National College of Ireland

# Configuration Manual for Enhancing Cryptocurrency Fraud Detection: A Hybrid Model Combining Transformers and Graph Neural Networks

MSc Research Project
Programme Name: Data Analytics

Sadda Rama Subba Reddy
Student ID: x23294850

School of Computing
National College of Ireland

Supervisor: John Kelly

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

**Student Name:** Sadda Rama Subba Reddy

**Student ID:** X23294850

**Programme:** Msc in Data Analytics **Year:** 2025

**Module:** Research Project

**Lecturer:** John Kelly

**Submission Due Date:** 11/08/2025

**Project Title:** A Hybrid Model Combining Transformers and Graph Neural Networks

**Word Count:** 908 **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Sadda Rama Subba Reddy

**Date:** 10/08/2025

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** |
|---|

| | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual for Enhancing Cryptocurrency Fraud Detection: A Hybrid Model Combining Transformers and Graph Neural Networks

Forename Surname
Student ID:

## 1. Introduction

This configuration manual provides a detailed, step-by-step setup to replicate and execute the research project: Enhancing Cryptocurrency Fraud Detection. The focus is on developing a hybrid machine learning framework that integrates traditional classifiers, deep learning models (GCN and FFNN), and a novel hybrid architecture combining Autoencoder, Transformer, and GNN techniques. The objective is to detect illicit cryptocurrency transactions using the Elliptic Dataset. This implementation uses Python, PyTorch, PyTorch Geometric, and Scikit-learn for modelling and evaluation.

## 2. System Hardware Requirements

Ensure your system meets the following minimum requirements:

- **Operating System:** Ubuntu 20.04+ or Windows 10/11 (Linux preferred for PyTorch GPU support)
- **Processor:** Intel Core i5 / AMD Ryzen 5 or higher (Quad-Core recommended)
- **RAM:** 16 GB Minimum (32 GB recommended)
- **GPU:** NVIDIA GPU with CUDA support (for deep learning models)
- **Disk Space:** At least 15–20 GB free for data, dependencies, and experiment outputs

## 3. Software Requirements:

- **Python:** Version 3.8 or newer
- **Core libraries:** pandas, numpy, matplotlib, seaborn, scikit-learn, plotly
- **Graph & Deep Learning:** orch, torchvision, torch_geometric, network
- **Utilities:**
  - DataLoader, TensorDataset, StandardScaler, CrossEntropyLoss, Adam, AdamW
- **Development environments:** Jupyter Notebook / VS Code / PyCharm
- **Environment Management:** conda or virtualenv for isolated Python environments

