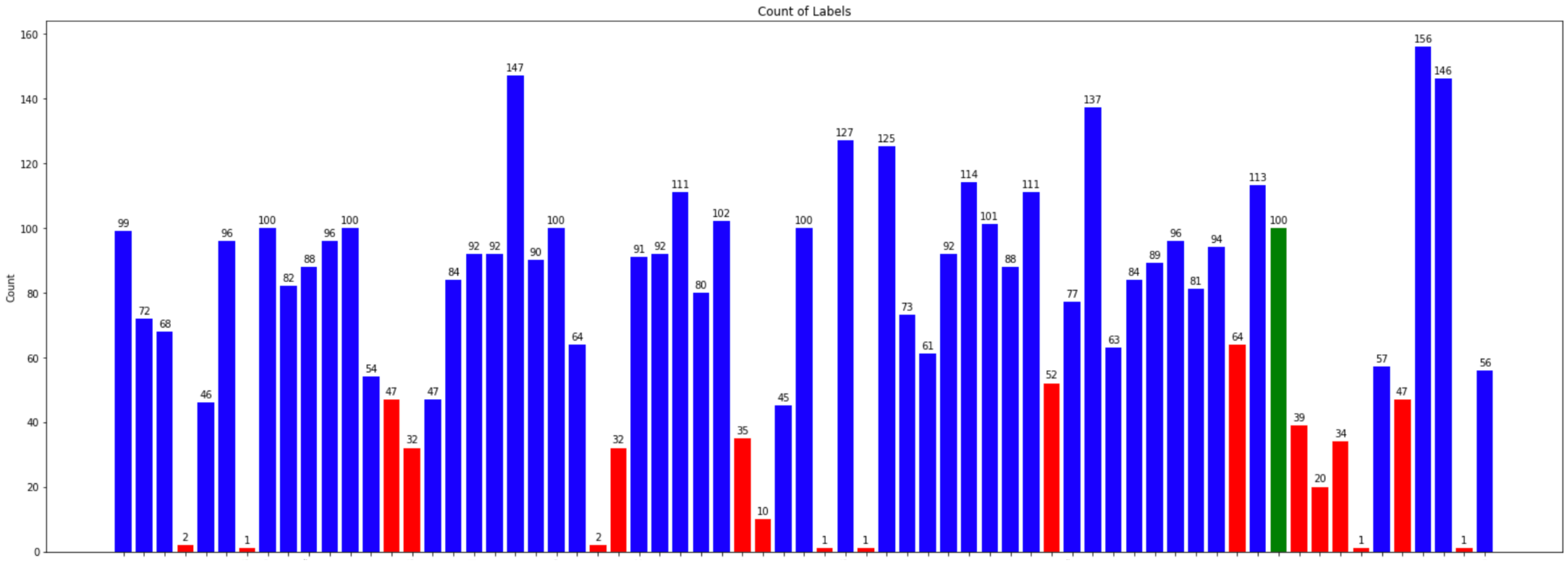
- Remove these 5 TTPs from the dataset and then train it again:
  - More prediction on the non-single triplet case is like this:




| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.97 | 309763 |
| macro avg | 0.63 | 0.62 | 0.62 | 309763 |
| weighted avg | 0.97 | 0.97 | 0.97 | 309763 |

# Remove the Popular TTPs

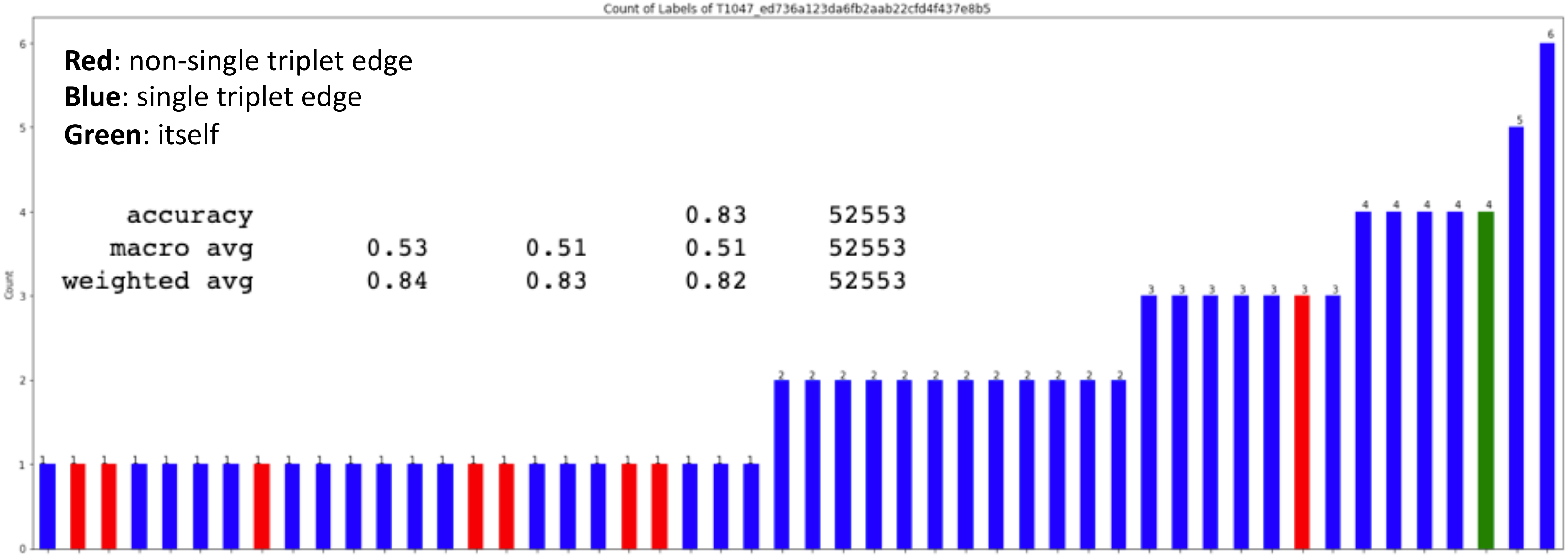
- Remove these 5 TTPs from the dataset and then train it again:
- The # of the predicted labels



