

Mini Project Report

ON

“Node Based Anomaly Detection in Ethereum”

Submitted in partial fulfillment for the award of degree in

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

SUBMITTED BY: Sai Krishna (CS21B1020)

Lakshmi Bhavya (CS21B1036)

# **ABSTRACT:**

In the face of escalating cybersecurity risks and growing susceptibility to financial fraud and bank account manipulation, Satoshi Nakamoto's 2008 white paper "Bitcoin: A Peer-to-Peer Electronic Cash System" presented a novel remedy. With the introduction of Bitcoin, a decentralised digital currency that functioned without the need for centralised middlemen like banks, was launched. Through the use of cryptographic techniques, Bitcoin made it possible for peer-to-peer transactions to be verified and safe, offering a degree of security and trust that conventional financial institutions frequently find difficult to provide.

While Ethereum advanced the blockchain concept by enabling smart contracts and decentralised applications (D-apps), Bitcoin concentrates on acting as a digital currency. Ethereum's blockchain facilitates peer-to-peer transactions as well as the creation of programmable contracts, which take action automatically in response to predetermined criteria being satisfied. A thriving ecosystem of non-fungible tokens (NFTs), decentralised finance (DeFi), and numerous other uses has resulted from this property.

A large dataset of Ethereum transactions and account metadata is used to extract essential properties such transaction patterns, balances, and code presence. Ethereum, being a decentralised blockchain platform, underlies a wide range of applications, including cryptocurrency transactions and smart contract execution. The network's dynamic nature and complexity make it vulnerable to abnormal behaviours such as fraudulent activity, hostile assaults, and irregular transactions. The effective identification and classification of anomalies inside the Ethereum network is critical to ensuring its integrity, security, and resilience.

This project explores the application of node classification techniques to identify anomalies within Ethereum's network graph. By leveraging graph-based analysis and machine learning, this study aims to enhance the understanding of node behaviors, detect irregular patterns, and contribute to the broader field of blockchain security and anomaly detection.

# **INTRODUCTION:**

Ethereum, one of the most widely used blockchain technologies, has transformed decentralised applications and digital transactions. Unlike traditional centralised systems, Ethereum runs on a decentralised network that employs accounts to store and transfer assets. These accounts are divided into two categories: externally owned accounts (EOAs) and smart contracts. Private keys govern EOAs, which often represent individuals or organisations, whereas Smart Contracts are self-executing code that is deployed on the blockchain to automate activities. Identifying the type of account is crucial for many applications, including transaction analysis, fraud detection, and resource optimisation in blockchain networks. However, Ethereum's decentralised structure renders it susceptible to abuse, such as fraud, hacking, and other unusual behaviours.  
  
 The classification of Ethereum accounts provides unique issues due to the large number of data and complicated interactions between accounts. While EOAs are associated with specific user actions, Smart Contracts frequently exhibit unique transaction patterns and behaviours. This research tackles the difficulty of reliably predicting account kinds by exploiting information extracted from Ethereum's transactional and structural data.

This research uses machine learning and graph-based techniques to investigate the inherent qualities of Ethereum accounts in order to construct a strong categorisation model. The findings aim to improve our understanding of Ethereum's ecosystem and provide useful tools for blockchain analytics, compliance monitoring, and smart contract audits. This study describes the methodology, implementation, and results of the account classification process, as well as its implications for the larger blockchain community.

# **ETHEREUM:**

Ethereum, introduced in 2015 by Vitalik Buterin, is a decentralized blockchain platform that extends the capabilities of cryptocurrencies like Bitcoin by enabling programmable smart contracts. Fundamentally, Ethereum uses accounts to store assets and communicate with the blockchain network. Ethereum is more adaptable than Bitcoin, which is solely a cryptocurrency, because it supports decentralised applications (dApps). There are two primary categories of Ethereum accounts:

## **Externally Owned Accounts:**

EOAs are user-managed accounts that are private key-controlled. They stand in for people or businesses that start transactions on the Ethereum network.

* **Key Features:**
* **Ownership:** Directly managed by private keys. Complete access to the account is possible with a working private key.
* **No Code:** Neither logic nor code are present in EOAs. They are only utilised to store and move assets or to start conversations with smart contracts.
* **Transactions**: EOAs can send and receive Ether (ETH) or ERC- 20 tokens.
* **Gas Fees:** An EOA must pay a gas fee in ETH for each transaction they start.
* **Security:** The private key determines an EOA's level of security. Access to the account is lost in the event that the private key is lost or compromised.

## **Smart Contracts:**

Accounts connected to self-governing code implemented on the Ethereum blockchain are known as smart contracts. When certain circumstances are met, these contracts carry out predetermined activities. They cannot start transactions on their own; they need to be contacted by EOAs or other smart contracts.

* **Key features:**

* **Code-Driven:** High-level programming languages like Solidity or Vyper are used to create smart contracts. The Ethereum Virtual Machine (EVM) is where the code is installed.
* **Autonomous:** Smart contracts function independently of direct human involvement once they are put into place. Deterministic execution adheres to the guidelines specified in the code.
* **Immutable:** A smart contract's code is unchangeable after it has been deployed, guaranteeing dependability and consistency. Nonetheless, certain design principles can be used to achieve upgradability.
* **Interactions:** Smart contracts have the ability to store information, carry out calculations, and communicate with other EOAs or smart contracts.  
  They enable flexible and composable systems by calling functions in other smart contracts.
* **Dependency on Transactions:** A smart contract is unable to start a transaction on its own. An EOA or similar smart contract must initiate it.

# **WHY CLASSIFICATION OF NODES:**

Nodes on the Ethereum blockchain stand in for accounts, which may be Smart Contracts or Externally Owned Accounts (EOAs). For a number of technical, analytical, and operational reasons, it is essential to classify these nodes. The following are the main justifications for the importance of this classification:

## **Blockchain Analytics:**

* **Understanding Network Behavior:** Researchers and analysts can better comprehend the behavioral patterns of various account types by using classification. For example, smart contracts usually react to transactions, but EOAs are more likely to start them.
* **Transaction Flow Analysis:** Better tracking of transaction flows is made possible by the identification of EOAs and smart contracts, which offer insights into the movement of assets throughout the network.

## **Security and Fraud Detection:**

* **Malicious Activity:** Malevolent individuals may use smart contracts to steal money or carry out illegal operations. Separating them from EOAs makes it easier to keep an eye on contract behaviour and spot possible weaknesses.
* **Avoiding frauds:** Malevolent smart contracts and fraudulent EOAs are frequently employed in frauds. Suspicious accounts may be flagged for additional investigation through classification.

## **Smart Contract Auditing and Verification:**

* **Improving Code Security:** By isolating smart contracts, auditors can focus on ensuring their security, correctness, and compliance with best practices.
* **Fault Isolation:** If issues occur in the system, identifying whether the source is an EOA or a smart contract can speed up debugging and resolution.

## **Network Optimization and Resource Allocation:**

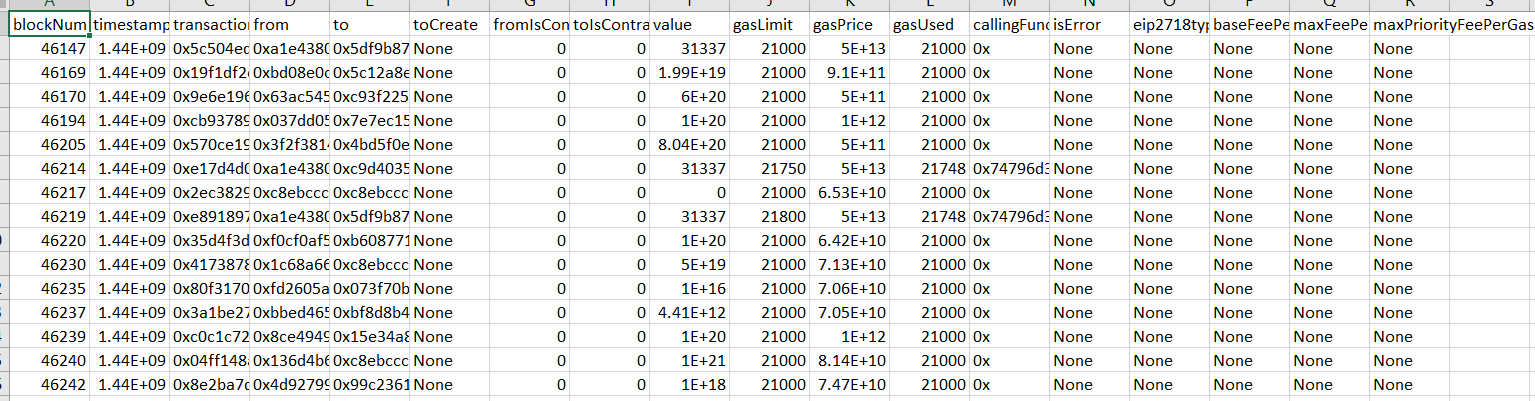
* **Load Balancing:** The Ethereum network's resource allocation can be optimised by designating nodes as EOAs or smart contracts. EOAs primarily manage transactions, whereas smart contracts might need greater processing power.
* **Scalability Analysis:** Classifying nodes aids in understanding network usage and scaling needs, particularly as the use of smart contracts continues to grow.

## **Token and Asset Management:**

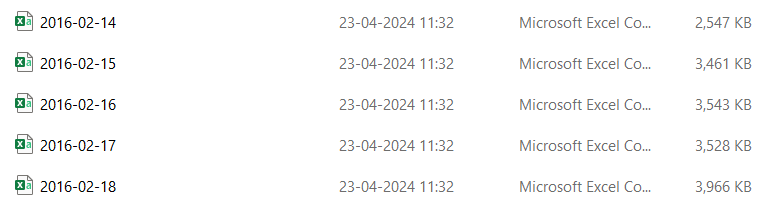
* **Token Tracking:** EOAs mostly store and move tokens, whereas smart contracts frequently oversee token transfers. Token flow analysis and tracking are made easier by classification.
* **Asset Management:** Portfolio analysis tools can differentiate between user-held assets (EOAs) and assets held in smart contracts for staking, lending, or liquidity provision.

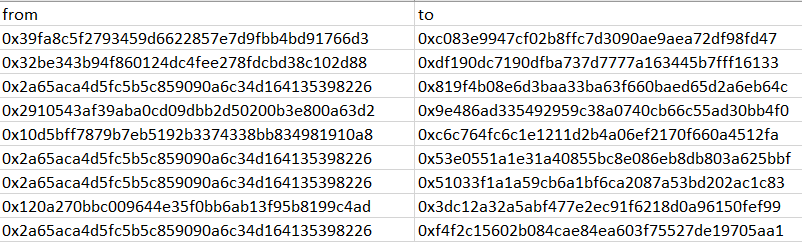
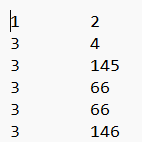
# **PREPROCESSING:**

This is my initial dataset format.

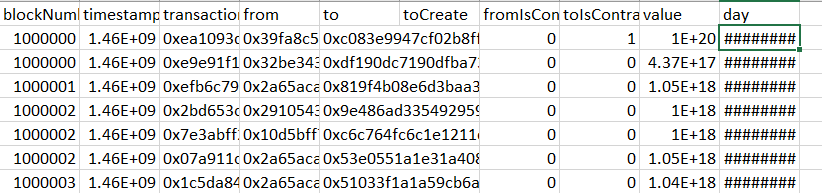


The dataset was initially divided based on the exact day that each record was created in order to analyse data on a daily basis. By transforming the raw timestamps into a more readable date format, this division was achieved. This conversion made it easier to split down the data by day, enabling more precise analysis and easy day-to-day comparisons.

