Configuration Manual for Leveraging Graph Convolutional Networks for the Detection of Illicit Bitcoin Transactions

MSc Research Project
Data Analytics

Kavitha Kannekanti
Student ID: x23237422

School of Computing
National College of Ireland

Supervisor: Cristina Hava Muntean

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Kavitha Kannekanti ……. ……………………………………………………………………………………………………		
Student ID:	x23237422 ……………………………………………………………………………………………………..……		
Programme:	Msc Data Analytics ……………………………………………………………	**Year:**	2024 - 2025 …………………………..
Module:	Msc Research Project ………………………………………………………………………………………………….………		
Lecturer:	Cristina Hava Muntean ………………………………………………………………………………………………….………		
Submission Due Date:	29/01/2025 ………………………………………………………………………………………………….………		
Project Title:	Leveraging Graph Convolutional Networks for the detection of Illicit Bitcoin Transactions. ………………………………………………………………………………………………….………		
Word Count:	……………730……………… **Page Count:** ……………13……………….…….………		

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: K.Kavitha
……

Date: 29/01/2025
……

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 Leveraging Graph Convolutional Networks for the Detection of Illicit Bitcoin Transactions

Kavitha Kannekanti
Student ID: x23237422

1. Introduction

This configuration manual provides a elaborated guide for implementing this research leverages Graph Convolutional Networks (GCNs) to detect illicit Bitcoin transactions using a graph-based model. The project involves preprocessing Bitcoin transaction data, constructing a graph from the data, and training machine learning models (both baseline models and the GCN model) to classify transactions as illicit or licit.

2. Requirements

This manual assumes that you are working in a Python-based environment, preferably in a virtual environment or Docker container.

Software Requirements

- Python (version 3.7 or higher)
- PyTorch (version 1.10 or higher)
- PyTorch Geometric (version 2.0 or higher)
- scikit-learn (version 0.24 or higher)
- Matplotlib (version 3.4 or higher)
- Pandas (version 1.2 or higher)
- NetworkX (version 2.5 or higher)
- Seaborn (version 0.11 or higher)

Hardware Requirements

- A machine with a CUDA-compatible GPU (for faster model training)
- At least 8GB of RAM for running the models
- Sufficient disk space for dataset storage and model checkpoints

3. Environment Setup

- **Create a Virtual Environment (Optional but recommended):**

```
python3 -m venv gcn-env
source gcn-env/bin/activate  # On Windows use gcn-env\Scripts\activate
```

- **Install Required Packages: Install dependencies using pip:**

```
pip install torch torchvision torchaudio torch-geometric pandas scikit-learn networkx
```

- **Verify GPU Availability: Ensure PyTorch can access the GPU:**

```
import torch
print(torch.cuda.is_available())
```

4. Dataset Preparation

Dataset Files:

The dataset used in this research is based on Bitcoin transaction data from the Elliptic dataset sourced from Kaggle, which is typically structured as follows:

- Features: elliptic_txs_features.csv (contains transaction features)
- Classes: elliptic_txs_classes.csv (contains transaction labels such as illicit or licit)
- Edgelist: elliptic_txs_edgelist.csv (contains edges representing transactions between Bitcoin addresses)

```
/elliptic_bitcoin_dataset
    elliptic_txs_features.csv
    elliptic_txs_classes.csv
    elliptic_txs_edgelist.csv
/code
```

5. Code Implementation

- Load the Dataset: Use Pandas to load the transaction features, classes, and edge list:

```
# Define file paths
feature_file = 'elliptic_bitcoin_dataset/elliptic_txs_features.csv'
class_file = 'elliptic_bitcoin_dataset/elliptic_txs_classes.csv'
edgelist_file = 'elliptic_bitcoin_dataset/elliptic_txs_edgelist.csv'


# Load the features, classes, and edges data
features_df = pd.read_csv(feature_file, header=None)
classes_df = pd.read_csv(class_file)
edges_df = pd.read_csv(edgelist_file)
```

