Deteksi Transfer Fraud pada Sistem Pembayaran dengan Graph Neural Network (GNN)

Pilot Analysis dalam rangka Hackathon Bank Indonesia 2024

Oleh : Tim JN and GNN - Jens Naki

1. Dengan GNN, diharapkan Bank Indonesia dapat mendeteksi aktivitas transfer fraud antar rekening dalam jaringan.
2. Pilot analysis dilakukan untuk menguji model sebelum diimplementasikan pada dataset yang sebenarnya.
3. Anonymized dataset yang digunakan adalah Elliptic Bitcoin Dataset (https://www.kaggle.com/datasets/ellipticco/elliptic-data-set). Dataset ini berisi informasi transfer Bitcoin antar alamat pada periode waktu tertentu. Sebagian transfer telah teridentifikasi sebagai fraud / illicit entities (penipuan, malware, organisasi teroris, ransomware, skema Ponzi, dll.). Sebagian transfer telah teridentifikasi sebagai non fraud / licit entities (Bitcoin exchange, wallet provider, miner, dll).
4. Model GNN yang digunakan adalah GAT dan GATv2 prebuilt dari Pytorch Geometric.
5. Code pada notebook ini dimodifikasi dari code Anthony Taing, Stanford CS224W GraphML Tutorials (https://medium.com/stanford-cs224w/fraud-detection-with-gat-edac49bda1a0).

Referensi:

1. Elliptic website, www.elliptic.co.
2. Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics, arXiv:1908.02591, 2019.
3. M. Weber G. Domeniconi J. Chen D. K. I. Weidele C. Bellei, T. Robinson, C. E. Leiserson, https://www.kaggle.com/ellipticco/elliptic-data-set, 2019.
4. Graph Attention Networks, Petar Veličković and Guillem Cucurull and Arantxa Casanova and Adriana Romero and Pietro Liò and Yoshua Bengio, arXiv:1710.10903, 2018.
5. PyG documentation: pytorch-geometric.readthedocs.io/en/latest/
6. https://medium.com/stanford-cs224w/fraud-detection-with-gat-edac49bda1a0

Instalasi package yang dibutuhkan:

1. Pytorch : https://pytorch.org/get-started/locally/
2. Pytorch Geometric: https://pytorch-geometric.readthedocs.io/en/latest/install/installation.html

In [20]:
```
import torch
```

In [21]:
```
import torch_geometric
```

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from torch.nn import Linear, LayerNorm, ReLU, Dropout
```

```
import torch.nn.functional as F
from torch_geometric.data import Data, DataLoader
from sklearn.metrics import accuracy_score,f1_score,roc_auc_score, precision_score, recall_score, confusion_matrix

import scipy.sparse as sp

import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from sklearn.model_selection import train_test_split

import torch.nn.functional as F
from sklearn.metrics import roc_auc_score
from torch.nn import Linear
from torch_geometric.nn import GATConv,GATv2Conv
```

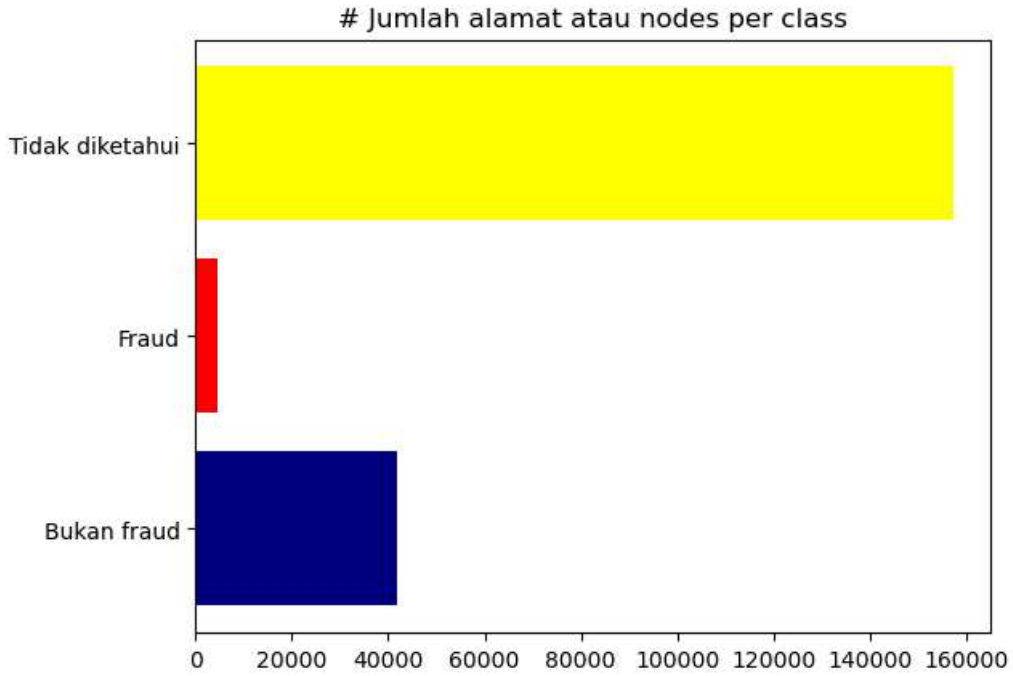
In [23]:
```
#Download data ke folder kita, menggunakan datasets dari Pytorch Geometric
#bisa juga didownload manual dari https://www.kaggle.com/datasets/ellipticco/elliptic-data-set

from torch_geometric.datasets import EllipticBitcoinDataset
dataset = EllipticBitcoinDataset(root="")
```

In [24]:
```
df_feature = pd.read_csv("./raw/elliptic_txs_features.csv",header=None)
df_edges = pd.read_csv("./raw/elliptic_txs_edgelist.csv")
df_classes =  pd.read_csv("./raw/elliptic_txs_classes.csv")
```

In [25]:
```
#Ubah format string jadi angka 0:Bukan fraud, 1:Fraud, 2:Tidak diketahui
df_classes["class"] = df_classes["class"].map({"unknown":2, "1":1, "2":0})
```

In [27]:
```
#Visualisasi proporsi transaksi NonFraud
group_class = df_classes.groupby("class").count()
plt.title ("# Jumlah alamat atau nodes per class")
plt.barh(["Bukan fraud","Fraud","Tidak diketahui"], group_class["txId"].values, color=["navy","red", "yellow"])
```

Out[27]:
```
<BarContainer object of 3 artists>
```



In [28]:
```
df_feature.head()
```

Out[28]:

	0	1	2	3	4	5	6	7	8	9	...	157	158	159	160	161	162	
0	230425980	1	-0.171469	-0.184668	-1.201369	-0.121970	-0.043875	-0.113002	-0.061584	-0.162097	...	-0.562153	-0.600999	1.461330	1.461369	0.018279	-0.087490	-0.13
1	5530458	1	-0.171484	-0.184668	-1.201369	-0.121970	-0.043875	-0.113002	-0.061584	-0.162112	...	0.947382	0.673103	-0.979074	-0.978556	0.018279	-0.087490	-0.13
2	232022460	1	-0.172107	-0.184668	-1.201369	-0.121970	-0.043875	-0.113002	-0.061584	-0.162749	...	0.670883	0.439728	-0.979074	-0.978556	-0.098889	-0.106715	-0.13
3	232438397	1	0.163054	1.963790	-0.646376	12.409294	-0.063725	9.782742	12.414558	-0.163645	...	-0.577099	-0.613614	0.241128	0.241406	1.072793	0.085530	-0.13
4	230460314	1	1.011523	-0.081127	-1.201369	1.153668	0.333276	1.312656	-0.061584	-0.163523	...	-0.511871	-0.400422	0.517257	0.579382	0.018279	0.277775	0.32

5 rows × 167 columns

In [29]:
```
df_edges.head()
```

Out[29]:

	txId1	txId2
0	230425980	5530458
1	232022460	232438397
2	230460314	230459870
3	230333930	230595899
4	232013274	232029206

In [30]:
```
df_classes.head()
```

Out[30]:

	txId	class
0	230425980	2
1	5530458	2
2	232022460	2
3	232438397	0
4	230460314	2

In [31]:
```
# Penggabungan features dengan classes
df_merge = df_feature.merge(df_classes, how="left", right_on="txId", left_on=0)
df_merge = df_merge.sort_values(0).reset_index(drop=True)
df_merge.head()
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	...	159	160	161	162	163	164	165
0	1076	48	-0.168500	0.270909	-0.091383	-0.046932	-0.043875	-0.029140	-0.061584	-0.163591	...	1.461330	1.461369	0.018279	0.470019	1.216796	1.151607	1.519700
1	2534	6	-0.170834	-0.131425	1.018602	0.028105	0.055376	0.054722	-0.061584	-0.163572	...	0.955101	0.459257	-0.098889	-0.087490	-0.099080	-0.122137	-0.379970
2	3181	34	1.305212	-0.210553	-1.756361	-0.121970	97.300650	-0.113002	-0.061584	1.348765	...	0.059948	0.113967	-0.098889	1.969527	0.037532	-0.131010	0.006994
3	3321	1	-0.169615	-0.184668	-1.201369	-0.121970	-0.043875	-0.113002	-0.061584	-0.160199	...	0.241128	0.241406	-0.098889	-0.087490	-0.084674	-0.140597	1.519700
4	3889	48	-0.086232	-0.101835	-0.646376	-0.121970	17.046997	-0.113002	-0.061584	-0.074885	...	0.082065	0.114773	-0.098889	8.948005	1.024948	-0.009570	-0.080708

