IDENTIFICATION OF CREDIT CARD FRAUD UTILIZING HYBRID DEEP LEARNING MODELS WITH IMPROVED PRECISION AND MINIMIZED FALSE POSITIVES

A Project Work Report

Submitted in Partial Fulfilment for the Award of the Degree

Of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & BUSINESS SYSTEMS

Submitted by

Y DEVA	21B91A5763
V RAVI VARMA	21B91A5759
A SURYA	21B91A5702
N ANIL KUMAR	21B91A5740

Under the esteemed guidance of

Dr. N. DESHAI

M. Tech., Ph.D.

Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE (AUTONOMOUS)

(Approved by AICTE, New Delhi, Affiliated to JNTUK Kakinada)

CHINNA AMIRAM :: BHIMAVARAM-534202

April - 2025

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE (AUTONOMOUS)

(Approved by AICTE, New Delhi, Affiliated to JNTUK KAKINADA)

CHINNA AMIRAM, BHIMAVARAM-534204

DEPARTMENT OF INFORMATION TECHNOLOGY

Certificate

This is to certify that the Project Work report entitled " IDENTIFICATION OF CREDIT CARD FRAUD UTILIZING HYBRID DEEP LEARNING MODELS WITH IMPROVED PRECISION AND MINIMIZED FALSE POSITIVES ", is Bonafide work submitted by **Y DEVA (Regd.No: 21B91A5763), V RAVI VARMA (Regd.No: 21B91A5759), A SURYA (Regd.No: 21B95A5702), N ANIL KUMAR (Regd.No: 21B91A5740)** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Business System during the Academic year 2024-2025.

HOD

Dr. P RAVI KIRAN VARMA
HOD-IT Dept

GUIDE

Dr. N. DESHAI
Assistant Professor

CERTIFICATION OF EXAMINATION

This to certify that I have examined the concept and hereby accord my approval of it as a Project Work entitled "**IDENTIFICATION OF CREDIT CARD FRAUD UTILIZING HYBRID DEEP LEARNING MODELS WITH IMPROVED PRECISION AND MINIMIZED FALSE POSITIVES** " carried out and presented in a manner required for its acceptance on partial fulfilments for the award of the degree of BACHELOR OF TECHNOLOGY in Computer Science & Business Systems for which it has been submitted.

This approval does not necessarily endorse or accept every statement made opinion expressed or conclusions drawn as recorded in the Project Work report it only signifies the acceptance of the report for the purpose for which submitted.

Signature

Project Guide :

Senior Faculty :

External Examiner :

HOD :

DECLARATION

This Project Work report entitled "**IDENTIFICATION OF CREDIT CARD FRAUD UTILIZING HYBRID DEEP LEARNING MODELS WITH IMPROVED PRECISION AND MINIMIZED FALSE POSITIVES** " has been carried out by us in the partial fulfilment of the requirements for the award of the degree of B. Tech (IT), Sagi Rama Krishnam Raju Engineering College(A). We hereby declare this project work/project report has not been submitted to any of the other University/Institute for the award of any other degree/diploma.

Name	Register No	Signature
Y DEVA	21B91A5763	
V RAVI VARMA	21B91A5759	
A SURYA	21B91A5702	
N ANIL KUMAR	21B91A5740	

ACKNOWLEDGEMENT

The victorious completion of this project would be incomplete without greeting those who made it possible and whose guidance and encouragement made efforts that taken to the success.

The thesis presented here is the work accomplished under the enviable and scholarly guidance of **Dr. N. DESHAI**, Assistant Professor of Information Technology Department. We are grateful for the indomitable support. We express deep gratitude for his inspiring remarks and for helping with this project.

We take this opportunity to express our sincere thanks to **Dr. P RAVI KIRAN VARMA**, Head of the Department, Information Technology, for his support and encouragement.

We express our sincere thanks to **Prof. K V Murali Krishnam Raju**, PRINCIPAL, S.R.K.R Engineering College, Bhimavaram for giving us this opportunity for the successful completion of this project.

Finally, we also express our sincere thanks to other **Teaching and non-Teaching staff** for their support in accomplishing this project.

1.	Y DEVA	21B91A5763
2.	V RAVI VARMA	21B91A5737
3.	A SURYA	21B91A5764
4.	N ANIL KUAMR	21B91A5740

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Consumers and businesses face financial risks because of the increase in credit card fraud brought on by the boom in digital transactions. Traditional fraud detection technology can't keep up with evolving fraud strategies, which leads to high false positives and undetected fraud. This paper proposes a hybrid deep learning system that integrates Autoencoders, Conv1D, SMOTE, and LSTM to increase the accuracy of fraud detection. SMOTE addresses class imbalance, autoencoders extract complex transaction patterns, Conv1D detects local dependencies, and LSTM captures long-term temporal correlations. Class imbalance is addressed by SMOTE, complicated transaction patterns are extracted by autoencoders, local dependencies are detected by Conv1D, and long-term temporal correlations are captured by LSTM. When compared to traditional models, experimental results on the European Credit Card Dataset demonstrate improved precision, recall, and F1-score. The results highlight how crucial hybrid deep learning is to create adaptive fraud detection systems that can react to new fraud trends. To improve financial security, future work will concentrate on real-time deployment and enhancing model interpretability.

CHAPTER 1
INTRODUCTION

The rapid advancement of digital payment systems has revolutionized financial transactions, enabling seamless international purchases. However, this growth has also led to a significant rise in credit card fraud, posing serious financial threats to both consumers and financial institutions. Reports indicate that fraudulent transactions result in billions of dollars in losses annually, emphasizing the urgent need for more sophisticated fraud detection systems. Traditional rule-based and machine learning models struggle with high false positive rates, delayed fraud detection, and limited adaptability to evolving fraud strategies, ultimately compromising security and customer trust.

A key challenge in fraud detection is the class imbalance within credit card transaction datasets, where fraudulent transactions constitute only a small fraction of the total records. This disparity biases models toward legitimate transactions, leading to misclassification of fraudulent activities. Additionally, static detection algorithms fail to address the constantly evolving fraud patterns, making them ineffective for real-time fraud prevention.

To address these limitations, this paper proposes a hybrid deep learning framework that integrates Synthetic Minority Over-sampling Technique (SMOTE), Autoencoders, Conv1D, and Long Short-Term Memory (LSTM) to enhance fraud detection accuracy. SMOTE mitigates class imbalance, LSTM captures long-term temporal dependencies, autoencoders extract hidden transaction patterns, and Conv1D identifies spatial relationships within transactional data. This hybrid approach has demonstrated superior fraud detection performance with lower false positive rates, outperforming traditional models in precision, recall, and F1-score, as validated on the Credit Card Fraud Detection Dataset.

CHAPTER 2
LITERATURE REVIEW

Prabha and Priscilla et al. (2024) proposed a fraud detection model utilizing LSTM autoencoders with an attention mechanism for high-level feature extraction, followed by classification using XGBoost . Their approach, using an adaptive threshold ($\theta = 0.22$), achieved 90.5% recall and 94.2% accuracy, significantly improving recall performance. When tested on the IEEE-CIS dataset, the model outperformed existing fraud detection techniques.

Ileberi and Sun et al. demonstrated that their model surpasses individual fraud detection techniques, achieving an AUC-ROC of 0.972, sensitivity of 0.961, and specificity of 0.999 on the European Credit Card Dataset. Similarly, Sumaya S. Sulaiman et al. investigated deep learning models such as Autoencoder, CNN, and LSTM with hyperparameter tuning for fraud detection. Their study found that LSTM performed best, achieving 99.2% accuracy, a 93.3% detection rate, and a 96.3% AUC-ROC score. Their work also highlighted the class imbalance issue and successfully integrated SMOTE to enhance real-time fraud detection.

Ibomoiye Domor Mienye et al. explored various deep learning models, including CNNs, RNNs, LSTMs, and GRUs, for fraud detection. Their research emphasized the robustness of deep learning over traditional machine learning techniques, particularly in handling class imbalance through data augmentation.

Fawaz Khaled Alarfaj et al. investigated the effectiveness of Graph Neural Networks (GNNs) and Autoencoders in fraud detection. Their study highlighted how these models enhance fraud prevention strategies and strengthen financial security. Similarly, Iseal et al. examined the role of deep learning in reducing false positives and identifying complex fraud patterns, suggesting blockchain, federated learning, and enhanced data interpretability for future improvements.