## 4. Dataset Details

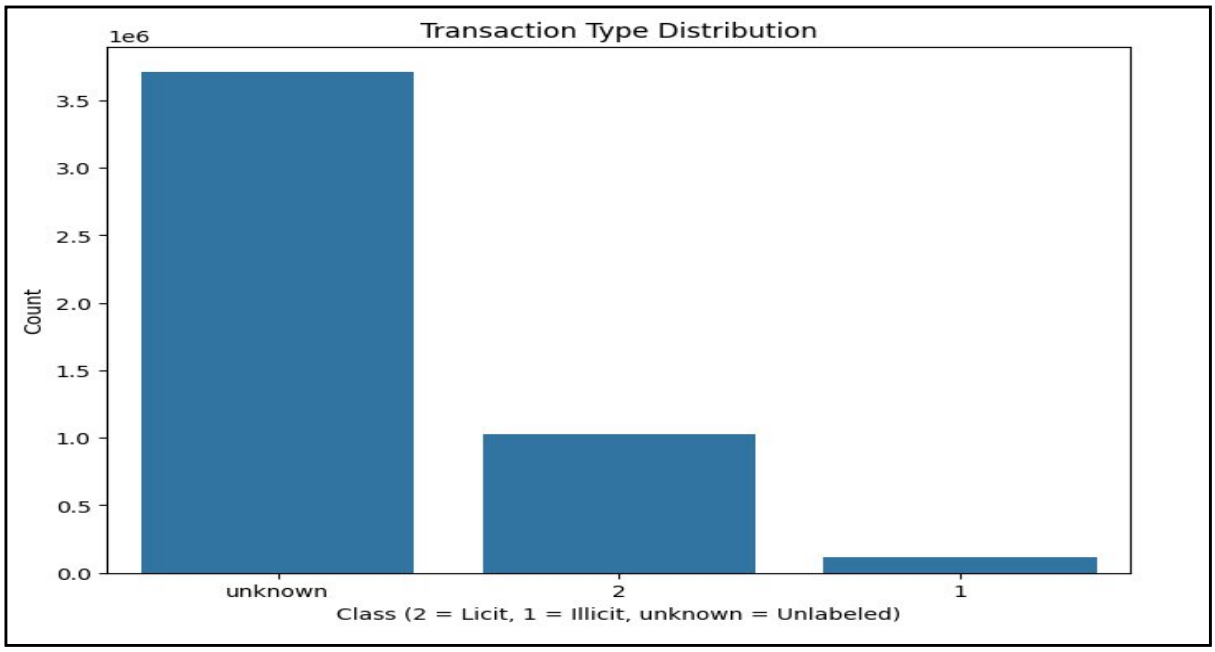
**Source:** Elliptic Data Set on Kaggle

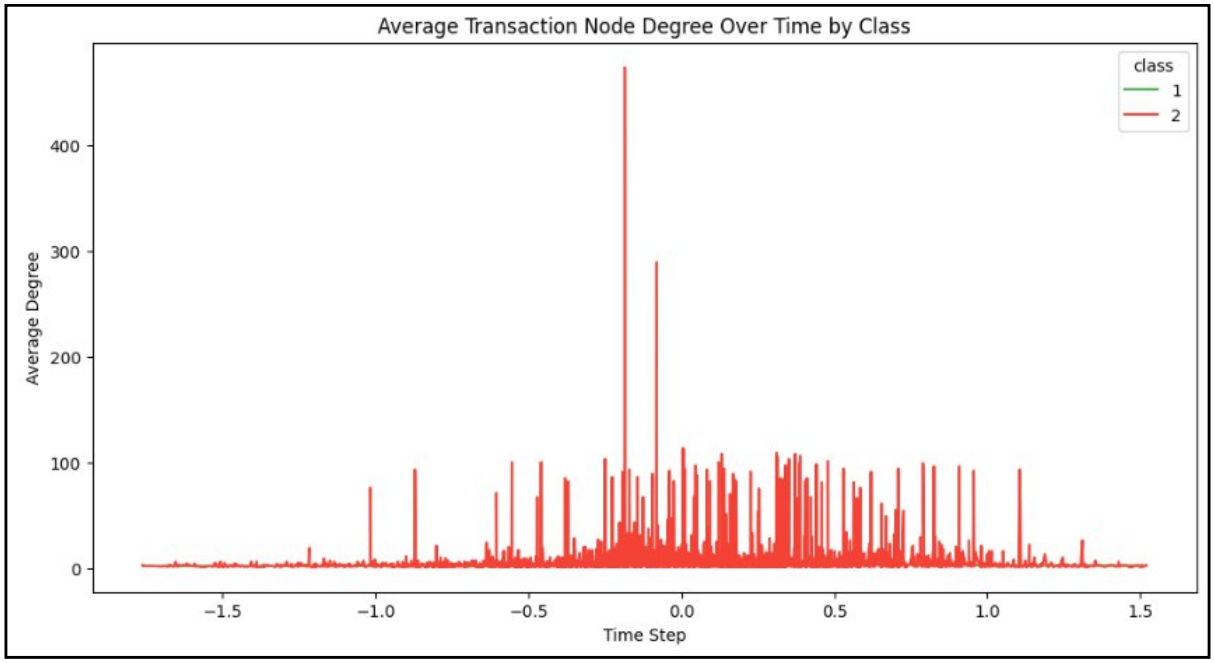
**4.1 Dataset Description:** The Elliptic Data Set maps Bitcoin transactions to real entities b elonging to licit categories (exchanges, wallet providers, miners, licit services, etc.) versus illi cit ones (scams, malware, terrorist organizations, ransomware, Ponzi schemes, etc.). The task on the dataset is to classify the illicit and licit nodes in the graph.