5 rows × 169 columns

In [32]:
```
#setup mapping trans ID ke node ID
nodes = df_merge[0].values

map_id = {j:i for i,j in enumerate(nodes)} #mapping nodes ke index
```

```
#Membuat edge dataframe yang trans ID nya telah dimap ke nodeID

edges = df_edges.copy()
edges.txId1 = edges.txId1.map(map_id) #get nodes idx1 from edges list and filtered data
edges.txId2 = edges.txId2.map(map_id)

edges = edges.astype(int)

edge_index = np.array(edges.values).T #convert into an array
edge_index = torch.tensor(edge_index, dtype=torch.long).contiguous() #create tensor

print(f"shape of edge index is {edge_index.shape}")
edge_index
```

```
shape of edge index is torch.Size([2, 234355])
```

Out[32]:
```
tensor([[138670, 141325, 139232,  ..., 100420,  54833, 101159],
        [  4142, 142201, 139223,  ..., 100419,  81951, 101163]])
```

In [33]:
```
#Membuat weights tensor dengan bentuk yang sama dengan edge_index
weights = torch.tensor([1] * edge_index.shape[1], dtype=torch.double)
```

In [34]:
```
#Define labels
labels = df_merge["class"].values
print("labels", np.unique(labels))
labels
```

```
labels [0 1 2]
```

Out[34]:
```
array([2, 0, 0, ..., 2, 2, 2], dtype=int64)
```

In [35]:
```
#Passing node features ke model

node_features = df_merge.drop(["txId"], axis=1).copy()
print("unique=",node_features["class"].unique())

#mempertahankan IDs yang sudah diketahui dan tidak
classified_idx = node_features["class"].loc[node_features["class"]!=2].index
unclassified_idx = node_features["class"].loc[node_features["class"]==2].index

#filter label fraud dan nonfraud
classified_fraud_idx = node_features["class"].loc[node_features["class"]==1].index
classified_nonfraud_idx = node_features["class"].loc[node_features["class"]==0].index

#Drop kolom yang tidak dibutuhkan, 0 = transID, 1 = time period, class = labels
node_features = node_features.drop(columns = [0,1,"class"])

#Konvert tensor
node_features_t = torch.tensor(np.array(node_features.values, dtype=np.double), dtype=torch.double)
node_features_t
```

```
unique= [2 0 1]
```

Out[35]:
```
tensor([[-0.1685,  0.2709, -0.0914,  ...,  1.1516,  1.5197,  1.5214],
        [-0.1708, -0.1314,  1.0186,  ..., -0.1221, -0.3800, -0.3793],
        [ 1.3052, -0.2106, -1.7564,  ..., -0.1310,  0.0070,  0.0178],
        ...,
        [-0.1727, -0.1588, -1.2014,  ..., -0.2698, -0.1206, -0.1198],
        [-0.1727, -0.1588, -1.2014,  ..., -0.2698, -0.1206, -0.1198],
        [-0.1433, -0.1588, -1.2014,  ..., -0.0975, -0.1206, -0.1198]],
       dtype=torch.float64)
```

In [36]:
```
# Melihat hasil node_features
node_features
```

Out[36]:

	2	3	4	5	6	7	8	9	10	11	...	157	158	159	160	161	
0	-0.168500	0.270909	-0.091383	-0.046932	-0.043875	-0.029140	-0.061584	-0.163591	-0.164980	-0.009283	...	0.073047	-0.039637	1.461330	1.461369	0.018279	0.47(
1	-0.170834	-0.131425	1.018602	0.028105	0.055376	0.054722	-0.061584	-0.163572	-0.167757	-0.038545	...	1.228858	0.379357	0.955101	0.459257	-0.098889	-0.08
2	1.305212	-0.210553	-1.756361	-0.121970	97.300650	-0.113002	-0.061584	1.348765	1.321754	-0.049707	...	1.348450	1.590664	0.059948	0.113967	-0.098889	1.96
3	-0.169615	-0.184668	-1.201369	-0.121970	-0.043875	-0.113002	-0.061584	-0.160199	-0.166062	-0.049707	...	-0.577099	-0.500080	0.241128	0.241406	-0.098889	-0.08
4	-0.086232	-0.101835	-0.646376	-0.121970	17.046997	-0.113002	-0.061584	-0.074885	-0.081943	-0.049707	...	0.501062	0.362510	0.082065	0.114773	-0.098889	8.94
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
203764	-0.172978	-0.172527	0.463609	-0.121970	-0.043875	-0.113002	-0.061584	-0.163640	-0.169455	-0.049707	...	-0.577099	-0.600999	0.241128	0.241406	0.018279	-0.08
203765	-0.172669	-0.158783	-1.201369	-0.121970	-0.063725	-0.113002	-0.061584	-0.163323	-0.169142	-0.049707	...	-0.577099	-0.626229	0.241128	0.241406	-0.216057	-0.12
203766	-0.172669	-0.158783	-1.201369	-0.121970	-0.063725	-0.113002	-0.061584	-0.163323	-0.169142	-0.049707	...	-0.577099	-0.626229	0.241128	0.241406	-0.216057	-0.12
203767	-0.172669	-0.158783	-1.201369	-0.121970	-0.063725	-0.113002	-0.061584	-0.163323	-0.169142	-0.049707	...	-0.577099	-0.626229	0.241128	0.241406	-0.216057	-0.12
203768	-0.143292	-0.158783	-1.201369	-0.121970	-0.043875	-0.113002	-0.061584	-0.133266	-0.139507	-0.049707	...	-0.577099	-0.613614	0.241128	0.241406	0.018279	-0.08

203769 rows × 165 columns

In [37]:
```
#membuat data training dengan test size = 15%
train_idx, valid_idx = train_test_split(classified_idx.values, test_size=0.15)
lentrain = len(train_idx)
lenvalid = len(valid_idx)
print(f"train_idx size {lentrain}")
print(f"test_idx size {lenvalid}")
```

```
train_idx size 39579
test_idx size 6985
```