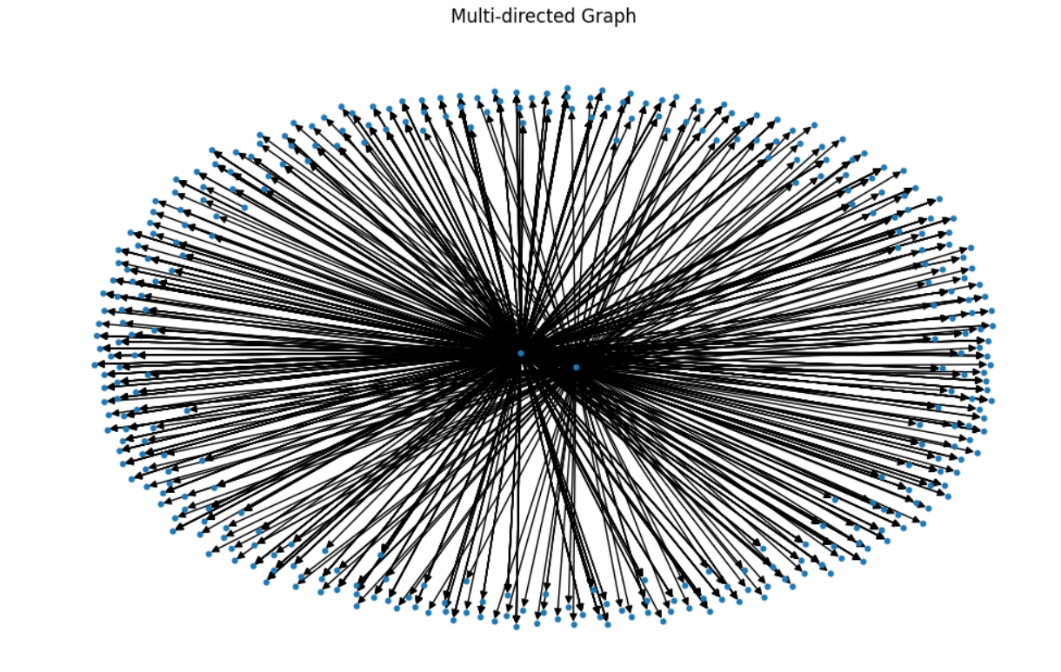
  
  
 We then created mappings to guarantee consistency throughout the collection and standardise important data elements. To monitor the flow of communications, transactions, or other pertinent activities between various entities within the dataset, "From" and "To" mappings were made. Building meaningful relationships and conducting accurate data analysis required this phase.

 🡪 

Reduce the columns also for easy computation and consider only necessary columns.



After the dataset was properly organised, we made graphs to show how the entities interacted with one another. In these graphs, edges indicated connections or interactions between nodes, which stood for distinct things. Based on the frequency or strength of the interactions each edge indicated, a weight was assigned to it. We were able to rank important links and spot patterns or trends in the network thanks to this weighting.

  
  
 Determining the kind of accounts the nodes represented was a crucial part of the investigation. This grouping may be based on behavioural patterns, particular characteristics, or other unique traits. We could study how various kinds of things interacted with one another and find any underlying dynamics or abnormalities by classifying these stories.

To classify the nodes, between externally owned accounts and smart contracts, we thought of graph neural network models than the regular model because it can capture more information and patterns than normal neural network models in this project field.

But when we are using the graph models, we need to be careful whether the graph is connected or not because if we give the not connected part also, then the model might not be able to capture the relations properly as the message passing is most important for graph models and it fails, if graph is not connected.

So, to make sure most of the data is present, extract the largest connected component and also extract the remaining columns associated with the connected edge into a text file.

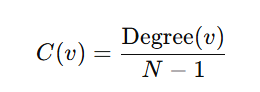
But here the node mapping in the largest connect component are not continuous, which in return will turnout into a problem while feature and label extraction. So, to avoid it renumber the nodes according to their ranking values and make sure same number is assigned same rank as the new node name.

Now, the text file is ready for feature and label extraction.

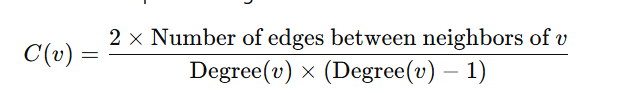
# **FEATURE EXTRACTION:**

The feature which we are going to extract are:

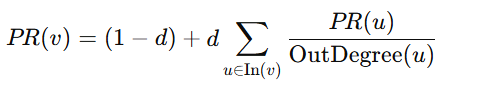
* In-degree - Number of transactions done to receive amount
* Out-degree – Number of transactions done to send amount
* Weighted-indegree – Amount received
* Weighted-outdegree – Amount sent
* Degree centrality – Represents how connected is a node



* Clustering coefficients – Measure of how interconnected the neighbours of a node are



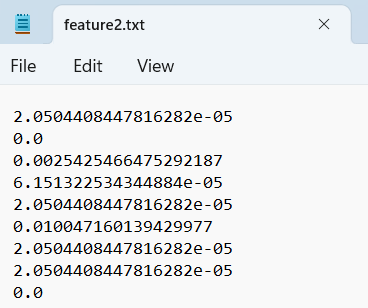
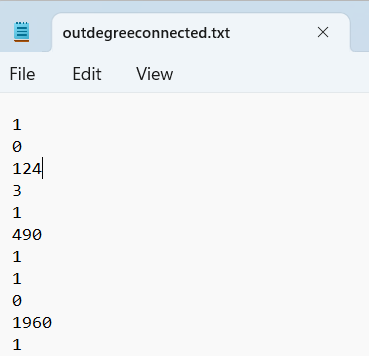
* Page rank – Measure of importance of node based on connections



We are these features because they are sufficient to capture the information like behavioural differences between the accounts and are efficient and can be extracted even for large volumes of data easily and also in past features like PageRank, clustering coefficient, degree-based metrics have shown strong performance in analysing blockchain networks.

Also, one more important point to be noted is, there will huge value difference in the features like when you deal with weighted in degree or something like that because some nodes receive high amount but might not be receiving any amount. When the features are vast spread like this, it creates a problem for the model to assign weights. So, we should normalize every feature and pass it into the model.

While coming into weighted indegree and weighted out degree, if we subtract weighted in degree with weighted out degree, we will get to know the amount transacted by that particular account on that day and then if the value is negative, keep zero or elso keep the same value and after that normalize this feature.



So, the final features are,

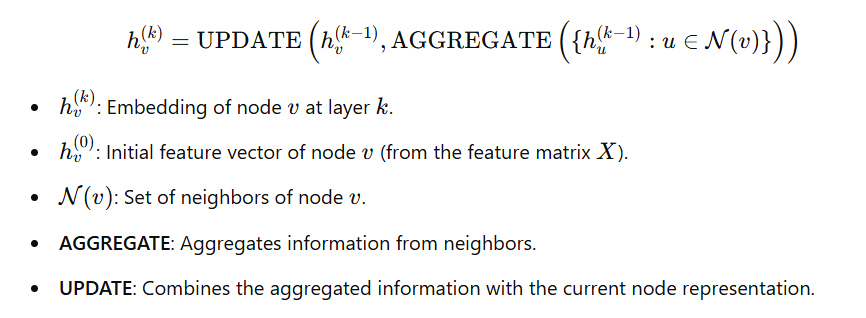
* Normalised in degree
* Normalised out degree
* Normalised amount transacted
* Normalised clustering coefficient
* Normalised degree centrality
* Normalised Page rank

The selected features are well-suited for this project as they capture both local (e.g., in-degree, out-degree) and global (e.g., PageRank, clustering coefficient) properties of accounts in the Ethereum graph.

# **MODELS**:

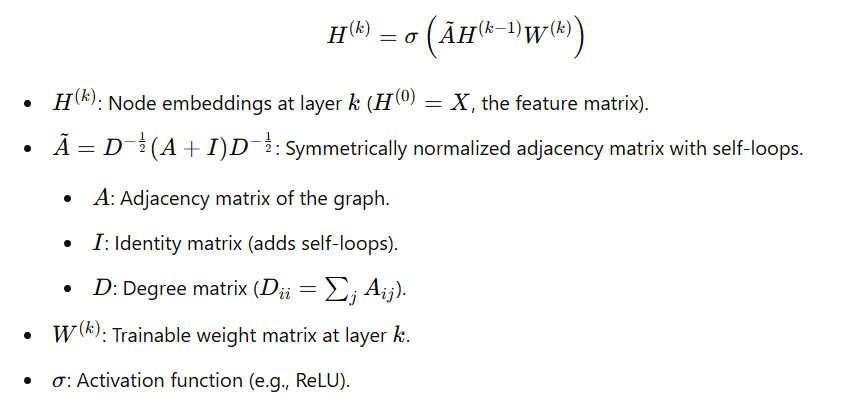
There are several models like GNN, GCN, GAT, GraphSAGE, GIN which can deal with this kind of work. Let’s go in detail for each model.

## **GNN:**

GNN follows a message-passing framework, where node embeddings are updated iteratively. 

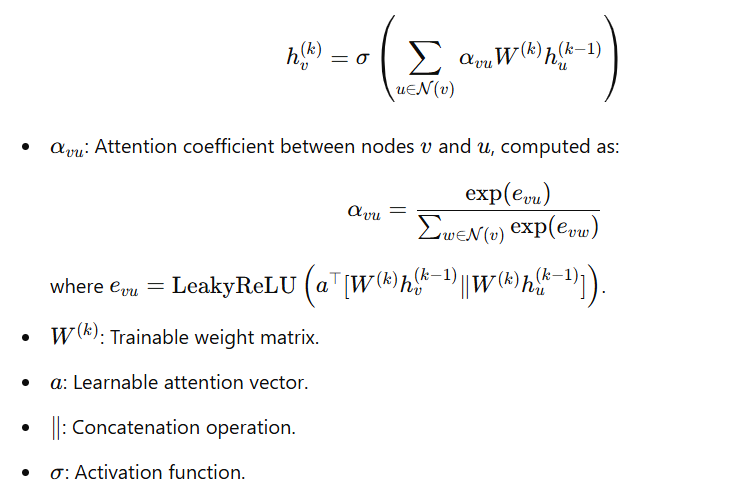
## **GCN:**

This creates a weight matrix which is multiplied with the product of normalised adjacent matrix and feature matrix.



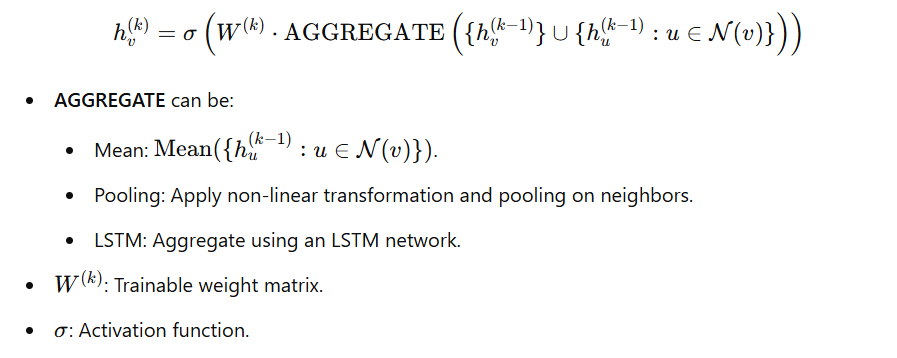
## **GAT:**

GATs introduce attention to focus more on relevant neighbours, dynamically adjusting the aggregation weights. GATs introduce attention to focus more on relevant neighbors, dynamically adjusting the aggregation weights.



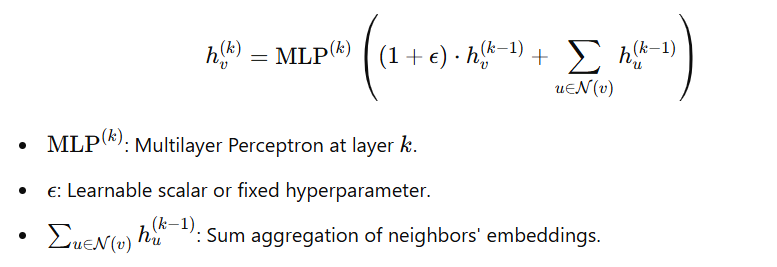
## **GraphSAGE:**

GraphSAGE aggregates neighbor information using a sampling strategy and combines it with the node's own features.



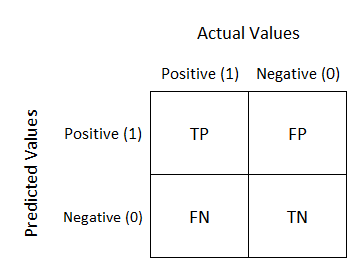
## **GIN:**

GINs are designed to capture graph structures uniquely, often outperforming other GNNs in graph classification tasks. GINs emphasize uniqueness in graph representations, achieving high expressiveness by mimicking the Weisfeiler-Lehman graph isomorphism test.