## 4.2 Files

| File Name | Description |
|---|---|
| `elliptic_txs_classes.csv` | Transaction labels: **1** = illicit, **2** = licit, **unknown** = unlabeled |
| `elliptic_txs_edgelist.csv` | Transaction graph edges: `txId1` → `txId2` |
| `elliptic_txs_features.csv` | 166 features per transaction + time step index |

**Size**: ~203,000 rows and over 166 features per transaction



# 5. Model Configuration

## 5.1 Baseline Models

Default algorithms and hyperparameters:

```
# Feature selection
X = df_clean.drop(['txId', 'class', 'time_step'], axis=1)
y = df_clean['class']
```

```
# Time-based split (critical for temporal data)
sorted_indices = np.argsort(df_clean['time_step'])
train_size = int(0.8 * len(sorted_indices))
train_idx = sorted_indices[:train_size]
test_idx = sorted_indices[train_size:]

X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
```

```
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score
)
```

```
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- **Linear Regression**
  - **Solver**: lbfgs
  - **Random State**: 42
  - **Max Iterations**: 2
- **Decision Tree**
  - **Variant**: GaussianNB (default settings)
- **Random Forest**
  - **Depth**: 1
  - **Class Weight**: Balanced
  - **Random State**: 42

```python
# Baseline models
models = {
    "Logistic Regression": LogisticRegression(max_iter=2, random_state=42),
    "Naive Bayes": GaussianNB(),
    "Random Forest": RandomForestClassifier(max_depth=1, random_state=42, class_weight='balanced'),
}
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Model training and evaluation
results = {}

for name, model in models.items():
    # Train model
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)
    y_proba = model.predict_proba(X_test_scaled)[:, 1] if hasattr(model, 'predict_proba') else None

    # Evaluation metrics
    results[name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred, average='weighted'),
        'Recall': recall_score(y_test, y_pred, average='weighted'),
        'F1-Score': f1_score(y_test, y_pred, average='weighted')
    }
```

- **Feature Scaling**: StandardScaler used before model training

```
Model Performance Comparison:
                     Accuracy  Precision    Recall  F1-Score
Logistic Regression  0.704392   0.935145  0.704392  0.795897
Naive Bayes          0.600021   0.954673  0.600021  0.717025
Random Forest        0.786105   0.936474  0.786105  0.849621
```

## 5.2 Deep Learning Models

### 5.2.1 Feed-Forward Neural Network (FFNN) Configuration

- **Input**: 166-dimensional feature vector
- **Hidden Layers**: 2 (128 units each)
- **Batch Normalization**: Yes
- **Dropout**: 0.7
- **Optimizer**: Adam
- **Loss**: CrossEntropyLoss
- **Epochs**: 4

```python
# Define FFNN Model
class FFNN(nn.Module):
    def __init__(self, input_dim, hidden_dim=128):
        super(FFNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.bn1 = nn.BatchNorm1d(hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.bn2 = nn.BatchNorm1d(hidden_dim)
        self.out = nn.Linear(hidden_dim, 2)

    def forward(self, x):
        x = F.relu(self.bn1(self.fc1(x)))
        x = F.dropout(x, 0.7, training=self.training)
        x = F.relu(self.bn2(self.fc2(x)))
        x = self.out(x)
        return x
```