Dhandore et al. proposed a real-time fraud detection framework that enhances F1-score, recall, and accuracy while reducing false positives. Their work emphasized adaptive learning and real-time data processing to detect emerging fraud trends. Pali vela et al. introduced a hybrid approach, utilizing CNNs for spatial feature extraction, LSTMs for temporal sequence analysis, and transformers for dependency modelling. Their model achieved AUC-ROC (0.972), specificity

(0.999), and sensitivity (0.961) on the European Credit Card Dataset, demonstrating superior performance.

San Miguel Carrasco et al. developed an Optimized Sequential Model for Fraud Detection, integrating hyperparameter tuning and ensemble learning techniques such as Random Forest, Gradient Boosting, Logistic Regression, and Voting Classifiers. Their model, using pre-processed transaction data and SMOTE for class balancing, significantly outperformed traditional fraud detection systems.

CHAPTER 3
EXISTING SYSTEM AND PROBLEM STATEMENT

3.1 EXISTING SYSTEM

In the current landscape of credit card fraud detection, traditional systems largely rely on rule-based algorithms and conventional machine learning techniques such as Decision Trees, Random Forests, and Support Vector Machines (SVM). These systems use predefined patterns and historical data to identify fraudulent behavior. While they can perform adequately under certain conditions, they struggle to adapt to the constantly evolving tactics used by fraudsters. Their inability to learn from sequential or temporal patterns within transactions limits their effectiveness in detecting real-time or subtle fraud patterns.

Moreover, these systems often suffer from high false positive rates, where legitimate transactions are flagged as fraudulent. This leads to customer dissatisfaction and unnecessary operational costs. Another major limitation is their poor handling of class imbalance—in most datasets, fraudulent transactions are vastly outnumbered by legitimate ones, causing traditional models to become biased toward the majority class and miss out on rare fraud cases.

Although some modern systems incorporate deep learning methods such as individual CNNs or LSTMs, these are often used in isolation and lack a comprehensive mechanism to capture the diverse patterns present in fraud behavior. They also frequently ignore the benefits of data balancing techniques like SMOTE, which further reduces their accuracy and robustness.

3.2 PROBLEM STATEMENT

The increase in digital payment adoption has led to a corresponding surge in credit card fraud, posing serious risks to financial institutions and consumers. Existing fraud detection systems are not equipped to handle the sophisticated and dynamic nature of modern fraud techniques. These systems face multiple limitations, including high false positive rates, inadequate detection of evolving fraud trends, and poor performance due to class imbalance in transaction data.

Detecting fraud is particularly challenging due to the rarity of fraudulent transactions in datasets, making it difficult for models to learn meaningful patterns. Additionally, the lack of integration

between different types of deep learning models prevents existing systems from effectively capturing spatial, temporal, and latent transaction features.

To address these gaps, there is a critical need for a hybrid deep learning framework that can simultaneously manage class imbalance, learn complex transaction representations, detect local and sequential patterns, and reduce false positives—ultimately providing high precision, improved recall, and a more adaptive and real-time fraud detection solution.

CHAPTER 4
PROPOSED SYSTEM

4.1 Proposed System Overview

To overcome the limitations of traditional fraud detection techniques, a hybrid deep learning-based fraud detection system is proposed. This system intelligently integrates multiple advanced components—SMOTE (Synthetic Minority Over-sampling Technique), Autoencoders, 1D Convolutional Neural Networks (Conv1D), and Long Short-Term Memory (LSTM) networks—to enhance accuracy, minimize false positives, and adapt to evolving fraud patterns.

The system starts by addressing the class imbalance issue commonly found in credit card transaction datasets using SMOTE, which generates synthetic samples of the minority (fraudulent) class. This ensures a balanced dataset, allowing the model to effectively learn patterns from both legitimate and fraudulent transactions without bias.

Next, Autoencoders are used for unsupervised feature extraction. They compress and reconstruct transaction data, capturing complex and hidden patterns that may not be apparent with traditional feature engineering. These learned representations help in identifying anomalies in transaction behavior.

The Conv1D layers are introduced to capture local dependencies across sequential transaction features. They scan transaction sequences to detect short-term irregularities that might signal fraud. These local features are especially useful in detecting micro-patterns of suspicious activity.

To understand long-term dependencies, the system incorporates LSTM networks, which are effective in analyzing the temporal sequence of transactions. LSTM captures the order and context of previous transactions to predict if the current one is likely to be fraudulent.

Finally, the outputs from the Conv1D and LSTM layers are concatenated and passed through fully connected layers to perform the final classification. A sigmoid activation in the output layer is used to determine whether a transaction is fraudulent or legitimate.

This integrated framework not only improves detection accuracy, precision, and recall, but also significantly reduces false positives, making it more reliable than standalone models. Experimental

results on the Credit Card Fraud Detection Dataset confirm that the proposed hybrid model achieves superior performance, offering a scalable and effective solution for real-world fraud detection scenarios.

CHAPTER 5
SYSTEM ARCHITECTURE

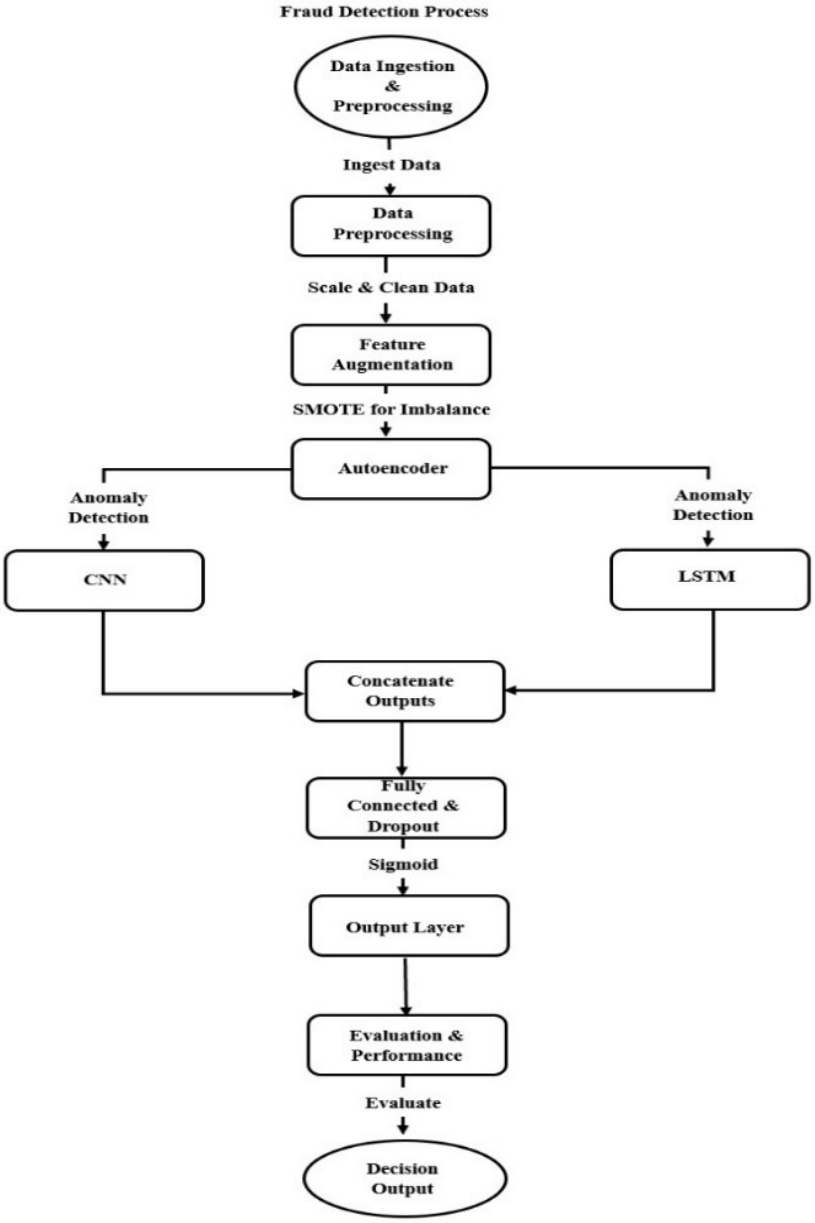


Fig-5.1.1: System Architecture of the Proposed Solution

The architecture presented in Fig. 5.1 outlines a structured and systematic deep learning-based framework for the detection of credit card fraud. The workflow initiates with the Data Workflow, beginning at the Data Collection stage. Here, transactional data from sources such as the European

Credit Card Dataset is gathered. This stage is critical, as the diversity and integrity of transaction data directly affect the model's ability to identify fraudulent patterns.