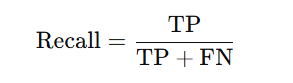
# **EVALUATION:**

Here, we are using confusion matrix for calculation of evaluation metrics like F1-score, Precision, Recall.



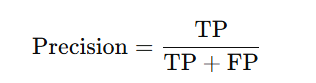
## **Recall:**

Proportion of actual positives that are correctly identified.



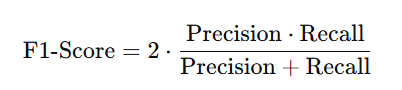
## **Precision:**

Proportion of predicted positives that are correct.



## **F1-Score:**

Harmonic mean of precision and recall, balancing the two metrics.

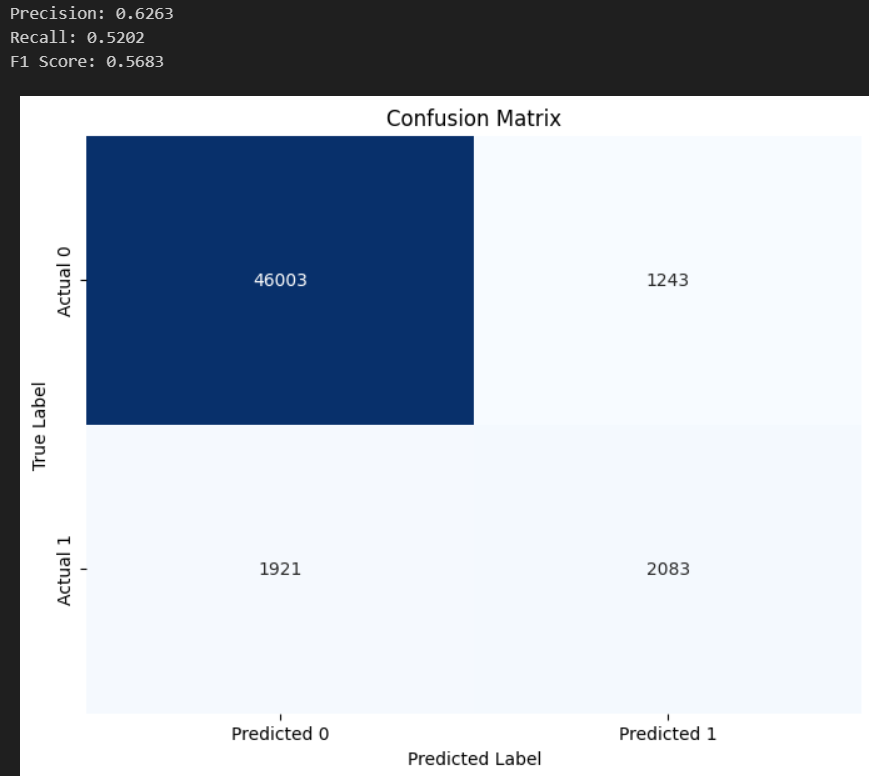
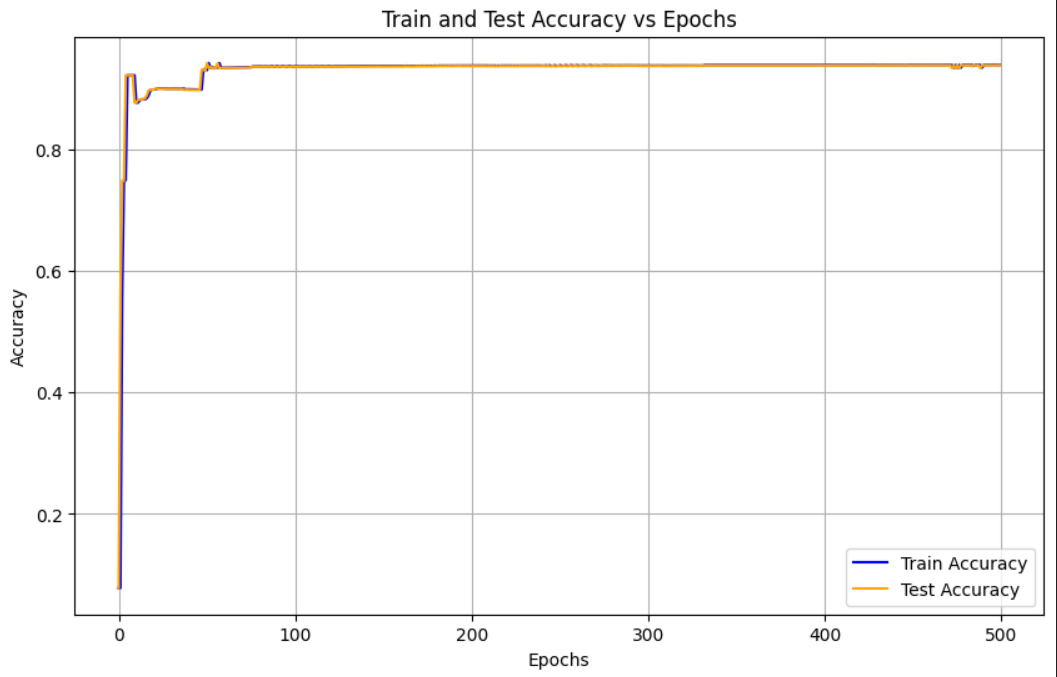
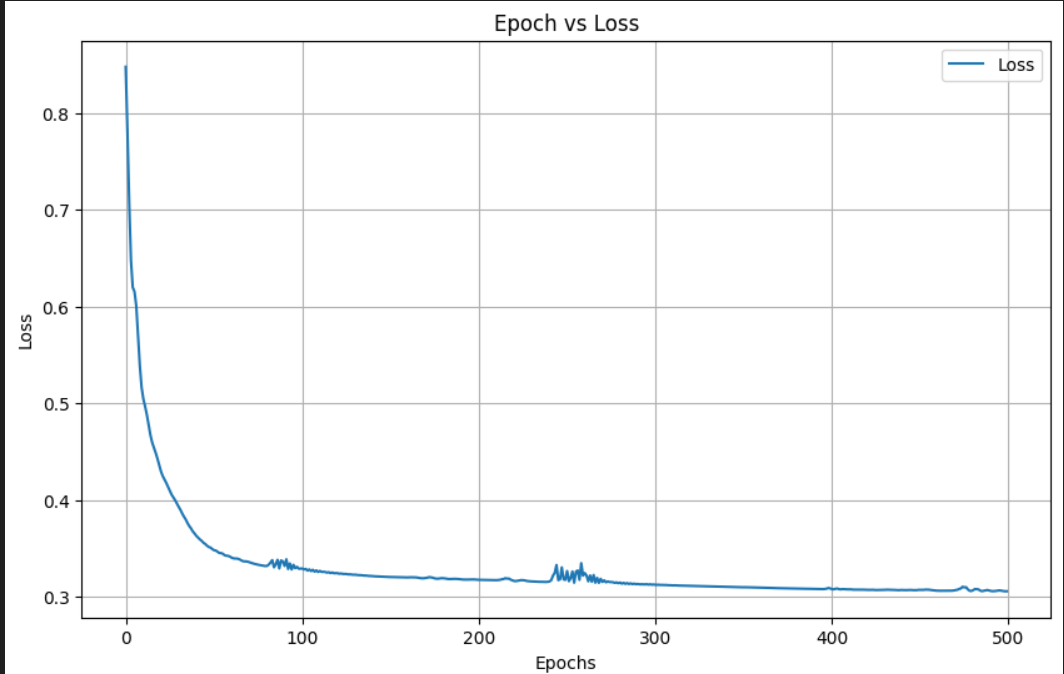