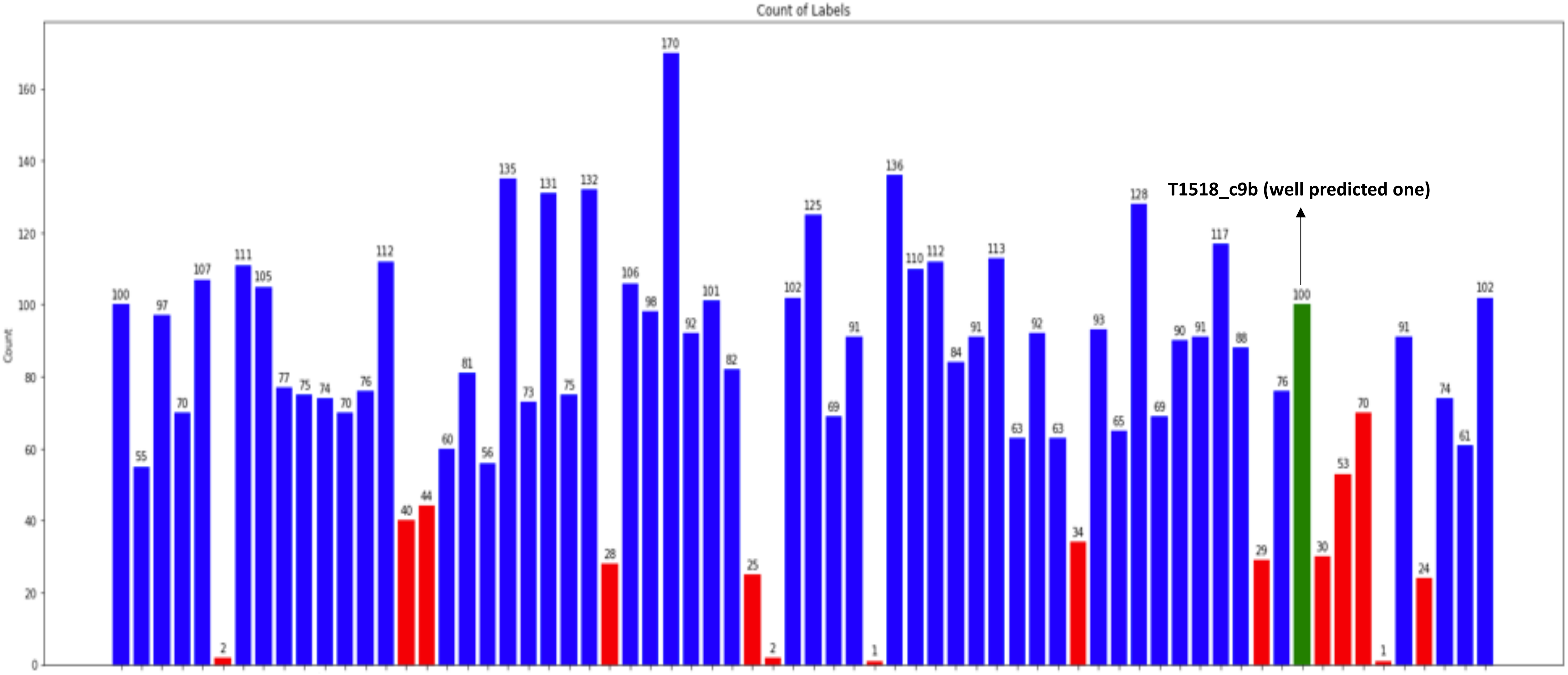
# Experiment – Change the Dataset

# Observation of the Prediction

- Remove the **TTPs with support > 1000** (Removed 32 TTPs)
  - The predictions on the single triplet cases are still a mess