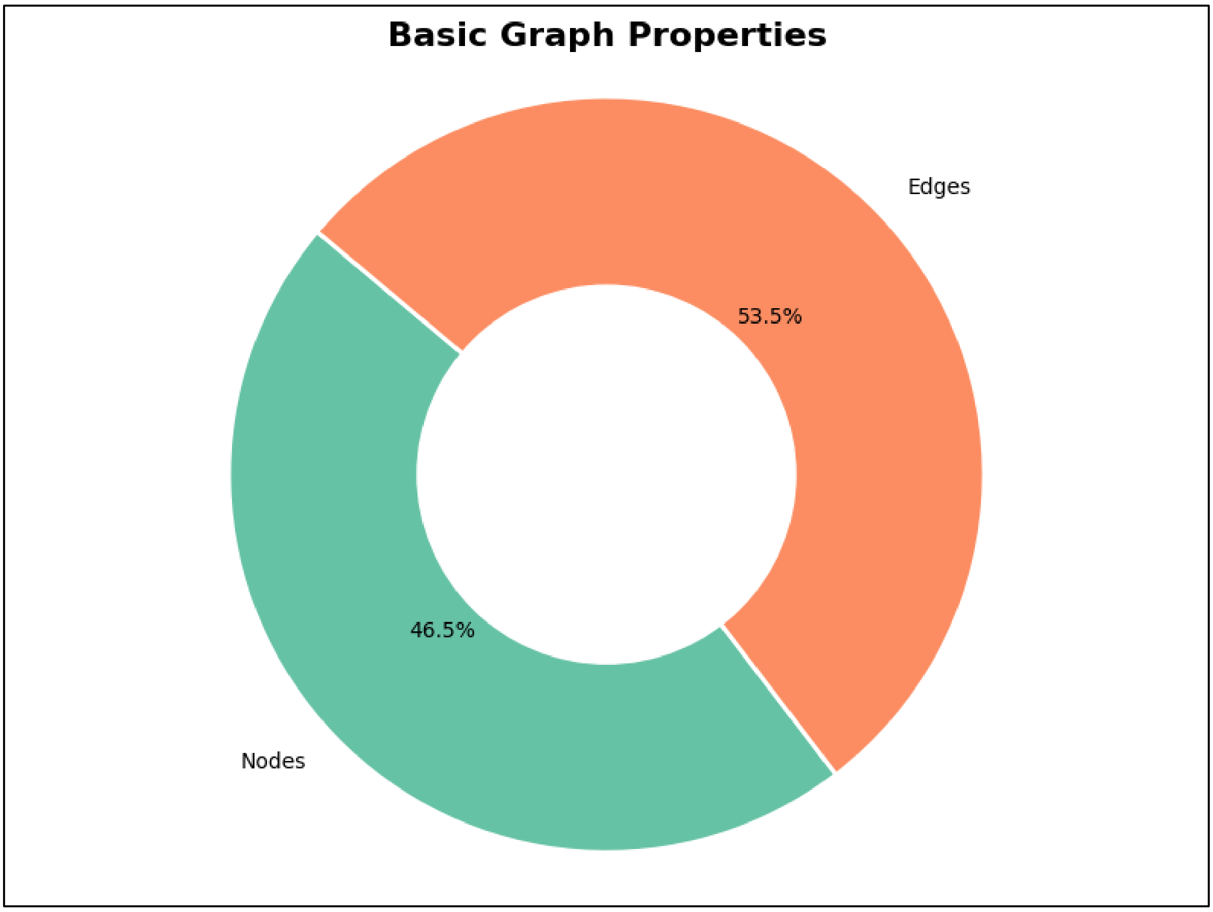
```
Dataset Shapes:
    Features  :  203,769 (rows)   167 (cols)
    Classes   :  203,769 (rows)     2 (cols)
    Edgelist  :  234,355 (rows)     2 (cols)
```

- Data Cleaning: Filter and prepare the dataset for model training. Ensure that there are no missing or malformed entries in the dataset.

