National College of Ireland

Enhancing Ethereum Fraud Detection Accuracy with Sparse-Attention-Based Model

MSc Research Project
MSc Data Analytics-C

Vineeth Kumar Reddy Chandravathi
Student ID: x23199091

School of Data Analytics
National College of Ireland

Supervisor: Harshani Nagahamulla

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vineeth Kumar Reddy Chandravathi
Student ID:	X23199091
Programme:	Data Analytics-C **Year:** 2024-25
Module:	MSc Research Project
Lecturer:	Harshani Nagahamulla
Submission Due Date:	11/08/2025
Project Title:	Enhancing Ethereum Fraud Detection Accuracy with Sparse- Attention-Based-Model
Word Count:	869 **Page Count:** 7

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: Vineeth Kumar Reddy Chandravathi

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):	

Enhancing Ethereum Fraud Detection Accuracy with Sparse-Attention-Based Model

Vineeth Kumar Reddy Chandravathi
Student ID: x23199091

1. Introduction

This manual details the **dataset preparation**, **model setup**, and **execution workflow** for detecting fraudulent Ethereum transactions.
The configuration aligns with the *Final Report – Transactions*, ensuring all dataset preprocessing steps, model configurations, and evaluation metrics match the original research.

System Infrastructure Prerequisites

- **Platform Compatibility**
    - **Primary Choice**: Windows 10 or Windows 11 — ensures broad compatibility with most development tools and frameworks
    - **Alternative Option**: Ubuntu 20.04 LTS or later — well-suited for open-source workflows and server-based deployments
- **Processing Requirements**
    - **Minimum Specification**: Multi-core processor (Intel Core i5 / AMD Ryzen 5 or newer)
    - **Recommended Specification**: Quad-core or higher for faster computation and efficient model training
- **Memory Capacity**
    - **Minimum**: 16 GB RAM — sufficient for smaller workloads and testing environments
    - **Recommended**: 32 GB RAM or above for handling larger datasets and concurrent experiments
- **Optional Performance Boost**
    - NVIDIA GPU with CUDA support — significantly accelerates deep learning and model training tasks

- **Storage Needs**
    - 15–20 GB of available space to store raw and processed datasets, trained model files, embeddings, visual outputs, and dependencies

Software Environment Configuration

- **Programming Language**
    - Python version 3.9 or newer is recommended for optimal compatibility
- **Essential Data Analysis Packages**
    - pandas — structured data processing and manipulation *(pip install pandas)*
    - numpy — fast numerical and array computations *(pip install numpy)*
    - matplotlib — creation of static plots and charts *(pip install matplotlib)*
    - seaborn — advanced statistical visualization *(pip install seaborn)*
    - plotly — interactive and dynamic charting *(optional) (pip install plotly)*
    - scikit-learn — preprocessing utilities, ML models, and evaluation tools *(pip install scikit-learn)*
- **Deep Learning Frameworks**
    - torch (PyTorch) — primary library for model design, training, and deployment *(pip install torch)*

2. Dataset Specification

2.1 Source

Ethereum wallet-level transaction summaries in CSV format.

2.2 Target Variable

- FLAG – Binary classification:
    - 0: Legitimate wallet
    - 1: Fraudulent wallet

2.3 Feature Types

- **Temporal:** Block intervals, timestamps, transaction time patterns
- **Behavioral:** Unique senders/receivers, ERC20 token counts, transaction frequency

- **Financial:** Total Ether sent/received, average transaction values
- **Categorical:** Token names, platforms

3. Data Preprocessing Pipeline

3.1 Data Loading

- Load CSV into pandas DataFrame.
- Verify column names and data types.

3.2 Missing Values

- Drop columns with excessive null values.
- Impute remaining numeric values with **median**.

3.3 Feature Engineering

- Convert timestamps to datetime.
- Extract day, month, hour for temporal analysis.
- Create behavioral metrics such as transaction ratios.

3.4 Feature Selection

- Correlation-based filtering to remove collinear features.
- Retain most informative predictors.

3.5 Scaling

- Apply **StandardScaler** to numerical features.

3.6 Data Split

- Features (X) and target (y)
- Stratified **80-20 split** to preserve fraud ratio.