# **RESULTS**:

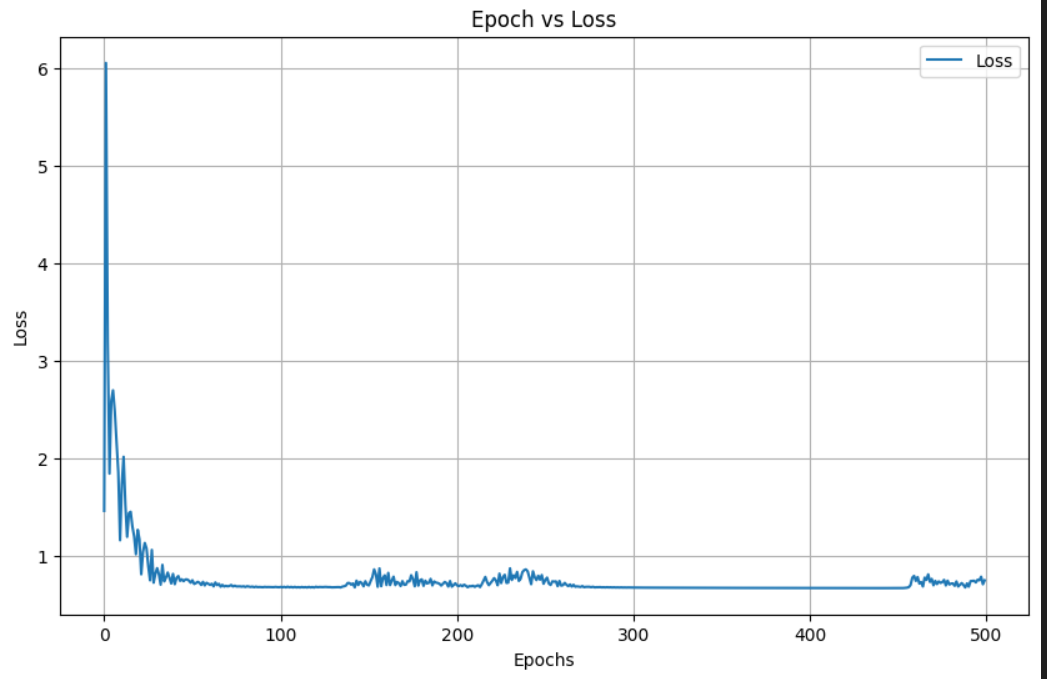
Results of each model are as follows

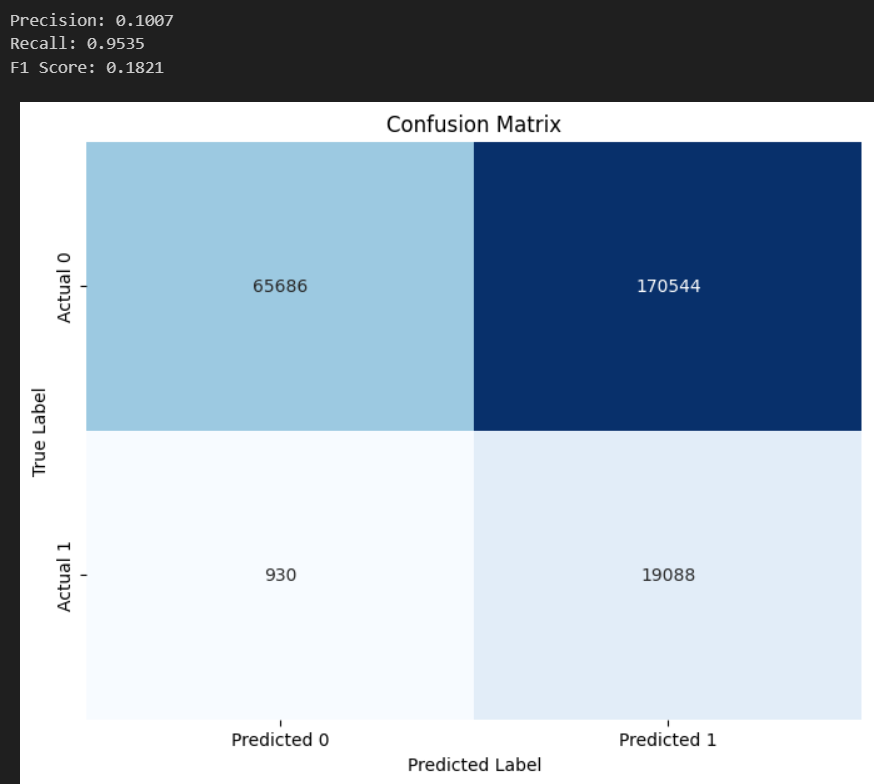
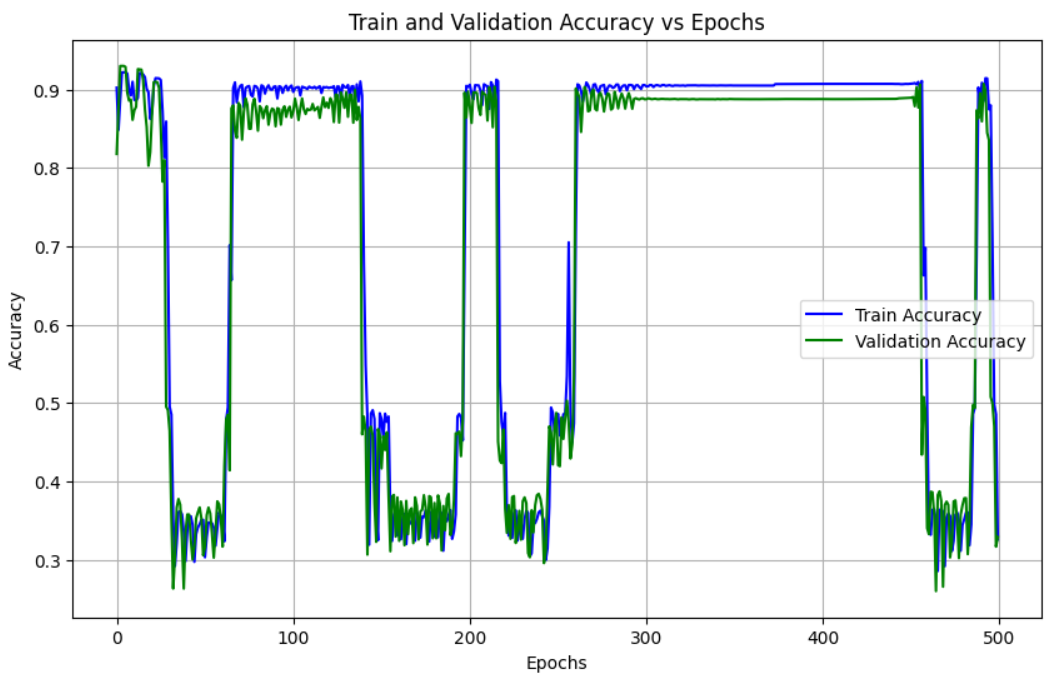
## **GNN:**

These below metrics are obtained when training and testing split is of 80% and 20%.

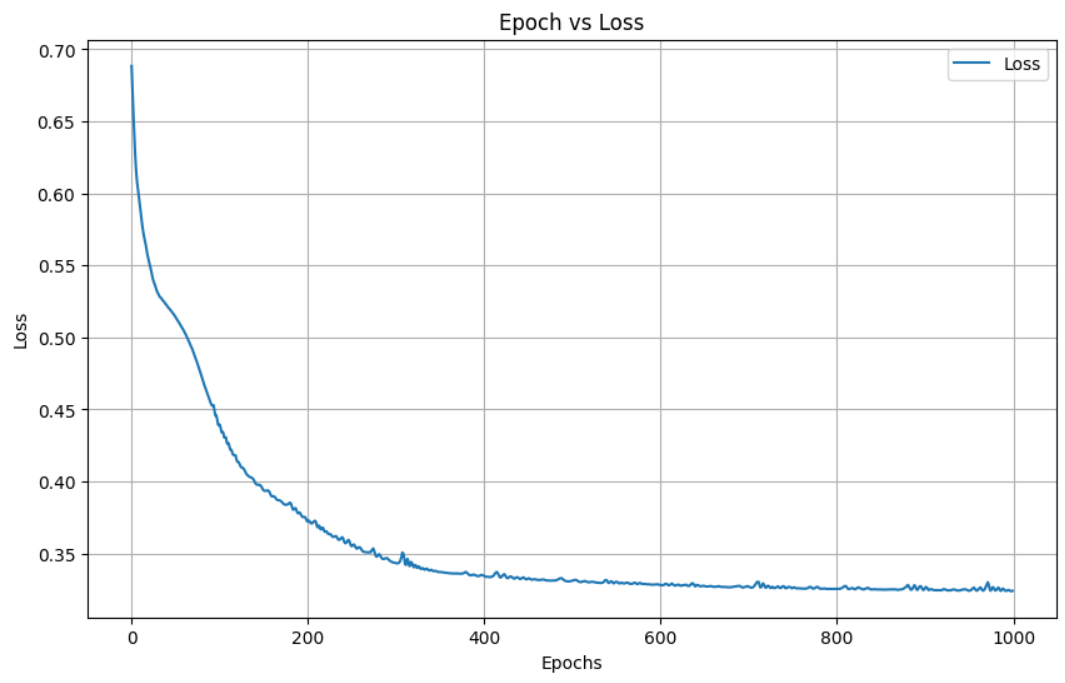


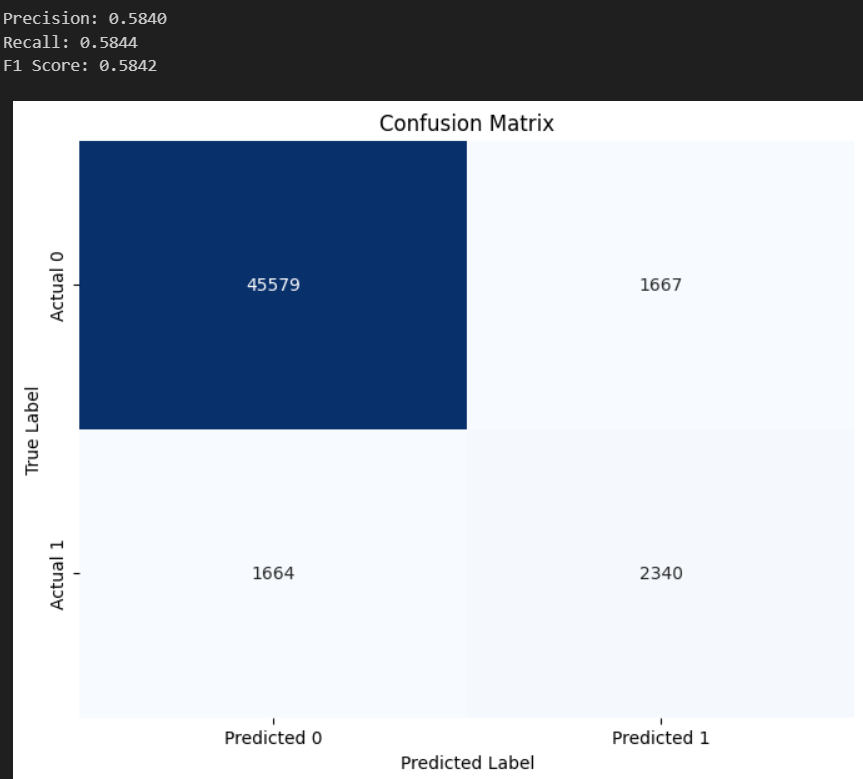
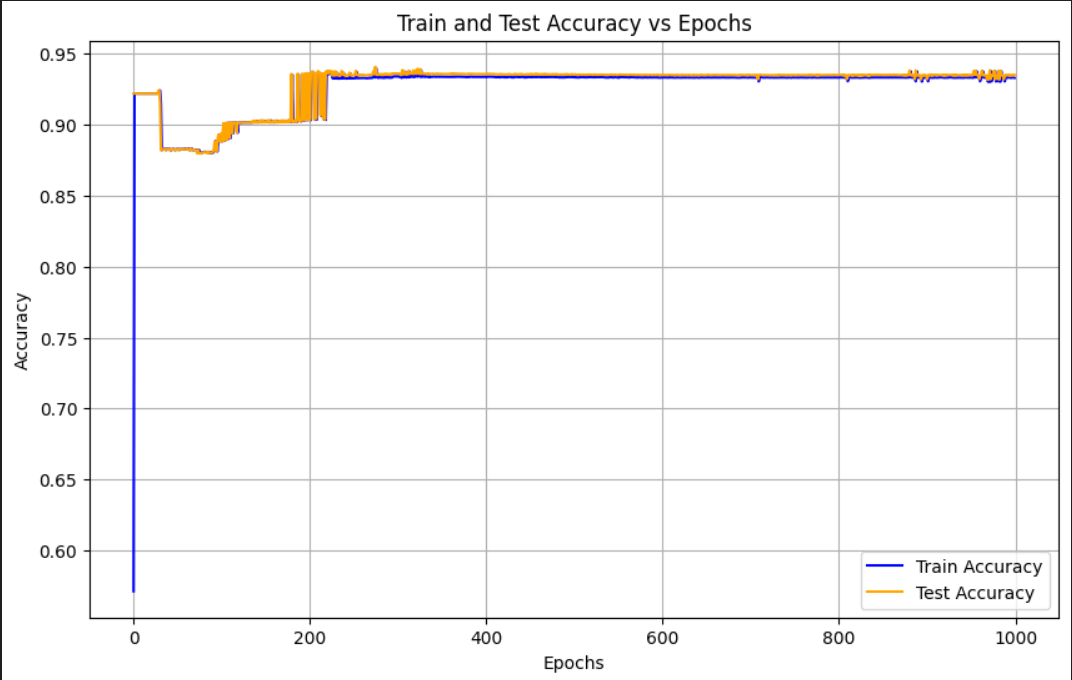
These below metrics are for validation dataset like Training over one dataset and testing over completely new dataset.



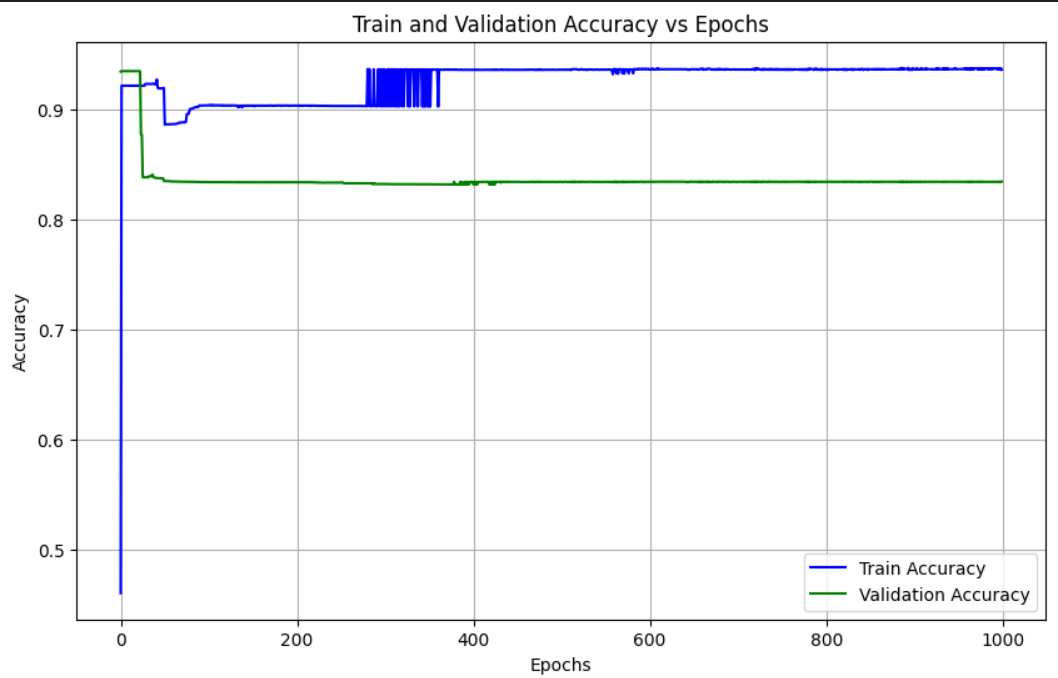
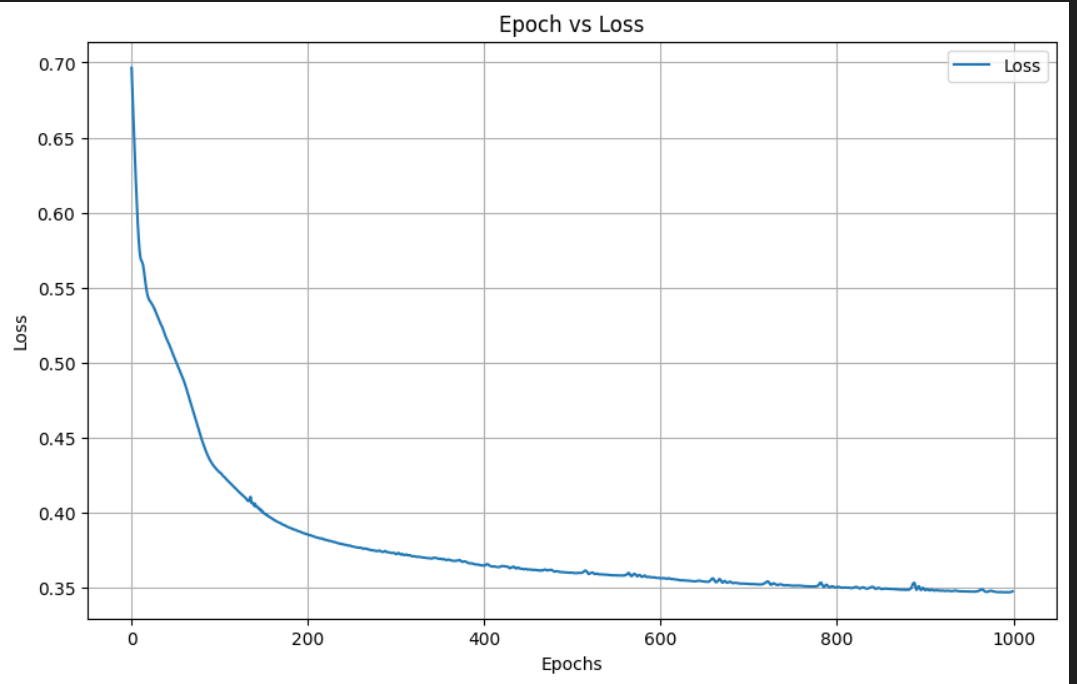