- Feature Normalization: Normalize the node features using standard scaling techniques:

```python
# Create a mapping of txId to index
tx_id_index = {tx_id: idx for idx, tx_id in enumerate(features_df['transaction_id'])}

# Filter edges to keep only those with valid txIds
valid_edges = edges_df[edges_df['txId1'].isin(tx_id_index) & edges_df['txId2'].isin(tx_id_index)]
valid_edges['Source'] = valid_edges['txId1'].map(tx_id_index)
valid_edges['Target'] = valid_edges['txId2'].map(tx_id_index)

# Prepare edge index for PyTorch Geometric
edge_index_tensor = torch.tensor(valid_edges[['Source', 'Target']].values.T, dtype=torch.long)

# Prepare node features
node_features_tensor = torch.tensor(features_df.drop(columns=['transaction_id']).values, dtype=torch.float)
print(node_features_tensor.shape)
```

- Graph Construction: Convert the edge list into a PyTorch Geometric-compatible format:

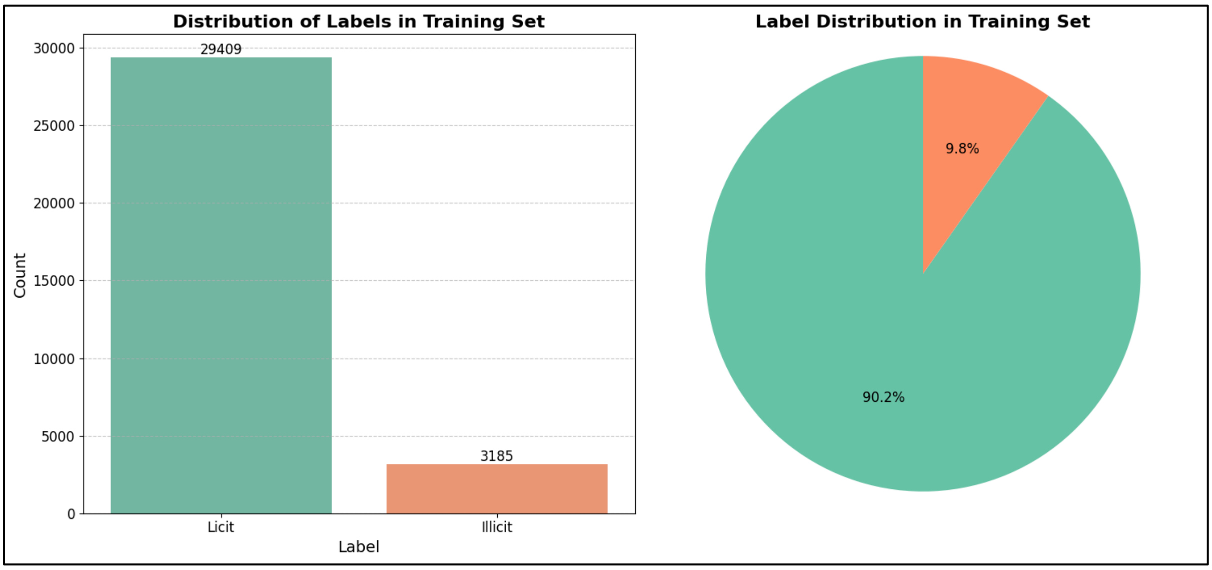


- Label Encoding: Convert class labels (licit, illicit) into numerical labels:

```
# Encode class labels (handle 'unknown' labels)
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(classes_df['class'])

# Convert the encoded labels to a tensor
node_labels_tensor = torch.tensor(encoded_labels, dtype=torch.long)
```