4. Model Framework

4.1 Baseline Machine Learning Models

1. **Ridge Classifier**

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report

# ==== Define hyperparameter grid ====
param_grid = {
    'alpha': np.logspace(-3, 3, 7)  # Try alpha from 0.001 to 1000
}

# ==== Grid search with cross-validation ====
ridge_grid = GridSearchCV(
    RidgeClassifier(),
    param_grid,
    cv=5,
    n_jobs=-1
)
ridge_grid.fit(X_train, y_train)

# ==== Best model predictions ====
ridge_pred = ridge_grid.predict(X_test)

# ==== Evaluation ====
ridge_acc = accuracy_score(y_test, ridge_pred)
```

2. Logistic Regression

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc, accuracy_score, classification_report

# Define hyperparameter grid for regularization strength
param_grid = {'C': np.logspace(-4, 4, 10)}

# GridSearch with 5-fold cross-validation
lr_grid = GridSearchCV(LogisticRegression(random_state=42, solver='liblinear'), param_grid, cv=5)
lr_grid.fit(X_train, y_train)

# Best model predictions
lr_pred = lr_grid.predict(X_test)
lr_prob = lr_grid.predict_proba(X_test)[:, 1]

# Evaluation metrics
lr_acc = accuracy_score(y_test, lr_pred)
fpr, tpr, _ = roc_curve(y_test, lr_prob)
lr_auc = auc(fpr, tpr)
```

3. Gaussian Naive Bayes

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, roc_curve, auc, classification_report

# ===== GaussianNB: for continuous features =====
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {'var_smoothing': np.logspace(-9, -5, 20)}
gnb = GridSearchCV(GaussianNB(), param_grid, cv=5)
gnb.fit(X_train_scaled, y_train)
y_pred = gnb.predict(X_test_scaled)
y_prob = gnb.predict_proba(X_test_scaled)[:, 1]
acc = accuracy_score(y_test, y_pred)
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
print("Best var_smoothing:", gnb.best_params_['var_smoothing'])
print("Accuracy:", acc)
print("AUC:", roc_auc)
print(classification_report(y_test, y_pred))
```

4.2 Deep Learning Models

There are two DL Models

1. **Feedforward Neural Network (FFNN)**
2. **Deep Autoencoder**

FNN Model

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Function to build the model with varying hyperparameters
def build_ffnn(dropout_rate=0.3, learning_rate=0.001):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
        BatchNormalization(),
        Dropout(dropout_rate),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(dropout_rate),
        Dense(1, activation='sigmoid')
    ])
    model.compile(
        optimizer=Adam(learning_rate=learning_rate),
        loss='binary_crossentropy',
        metrics=['accuracy', 'precision', 'recall']
    )
    return model

# Hyperparameter values to try
dropout_rates = [0.2, 0.3, 0.4]
learning_rates = [0.001, 0.0005]

# Dictionary to track validation accuracy for each combination
results = {}
```

4.3 TabNet – Primary Model

Architecture: Sparse attention-based feature selection for tabular data.

```python
# ==== TabNet Model ====
def build_tabnet(input_dim, n_steps=7, feature_dim=256, output_dim=128, n_glu=4, dropout_rate=0.25):
    inputs = Input(shape=(input_dim,))
    x = Dense(feature_dim)(inputs)
    x = BatchNormalization()(x)

    prior = Lambda(lambda x: tf.ones_like(x))(x)
    outputs = []

    for step in range(n_steps):
        transformer = FeatureTransformer(feature_dim, n_glu, dropout_rate)
        features = transformer(x)

        attn = AttentiveTransformer(feature_dim)(features, prior)
        masked = Multiply()([features, attn])

        decision = Dense(output_dim)(masked)
        decision = BatchNormalization()(decision)
        decision = Activation('relu')(decision)
        outputs.append(decision)

        prior = Multiply()([prior, Lambda(lambda a: 1.0 - a)(attn)])

    agg = Lambda(lambda x: tf.reduce_sum(tf.stack(x, axis=1), axis=1))(outputs)
    out = Dense(1, activation="sigmoid")(agg)

    return Model(inputs, out)
```

5. Evaluation Protocol

5.1 Metrics

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC

```
Final TabNet Accuracy: 85.47%
AUC Score: 0.9033

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.89      0.91      1533
           1       0.66      0.71      0.69       436

    accuracy                           0.85      1969
   macro avg       0.79      0.80      0.80      1969
weighted avg       0.86      0.85      0.86      1969
```

6. Future Enhancements

1. Add temporal and graph-based features.
2. Use advanced imbalance handling (SMOTE, focal loss).

3. Test model across multiple blockchain datasets.
4. Implement real-time drift detection and monitoring.

References

Python Official Documentation – https://docs.python.org/3/

NumPy Documentation – https://numpy.org/doc/

Pandas Documentation – https://pandas.pydata.org/docs/

Scikit-learn Documentation – https://scikit-learn.org/stable/documentation.html

PyTorch Documentation – https://pytorch.org/docs/stable/index.html

LightGBM Documentation – https://lightgbm.readthedocs.io/en/latest/

XGBoost Documentation – https://xgboost.readthedocs.io/en/stable/

Matplotlib Documentation – https://matplotlib.org/stable/users/index.html

Seaborn Documentation – https://seaborn.pydata.org/

Plotly for Python Documentation – https://plotly.com/python/

Jupyter Notebook Documentation – https://jupyter.org/documentation