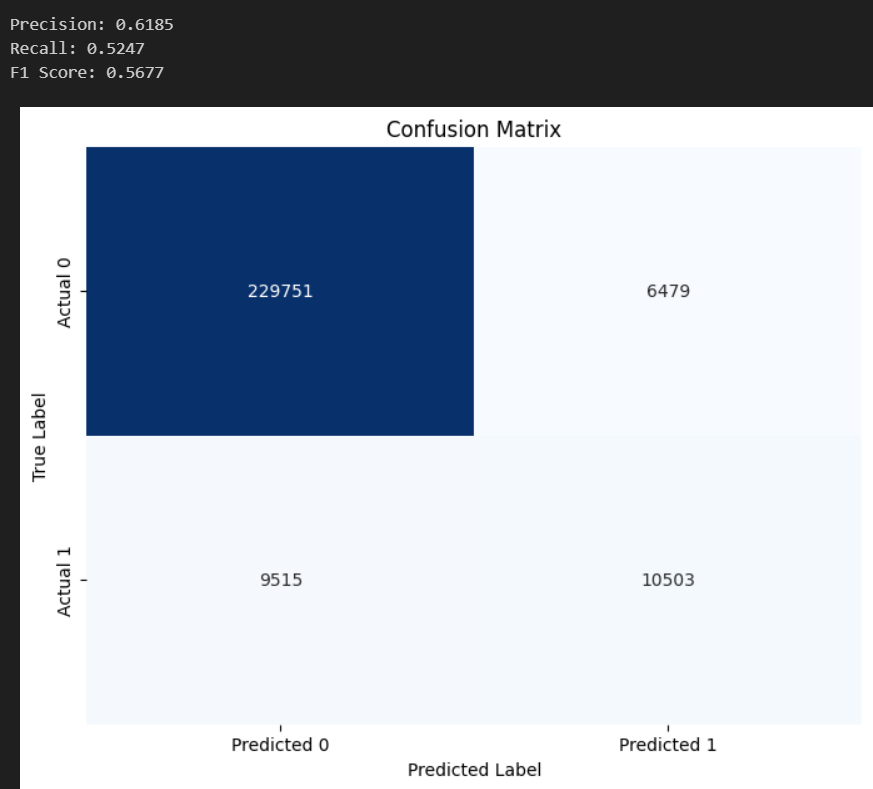
## **GCN:**

These below metrics are obtained when training and testing split is of 80% and 20%.



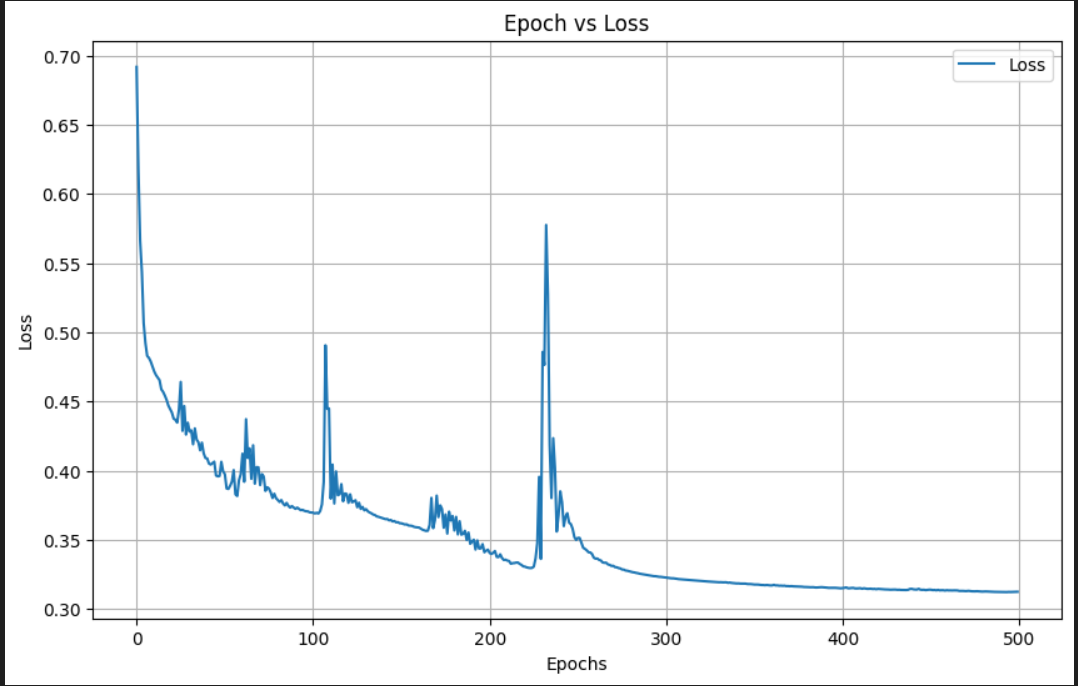
These below metrics are for validation dataset like Training over one dataset and testing over completely new dataset.

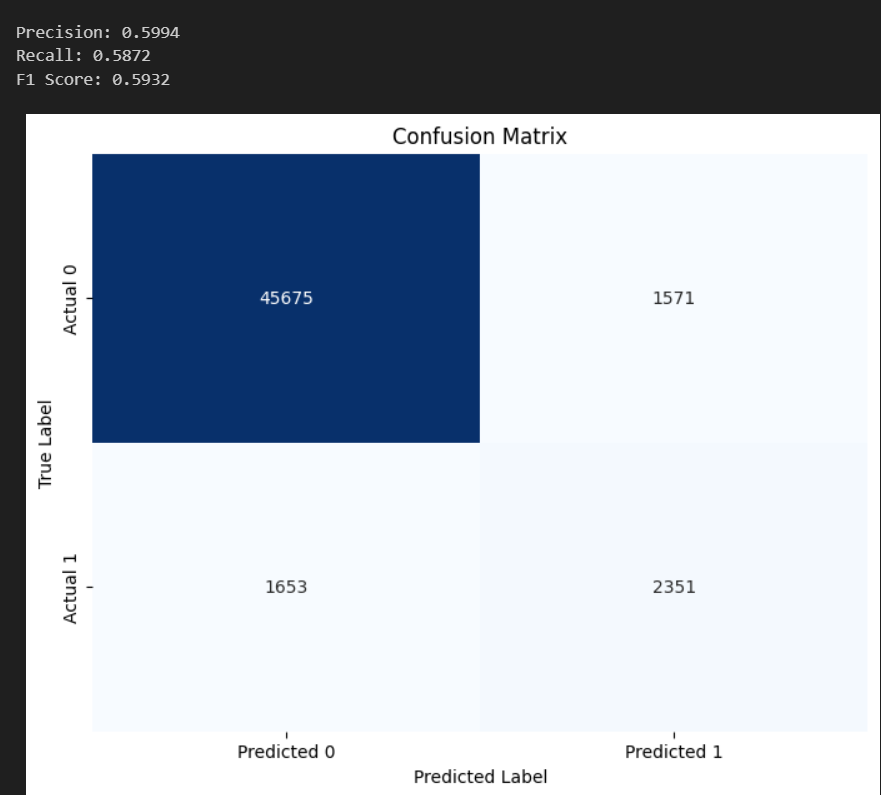
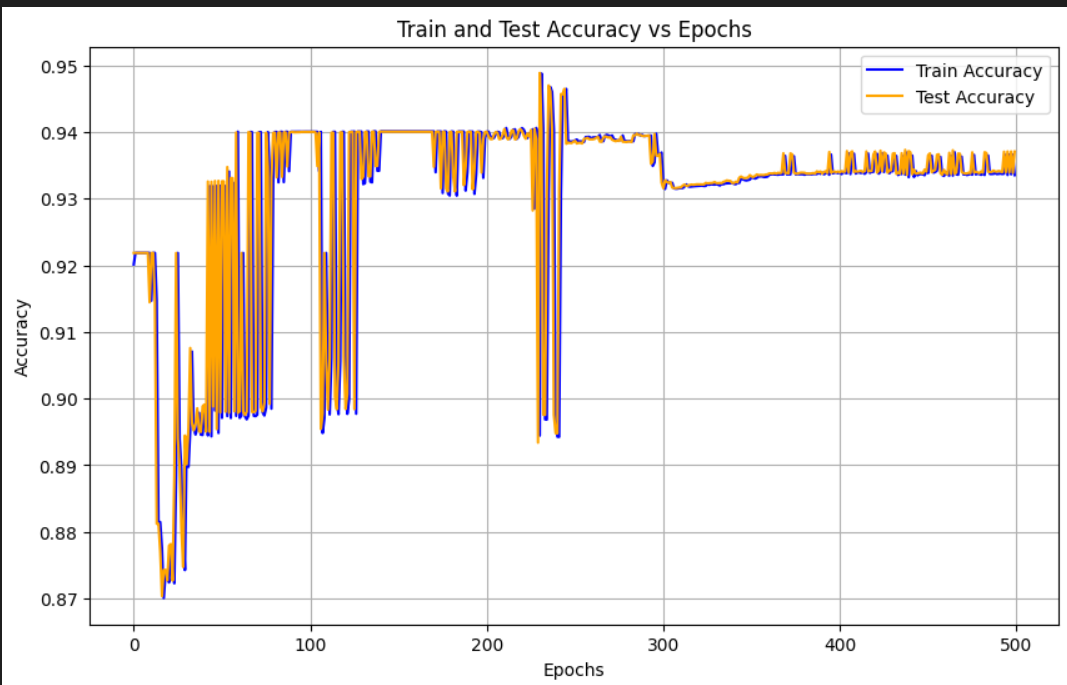