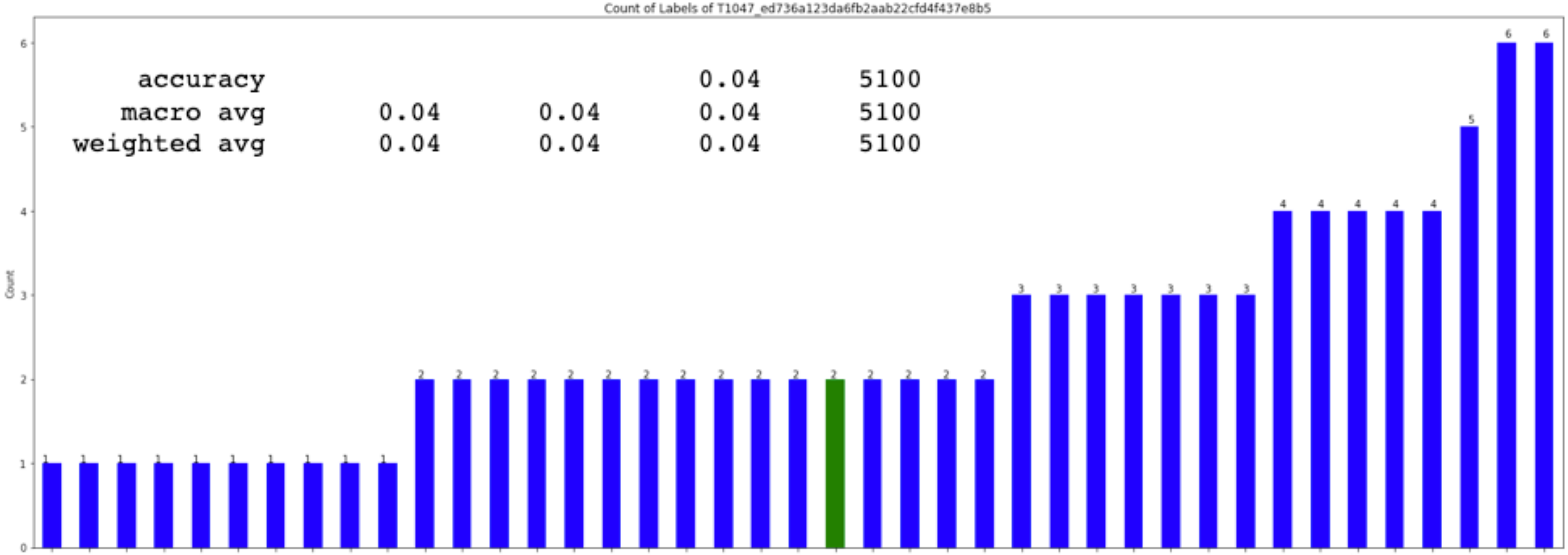


# Observation of the Prediction

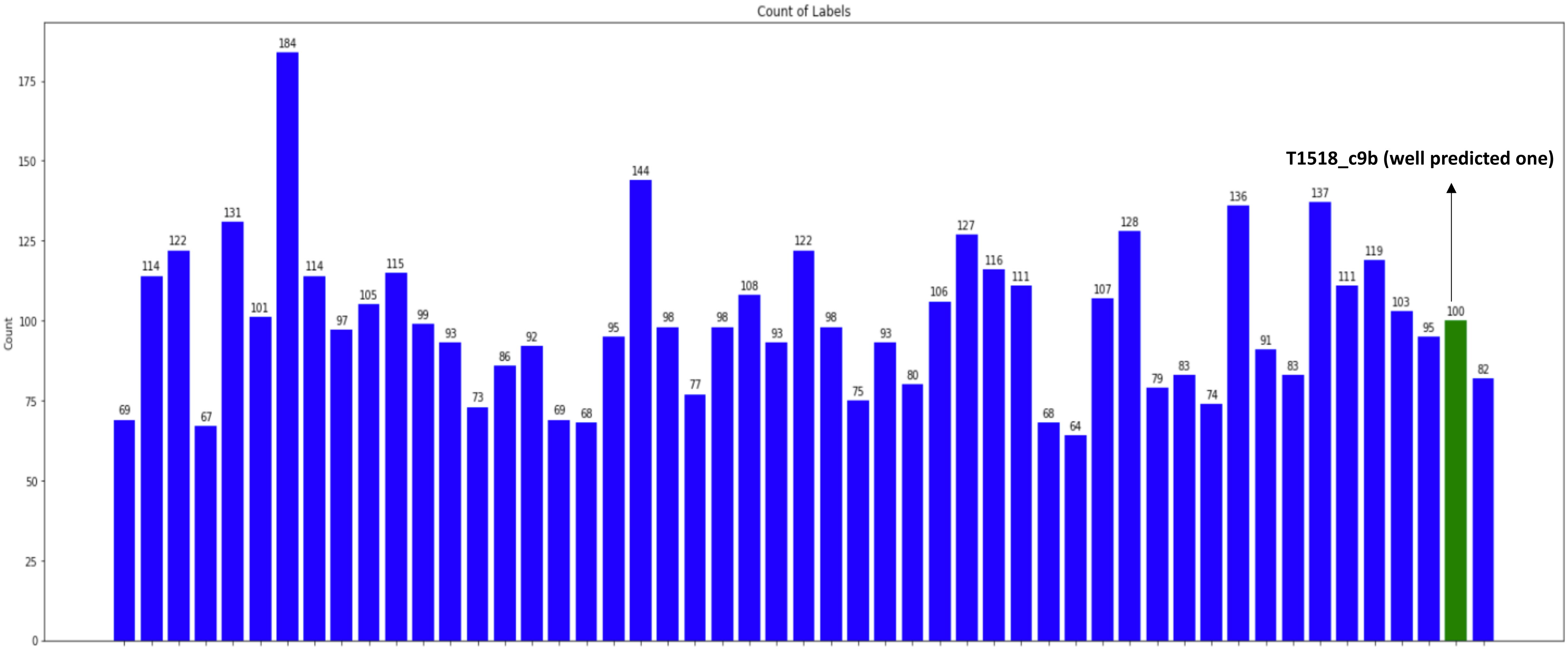


# Observation of the Prediction

- Only use the **single triplet (support=100) cases**:
  - Still messy and even have a worse performance