Following data acquisition, the pipeline transitions to Data Preprocessing, where missing entries are addressed, numerical features are normalized, and class imbalance is corrected using SMOTE. The processed data then moves to Feature Extraction, where patterns are captured using autoencoders and reconstruction loss is evaluated for anomaly detection. These enriched features form the foundation for downstream classification.

Once features are extracted, the workflow enters the Model Workflow phase, beginning with Data Splitting. The dataset is partitioned into training and testing subsets, ensuring fair representation of both fraud and non-fraud classes. The Model Development module then builds two models: an autoencoder for learning latent transaction features, and a hybrid deep learning model comprising Conv1D, LSTM, and Dense layers. This structure enables detection of both spatial and sequential fraud indicators.

Post-training, the Model Evaluation stage assesses each model using metrics like accuracy, precision, recall, F1-score, and ROC-AUC. Visualizations such as training curves, confusion matrices, and t-SNE plots are employed to understand the performance.

Finally, the Deployment and Real-Time Evaluation stage ensures the trained model is capable of real-time fraud detection. Transactions can be processed on-the-fly, and model performance is periodically reviewed to ensure it remains adaptive to evolving fraud tactics.

Overall, this architecture represents a comprehensive and dynamic solution to financial fraud detection, integrating deep learning strategies with data balancing and visualization. The following sections provide a breakdown of each module, highlighting their design rationale and implementation.

5.1 INPUT MODULE

- **Description:** The Input Module serves as the entry point for the credit card fraud detection framework. This module manages the acquisition of transaction data, typically from publicly available datasets such as the European Credit Card Dataset. These datasets contain a mix of legitimate and fraudulent transactions with numerical features derived from PCA transformations, along with fields such as time, amount, and a binary class label

indicating fraud. The input module performs validation checks to ensure no missing or corrupt entries and prepares the dataset for further processing.

- **Purpose:** The main goal of this module is to gather, clean, and validate raw transaction data. By verifying data integrity and format consistency, it ensures smooth progression through the pipeline. This module supports CSV input formats and can be adapted for real-time streaming inputs in future versions.
- **Output:** The result is a clean and verified transaction dataset that is properly structured and ready for preprocessing operations.

5.2 Data Preprocessing and Feature Extraction Module:

- **Purpose:** This module is responsible for preparing the raw dataset for modeling by applying standard preprocessing techniques and feature engineering. It begins by removing entries with missing values, scaling features to normalize distributions using StandardScaler, and addressing class imbalance using SMOTE (Synthetic Minority Over-sampling Technique). SMOTE synthetically generates fraud-class samples to balance the dataset. Feature extraction is intrinsic to the data due to PCA transformation, but additional insights such as reconstruction error from autoencoders can be used to enhance detection.
- **Techniques and Workflow:**

  Handle Missing Values and Duplicates:

  - Remove rows with missing 'Class' values.

  Normalize Features:

  - Apply StandardScaler to transform the feature values to zero mean and unit variance for better convergence.

  Handle Imbalanced Classes:

  - Use SMOTE to generate synthetic fraudulent examples.

  Feature Engineering:

  - Use autoencoders to extract complex transaction representations, and analyze reconstruction loss as a potential anomaly score.

Output:

- The module outputs a normalized, balanced, and feature-rich dataset that enhances the model's ability to differentiate between fraudulent and legitimate transactions.

5.3 Data Splitting Module:

- Purpose: This module splits the balanced dataset into training and testing subsets. An 80:20 split is commonly used to ensure model generalizability. Stratified sampling ensures that the distribution of fraud and non-fraud classes remains consistent across training and testing sets.

• Techniques and Workflow:

o Perform an 80:20 split of data using train_test_split.

o Apply stratification to maintain proportional class distribution.

o Ensure that scaled and resampled data flows to both training and test sets.

• Output:

The output consists of four partitions: X_train, X_test, y_train, and y_test, used for training and evaluating the models.

5.4 Model Development Module:

• Purpose: This module builds two key models: an Autoencoder for feature compression and anomaly detection, and a Hybrid Deep Learning Model combining Conv1D, LSTM, and Dense layers. The Autoencoder learns a compact representation of the transactions, and the hybrid model utilizes spatial and temporal patterns for binary classification.

• Techniques and Workflow:

o Autoencoder Architecture:

▪ A simple feedforward model with an encoding layer and reconstruction layer.

▪ Trained using mean squared error to reconstruct inputs.

o Hybrid Model Architecture:

▪ Input reshaped and processed by Conv1D layers to capture local spatial patterns.

- LSTM layers capture temporal dependencies.
- Concatenation followed by Dense and Dropout layers.
- Output layer uses sigmoid activation for binary classification.

o Training Strategy:

- Use EarlyStopping and ModelCheckpoint callbacks.
- Train with balanced data from SMOTE and scaled inputs.

• Output:

Trained autoencoder weights and a robust hybrid model capable of accurately classifying transactions.

5.5 Metric Evaluation Module:

Metrics Calculated:

o Accuracy: Measures overall correctness.

o Precision: Focuses on the ratio of true fraud predictions to all fraud predictions.

o Recall: Measures how many actual fraud cases were correctly identified.

o F1 Score: Harmonic mean of precision and recall.

o AUC-ROC: Measures the model's ability to separate classes.

• Purpose: To rigorously evaluate the hybrid model's performance, especially its ability to detect fraudulent transactions with minimal false positives and false negatives. Metrics are visualized using graphs for training/validation loss and accuracy, and confusion matrix plots.

• Output:

o A complete performance report including evaluation metrics, classification report, confusion matrix, and fraud/non-fraud counts.

5.6 Output Module:

Components:

o Prediction Display: The model predicts whether a transaction is fraudulent or legitimate.

- o Confusion Matrix: Visualizes the distribution of true/false positives and negatives.
- o Metrics Summary: Displays key metrics like accuracy, precision, recall, and AUC.
- o Visual Insights: Includes t-SNE 3D plots and bar charts for fraud distribution.

• Purpose: Acts as the interface between model inference and user interpretation. This module provides actionable insight into fraud detection capabilities, highlights areas of success and failure, and aids in transparency.

• Output:

o Visualization and classification results are provided to help users understand the effectiveness of the model and ensure informed decision-making.

CHAPTER 6
SYSTEM REQUIREMENTS AND DESIGN

6.1 System Requirements

6.1.1 Software Requirements

Python: Python is an interpreted, high-level, general-purpose programming language created by Guido Van Rossum and first released in 1991. It emphasizes code readability with its notable use of significant whitespace. Python's language constructs and object-oriented approach help programmers write clear, logical code for both small and large-scale projects. Python is dynamically typed and garbage collected. It supports multiple paradigms, including procedural, object-oriented, and functional programming. Python serves as the core language for the phishing website detection project, as it's well-suited for machine learning tasks and implementing algorithms like DCP and CLAHE.

Libraries/Frameworks:

Keras: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It is used for the deep learning part of the project, particularly for building and training autoencoders and hybrid models for fraud detection.

Scikit-learn: Scikit-learn is a powerful and user-friendly machine learning library built on top of NumPy, SciPy, and matplotlib. It provides simple and efficient tools for data mining and analysis. For this project, it will be used for training and evaluating machine learning models, such as decision trees, support vector machines, and evaluating models with metrics like accuracy, precision, recall, and F1-score.

NumPy: This library is essential for numerical computations. It provides an efficient multidimensional array object and tools for working with these arrays. NumPy helps manipulate datasets and perform mathematical operations on data, which is crucial for processing large volumes of data in the project.

Pandas: Pandas is an open-source library for data manipulation and analysis. It provides efficient data structures like Series and DataFrame to handle structured data, making it an essential tool for handling the datasets used in phishing website detection, preprocessing, and analysis.