In [38]:
```
#membuat Pytorch Geometric dataset
data_train = Data(x=node_features_t, edge_index=edge_index, edge_attr=weights,
                  y=torch.tensor(labels, dtype=torch.double))
```

```
#masukan train dan valid idx
data_train.train_idx = train_idx
data_train.valid_idx = valid_idx
data_train.test_idx = unclassified_idx
```

In [39]:
```
#import package yang dibutuhkan
import torch
import torch.nn as nn
import torch.nn.functional as F

import torch_geometric.nn as pyg_nn
import torch_geometric.utils as pyg_utils

from torch import Tensor
from typing import Union, Tuple, Optional
from torch_geometric.typing import (OptPairTensor, Adj, Size, NoneType, OptTensor)

from torch.nn import Parameter, Linear
from torch_geometric.nn.conv import MessagePassing
from torch_geometric.utils import remove_self_loops, add_self_loops, softmax, degree

from torch_geometric.nn import GATConv, GATv2Conv
import pickle
```

In [41]:
```
#Model GAT Prebuilt dari Pytorch Geometric

class GAT(torch.nn.Module):
    def __init__(self,input_dim, hidden_dim, output_dim,args):
        super(GAT, self).__init__()
        #use our gat message passing
        self.conv1 = GATConv(input_dim, hidden_dim, heads=args['heads'])
        self.conv2 = GATConv(args['heads'] * hidden_dim, hidden_dim, heads=args['heads'])

        self.post_mp = nn.Sequential(
            nn.Linear(args['heads'] * hidden_dim, hidden_dim), nn.Dropout(args['dropout'] ),
            nn.Linear(hidden_dim, output_dim))

    def forward(self, data, adj=None):
        x, edge_index = data.x, data.edge_index
        # Layer 1
        x = self.conv1(x, edge_index)
        x = F.dropout(F.relu(x), p=args['dropout'], training=self.training)
        # Layer 2
        x = self.conv2(x, edge_index)
        x = F.dropout(F.relu(x), p=args['dropout'], training=self.training)
        # MLP output
        x = self.post_mp(x)
        return F.sigmoid(x)
```

In [42]:
```
#Model GATv2 dari Pytorch Geometric
class GATv2(torch.nn.Module):
    def __init__(self,input_dim, hidden_dim, output_dim,args):
        super(GATv2, self).__init__()
        #use our gat message passing
        self.conv1 = GATv2Conv(input_dim, hidden_dim, heads=args['heads'])
        self.conv2 = GATv2Conv(args['heads'] * hidden_dim, hidden_dim, heads=args['heads'])

        self.post_mp = nn.Sequential(
            nn.Linear(args['heads'] * hidden_dim, hidden_dim), nn.Dropout(args['dropout'] ),
            nn.Linear(hidden_dim, output_dim))

    def forward(self, data, adj=None):
        x, edge_index = data.x, data.edge_index
        # Layer 1
        x = self.conv1(x, edge_index)
        x = F.dropout(F.relu(x), p=args['dropout'], training=self.training)
        # Layer 2
        x = self.conv2(x, edge_index)
        x = F.dropout(F.relu(x), p=args['dropout'], training=self.training)
        # MLP output
        x = self.post_mp(x)
        return F.sigmoid(x)
```

In [49]:
```
#Model Training
class GnnTrainer(object):

  def __init__(self, model):
    self.model = model
    self.metric_manager = MetricManager(modes=["train", "val"])

  def train(self, data_train, optimizer, criterion, scheduler, args):

    self.data_train = data_train
    for epoch in range(args['epochs']):
        self.model.train()
        optimizer.zero_grad()
        out = self.model(data_train)

        out = out.reshape((data_train.x.shape[0]))
        loss = criterion(out[data_train.train_idx], data_train.y[data_train.train_idx])
        ## Metric calculations
        # train data
        target_labels = data_train.y.detach().cpu().numpy()[data_train.train_idx]
        pred_scores = out.detach().cpu().numpy()[data_train.train_idx]
        train_acc, train_f1,train_f1macro, train_aucroc, train_recall, train_precision, train_cm = self.metric_manager.store_metrics("train", pred_scores, target_


        ## Training Step
        loss.backward()
```

```
        optimizer.step()

        # validation data
        self.model.eval()
        target_labels = data_train.y.detach().cpu().numpy()[data_train.valid_idx]
        pred_scores = out.detach().cpu().numpy()[data_train.valid_idx]
        val_acc, val_f1,val_f1macro, val_aucroc, val_recall, val_precision, val_cm = self.metric_manager.store_metrics("val", pred_scores, target_labels)

        if epoch%5 == 0:
          print("epoch: {} - loss: {:.4f} - accuracy train: {:.4f} -accuracy valid: {:.4f}  - val roc: {:.4f}  - val f1micro: {:.4f}".format(epoch, loss.item(), t

  # To predict labels
  def predict(self, data=None, unclassified_only=True, threshold=0.5):
    # evaluate model:
    self.model.eval()
    if data is not None:
      self.data_train = data

    out = self.model(self.data_train)
    out = out.reshape((self.data_train.x.shape[0]))

    if unclassified_only:
      pred_scores = out.detach().cpu().numpy()[self.data_train.test_idx]
    else:
      pred_scores = out.detach().cpu().numpy()

    pred_labels = pred_scores > threshold

    return {"pred_scores":pred_scores, "pred_labels":pred_labels}

  # To save metrics
  def save_metrics(self, save_name, path="./save/"):
    file_to_store = open(path + save_name, "wb")
    pickle.dump(self.metric_manager, file_to_store)
    file_to_store.close()

  # To save model
  def save_model(self, save_name, path="./save/"):
    torch.save(self.model.state_dict(), path + save_name)
```

In [50]:

```
#Metric Manager
class MetricManager(object):
  def __init__(self, modes=["train", "val"]):

    self.output = {}

    for mode in modes:
      self.output[mode] = {}
      self.output[mode]["accuracy"] = []
      self.output[mode]["f1micro"] = []
      self.output[mode]["f1macro"] = []
```

```
      self.output[mode]["aucroc"] = []
      #new
      self.output[mode]["precision"] = []
      self.output[mode]["recall"] = []
      self.output[mode]["cm"] = []

  def store_metrics(self, mode, pred_scores, target_labels, threshold=0.5):

    # calculate metrics
    pred_labels = pred_scores > threshold
    accuracy = accuracy_score(target_labels, pred_labels)
    f1micro = f1_score(target_labels, pred_labels,average='micro')
    f1macro = f1_score(target_labels, pred_labels,average='macro')
    aucroc = roc_auc_score(target_labels, pred_scores)
    #new
    recall = recall_score(target_labels, pred_labels)
    precision = precision_score(target_labels, pred_labels)
    cm = confusion_matrix(target_labels, pred_labels)

    # Collect results
    self.output[mode]["accuracy"].append(accuracy)
    self.output[mode]["f1micro"].append(f1micro)
    self.output[mode]["f1macro"].append(f1macro)
    self.output[mode]["aucroc"].append(aucroc)
    #new
    self.output[mode]["recall"].append(recall)
    self.output[mode]["precision"].append(precision)
    self.output[mode]["cm"].append(cm)

    return accuracy, f1micro,f1macro, aucroc,recall,precision,cm

  # Get best results
  def get_best(self, metric, mode="val"):

    # Get best results index
    best_results = {}
    i = np.array(self.output[mode][metric]).argmax()

    # Output
    for m in self.output[mode].keys():
      best_results[m] = self.output[mode][m][i]

    return best_results
```