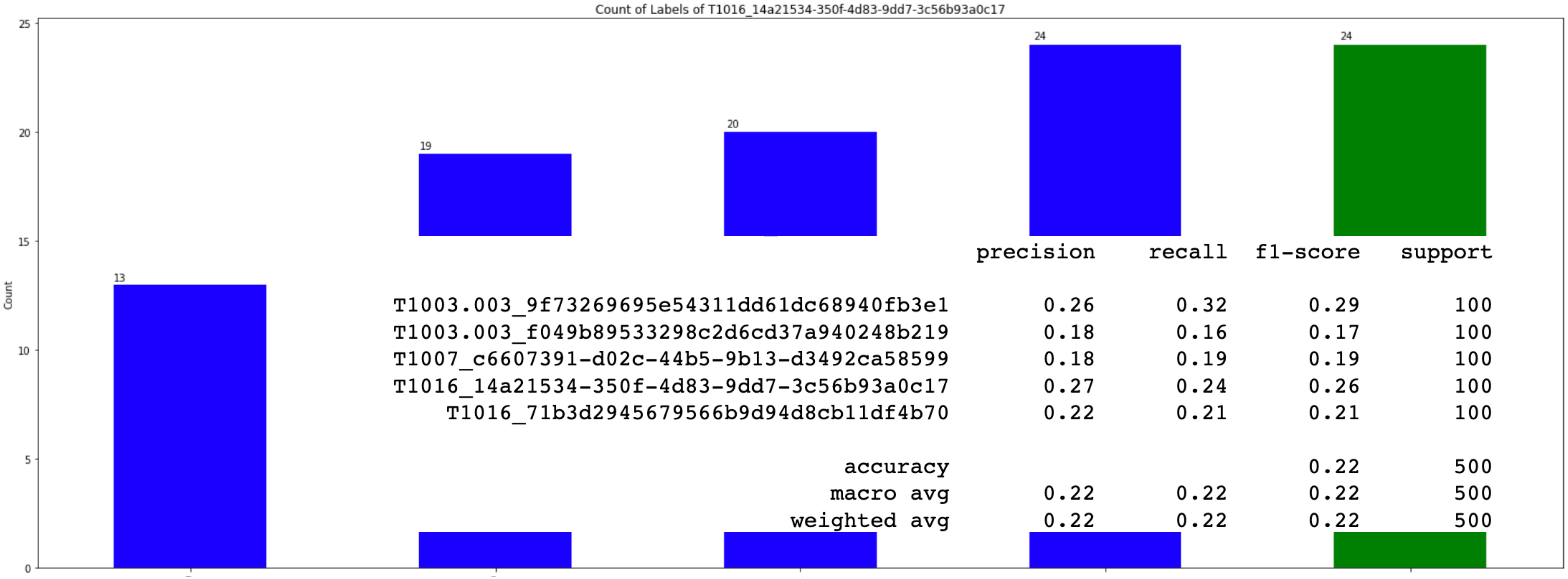


# Observation of the Prediction



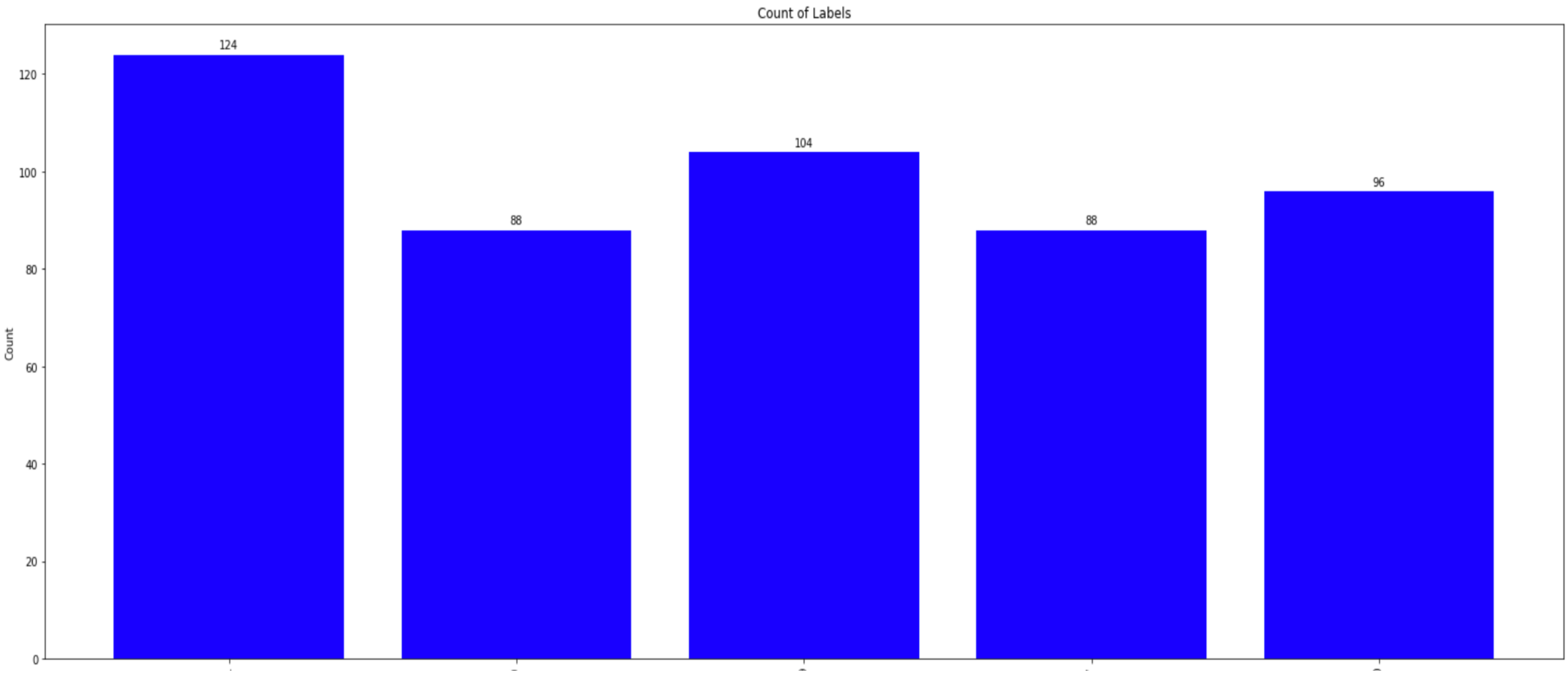
# Observation of the Prediction

- Dataset only has the **5 single triplet labels**



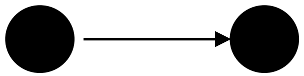

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| T1003.003_9f73269695e54311dd61dc68940fb3e1 | 0.26 | 0.32 | 0.29 | 100 |
| T1003.003_f049b89533298c2d6cd37a940248b219 | 0.18 | 0.16 | 0.17 | 100 |
| T1007_c6607391-d02c-44b5-9b13-d3492ca58599 | 0.18 | 0.19 | 0.19 | 100 |
| T1016_14a21534-350f-4d83-9dd7-3c56b93a0c17 | 0.27 | 0.24 | 0.26 | 100 |
| T1016_71b3d2945679566b9d94d8cb11df4b70 | 0.22 | 0.21 | 0.21 | 100 |
| accuracy | | | 0.22 | 500 |
| macro avg | 0.22 | 0.22 | 0.22 | 500 |
| weighted avg | 0.22 | 0.22 | 0.22 | 500 |