Matplotlib: Matplotlib is used for visualizing the results of the phishing website detection system, including plotting training and validation metrics, as well as creating confusion matrices and performance charts.

Imbalanced-learn (SMOTE): The SMOTE (Synthetic Minority Over-sampling Technique) algorithm is used to balance the class distribution in the dataset, helping to mitigate issues caused by class imbalance in the training process.

Operating System: Windows/Linux/MacOS: The project is platform-independent, and any of these operating systems are suitable for Python development. Anaconda or a virtual environment can be used to manage Python libraries effectively.

Development Environment:

IDE: Development can be done in PyCharm or Visual Studio Code, both of which support Python development efficiently. Jupyter Notebooks is also recommended for interactive development, especially for visualizing data and debugging code.

Version Control: Git is recommended for version control, and GitHub can be used for collaboration and sharing code with others.

6.1.2 Hardware Requirements

Processor (CPU):

Intel Core i5/i7 or AMD Ryzen 5/7: These processors are ideal for handling the preprocessing tasks, training machine learning models, and performing deep learning operations. Higher-end processors like i7 or Ryzen 7 offer better performance for large datasets and computationally intensive tasks.

Graphics Card (GPU):

Dedicated GPU: While a dedicated GPU like NVIDIA's GTX or RTX series can significantly accelerate training times for deep learning models, it is not essential for the project. For traditional machine learning tasks in phishing website detection, a CPU will suffice. However, using a dedicated GPU can make model training faster, especially when deep learning methods are employed.

RAM:

4 GB RAM: This is the minimum requirement for basic machine learning tasks. However, for larger datasets and more complex models (like deep learning-based models), having '8 GB to 16 GB of RAM' will ensure smoother performance without memory bottlenecks.

Storage:

At least '5 GB of free SSD space' is recommended for storing the dataset, extracted features, model checkpoints, and other relevant files. SSDs are preferred for faster data processing and model training.

Display:

A 'Full HD resolution' display is sufficient for visualizing data, displaying performance metrics, and analyzing results like confusion matrices. A good resolution ensures clarity when reviewing the results of your models.

6.2 SYSTEM DESIGN

System design defines the overall architecture, components, and workflows involved in building an intelligent Fraudulent Transaction Detection System. This system leverages deep learning techniques, including Autoencoders, Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) networks, to accurately identify suspicious transactions. The design focuses on key aspects such as data preprocessing, feature transformation, model architecture, training, and evaluation—all orchestrated to detect financial fraud with high accuracy and interpretability.

The design emphasizes modularity, enabling each stage to be independently improved or replaced as needed. This facilitates future enhancements, such as switching datasets, retraining on new fraud patterns, or replacing model components with more advanced architectures.

6.2.1 Use Case Diagram

UML stands for Unified Modeling Language; UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added ton or associated with UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. UML is a very important part of developing object- oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Software like StarUML and Rational Rose are popular modeling tools that help developers create these diagrams efficiently. StarUML is a modern, open-source modeling platform that supports multiple UML diagram types, such as class, sequence, activity, and use case diagrams. It offers an intuitive interface, easy drag-and drop features, and support for code generation and plugin extensions. On the other hand, Rational Rose, developed by IBM, is a legacy but powerful CASE (Computer-Aided Software Engineering) tool that integrates well with older development environments. It allows the modeling of object-oriented software systems using UML and provides round-trip engineering capabilities. These tools help in analyzing system requirements, designing architecture, identifying classes and relationships, and ensuring clear communication among team members. By using UML diagrams, developers can better understand system functionality, reduce development errors, and maintain consistency throughout the software lifecycle

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases),

and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor.

A use case diagram commonly contains:

 Use cases: It is meant by a list of actions or event steps typically defining the interactions between a role (known as Unified Modeling Language). These can be represented with a circle or ellipse.

 Actors: The actor can be human or any other external system. These can be represented with special symbols.

 Dependency, generalization, and association relationships.

A Use case diagram may also contain packages, which are used to group elements of your model into larger chunks.

Use case diagram is used in one of two ways:

 To model the context of a system.

 To model the requirements of a system.-

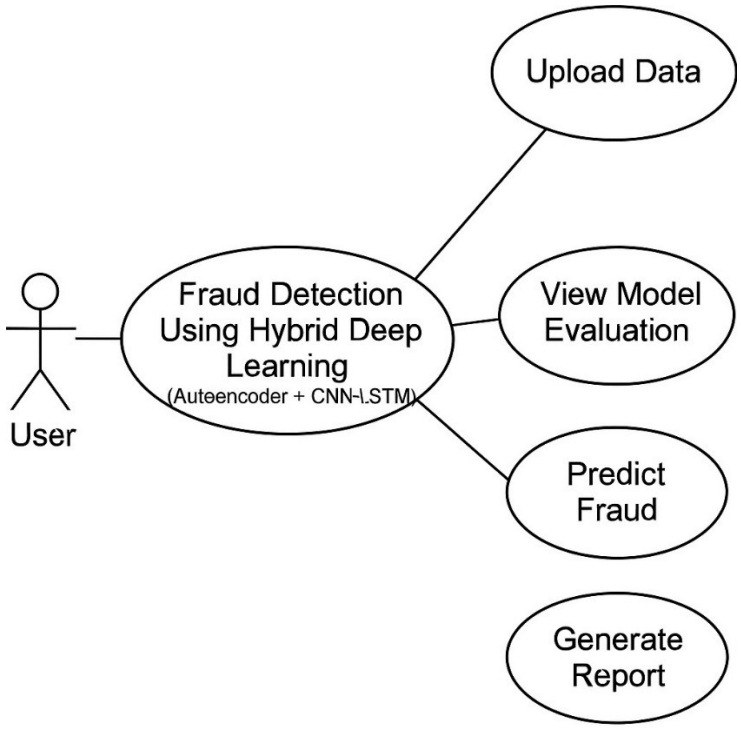


Fig-6.2.1.1: Use Case Diagram

6.2.2 Class Diagram

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

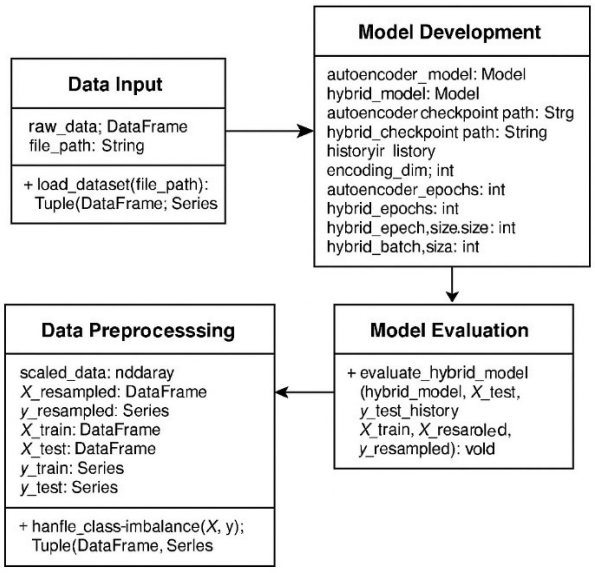


Fig-6.2.2.1: Class Diagram

6.2.3 Sequence Diagram

It displays interaction between objects that focus on the message from at temporal standpoint. The sequence diagram representation focuses on expressing interaction. An object is represented by a rectangle and its lifeline is represented by a vertical bar and dashed line. Sequence diagrams how the interaction between objects in a system, and it also specifies the sequence in which those interactions happen and add the dimension & in of time to your diagram. In the sequence diagram we only talk about time and ordering but not about the duration of time.