- Prepare the Dataset for Baseline Models & GTAD Model

```
# Split the dataset into training, validation, and testing sets (70% train, 15% validation, 15% test)
known_nodes_count = filtered_node_labels.shape[0]
random_permutations = torch.randperm(known_nodes_count)  # Shuffle the indices

train_count = int(0.7 * known_nodes_count)
val_count = int(0.15 * known_nodes_count)
test_count = known_nodes_count - train_count - val_count

# Create indices for the splits
train_indices = random_permutations[:train_count]
val_indices = random_permutations[train_count:train_count + val_count]
test_indices = random_permutations[train_count + val_count:]

# Split the data into train, validation, and test sets
X_train = filtered_node_features[train_indices]
y_train = filtered_node_labels[train_indices]
X_val = filtered_node_features[val_indices]
y_val = filtered_node_labels[val_indices]
X_test = filtered_node_features[test_indices]
y_test = filtered_node_labels[test_indices]
```



- Train & Evaluation the Baseline Models

```python
# Define the Logistic Regression model
logistic_model = LogisticRegression(class_weight='balanced')

# Define the hyperparameter grid
param_grid = {
    'max_iter':[10000],
    'C': [0.01, 0.1],
    'penalty': ['l2'],
    'solver': ['lbfgs', 'liblinear'],
}

# Set up the grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=logistic_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='accuracy', verbose=1)

# Fit grid search to the training data
grid_search.fit(X_train, y_train)
```

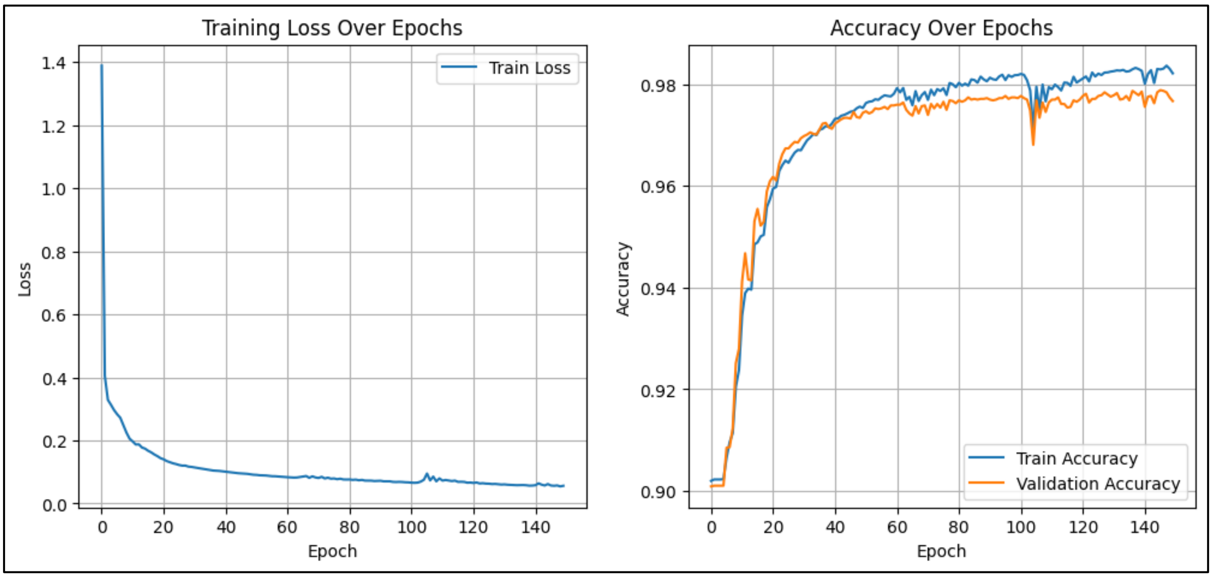
```python
# Define the Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Define the hyperparameter grid
param_grid = {
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4],
   'max_depth':[1,2],
    'criterion': ['gini', 'entropy'],
    'max_features': [ 'auto','sqrt'],
}

# Set up the grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=decision_tree_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='accuracy', verbose=1)

# Fit grid search to the training data
grid_search.fit(X_train, y_train)
```

```python
# Define the Random Forest model
random_forest_model = RandomForestClassifier(random_state=42)

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [ 5, 10, 20,100,500],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_depth':[1,2],
    'max_features': ['auto', 'sqrt'],
}

# Set up the grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=random_forest_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='accuracy', verbose=1)

# Fit grid search to the training data
grid_search.fit(X_train, y_train)
```