# Observation of the Prediction



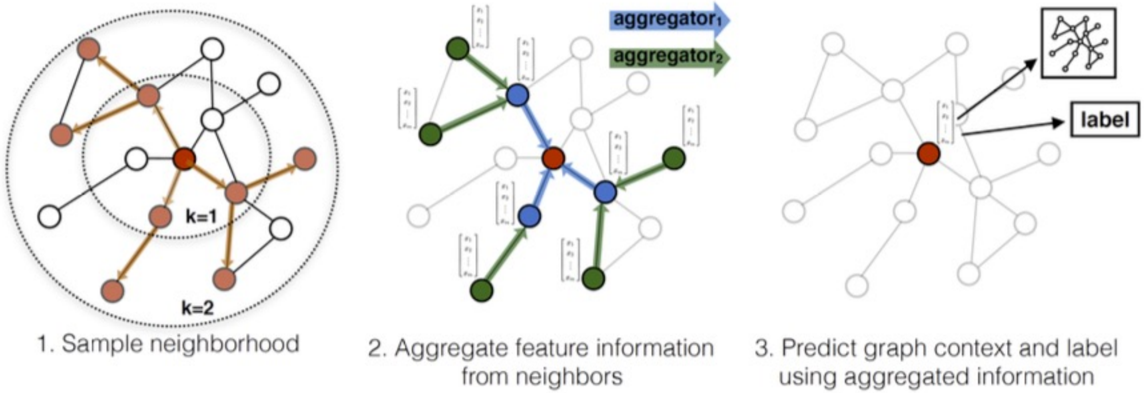
# Conclusion

- GraphSAGE really have a bad performance on the **isolated** triplet case
  - Even training dataset consists of isolated triplet case only
  - Even training dataset consists of only 5 isolated triplet cases



- Possible Reason:
  - **Lack of Neighborhood Information**: GraphSAGE aggregates features from a node's local neighborhood to learn its representation. In the case of isolated triplets, each node has very limited neighborhood information (only one neighbor), which restricts the model's ability to learn complex or rich representations

Visual illustration of the GraphSAGE sample and aggregate approach:



# Experiment – Predict More Labels

# Predict the Top 3

- Thought:
  - The experts can use the DL prediction and combine with their expertise to make the final prediction

- Prediction Format:
  - Predict the **top 3** classes of the edge
  - True:

```
65,
65,
70,
```

Predicted:

```
[65, 28, 46],
[65, 28, 46],
[70, 154, 149],
```

- Performance:

| | precision | recall | f1-score |
|---|---|---|---|
| **macro avg** | 0.656194 | 0.643040 | 0.634166 |
| **micro avg** | 0.977822 | 0.977822 | 0.977822 |

Previous one:

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.97 | 310263 |
| macro avg | 0.61 | 0.60 | 0.60 | 310263 |
| weighted avg | 0.97 | 0.97 | 0.97 | 310263 |

# Predict the Top 5

- Prediction Format:
  - Predict the **top 5** classes of the edge
  - True:

```
65,
65,
70,
```

Predicted:

```
[65, 28, 46, 154, 148],
[65, 28, 46, 154, 135],
[70, 154, 149, 52, 135],
```

- Performance:

| | precision | recall | f1-score |
|---|---|---|---|
| macro avg | 0.701622 | 0.665422 | 0.656189 |
| micro avg | 0.979798 | 0.979798 | 0.979798 |

Version of top3:

| | precision | recall | f1-score |
|---|---|---|---|
| macro avg | 0.656194 | 0.643040 | 0.634166 |
| micro avg | 0.977822 | 0.977822 | 0.977822 |

Previous one:

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.97 | 310263 |
| macro avg | 0.61 | 0.60 | 0.60 | 310263 |
| weighted avg | 0.97 | 0.97 | 0.97 | 310263 |

# Experiment – DAPRA Format

# Change the DARPA Format

- Original Format:
  - {"subj": {"uuid": "F9F5DC58-9431-4519-82CF-2BBFF40796E9", "type": "Subject", "n_attrbiute": {"cmdline": "None", "type": "SUBJECT_THREAD", "pid": 8920}}, "relation": "EVENT_WRITE", "obj": {"uuid": "F5D43721-2AD3-487C-B3DA-0A5D551FAE0D", "type": "FileObject", "n_attrbiute": {"filepath": "\\\\Device\\\\HarddiskVolume2\\\\ProgramData\\\\Microsoft\\\\Windows Defender\\\\Network Inspection System\\\\Support\\\\NisLog.txt"}}, "timestamp": 1523295276588000000, "label": "benign"}

- Final Format:
  - Encode the subj uuid, relation, obj uuid and combine them with the label
  - Keep the mapping information into 2 mapping file
  - a is attack; b is benign

```
872529 1086092 5 a
872529 61317 7 a
1059095 759218 19 b
280074 759218 19 b
```

# Future Work

# Future Work

- Figure out why the model can detect the **T1518_c9b** (if available?)
  - Is the label overlap or not overlap at all with other labels
- Figure out what the model really predict about
  - Observe the real data in the original form (before embedding)
- Run the training of the DAPRA dataset
- GraphSMOTE

Thanks!!