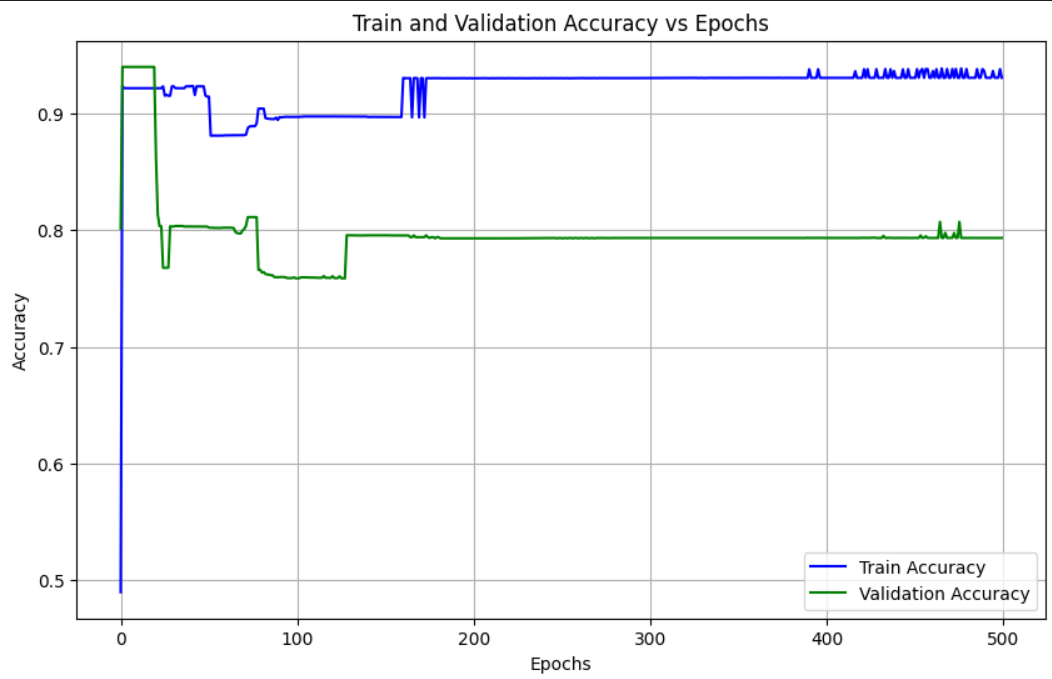
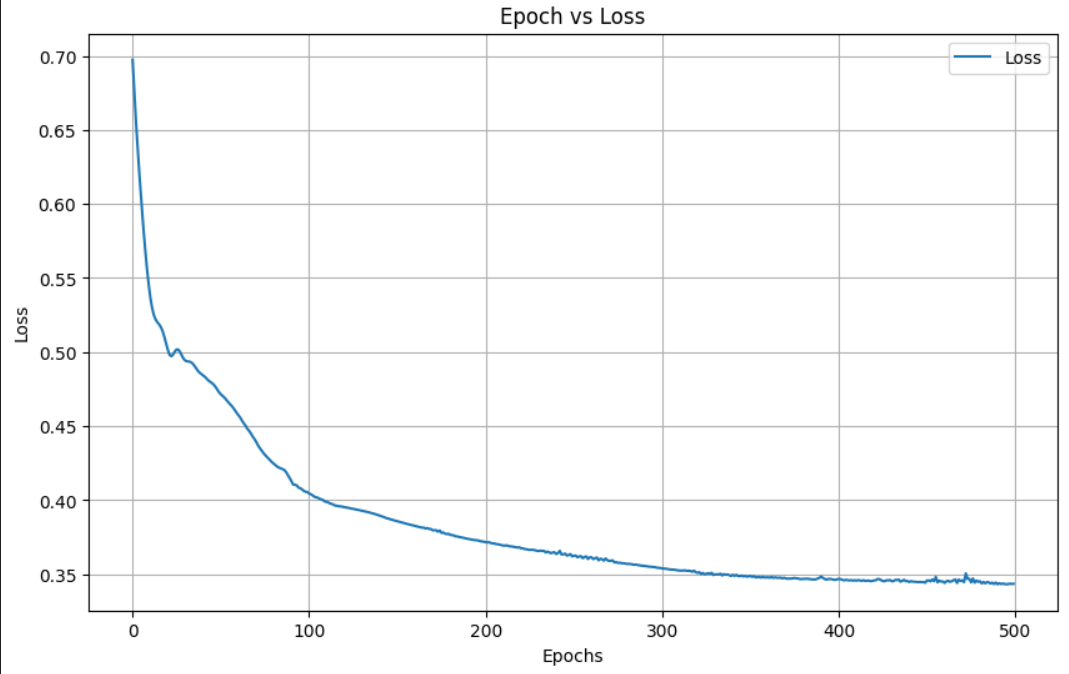


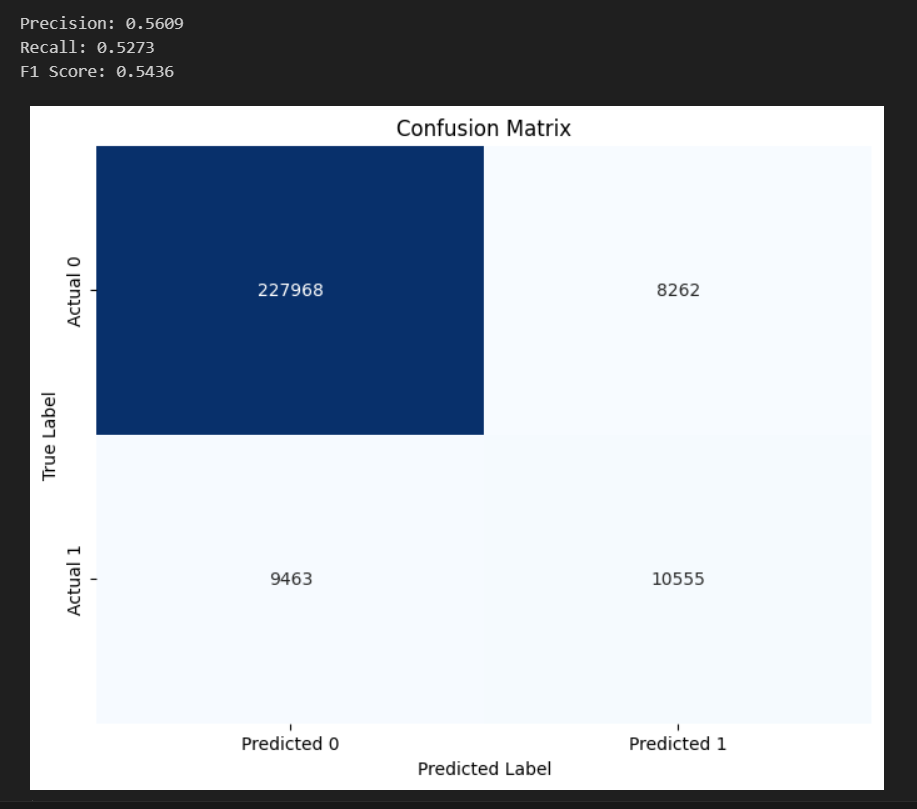
## **GAT:**

These below metrics are obtained when training and testing split is of 80% and 20%.

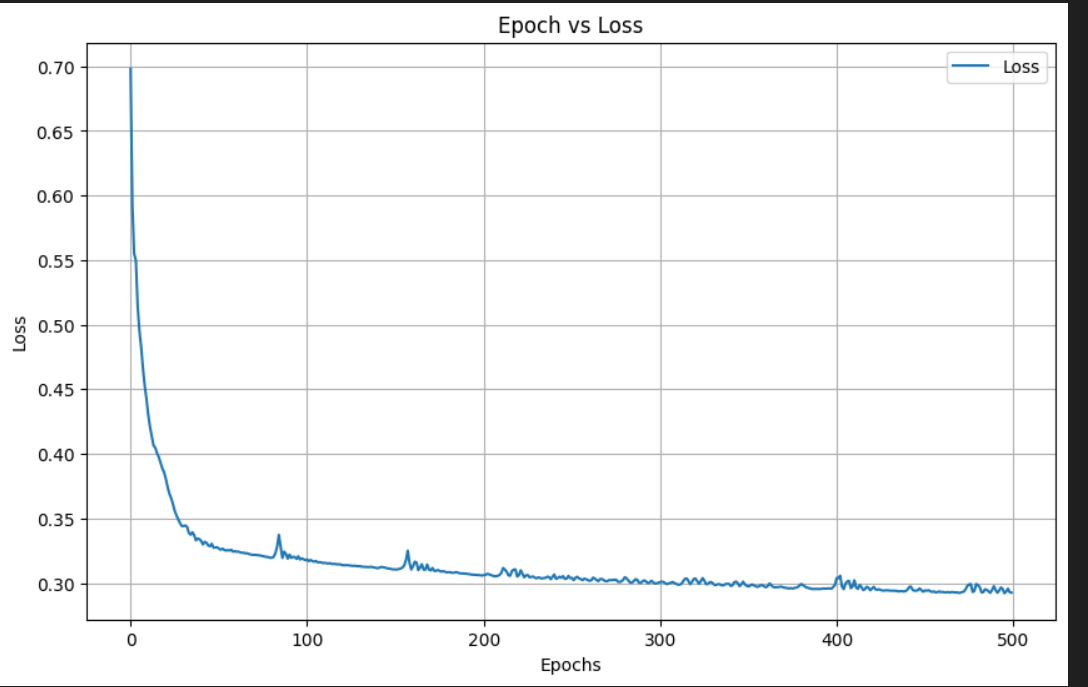
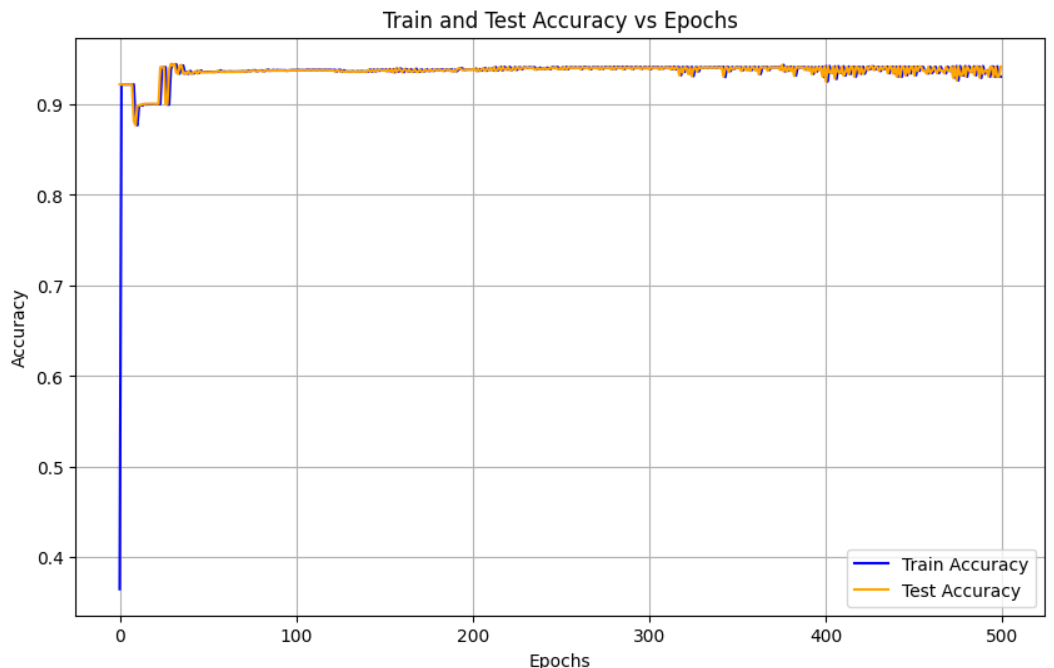