```python
# Train the Model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_ffnn = FFNN(x.shape[1]).to(device)

optimizer = torch.optim.Adam(model_ffnn.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

x_train, y_train = x_train.to(device), y_train.to(device)
x_test, y_test = x_test.to(device), y_test.to(device)

print("\nTraining FFNN...")
for epoch in range(1,5):
    model_ffnn.train()
    optimizer.zero_grad()
    out = model_ffnn(x_train)
    loss = criterion(out, y_train)
    loss.backward()
    optimizer.step()

    if epoch % 1 == 0:
        print(f"Epoch {epoch} - Loss: {loss.item():.4f}")
```

```python
# FFNN Evaluation
model_ffnn.eval()
with torch.no_grad():
    logits = model_ffnn(x_test)
    preds = logits.argmax(dim=1).cpu()
    y_true = y_test.cpu()

# Compute FFNN metrics
accuracy_ffnn = accuracy_score(y_true, preds)
precision_ffnn = precision_score(y_true, preds, average='weighted')
recall_ffnn = recall_score(y_true, preds, average='weighted')
f1_ffnn = f1_score(y_true, preds, average='weighted')
```

### 5.2.2 Graph Convolutional Network (GCN) Configuration

- **Graph**: Built using networkx and torch_geometric from edgelist
- **Layers**: 2 GCNConv layers
- **Hidden Units**: 64
- **Dropout**: 0.6
- **Activation**: ReLU

- **Optimizer**: Adam (lr=0.01, weight_decay=5e-4)
- **Epochs**: 2

```python
# Define GCN Model
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels=64, out_channels=2):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, out_channels)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.relu(self.conv1(x, edge_index))
        x = F.dropout(x, p=0.6, training=self.training)
        return self.conv2(x, edge_index)
```

```python
# Train the Model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GCN(in_channels=data.num_node_features).to(device)
data = data.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

print("Training GCN...")
for epoch in range(1,3):
    model.train()
    optimizer.zero_grad()
    out = model(data)
    # Only compute loss on labeled training data
    mask = data.train_mask & (data.y != -1)
    loss = F.cross_entropy(out[mask], data.y[mask])
    loss.backward()
    optimizer.step()

    if epoch % 1 == 0:
        print(f"Epoch {epoch} - Loss: {loss.item():.4f}")
```

```python
# GCN Evaluation
model.eval()
with torch.no_grad():
    out = model(data)
    pred = out.argmax(dim=1)
    mask = data.test_mask & (data.y != -1)

    y_true = data.y[mask].cpu()
    y_pred = pred[mask].cpu()

# Compute GCN evaluation metrics
accuracy_gcn = accuracy_score(y_true, y_pred)
precision_gcn = precision_score(y_true, y_pred, average='weighted')
recall_gcn = recall_score(y_true, y_pred, average='weighted')
f1_gcn = f1_score(y_true, y_pred, average='weighted')
```

## 5.3 Hybrid Model: Transformer + GNN Inspired

This model combines an **Autoencoder** for feature learning, and a **Graph-Inspired Neural Network** for classification.

**Autoencoder (AE)**

- **Encoder**: [Input → 256 → 128 → 64]
- **Decoder**: [64 → 128 → 256 → Input]
- **Activation**: ReLU
- **Loss**: MSELoss
- **Optimizer**: AdamW (lr=0.001)

```python
# Define Hybrid Model Models
class AE(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 256), nn.ReLU(),
            nn.Linear(256, 128), nn.ReLU(),
            nn.Linear(128, 64), nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(64, 128), nn.ReLU(),
            nn.Linear(128, 256), nn.ReLU(),
            nn.Linear(256, input_dim)
        )

    def forward(self, x):
        z = self.encoder(x)
        recon = self.decoder(z)
        return recon, z

class GNN_Model(nn.Module):
    def __init__(self, input_dim=3):
        super().__init__()
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, 64), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(64, 32), nn.ReLU(),
            nn.Linear(32, 2)
        )

    def forward(self, x):
        return self.mlp(x)
```

**Fusion Network (GNN-Inspired Classifier)**

- **Input**: 3 aggregated features (latent vector mean, reconstruction error)
- **Hidden Layers**: [3 → 64 → 32 → 2]
- **Dropout**: 0.5
- **Activation**: ReLU
- **Loss**: CrossEntropyLoss (class-weighted)

- **Epochs**: 200
- **Batch Size**: 2048