# NOTE!!

# NOTE

- 每次都要再複習一下問題點
  - Show 以前的數據等等
  - 為何要做現在這個實驗
- 圖表要清楚一點
  - X and Y-axis
- 給的example要是真的有解決那個問題的example

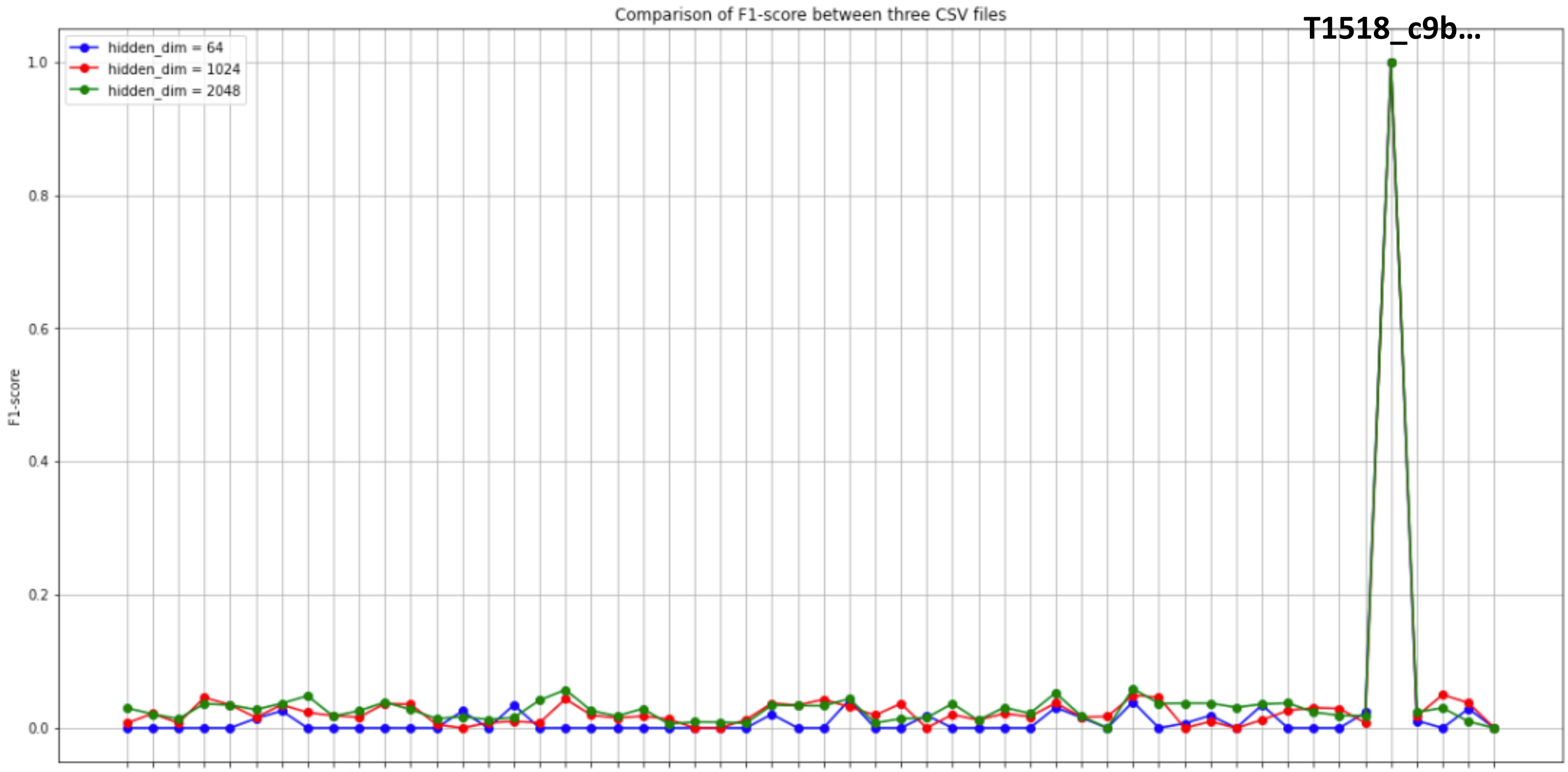
# Appendix

# Recap

- Noticed that **T1518_c9b** always got predicted in all the experiments
  - MLP, RNN, GNN
  - **Ensemble** is not useful here

- **Hidden Dimension** do have an effect on the result → but it's all about from 0 to 0.05
- **Embedding** (transR_50 and tansH_150) seems to have the similar result

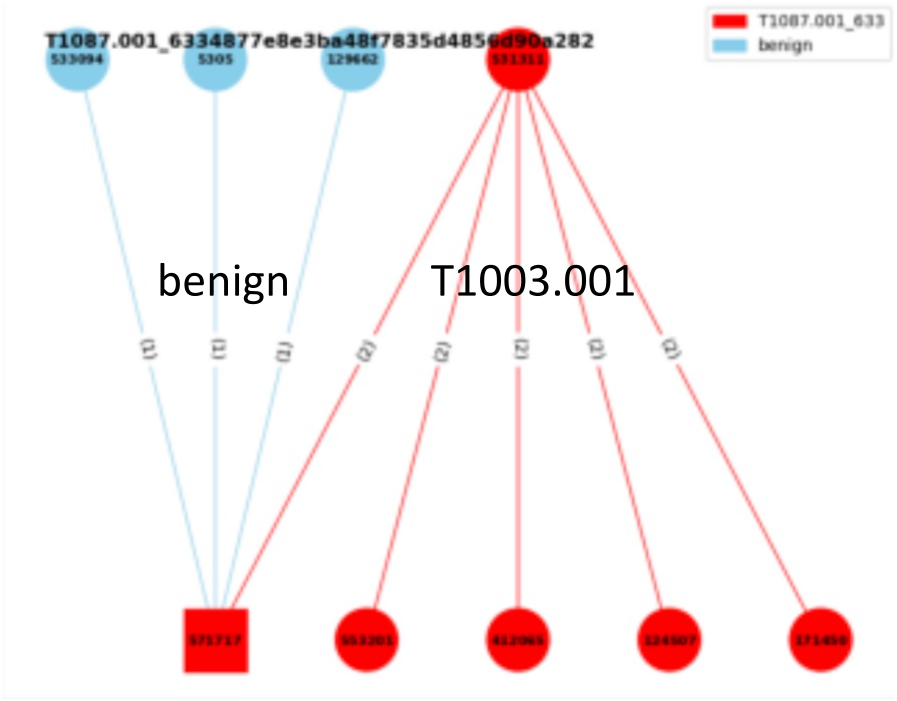
# Observation on Different Dimension



# Experiment 3

- **Experiment 3:**
  - Consider the **neighbor** benign nodes
  - Edge classification