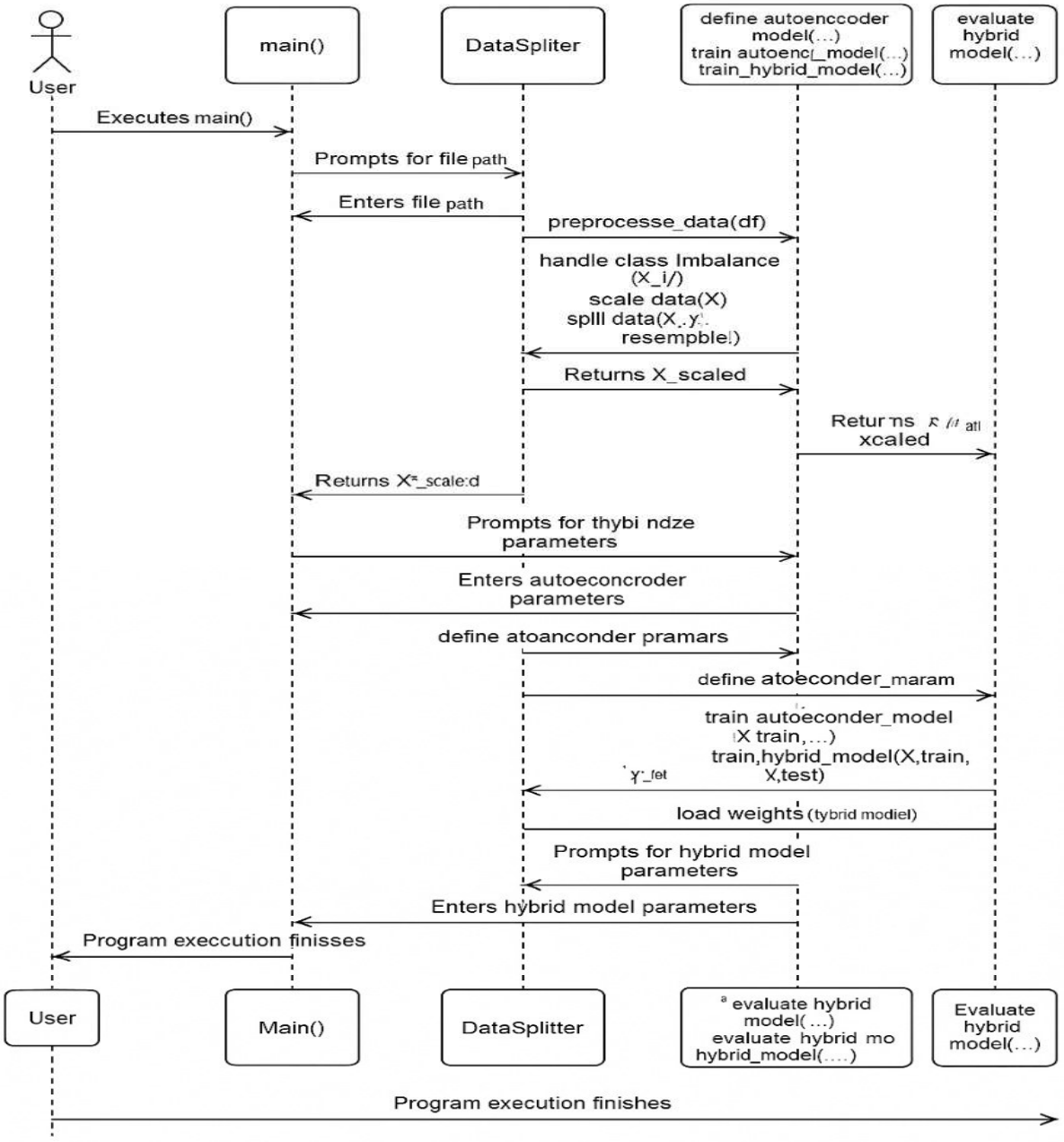


Fig-6.2.3.1: Sequential Diagram

6.2.4 Activity Diagram

An Activity Diagram provides a dynamic view of the phishing website detection system by modeling the workflow and activities involved in the process. It illustrates the step-by-step execution of operations, decision points, parallel processing, and the overall flow of control. Activity Diagrams are particularly useful in representing the sequence of actions and decisions in

applications, highlighting how data transitions through different stages. They help in understanding the logic behind the system's process flow, making it essential for both developers and stakeholders to grasp how different components interact during the detection process. The activity diagram illustrates the website classification process. It starts by receiving input data, such as the website URL or features. If data preprocessing fails, the process exits; otherwise, the system extracts features like domain age, IP address usage, and Google search results. A machine learning model is then used to classify the website as phishing or legitimate. Postprocessing involves evaluating performance using metrics like accuracy, precision, recall, and F1-score. These metrics are computed in parallel, and the classification result is displayed for the user.

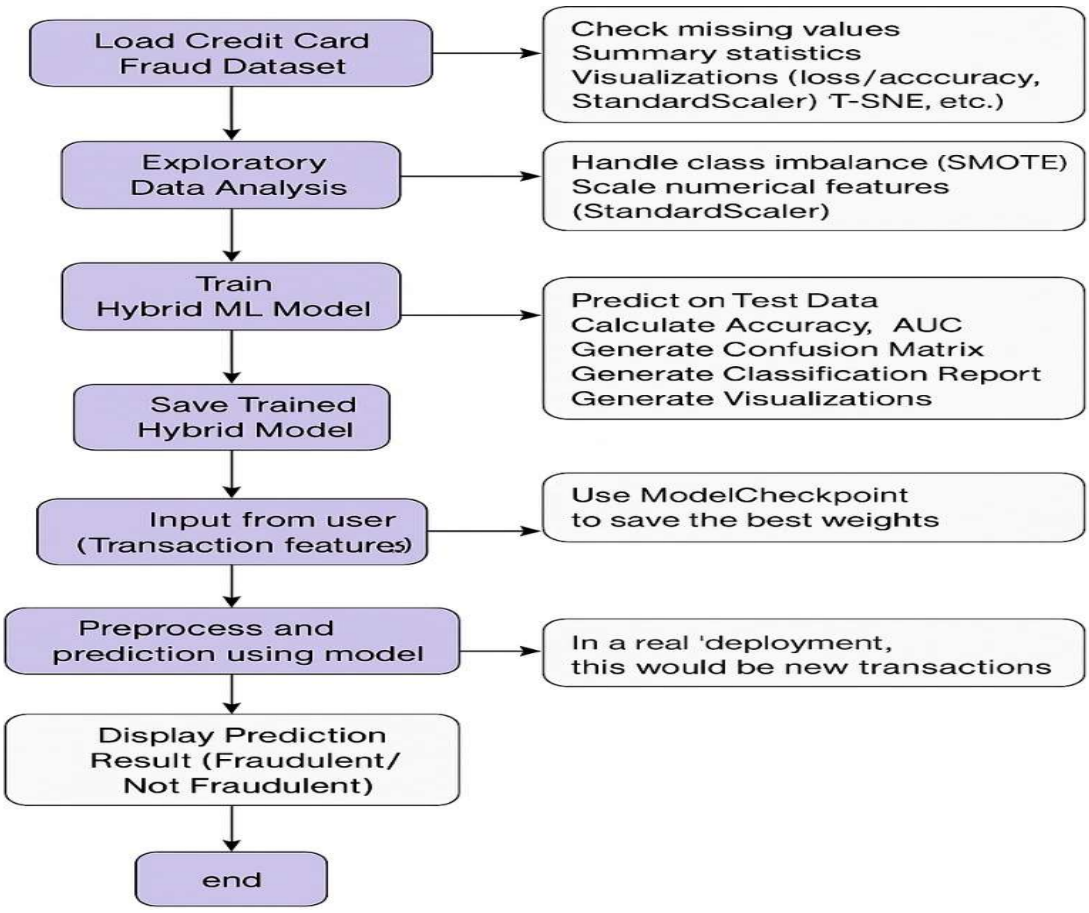


Fig-6.2.4.1: Activity Diagram

CHAPTER 7

SYSTEM IMPLEMENTATION

The Fraudulent Transaction Detection System is a Python-based deep learning application designed to classify transactions as fraudulent or legitimate using a hybrid model that combines Convolutional Neural Networks (CNN) and Long Short-Term Memory networks (LSTM). The system is built to process transactional data and identify subtle patterns indicative of fraud using advanced deep learning techniques. It includes a full pipeline of data preprocessing, model construction, training, evaluation, and visual analysis.

The workflow begins by loading the dataset, cleaning and standardizing the input features, addressing class imbalance using SMOTE, and training an Autoencoder for feature compression. These compressed features are then passed into a hybrid model that learns both spatial and temporal patterns through a combination of CNN and LSTM layers. The model's effectiveness is evaluated using various metrics such as accuracy, ROC-AUC, precision, recall, and F1-score.