In [51]:

```
# Setting training argument
args={"epochs":100,
      'lr':0.01,
      'weight_decay':1e-5,
      'prebuild':True,
      'heads':2,
      'hidden_dim': 128,
```

```
      'dropout': 0.5
      }

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [55]:
```
# Model selector GAT atau GATv2
net = "GAT"

if net == "GAT":
    if args['prebuild']==True:
      model = GAT(data_train.num_node_features, args['hidden_dim'], 1, args)
      print("Prebuilt GAT from PyG ")
elif net == "GATv2":
    # args['heads'] = 1
    if args['prebuild']==True:
      model = GATv2(data_train.num_node_features, args['hidden_dim'], 1, args)
      print("Prebuilt GATv2 from PyG ")

model.double().to(device)
```

```
Prebuilt GAT from PyG
```

Out[55]:
```
GAT(
  (conv1): GATConv(165, 128, heads=2)
  (conv2): GATConv(256, 128, heads=2)
  (post_mp): Sequential(
    (0): Linear(in_features=256, out_features=128, bias=True)
    (1): Dropout(p=0.5, inplace=False)
    (2): Linear(in_features=128, out_features=1, bias=True)
  )
)
```

In [33]:
```
#Running Model GAT
data_train = data_train.to(device)

# Setup training settings
optimizer = torch.optim.Adam(model.parameters(), lr=args['lr'], weight_decay=args['weight_decay'])
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')
criterion = torch.nn.BCELoss()

# Train
gnn_trainer_gat = GnnTrainer(model)
gnn_trainer_gat.train(data_train, optimizer, criterion, scheduler, args)

gnn_trainer_gat.save_metrics("GATprebuilt.results", path="./")
gnn_trainer_gat.save_model("GATprebuilt.pth", path="./")
```

```
epoch: 0 - loss: 0.0409 - accuracy train: 0.9871 -accuracy valid: 0.9770  - val roc: 0.9810  - val f1micro: 0.9770
epoch: 5 - loss: 0.0853 - accuracy train: 0.9721 -accuracy valid: 0.9721  - val roc: 0.9740  - val f1micro: 0.9721
epoch: 10 - loss: 0.0694 - accuracy train: 0.9750 -accuracy valid: 0.9741  - val roc: 0.9767  - val f1micro: 0.9741
epoch: 15 - loss: 0.0697 - accuracy train: 0.9801 -accuracy valid: 0.9761  - val roc: 0.9792  - val f1micro: 0.9761
epoch: 20 - loss: 0.0572 - accuracy train: 0.9829 -accuracy valid: 0.9765  - val roc: 0.9816  - val f1micro: 0.9765
epoch: 25 - loss: 0.0520 - accuracy train: 0.9842 -accuracy valid: 0.9768  - val roc: 0.9790  - val f1micro: 0.9768
epoch: 30 - loss: 0.0466 - accuracy train: 0.9864 -accuracy valid: 0.9791  - val roc: 0.9806  - val f1micro: 0.9791
epoch: 35 - loss: 0.0438 - accuracy train: 0.9867 -accuracy valid: 0.9780  - val roc: 0.9799  - val f1micro: 0.9780
epoch: 40 - loss: 0.0406 - accuracy train: 0.9877 -accuracy valid: 0.9791  - val roc: 0.9799  - val f1micro: 0.9791
epoch: 45 - loss: 0.0391 - accuracy train: 0.9882 -accuracy valid: 0.9778  - val roc: 0.9792  - val f1micro: 0.9778
epoch: 50 - loss: 0.0362 - accuracy train: 0.9884 -accuracy valid: 0.9784  - val roc: 0.9784  - val f1micro: 0.9784
epoch: 55 - loss: 0.0317 - accuracy train: 0.9897 -accuracy valid: 0.9778  - val roc: 0.9781  - val f1micro: 0.9778
epoch: 60 - loss: 0.0358 - accuracy train: 0.9892 -accuracy valid: 0.9791  - val roc: 0.9807  - val f1micro: 0.9791
epoch: 65 - loss: 0.0350 - accuracy train: 0.9897 -accuracy valid: 0.9781  - val roc: 0.9806  - val f1micro: 0.9781
epoch: 70 - loss: 0.0349 - accuracy train: 0.9893 -accuracy valid: 0.9782  - val roc: 0.9782  - val f1micro: 0.9782
epoch: 75 - loss: 0.0334 - accuracy train: 0.9900 -accuracy valid: 0.9787  - val roc: 0.9769  - val f1micro: 0.9787
epoch: 80 - loss: 0.0310 - accuracy train: 0.9909 -accuracy valid: 0.9778  - val roc: 0.9802  - val f1micro: 0.9778
epoch: 85 - loss: 0.0303 - accuracy train: 0.9902 -accuracy valid: 0.9784  - val roc: 0.9797  - val f1micro: 0.9784
epoch: 90 - loss: 0.0319 - accuracy train: 0.9897 -accuracy valid: 0.9791  - val roc: 0.9784  - val f1micro: 0.9791
epoch: 95 - loss: 0.0298 - accuracy train: 0.9904 -accuracy valid: 0.9787  - val roc: 0.9779  - val f1micro: 0.9787
```

In [34]:

```python
#Running model GATv2

model = GATv2(data_train.num_node_features, args['hidden_dim'], 1, args)
model.double().to(device)
# Push data to GPU
data_train = data_train.to(device)

# Setup training settings
optimizer = torch.optim.Adam(model.parameters(), lr=args['lr'], weight_decay=args['weight_decay'])
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')
criterion = torch.nn.BCELoss()
# Train
gnn_trainer_gatv2 = GnnTrainer(model)
gnn_trainer_gatv2.train(data_train, optimizer, criterion, scheduler, args)

gnn_trainer_gatv2.save_metrics("GATv2prebuilt.results", path="./")
gnn_trainer_gatv2.save_model("GATv2prebuilt.pth", path="./")
```