  - Given a graph → label the triplets with the benign or the specific AP



# Experiment 3 - Model

- Concept from the DGL official website:
  1. Let the dgl graph's edge data have the attribute: **edata["label"]**
  2. Use **GraphSAGE** model to get the new **node embedding**
  3. Use **MLP** model to get the **score** of the edge
  4. Concatenate these two models
  5. Train the final model

```
g.ndata['feat'] = th.tensor(data["node_feat"])
g.edata['feat'] = th.tensor(data["edge_attr"])
g.edata['label'] = th.tensor(data["labels"])
```

```
def model_fn(batched_g, model, criterion, device, count=1, which_type='train'):
    """Forward a batch through the model."""
    batched_g = batched_g.to(device)
    labels = batched_g.edata['label'].to(device)

    logits = model(batched_g, batched_g.ndata['feat'].float())
    loss = criterion(logits, labels)

    output = torch.softmax(logits, dim=1)
    preds = output.argmax(1)

    accuracy = torch.mean((preds == labels).float())
```

# Experiment 3 - Model

```
class GraphSAGE(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(GraphSAGE, self).__init__()
        self.layer1 = dglnn.SAGEConv(in_dim, hidden_dim, 'pool')
        self.layer2 = dglnn.SAGEConv(hidden_dim, out_dim, 'pool')

    def forward(self, g, inputs):
        h = self.layer1(g, inputs)
        h = torch.relu(h)
        h = self.layer2(g, h)
        return h
```

```
class MLPPredictor(nn.Module):
    def __init__(self, out_feats, out_classes):
        super().__init__()
        self.W = nn.Linear(out_feats*2, out_classes)

    def apply_edges(self, edges):
        h_u = edges.src['h']
        h_v = edges.dst['h']
        score = self.W(torch.cat([h_u, h_v], 1))
        return {'score': score}

    def forward(self, graph, h):
        with graph.local_scope():
            graph.ndata['h'] = h
            graph.apply_edges(self.apply_edges)
            return graph.edata['score']
```

```
class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, num_classes):
        super().__init__()
        self.sage = GraphSAGE(in_features, hidden_features, out_features)
        self.pred = MLPPredictor(out_features, num_classes)

    def forward(self, g, node_feat, return_logits=False):
        h = self.sage(g, node_feat)
        logits = self.pred(g, h)

        return logits
```

# GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks

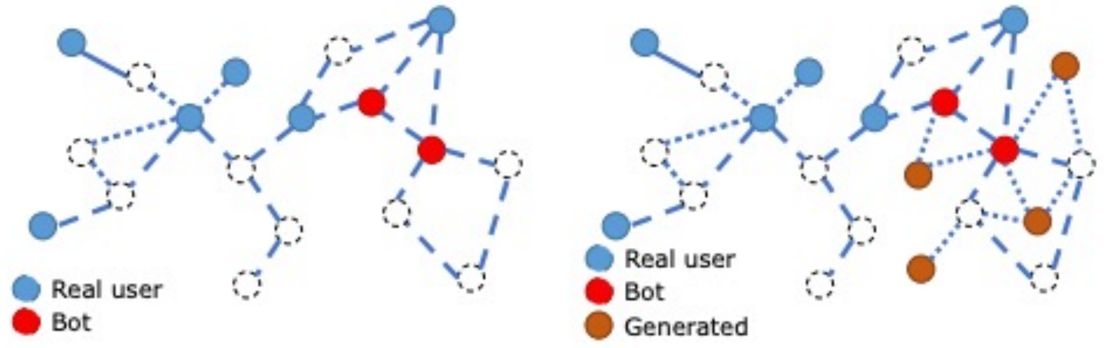
*WSDM ’21, March 8–12, 2021, Virtual Event, Israel*

***Tianxiang Zhao, Xiang Zhang, Suhang Wang***
{tkz5084,xzz89,szw494}@psu.edu
College of Information Science and Technology, Penn State University State College, The USA