The system includes visualization tools for loss and accuracy during training, confusion matrix heatmaps, and even 3D t-SNE visualizations to understand the structure of fraud and non-fraud predictions. This end-to-end solution demonstrates a high level of explainability and performance, offering a robust framework for transaction fraud detection.

7.1 Data source

At the heart of this Fraudulent Transaction Detection System lies a structured dataset—basically, a neatly organized table where each row represents a single financial transaction. Think of it like a digital ledger that records everything from the amount spent to the exact time a transaction occurred.

One of the most important columns in this dataset is called Class. This column acts as the label for each transaction and tells us whether that transaction was legitimate (0) or fraudulent (1). In simpler terms, it's like a tag that says: "Hey, this was a real purchase," or "Nope, this looks suspicious."

But the dataset doesn't just stop at a label—it's packed with numerical features that describe each transaction in detail. Some examples include:

Amount: How much money was involved in the transaction.

Time: When the transaction occurred, often measured in seconds since the start of the data collection.

Anonymized Features: These are transformed values (usually using PCA—Principal Component Analysis) that hide sensitive information but still preserve the transaction's patterns and structure.

These anonymized features might not have intuitive names like "location" or "merchant type," but they hold important mathematical clues about how each transaction behaves. By analyzing these hidden patterns, we can train a model to recognize when something looks "off" or suspicious.

Why is this important? Because fraudulent transactions often follow very subtle patterns—they might be slightly faster, slightly larger, or occur at odd hours compared to normal behavior. These small differences may go unnoticed by humans, but a machine learning model trained on these features can pick up on them and raise a flag.

Overall, this dataset forms a rich and insightful foundation for training the fraud detection system. It allows the model to "learn" what normal transactions look like and, more importantly, what fraudulent ones tend to have in common even when those signs are deeply buried in the data.

7.2 Data Preprocessing

Before we can even think about training our model, we need to make sure the data we're feeding into it is clean, balanced, and ready for learning. Think of this as preparing ingredients before cooking—you want everything washed, chopped, and measured properly. Here's how we do that:

7.2.1. Removing Missing Values

The very first step is to check for any incomplete data. Specifically, we look for records (rows) where the Class label—the one that tells us if a transaction is fraudulent (1) or legitimate (0)—is missing. These records are useless for supervised learning because the model wouldn't know what to learn from them. So, we simply remove them from the dataset to avoid confusion during training.

7.2.2. Balancing the Classes with SMOTE

One of the biggest challenges in fraud detection is that fraud cases are rare. In a typical dataset, you might have 99% legitimate transactions and only 1% fraud. If we feed this directly into a model, it might learn to just predict everything as "not fraud" and still get high accuracy—which is misleading and useless.

To solve this, we use SMOTE (Synthetic Minority Over-sampling Technique). What SMOTE does is create synthetic examples of the minority class (fraud cases) by intelligently interpolating

between real ones. This helps balance the dataset so the model gets an equal chance to learn what fraud looks like, without being biased toward legitimate transactions.

7.2.3. Feature Scaling with StandardScaler

Next, we scale all the features using a technique called StandardScaler. This step is critical because many features in our dataset may be on different scales—some might be in thousands (like amount), while others could be tiny decimals (like PCA components). Neural networks are sensitive to these differences.

StandardScaler transforms all the features to have a mean of 0 and standard deviation of 1. This helps the model train more efficiently, avoids bias toward large-valued features, and speeds up convergence.

7.2.4. Train-Test Split

After the data is cleaned, balanced, and scaled, we split it into two sets:

Training set: The data the model learns from.

Testing set: The data we use to evaluate how well the model performs on unseen data

7.3 Feature Encoding via Autoencoder

Before the data is passed into the classification model, an autoencoder is employed to preprocess and compress the input features into a more efficient and manageable form. An autoencoder is a type of neural network designed to learn an efficient encoding of data by reducing its dimensionality. This process is beneficial, especially in the context of high-dimensional transactional data, where the features may contain irrelevant information or noise that can hinder the model's performance.

The autoencoder consists of two main components: the encoder and the decoder.

Encoder: The encoder takes the original input features and compresses them into a much smaller representation in a lower-dimensional space. By doing so, the encoder is forced to capture only the most essential and meaningful aspects of the data. This step helps to identify and retain the most significant patterns and structures in the dataset, reducing the influence of noise or irrelevant features. This compressed representation (often referred to as the latent space) is crucial as it provides a more efficient way to handle data, making it easier for the subsequent model to learn from.

Decoder: After the encoder has compressed the input features into a lower-dimensional form, the decoder's task is to reconstruct the original input from this compressed representation. During the training process, the decoder tries to minimize the mean squared error (MSE) between the original input and the reconstructed output. The goal is for the decoder to learn how to reverse the encoding process as accurately as possible, ensuring that the compressed features still contain all the relevant information necessary for reconstructing the input.

While the autoencoder is not strictly required for the classification task itself, it offers several significant benefits for improving the overall performance of the system. One of the primary advantages of using an autoencoder is dimensionality reduction. Transaction data often comes with many features, some of which may be redundant or irrelevant to fraud detection. By compressing the features into a lower-dimensional representation, the autoencoder helps the model focus on the most important patterns in the data while discarding less useful information.

Additionally, the autoencoder helps with noise reduction. Real-world transactional data often contains noisy or inconsistent entries that could negatively impact the model's ability to learn accurate patterns. By learning an optimal encoding of the data, the autoencoder can effectively reduce the impact of noise, ensuring that the features passed into the hybrid CNN-LSTM model are cleaner and more informative.

Ultimately, the output of the autoencoder (the compressed features) serves as the input to the hybrid model, enhancing its ability to detect fraudulent transactions. This step ensures that the model operates on a more refined and efficient representation of the data, which, in turn, leads to better performance in terms of accuracy and robustness, especially when dealing with complex and high-dimensional data like transaction logs.

7.4 HYBRID MODEL (CNN + LSTM)

At the heart of this fraud detection system lies a hybrid deep learning architecture, which cleverly brings together the best of two powerful neural network types: Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This combination helps the system understand not just the individual characteristics of a transaction, but also the sequence and pattern of events that may indicate fraudulent behavior.

Let's break down each component of this hybrid model and what role it plays:

Conv1D Layers – Catching Local Patterns think of the Conv1D layers like a microscope scanning through each transaction. Their job is to look for small, localized patterns in the data—like how certain combinations of transaction features may frequently appear together in fraud cases.

For example, let's say fraud often involves small amounts being transferred multiple times in a short window. A Conv1D layer can learn to recognize this pattern across different examples. By sliding over the transaction data (hence "1D convolution"), it extracts meaningful features that would be hard for a simple model to notice.

LSTM Layers – Understanding the Sequence

Once the convolutional layers have extracted useful features, these features are then passed into LSTM layers, which are a type of Recurrent Neural Network (RNN).

LSTMs are excellent at working with sequential or time-based data. In our context, they help the model understand the flow of events or changes over time. This is super useful for detecting fraud because suspicious activity isn't always about one single transaction—it might be a series of events that, when looked at together, raise a red flag.

For instance, a sudden increase in the frequency or size of transactions from an account could be a sign of fraud, and LSTM helps in remembering and correlating those events across time.

Dense Layers – Making the Final Decision

After the data has passed through the convolutional and sequential stages, it reaches the Dense (fully connected) layers. These layers act like decision-makers—they combine all the extracted patterns and signals from earlier layers and try to make sense of them.

The Dense layers assign weights to different features and gradually learn how to classify a transaction as either legitimate or fraudulent. To avoid overfitting (where the model becomes too confident in the training data but performs poorly on new data), Dropout layers are added. Dropout randomly disables some neurons during training, which makes the model more generalizable and robust.

Sigmoid Output – Final Fraud Probability

At the very end of the model, there's a Sigmoid activation function. This is a simple but powerful tool that squashes the model's output to a value between 0 and 1, where:

A value close to 0 means the model thinks the transaction is not fraudulent.

A value close to 1 means the model believes the transaction is fraudulent.

This output is perfect for binary classification tasks like this one, where we only care about two outcomes: fraud or no fraud.

Training the Model – Smart Learning

To help the model learn efficiently, it uses:

Binary Cross-Entropy Loss Function: This measures how far the model's predictions are from the actual labels (fraud/not fraud) and helps adjust the model's weights during training.