```python
# Instantiate models
input_dim = x.shape[1]
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ae = AE(input_dim).to(device)
fusion = GNN_Model().to(device)

# Optimizers and loss
ae_opt = torch.optim.AdamW(ae.parameters(), lr=0.001)
fusion_opt = torch.optim.AdamW(fusion.parameters(), lr=0.001)
cls_loss_fn = nn.CrossEntropyLoss(weight=torch.tensor([5.0, 1.0], device=device))
recon_loss_fn = nn.MSELoss()
```

```python
# Train the hybrid model
print("Training hybrid model...")
for epoch in range(1, 201):
    ae.train(); fusion.train()
    total_loss = 0
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        ae_opt.zero_grad(); fusion_opt.zero_grad()

        recon_x, z = ae(xb)
        rec_err = torch.mean((xb - recon_x) ** 2, dim=1)
        z_mean = z.mean(dim=1)

        # Normalize features
        rec_norm = (rec_err - rec_err.min()) / (rec_err.max() - rec_err.min() + 1e-6)
        z_norm = (z_mean - z_mean.min()) / (z_mean.max() - z_mean.min() + 1e-6)

        fuse_input = torch.stack([z_norm, rec_norm, z_norm], dim=1)
        out = fusion(fuse_input)

        loss = cls_loss_fn(out, yb) + 0.1 * recon_loss_fn(recon_x, xb)
        loss.backward()
        ae_opt.step(); fusion_opt.step()

        total_loss += loss.item()

    if epoch % 1 == 0:
        print(f"Epoch {epoch}: Loss = {total_loss:.4f}")
```

```python
# Evaluate the Hybrid Model (Transformer + GNN Model)
fusion.eval(); ae.eval()
preds_list, true_list = [], []

with torch.no_grad():
    for xb, yb in test_loader:
        xb = xb.to(device)
        recon_x, z = ae(xb)
        rec_err = torch.mean((xb - recon_x) ** 2, dim=1)
        z_mean = z.mean(dim=1)
        rec_norm = (rec_err - rec_err.min()) / (rec_err.max() - rec_err.min() + 1e-6)
        z_norm = (z_mean - z_mean.min()) / (z_mean.max() - z_mean.min() + 1e-6)
        fuse_input = torch.stack([z_norm, rec_norm, z_norm], dim=1)
        out = fusion(fuse_input)
        preds = out.argmax(dim=1).cpu()
        preds_list.append(preds)
        true_list.append(yb)