```
epoch: 0 - loss: 0.7775 - accuracy train: 0.5748 -accuracy valid: 0.5771  - val roc: 0.4718  - val f1micro: 0.5771
epoch: 5 - loss: 0.2524 - accuracy train: 0.9014 -accuracy valid: 0.9081  - val roc: 0.8897  - val f1micro: 0.9081
epoch: 10 - loss: 0.2180 - accuracy train: 0.9014 -accuracy valid: 0.9081  - val roc: 0.9210  - val f1micro: 0.9081
epoch: 15 - loss: 0.1767 - accuracy train: 0.9013 -accuracy valid: 0.9079  - val roc: 0.9472  - val f1micro: 0.9079
epoch: 20 - loss: 0.1605 - accuracy train: 0.9014 -accuracy valid: 0.9081  - val roc: 0.9600  - val f1micro: 0.9081
epoch: 25 - loss: 0.1460 - accuracy train: 0.9032 -accuracy valid: 0.9097  - val roc: 0.9682  - val f1micro: 0.9097
epoch: 30 - loss: 0.1335 - accuracy train: 0.9529 -accuracy valid: 0.9553  - val roc: 0.9704  - val f1micro: 0.9553
epoch: 35 - loss: 0.1236 - accuracy train: 0.9676 -accuracy valid: 0.9671  - val roc: 0.9740  - val f1micro: 0.9671
epoch: 40 - loss: 0.1159 - accuracy train: 0.9718 -accuracy valid: 0.9705  - val roc: 0.9754  - val f1micro: 0.9705
epoch: 45 - loss: 0.1034 - accuracy train: 0.9742 -accuracy valid: 0.9739  - val roc: 0.9783  - val f1micro: 0.9739
epoch: 50 - loss: 0.0975 - accuracy train: 0.9758 -accuracy valid: 0.9755  - val roc: 0.9763  - val f1micro: 0.9755
epoch: 55 - loss: 0.0875 - accuracy train: 0.9780 -accuracy valid: 0.9764  - val roc: 0.9774  - val f1micro: 0.9764
epoch: 60 - loss: 0.0787 - accuracy train: 0.9800 -accuracy valid: 0.9757  - val roc: 0.9807  - val f1micro: 0.9757
epoch: 65 - loss: 0.0709 - accuracy train: 0.9811 -accuracy valid: 0.9744  - val roc: 0.9797  - val f1micro: 0.9744
epoch: 70 - loss: 0.0643 - accuracy train: 0.9818 -accuracy valid: 0.9745  - val roc: 0.9796  - val f1micro: 0.9745
epoch: 75 - loss: 0.0587 - accuracy train: 0.9828 -accuracy valid: 0.9754  - val roc: 0.9790  - val f1micro: 0.9754
epoch: 80 - loss: 0.0531 - accuracy train: 0.9833 -accuracy valid: 0.9759  - val roc: 0.9791  - val f1micro: 0.9759
epoch: 85 - loss: 0.0488 - accuracy train: 0.9852 -accuracy valid: 0.9745  - val roc: 0.9771  - val f1micro: 0.9745
epoch: 90 - loss: 0.0481 - accuracy train: 0.9860 -accuracy valid: 0.9759  - val roc: 0.9807  - val f1micro: 0.9759
epoch: 95 - loss: 0.0431 - accuracy train: 0.9865 -accuracy valid: 0.9771  - val roc: 0.9806  - val f1micro: 0.9771
```

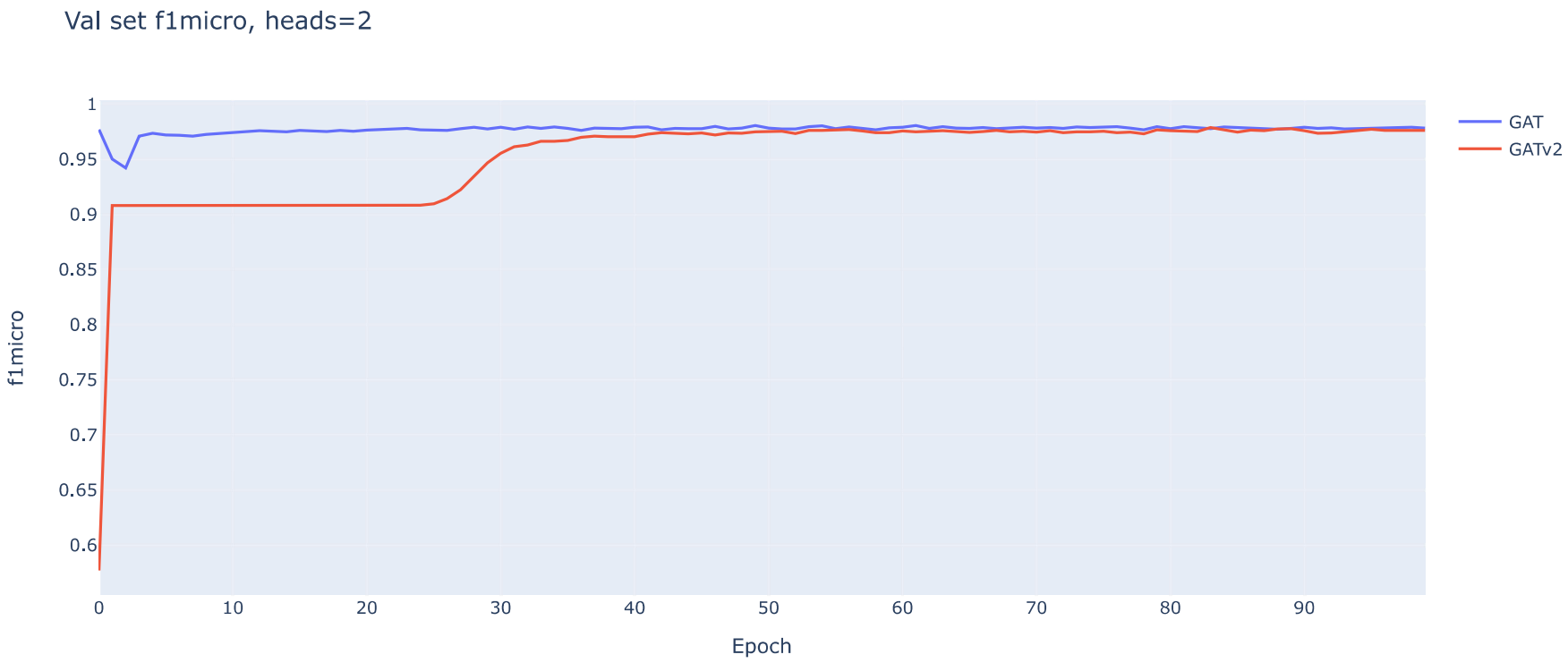
In [11]:
```
# Mengambil hasil running model yang sudah disimpan
fn="GATprebuilt.results"
mmGATprebuilt = pickle.load(open("./" + fn, "rb"))
mmGATprebuilt.get_best("aucroc")