- **Building & Training of the GTAD Model**

```python
# Define the GTAD Model
# Temporal Graph Convolution Layer (T-GCN)
class TGConv(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(TGConv, self).__init__()
        self.gcn_conv = GCNConv(in_channels, out_channels)

    def forward(self, x, edge_index, edge_attr):
        # edge_attr can represent the temporal information (like time difference)
        return self.gcn_conv(x, edge_index, edge_attr)

# Define the combined GCN, T-GCN, and Transformer model
class GTADModel(torch.nn.Module):
    def __init__(self, input_features, output_classes):
        super(GTADModel, self).__init__()

        # GCN Layer (first layer for learning graph structure)
        self.layer1 = GCNConv(input_features, 16)

        # Temporal GCN Layer (second layer to capture temporal information)
        self.temporal_layer = TGConv(16, 32)

        # Transformer Layer (to capture temporal dependencies)
        self.transformer_layer = TransformerEncoderLayer(d_model=32, nhead=4, dim_feedforward=64)
        self.transformer_encoder = TransformerEncoder(self.transformer_layer, num_layers=2)

        # Final output layer (classification)
        self.fc = torch.nn.Linear(32, output_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        edge_attr = data.edge_attr  # Temporal information encoded in edge attributes

        # Apply GCN Layer (first step in graph feature learning)
        x = self.layer1(x, edge_index)
        x = F.relu(x)

        # Apply Temporal GCN Layer (using edge attributes to encode time-related data)
        x = self.temporal_layer(x, edge_index, edge_attr)
```

```
Epoch 10: Loss = 0.2057, Train Acc = 0.9238, Val Acc = 0.9280
Epoch 20: Loss = 0.1438, Train Acc = 0.9574, Val Acc = 0.9609
Epoch 30: Loss = 0.1161, Train Acc = 0.9670, Val Acc = 0.9694
Epoch 40: Loss = 0.1022, Train Acc = 0.9722, Val Acc = 0.9712
Epoch 50: Loss = 0.0913, Train Acc = 0.9753, Val Acc = 0.9744
Epoch 60: Loss = 0.0839, Train Acc = 0.9781, Val Acc = 0.9758
Epoch 70: Loss = 0.0827, Train Acc = 0.9784, Val Acc = 0.9758
Epoch 80: Loss = 0.0761, Train Acc = 0.9794, Val Acc = 0.9762
Epoch 90: Loss = 0.0720, Train Acc = 0.9805, Val Acc = 0.9772
Epoch 100: Loss = 0.0671, Train Acc = 0.9818, Val Acc = 0.9772
Epoch 110: Loss = 0.0798, Train Acc = 0.9794, Val Acc = 0.9764
Epoch 120: Loss = 0.0666, Train Acc = 0.9807, Val Acc = 0.9769
Epoch 130: Loss = 0.0602, Train Acc = 0.9825, Val Acc = 0.9775
Epoch 140: Loss = 0.0571, Train Acc = 0.9826, Val Acc = 0.9785
Epoch 150: Loss = 0.0563, Train Acc = 0.9821, Val Acc = 0.9767
```