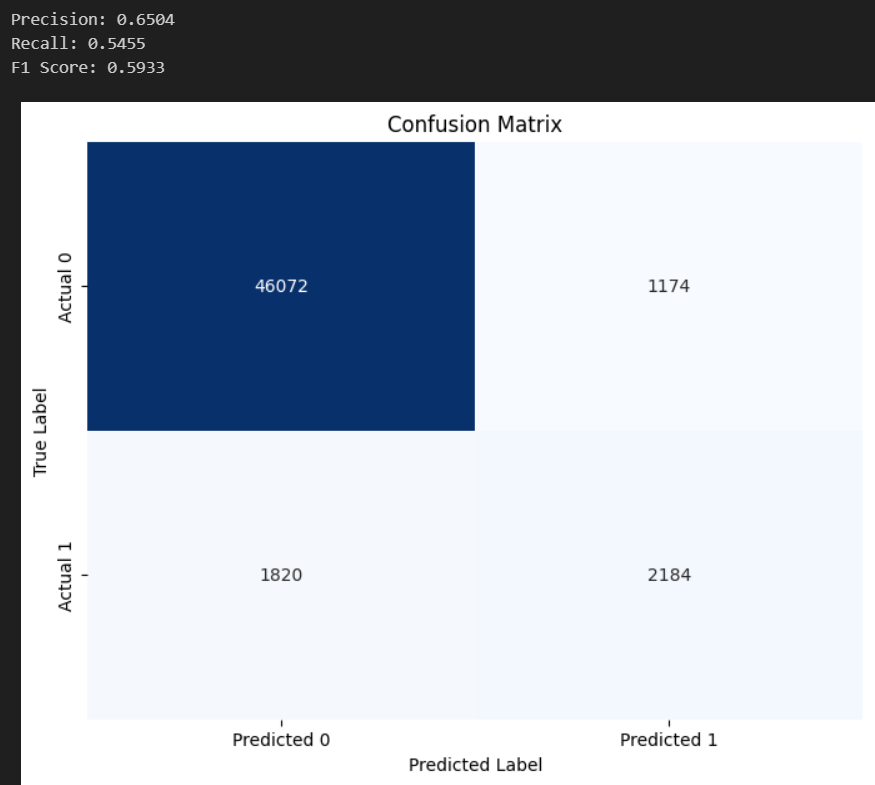


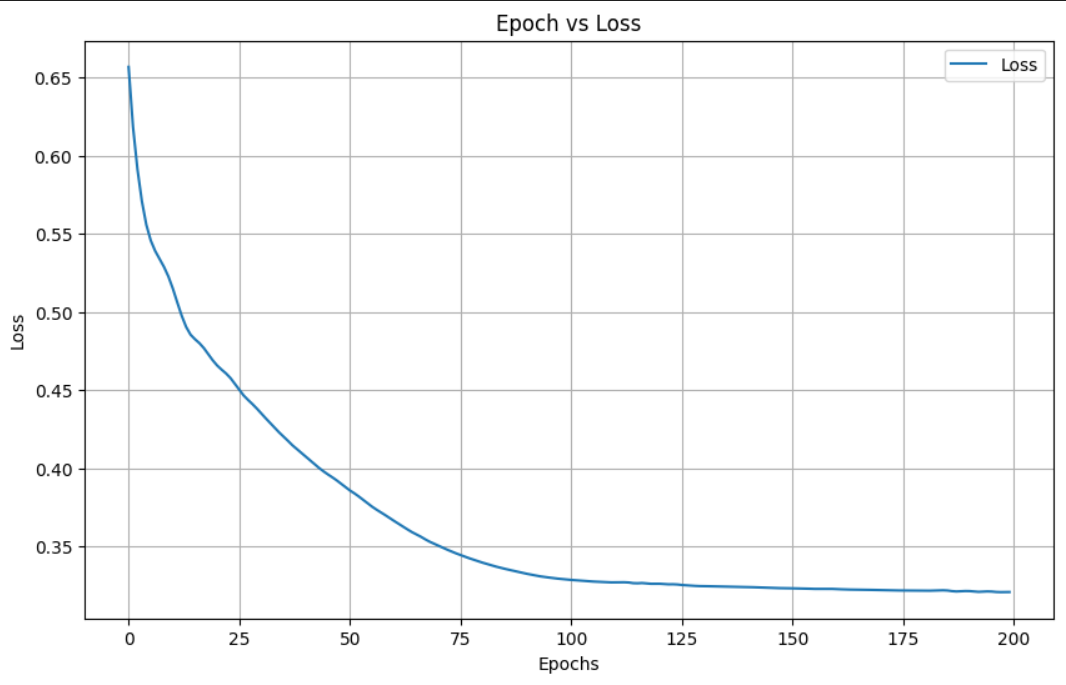
These below metrics are for validation dataset like Training over one dataset and testing over completely new dataset. 

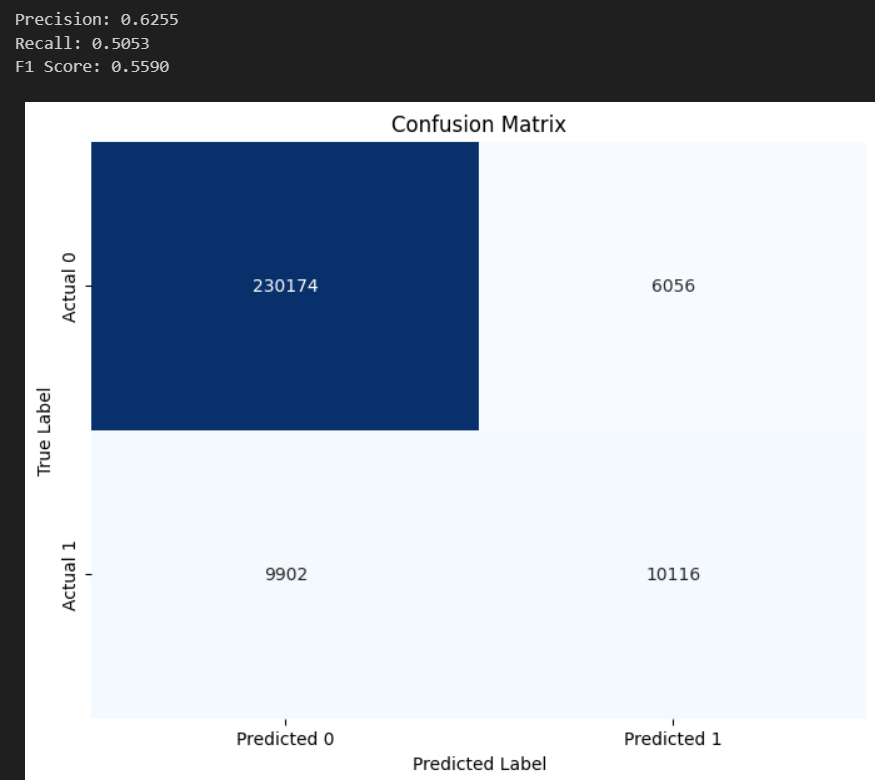
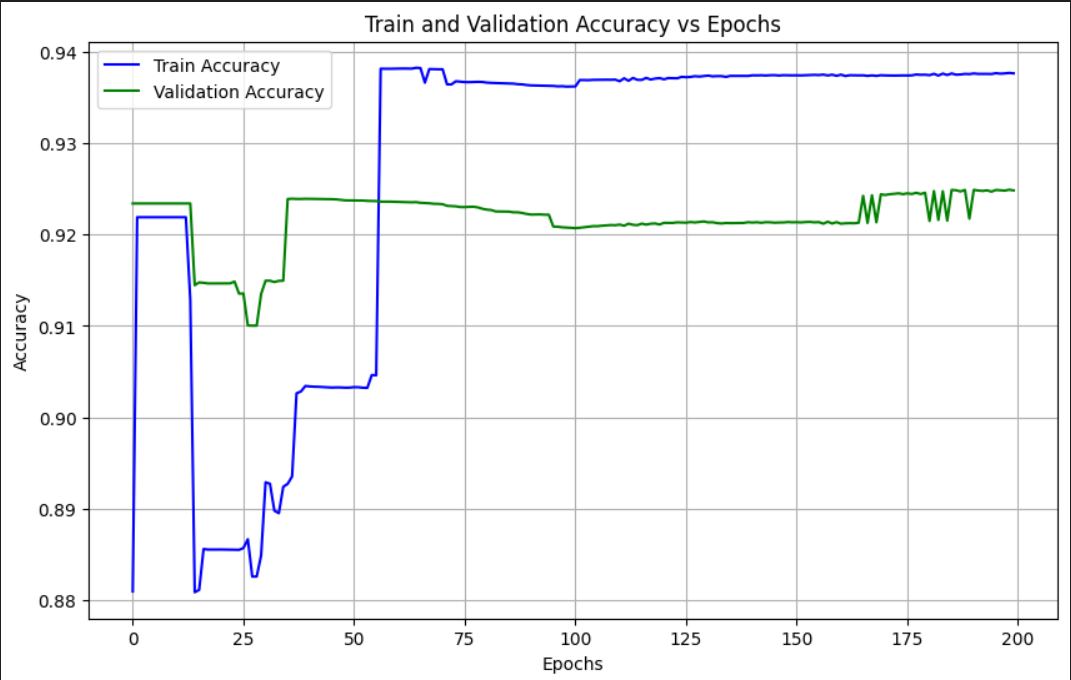


## **GraphSAGE:**

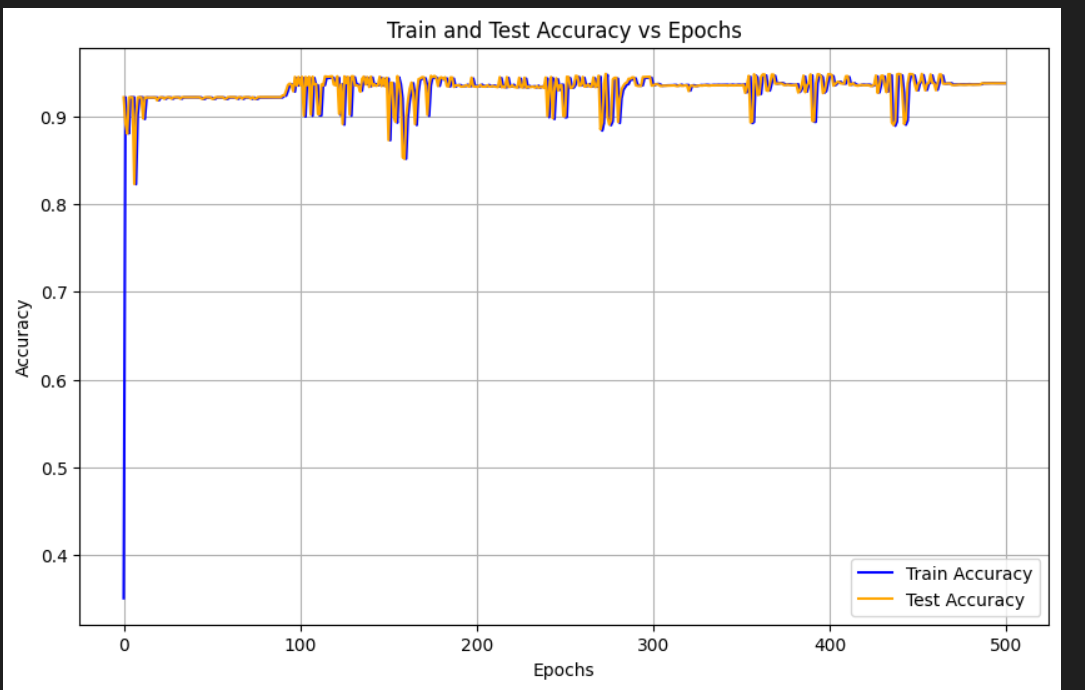
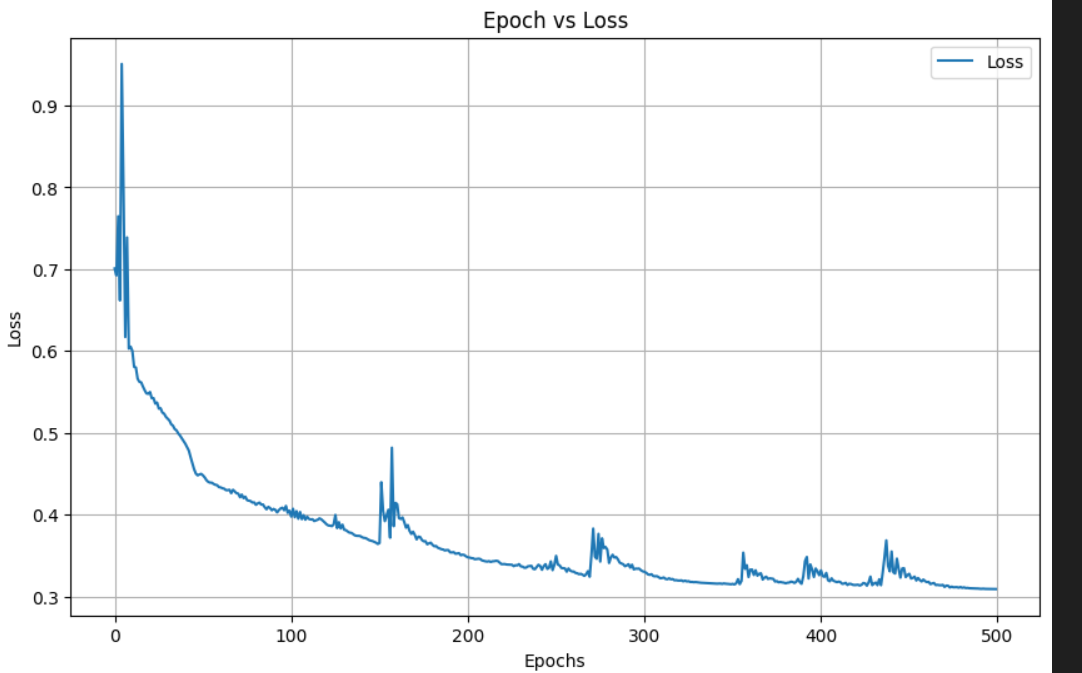
These below metrics are obtained when training and testing split is of 80% and 20%.