fn="GATv2prebuilt.results"
mmGATv2prebuilt = pickle.load(open("./" + fn, "rb"))
mmGATv2prebuilt.get_best("aucroc")
```

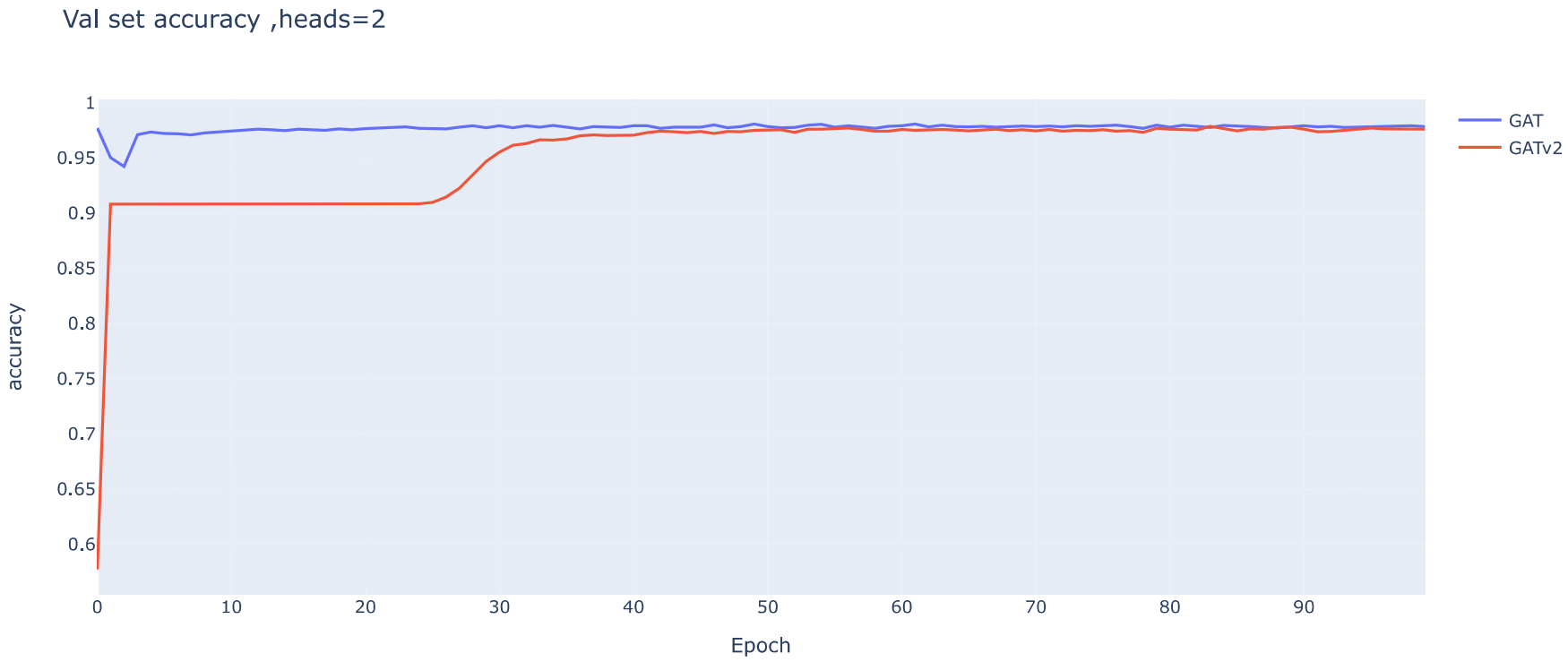
Out[11]:
```
{'accuracy': 0.9786685755189692,
 'f1micro': 0.9786685755189692,
 'f1macro': 0.9351505498088628,
 'aucroc': 0.9814292302501396,
 'precision': 0.8969404186795491,
 'recall': 0.867601246105919,
 'cm': array([[6279,   64],
        [  85,  557]], dtype=int64)}
```

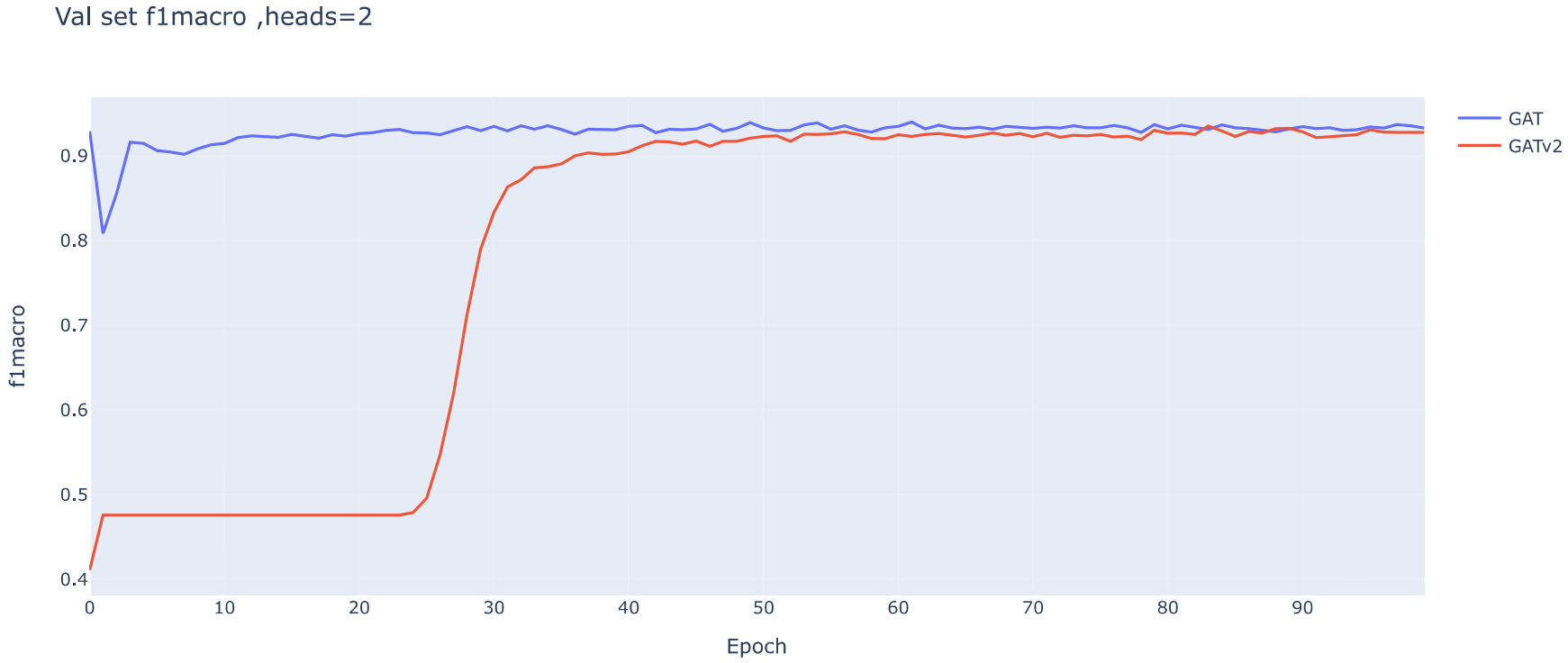
In [12]:
```
import plotly.express as px
import plotly.graph_objects as go

# Komparasi akurasi validasi
def plot_training_comp(metric_manager_list, names, metric="aucroc", version="val",title="Val set accuracy comparison"):

  fig = go.Figure()
  fig = fig.update_layout(title=title,
                          xaxis_title="Epoch",
                          yaxis_title=metric)

  for i, metric_manager in enumerate(metric_manager_list):
```

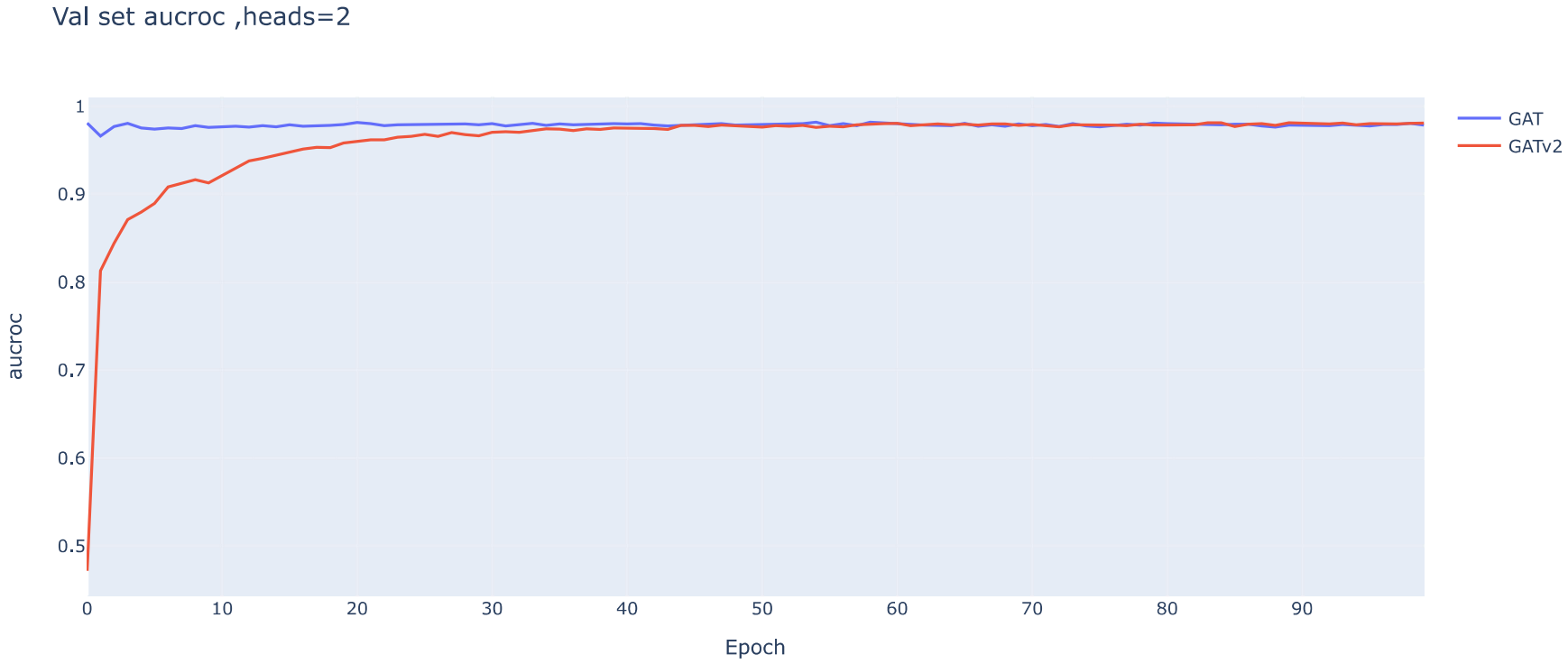
```
    epochs = np.arange(len(metric_manager.output[version][metric]))
    fig.add_trace(go.Scatter(x=epochs,
                             y=metric_manager.output[version][metric],
                             name=names[i]))
  fig.show()