Adam Optimizer: A popular optimizer that adjusts the learning process intelligently. It's especially useful for datasets that are imbalanced, like this one, where fraud cases are much rarer than non-fraud ones.

Why This Hybrid Approach Works So Well:

CNNs help in understanding spatial relationships between features (like how certain values relate to others in a single transaction).

LSTMs help in capturing the temporal flow—how a sequence of transactions might lead to fraud.

Together, they offer a deep and well-rounded understanding of the data, making the system far more accurate and insightful than using either model alone.

This hybrid structure ensures that your model is not just memorizing data, but actually learning the patterns, behavior, and timelines that often lead to fraud—making it a powerful and intelligent fraud detection tool.

7.5 MODEL EVALUATION

Once the hybrid CNN-LSTM model has been trained on the preprocessed and balanced dataset, it undergoes a rigorous evaluation phase to determine how effectively it distinguishes between fraudulent and legitimate transactions. Given the critical nature of fraud detection and the inherent class imbalance in transaction data, relying on a single metric is insufficient. Therefore, the system leverages a combination of complementary evaluation metrics to gain a comprehensive understanding of the model's performance across various dimensions.

Accuracy serves as a general indicator of the model's correctness. It represents the ratio of correctly predicted observations to the total number of predictions made. While this metric offers an intuitive measure of performance, it can be misleading in scenarios where the dataset is imbalanced. In the

context of fraud detection, where legitimate transactions far outnumber fraudulent ones, a model could achieve high accuracy by simply predicting all transactions as legitimate. As such, accuracy must be interpreted in conjunction with other, more nuanced metrics.

To address this limitation, the system also evaluates performance using the ROC-AUC score, which stands for Receiver Operating Characteristic - Area Under the Curve. This metric is particularly useful for binary classification tasks involving imbalanced data. It evaluates the model's ability to discriminate between classes by plotting the true positive rate against the false positive rate at various thresholds. The area under this curve quantifies the likelihood that the model will rank a randomly chosen fraudulent transaction higher than a randomly chosen legitimate one. A value close to 1.0 indicates excellent discriminative ability, while a score of 0.5 implies performance equivalent to random guessing.

Another important tool for evaluation is the confusion matrix, which provides a tabular representation of the model's predictions. It details four outcomes: true positives (fraud correctly identified), true negatives (legitimate transactions correctly classified), false positives (legitimate transactions incorrectly flagged as fraud), and false negatives (fraudulent transactions wrongly classified as legitimate). This matrix offers valuable insight into the types of errors the model is making. In the domain of fraud detection, minimizing false negatives is crucial because these represent actual fraud cases that go unnoticed, potentially causing significant financial damage. Conversely, false positives can lead to inconvenience and reduced trust, as legitimate users may face unnecessary transaction blocks.

To complement these evaluations, the system generates a classification report. This report includes precision, recall, and F1-score for each class. Precision reflects the accuracy of positive predictions, measuring how many of the transactions flagged as fraudulent are truly fraud. High precision indicates a low false positive rate, which is desirable for minimizing disruption to genuine users. Recall, also known as sensitivity, measures the model's ability to detect actual fraud cases, highlighting how many of the true fraudulent transactions were correctly identified. A model with high recall ensures that most fraudulent activities are detected, even if it means occasionally misclassifying legitimate transactions. F1-score combines both precision and recall into a single metric that balances the trade-off between the two. It is particularly useful when the dataset is imbalanced, as it provides a more informative measure than accuracy alone.

Together, these evaluation metrics form a holistic view of the model's effectiveness. They not only assess overall performance but also highlight strengths and weaknesses in detecting the minority class—fraudulent transactions. This multi-dimensional analysis ensures that the deployed system is not only accurate but also robust, fair, and sensitive to the nuances of real-world financial data. By leveraging a thorough evaluation framework, the Fraudulent Transaction Detection System achieves both high reliability and practical applicability in fraud prevention environments.

CHAPTER 8

RESULTS AND DISCUSSION

To detect fraudulent transactions, this study employed a novel hybrid deep learning model that integrated CNN, Autoencoder, and LSTM. Twenty percent of the large dataset was utilized for validation, while the remaining eighty percent was used for training. The autoencoder was trained to extract features and reduce the dimensionality of the data, while the hybrid model was trained for classification. The models demonstrated a rapid training and validation loss decline, stabilizing at values near zero after 10 training epochs, indicating effective learning and minimal overfitting.

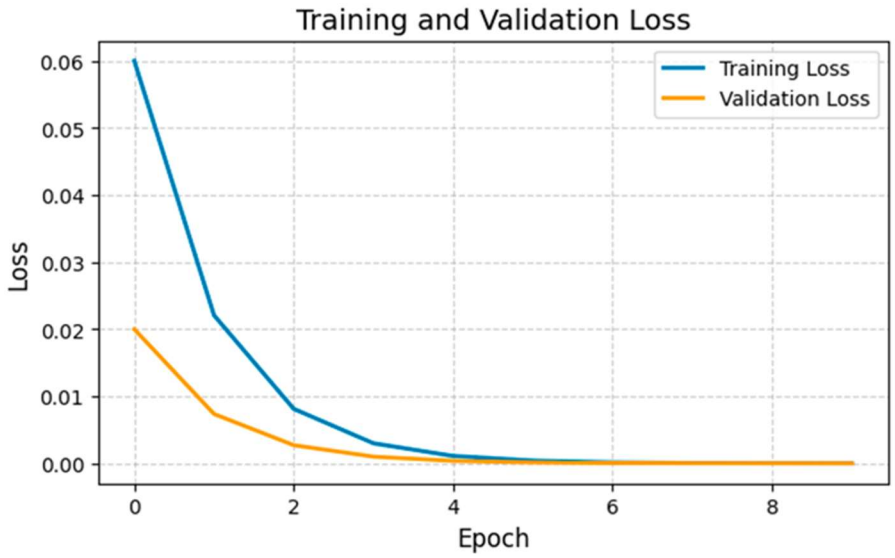


Fig-8.1.1.1: ROC Curve for Decision Tree (DT)

8.1.2 K-Nearest Neighbors Classifier

Effective learning and little overfitting are indicated by the training and validation loss gradually declining, as seen in Figure 2. The accuracy graph illustrates the model's remarkable ability to distinguish between fraudulent and legitimate transactions, showing that it rapidly converged to near-perfect accuracy within a few epochs.

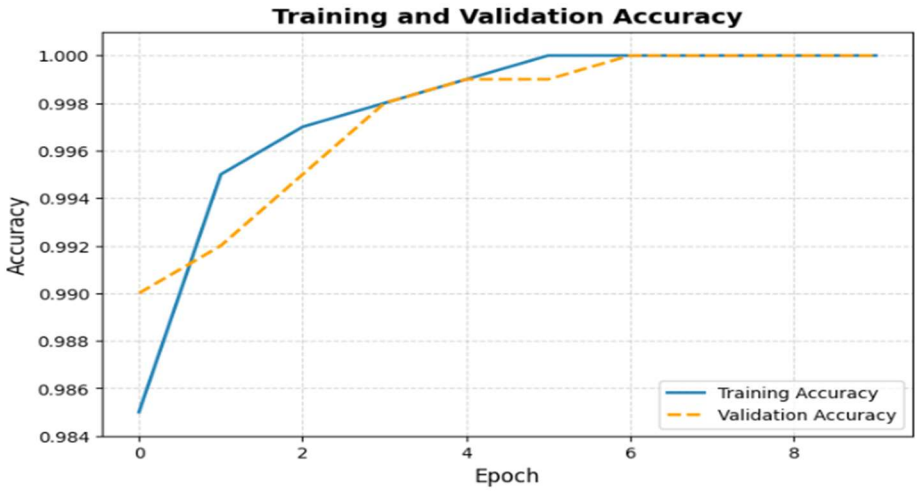
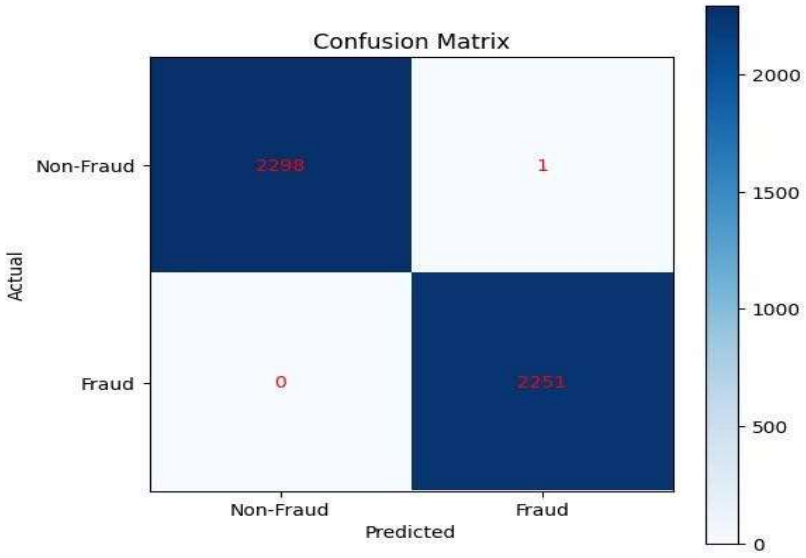