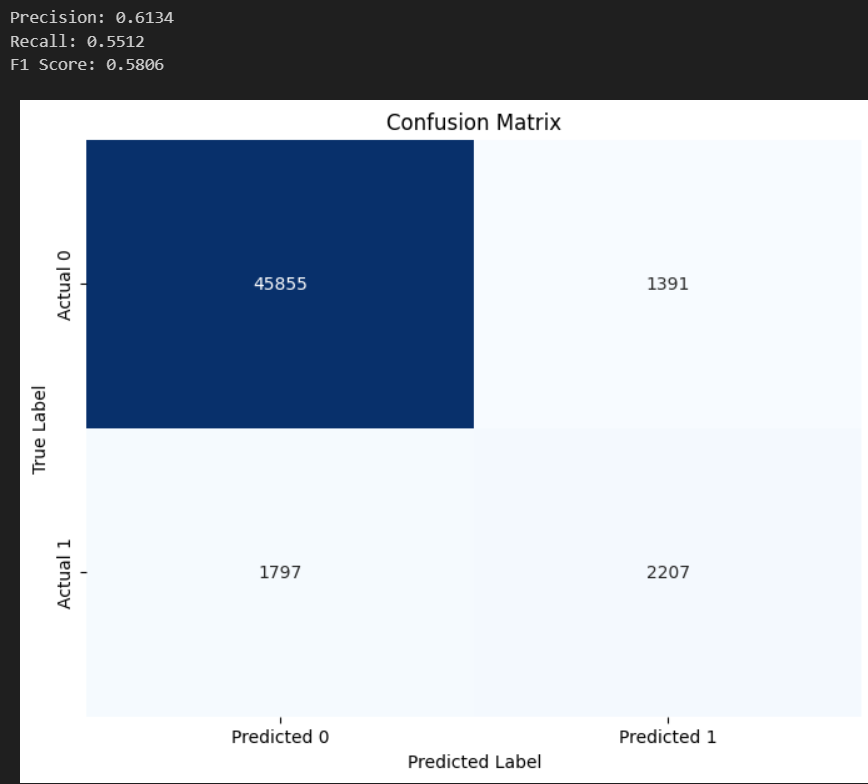


These below metrics are for validation dataset like Training over one dataset and testing over completely new dataset. 

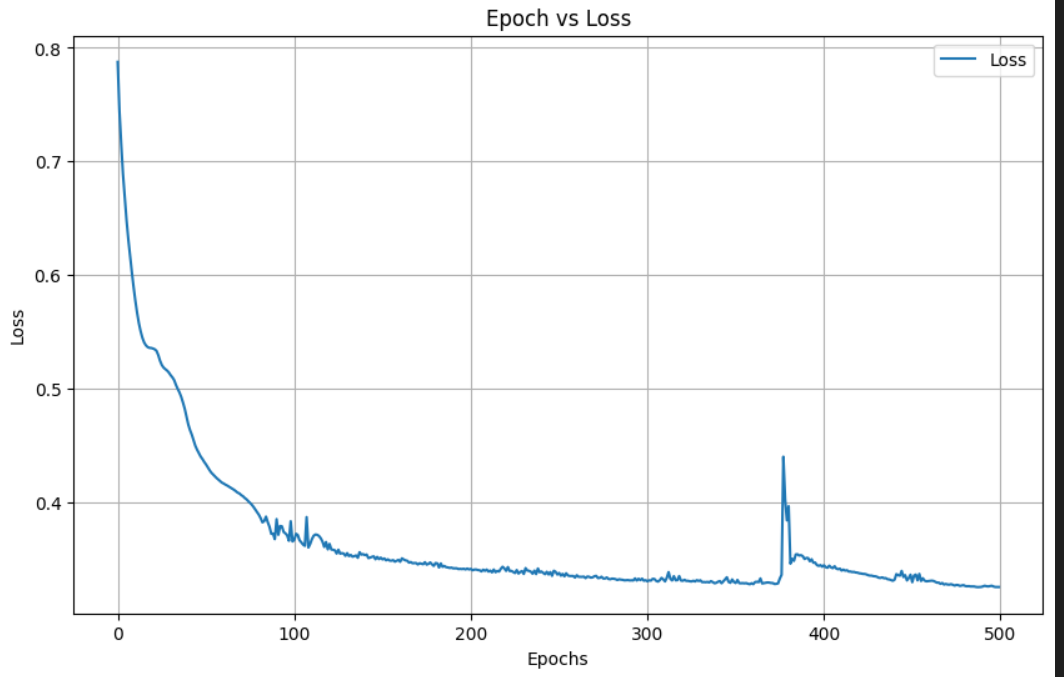


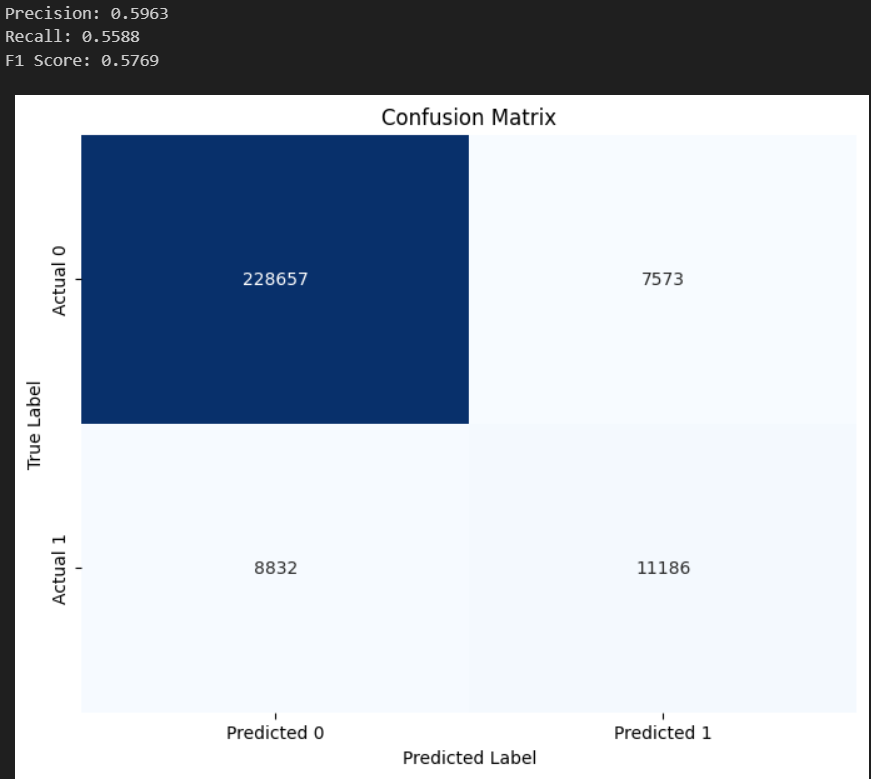
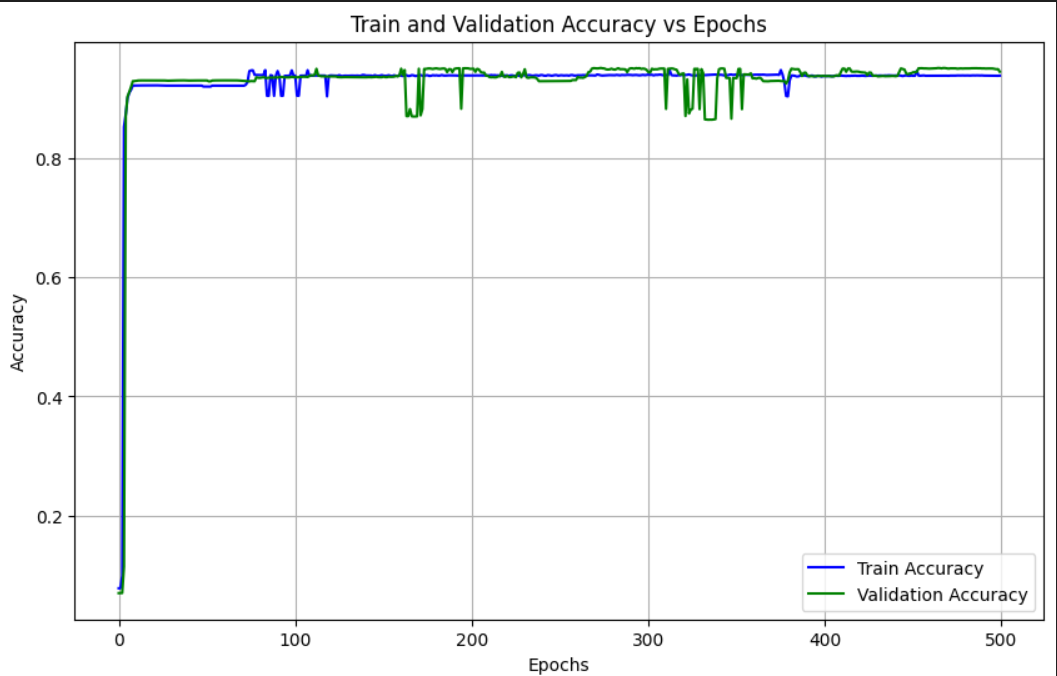
## **GIN:**

These below metrics are obtained when training and testing split is of 80% and 20%.



These below metrics are for validation dataset like Training over one dataset and testing over completely new dataset.





# **CONCLUSION:**

This study effectively illustrates the significance and efficacy of employing graph-based analysis to categorise Ethereum accounts into Externally Owned Accounts (EOAs) and Smart Contracts. We have extracted significant features—like in-degree, out-degree, weighted degrees, degree centrality, clustering coefficient, and PageRank—that represent the distinct relational contexts and behavioural patterns of these account types by utilising the interconnectedness of blockchain data.

Graph analysis proved to be a powerful tool, capturing both local interactions (e.g., transaction patterns) and global importance (e.g., centrality, connectivity) of accounts. These features enabled precise classification and enhanced interpretability compared to traditional analysis.

# **FUTURE WORK:**

This project lays the foundation for more advanced applications, such as anomaly detection, predictive modelling of smart contract behaviours, and enhancing the security of decentralized applications. Future research could incorporate temporal graph analysis or advanced techniques like Graph Neural Networks (GNNs) to further improve classification accuracy and explore dynamic account behaviours over time.

In conclusion, the project highlights the significance of graph analysis in blockchain research and contributes to a deeper understanding of Ethereum's account dynamics, paving the way for innovations in decentralized systems.

# **REFERENCES:**

* P. Zheng, Z. Zheng, J. Wu and H. -N. Dai, "XBlock-ETH: Extracting and Exploring Blockchain Data From Ethereum," in *IEEE Open Journal of the Computer Society*, vol. 1, pp. 95-106, 2020, doi: 10.1109/OJCS.2020.2990458. keywords: {Blockchain;Smart contracts;Data mining;Bitcoin;Blockchain;data analytics;ethereum;smart contracts},
* Patel, V., Pan, L. and Rajasegarar, S., 2020. Graph deep learning based anomaly detection in ethereum blockchain network. In *International conference on network and system security* (pp. 132-148). Springer, Cham.
* Huang, T., Lin, D. and Wu, J., 2022. Ethereum account classification based on graph convolutional network. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *69*(5), pp.2528-2532.