# Load saved outputs
def load_results(fn):
  mm1 = pickle.load(open("./" + fn, "rb"))
  return mm1
```

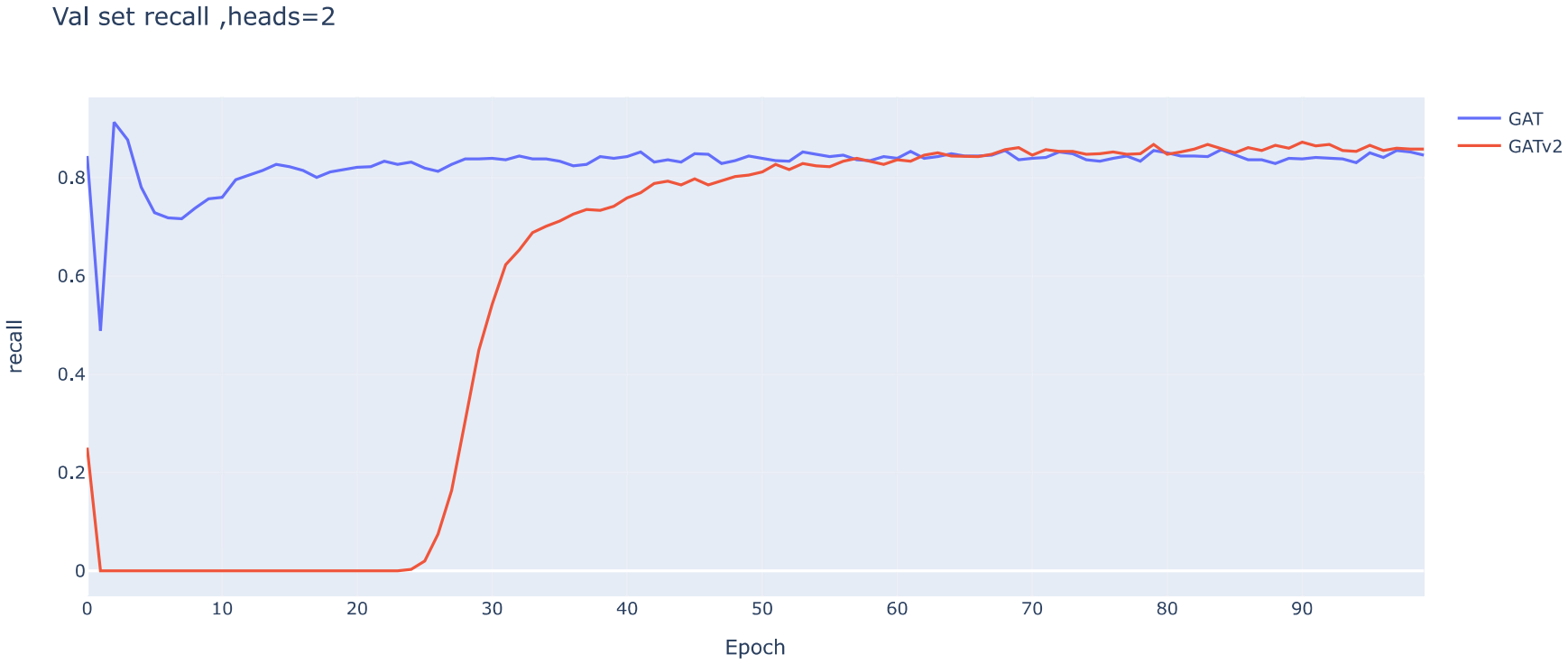
In [13]:
```
# Visualisasi komparasi hasil model GAT vs GATv2
plot_training_comp([mmGATprebuilt,mmGATv2prebuilt], ["GAT" ,"GATv2" ],"f1micro",title="Val set f1micro, heads=2")
plot_training_comp([mmGATprebuilt,mmGATv2prebuilt], ["GAT" ,"GATv2" ],"accuracy",title="Val set accuracy ,heads=2")
plot_training_comp([mmGATprebuilt,mmGATv2prebuilt], ["GAT" ,"GATv2" ],"f1macro",title="Val set f1macro ,heads=2")
plot_training_comp([mmGATprebuilt,mmGATv2prebuilt], ["GAT" ,"GATv2" ],"aucroc",title="Val set aucroc ,heads=2")
plot_training_comp([mmGATprebuilt,mmGATv2prebuilt], ["GAT" ,"GATv2" ],"recall",title="Val set recall ,heads=2")
plot_training_comp([mmGATprebuilt,mmGATv2prebuilt], ["GAT" ,"GATv2" ],"precision",title="Val set precision ,heads=2")
```

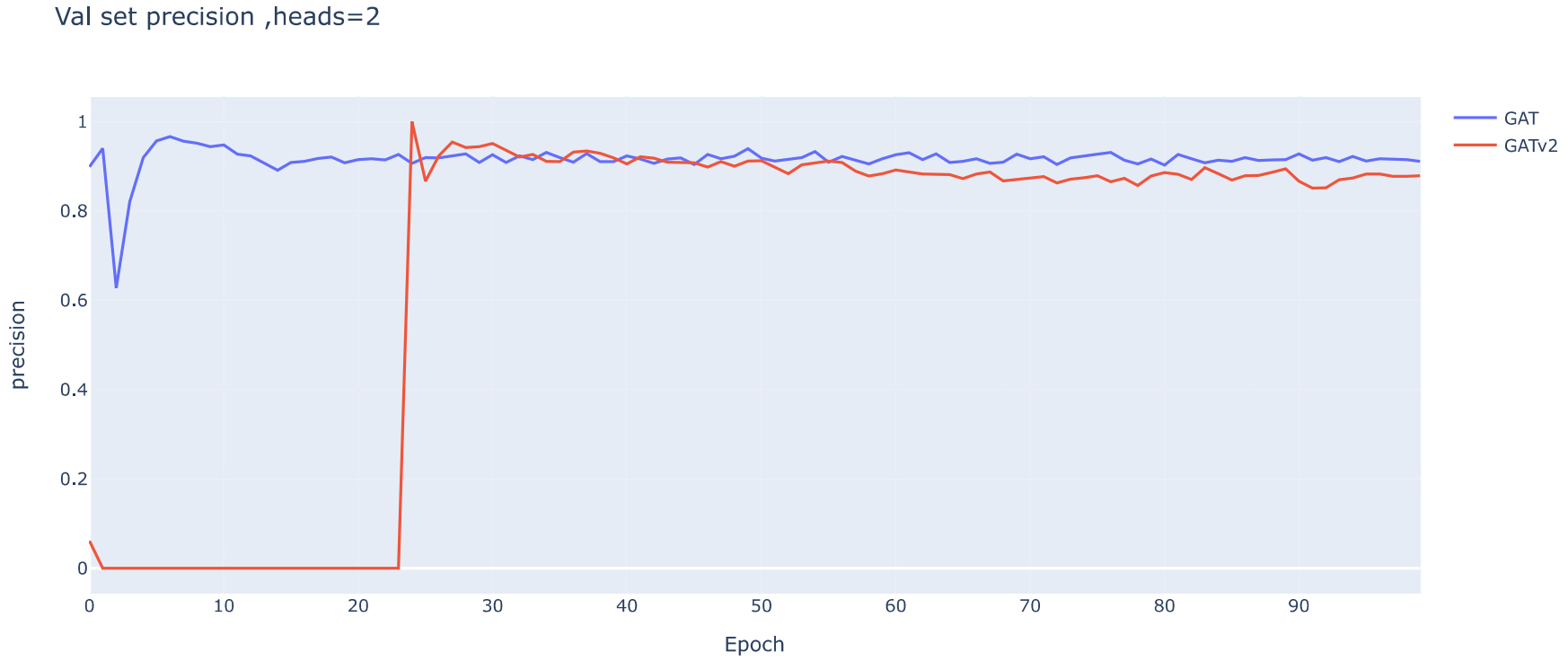



Val set accuracy ,heads=2
accuracy
Epoch
GAT
GATv2
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0
10
20
30
40
50
60
70
80
90


Val set f1macro ,heads=2
f1macro
Epoch
GAT
GATv2
0.4
0.5
0.6
0.7
0.8
0.9
0
10
20
30
40
50
60
70
80
90


Val set aucroc ,heads=2
aucroc
Epoch
GAT
GATv2
1
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50
60
70
80
90


Val set recall ,heads=2
recall
Epoch
GAT
GATv2
0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
70
80
90

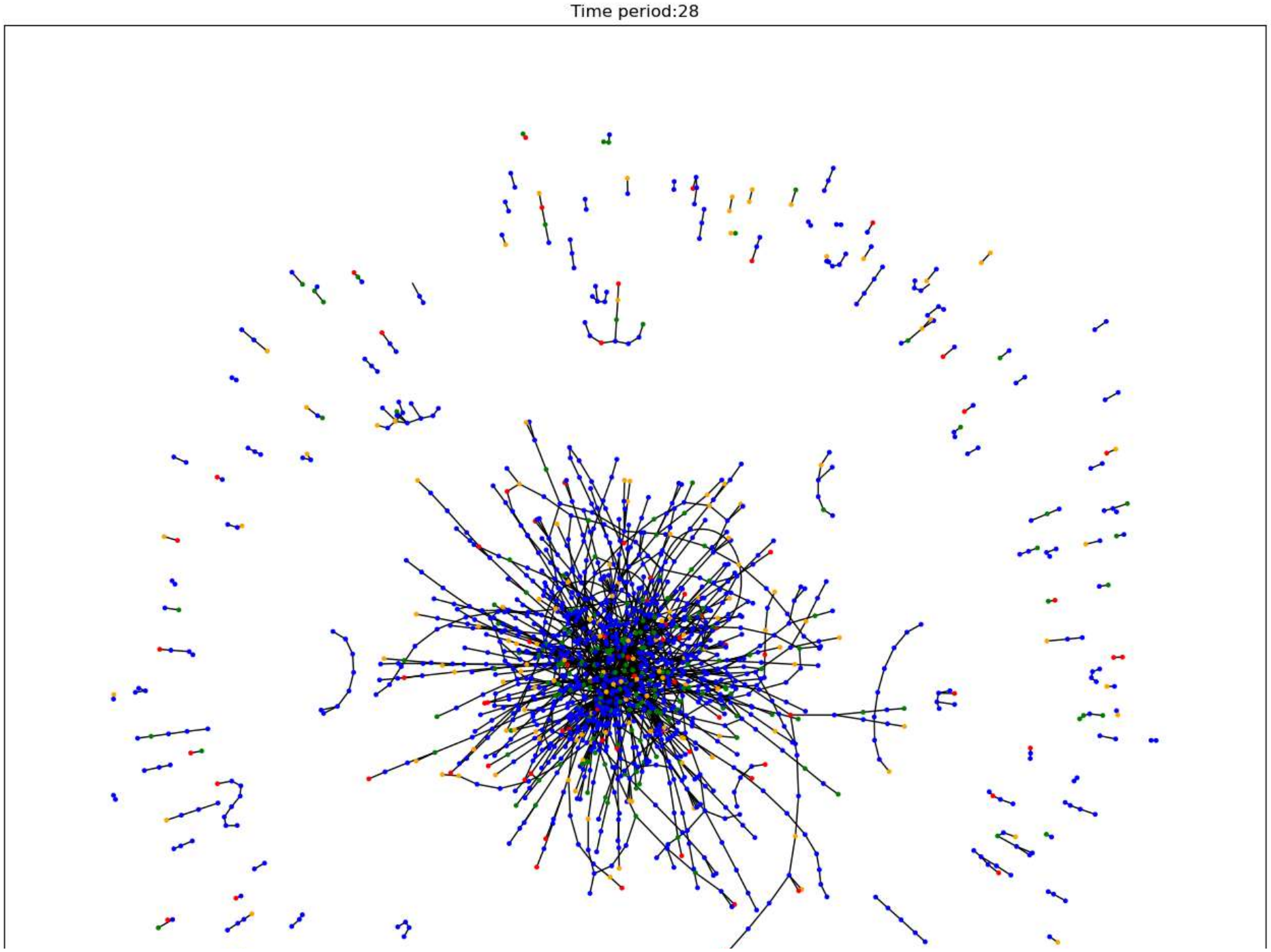


In [52]:
```
#Visualisasi Graph
import networkx as nx
import matplotlib.pyplot as plt