- Evaluation of the Baseline Models

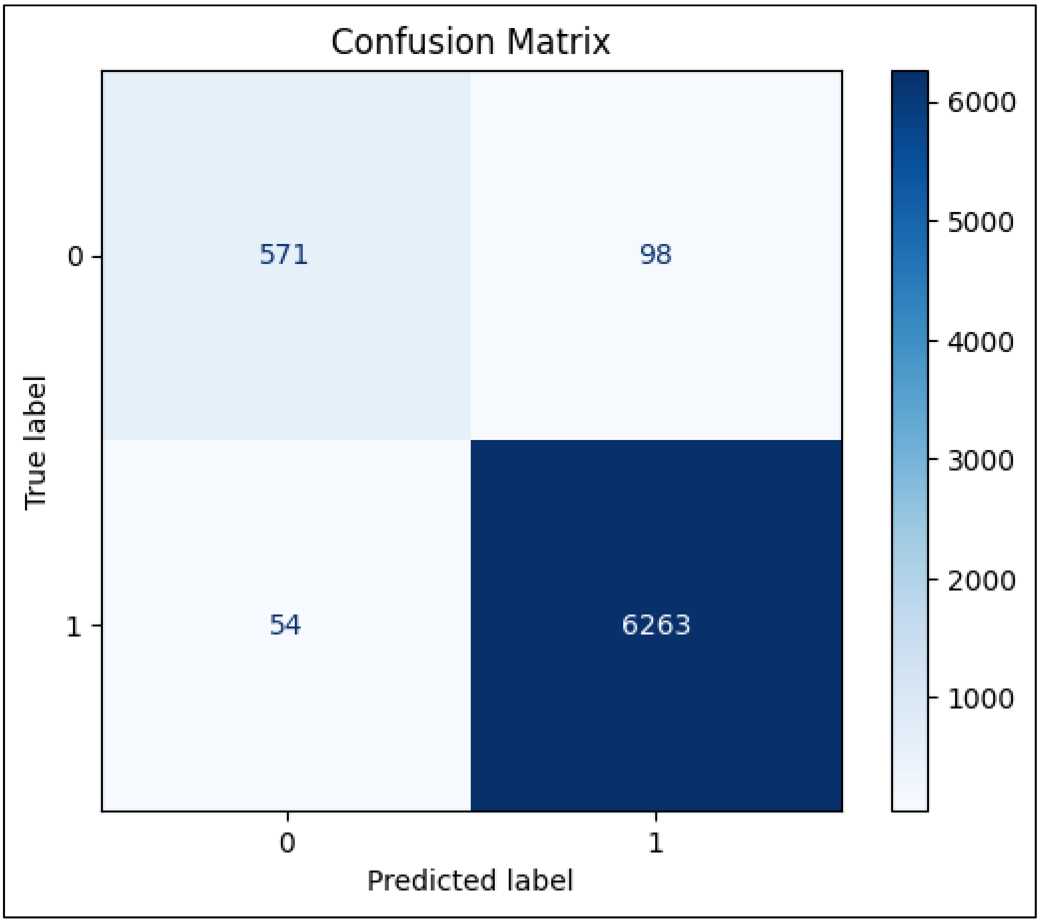
```
Logistic Regression Test Evaluation:

Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.95      0.60       669
           1       0.99      0.87      0.93      6317

    accuracy                           0.88      6986
   macro avg       0.72      0.91      0.77      6986
weighted avg       0.94      0.88      0.90      6986
```