https://github.com/TianxiangZhao/GraphSmote

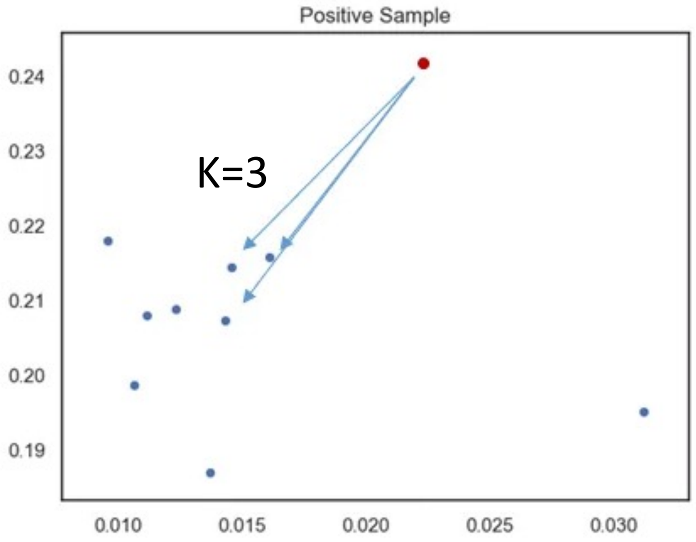
# GraphSMOTE

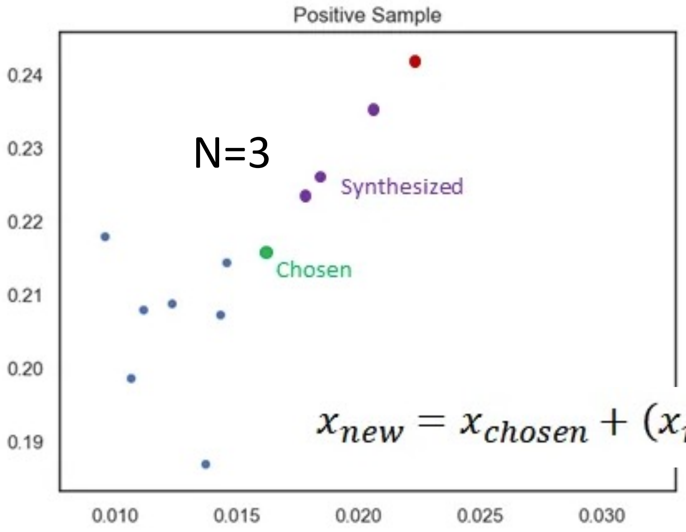
- **Task:**




(a) Bot detection task (b) After over-sampling

- **Synthesized Minority Oversampling Technique (SMOTE)**




- Used on i.i.d. data
- Doesn't consider the structure of the graph
- No edge connection between the nodes

$$x_{new} = x_{chosen} + (x_{nearest} - x_{chosen}) * \delta \; ; \; \delta \in [0,1]$$

# GraphSMOTE

- **Framework**
  - a GNN-based feature extractor
    → Use **GraphSAGE**
  - Synthetic Node Generation
    → Use **SMOTE** algorithm
  - Edge Generator
    → weighted inner product **decoder**
  - GNN Classifier (downstream task)

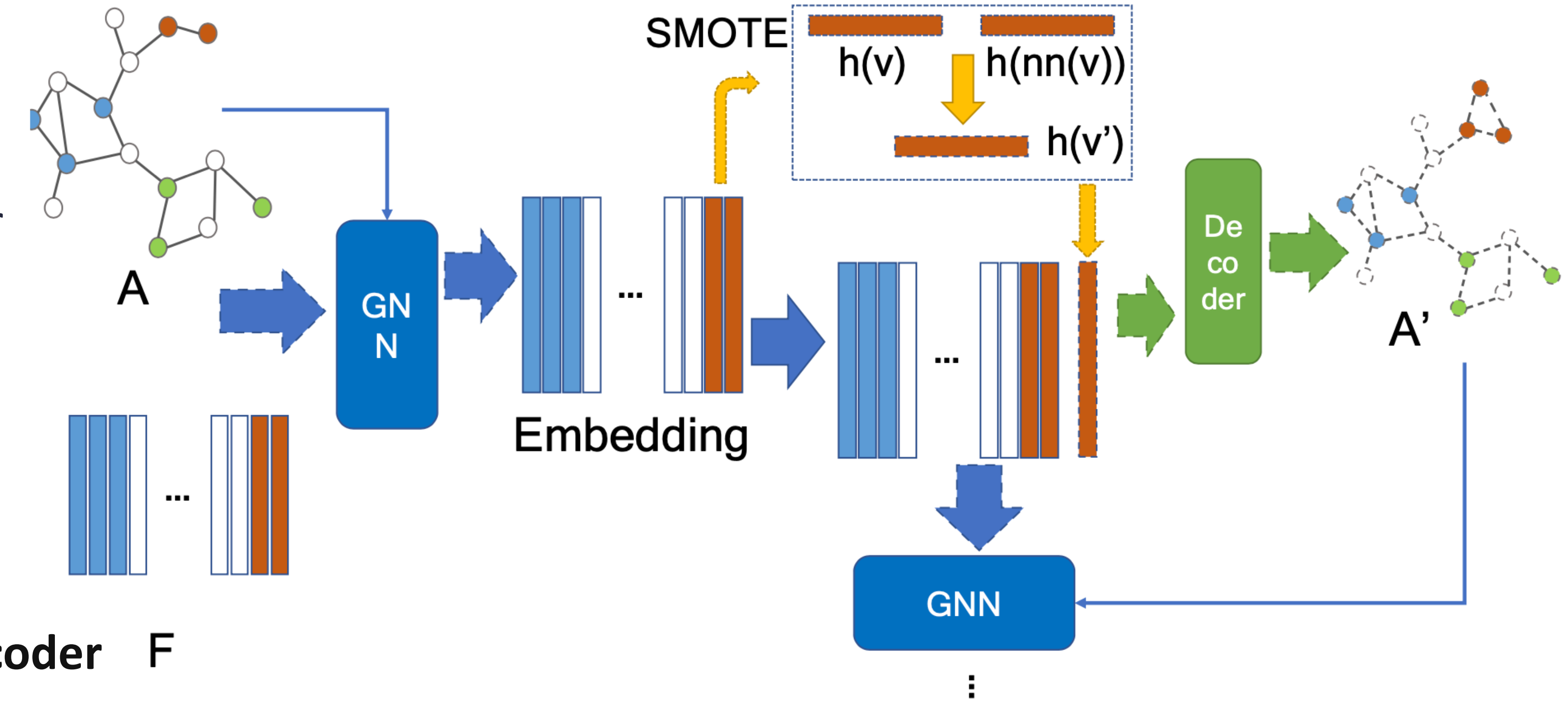



**Figure 2: Overview of the framework**