# Misalkan kita ambil model GATv2
m1 = GATv2(data_train.num_node_features, args['hidden_dim'], 1, args).to(device).double()
m1.load_state_dict(torch.load("./GATv2prebuilt.pth"))
gnn_t2 = GnnTrainer(m1)
output = gnn_t2.predict(data=data_train, unclassified_only=False)
output
```

Out[52]:
```
{'pred_scores': array([2.62891543e-03, 3.46962787e-07, 0.00000000e+00, ...,
        4.96969057e-64, 2.54770454e-02, 2.06414141e-05]),
 'pred_labels': array([False, False, False, ..., False, False, False])}
```

In [54]:
```
# Mengambil satu periode waktu
time_period = 28
sub_node_list = df_merge.index[df_merge.loc[:, 1] == time_period].tolist()
```

```
# Mengambil daftar edges untuk periode watu yang sudah dipilih
edge_tuples = []
for row in data_train.edge_index.view(-1, 2).numpy():
  if (row[0] in sub_node_list) | (row[1] in sub_node_list):
    edge_tuples.append(tuple(row))
len(edge_tuples)

# Mengambil hasil prediksi untuk periode waktu yang sudah dipilih
# Sudah teridentifikasi: merah = fraud, hijau = non fraud
# Hasil prediksi model: orange = diprediksi fraud, biru = diprediksi non fraud
node_color = []
for node_id in sub_node_list:
  if node_id in classified_fraud_idx: #
     label = "red" # fraud
  elif node_id in classified_nonfraud_idx:
     label = "green" # not fraud
  else:
    if output['pred_labels'][node_id]:
      label = "orange" # Predicted fraud
    else:
      label = "blue" # Not fraud predicted

  node_color.append(label)

# Membuat graph dengan networkX
G = nx.Graph()
G.add_edges_from(edge_tuples)

# Plot graph
plt.figure(3,figsize=(16,16))
plt.title("Time period:"+str(time_period))
nx.draw_networkx(G, nodelist=sub_node_list, node_color=node_color, node_size=6, with_labels=False)
```


Time period:28