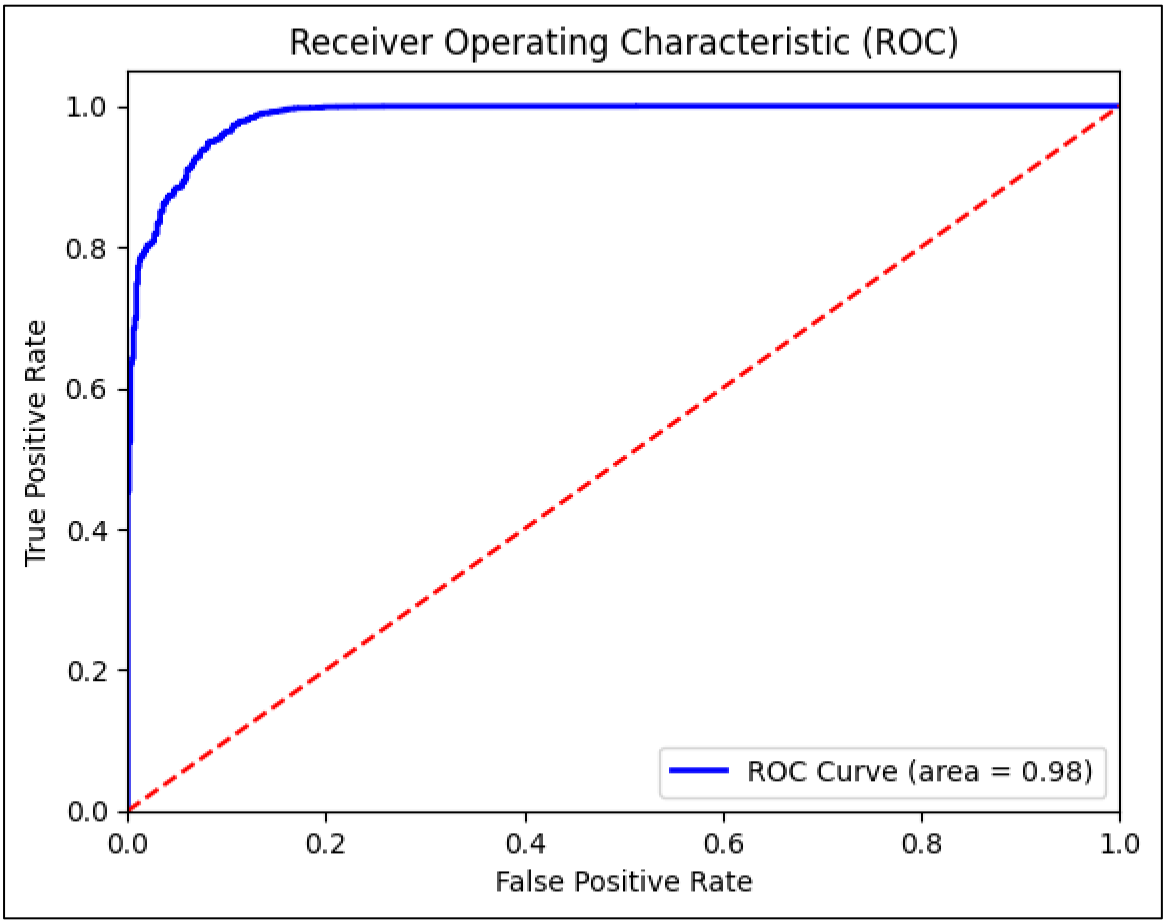
```
Decision Tree Test Evaluation:

Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.68      0.60       669
           1       0.96      0.94      0.95      6317

    accuracy                           0.91      6986
   macro avg       0.75      0.81      0.78      6986
weighted avg       0.92      0.91      0.92      6986
```

```
Random Forest Test Evaluation:

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.32      0.47       669
           1       0.93      0.99      0.96      6317

    accuracy                           0.93      6986
   macro avg       0.89      0.66      0.71      6986
weighted avg       0.92      0.93      0.91      6986
```

- Evaluation of the GTAD Model

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       669
           1       0.98      0.99      0.99      6317

    accuracy                           0.98      6986
   macro avg       0.95      0.92      0.94      6986
weighted avg       0.98      0.98      0.98      6986
```






```
# Print Overall Metrics
print("\nOverall Test Metrics:")
print(f"Accuracy: {test_metrics['accuracy']:.4f}")
print(f"Precision: {test_metrics['precision']:.4f}")
print(f"Recall: {test_metrics['recall']:.4f}")
print(f"F1 Score: {test_metrics['f1_score']:.4f}")
```

```
Overall Test Metrics:
Accuracy: 0.9782
Precision: 0.9778
Recall: 0.9782
F1 Score: 0.9779
```

Conclusion

This manual outlines how to configure and set up the system for detecting illicit Bitcoin transactions using Graph Convolutional Networks (GTAD Model). By following the steps for dataset preprocessing, model configuration, and evaluation, you can replicate the research results and experiment with variations of the model.

References

Python: https://www.python.org
Dataset Source: https://www.kaggle.com/datasets/ellipticco/elliptic-data-set