preds_all = torch.cat(preds_list)
true_all = torch.cat(true_list)
```

# 6. Evaluation and Results

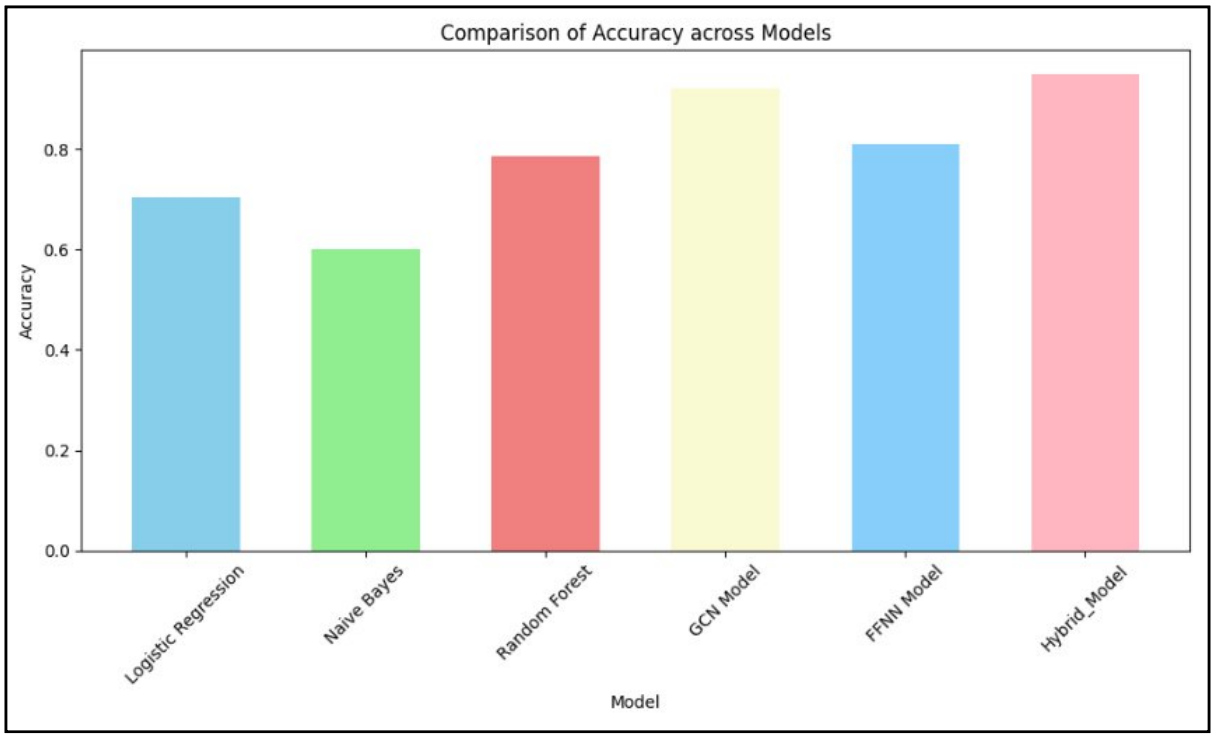
### 6.1 Evaluation Metrics (for all models):

- **Accuracy**
- **Precision** (weighted)
- **Recall** (weighted)
- **F1 Score** (weighted)

### 6.2 Final Results:

Updated Model Performance Comparison (Including Hybrid Model):

| | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 0.704392 | 0.935145 | 0.704392 | 0.795897 |
| Naive Bayes | 0.600021 | 0.954673 | 0.600021 | 0.717025 |
| Random Forest | 0.786105 | 0.936474 | 0.786105 | 0.849621 |
| GCN Model | 0.919496 | 0.882982 | 0.919496 | 0.899374 |
| FFNN Model | 0.808818 | 0.925429 | 0.808818 | 0.851927 |
| Hybrid_Model | 0.949010 | 0.980817 | 0.980817 | 0.972952 |




# 7. Conclusion

This manual provides the end-to-end setup and configuration for replicating the research titled *Enhancing Cryptocurrency Fraud Detection*. By combining deep learning, graph analytics, and hybrid architectures, the study advances fraud detection capabilities using a multi-model comparative approach. Users are encouraged to further explore parameter tuning, different feature aggregations, or extended architectures to enhance performance. Review error logs and consult relevant library documentation for troubleshooting.

# References

**Python Official Docs**: https://www.python.org/doc/

**PyTorch**: https://pytorch.org/

**PyTorch Geometric**: https://pytorch-geometric.readthedocs.io/en/latest/

**Scikit-learn**: https://scikit-learn.org/

**Seaborn**: https://seaborn.pydata.org/

**Matplotlib**: https://matplotlib.org/

**Elliptic Dataset**: https://www.kaggle.com/datasets/ellipticco/elliptic-data-set