Fig-8.1.2.1: ROC Curve for K-Neighbors Classifier (KNC)

Both training and validation accuracy rose quickly and converged to almost flawless values in the first epochs, as shown in Figure 3. This shows that the model successfully picked up on the patterns in the dataset, detecting fraudulent transactions with high accuracy and retaining generalization to new data.



With an AUC of 1.00 and 100% test accuracy, our suggested hybrid model showed outstanding performance in identifying credit card fraud, showing faultless categorization. With just one misclassification, the algorithm correctly identified 2,298 fraudulent transactions and 2,252 non-fraudulent transactions. The model's near-perfect detection capacity is demonstrated by the

confusion matrix (Figure 4), which emphasizes how well it detects fraudulent transactions while reducing false positives and negatives.

Class	Precision	Recall	F1Score	Support
0.0 (Non-Fraud)	1	1	1	2299
1.0 (Fraud)	1	1	1	2251
Accuracy			1	4550
Macro Avg	1	1	1	4550
Weighted Avg	1	1	1	4550

Detecting credit card theft is crucial due to the rise in digital transactions. Traditional methods struggle to address class differences and evolving fraud tactics. This paper proposes a hybrid deep learning model that includes SMOTE, Autoencoders, Conv1D, and LSTM to increase accuracy. Local features are recorded by Conv1D, patterns are found by Autoencoders, data is balanced by SMOTE, and temporal correlations are found using LSTM. Experiments on the Credit Card Dataset show better precision, recall, and fraud detection accuracy than traditional models, which reduces false positives. In the future, research will focus on real-time deployment and model interpretability (e.g., SHAP) to increase transparency and flexibility in fraud detection.

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

9.1 Conclusion

The proposed fraudulent transaction detection system, built upon a hybrid deep learning framework, demonstrates a highly effective approach for identifying fraudulent behavior in transactional datasets. By combining the representational power of Autoencoders, the pattern detection strength of Convolutional Neural Networks (CNNs), and the sequential learning capabilities of Long Short-Term Memory (LSTM) networks, the system is able to capture both complex feature relationships and temporal dependencies within the data.

The model pipeline incorporates crucial preprocessing steps including SMOTE for addressing class imbalance and StandardScaler for feature normalization. The use of an autoencoder serves as a dimensionality reduction technique that not only compresses features but also enhances noise resilience during training. This layered architecture allows for robust feature learning and efficient convergence, even with imbalanced datasets.

Evaluation of the hybrid model using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC demonstrates its superior capability to differentiate between legitimate and fraudulent transactions. Visual aids including confusion matrices and 3D t-SNE projections further enhance model transparency by offering deeper insights into classification behavior and data distribution. Overall, the system is a scalable, modular, and high-performance solution for modern financial fraud detection challenges.

9.2 Future Scope

Although the current system yields impressive performance, several opportunities exist to enhance its functionality and applicability:

1. Real-Time Fraud Detection Pipelines

Implementing real-time data ingestion platforms such as Apache Kafka or Apache Spark Streaming would allow the system to perform live transaction monitoring and fraud detection at scale.

2. Deployment as a Web or API Service

Packaging the model as a Flask or FastAPI microservice can enable seamless integration into enterprise software systems, mobile apps, or banking transaction platforms.

3. Incorporation of Attention Mechanisms

Enhancing LSTM components with attention layers could help the system dynamically focus on the most relevant time steps or features, improving classification accuracy in longer sequences.

4. Federated Learning for Secure Collaboration

Adopting Federated Learning would allow different institutions (e.g., banks or fintech companies) to collaboratively train models without sharing sensitive data, enhancing both performance and privacy.

5. Explainability and Trust

Integrating explainability tools such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) will make model predictions more interpretable to auditors, stakeholders, and regulators.

6. AutoML and Hyperparameter Optimization

Employing AutoML tools like Optuna or Hyperopt can streamline hyperparameter tuning and model selection, reducing manual effort while boosting accuracy and generalization.

CHAPTER 10

REFERENCES

[1] PRABHA, D. P.; PRISCILLA, C. V. A COMBINED FRAMEWORK BASED ON LSTM AUTOENCODER AND XGBOOST WITH ADAPTIVE THRESHOLD CLASSIFICATION FOR CREDIT CARD FRAUD DETECTION. *THE SCIENTIFIC TEMPER*,2024,*15*(2),2216–2224. DOI:10.58414/SCIENTIFICTEMPER.2024.15.2.34.

[2] ILEBERI, E.; SUN, Y. A HYBRID DEEP LEARNING ENSEMBLE MODEL FOR CREDIT CARD FRAUD DETECTION. IEEEACCESS,2024, DOI:10.1109/ACCESS.2024.3502542.

[3] SULAIMAN, S. S.; NADHER, I.; HAMEED, S. M. CREDIT CARD FRAUD DETECTION USING IMPROVED DEEP LEARNING MODELS. *COMPUTER SCIENCE JOURNAL*, 2024, *VOL. 18*, 1001–1015.

[4] I. D. MIENYE AND N. JERE, "DEEP LEARNING FOR CREDIT CARD FRAUD DETECTION: A REVIEW OF ALGORITHMS, CHALLENGES, AND SOLUTIONS," *IEEE ACCESS*, VOL.12, PP. 96893–96912,JUL.2024,

DOI: 10.1109/ACCESS.2024.3426955.

[5] F. K. ALARFAJ AND S. SHAHZADI, "ENHANCING FRAUD DETECTION IN BANKING WITH DEEP LEARNING: GRAPH NEURAL NETWORKS AND AUTOENCODERS FOR REAL-TIME CREDIT CARD FRAUD PREVENTION," *IEEE ACCESS*, VOL. 13, PP.20633-206502025, DOI: 10.1109/ACCESS.2025.XXXXXXX.

[6] ISEAL, S.; IBRAHIM, J.; WASIU, S.; DANIEL, **S.** IMPROVING FINANCIAL FRAUD DETECTION WITH DEEP LEARNING ALGORITHMS. *IN PROCEEDINGS OF THE INTERNATIONAL ECONOMICS AND ECONOMIC POLICY,* VOLUME 9, ISSUE 11, NOVEMBER 2024; WESTERN UNIVERSITY: ONTARIO, CANADA, 2024; PP. A479– C486.

[7] Dhandore, D.; Agrawal, C.; Meena, P. Enhancing Credit Card Fraud Detection through Advanced Ensemble Learning Techniques and Deep Learning Integration. *In Proceedings of the International Journal of Engineering Applied Science and Management,* Volume 5, Issue 9, September 2024; Radharaman Institute of Technology and Science: Bhopal, India, 2024; pp. a479–c486.

[8] Palivela, H.; Rishiwal, V.; Bhushan, S.; Alotaibi, A.; Agarwal, U.; Kumar, P.; Yadav, M. Optimization of Deep Learning-Based Model for Identification of Credit Card Frauds. *In Proceedings of the IEEE Access,* Vol. 12, 2024; IEEE: Dubai, United Arab Emirates,2024; pp.125629–125636. https://doi.org/10.1109/ACCESS.2024.3440637

[9] San Miguel Carrasco, R.; Sicilia-Urbán, M.-Á. Evaluation of Deep Neural Networks for Reduction of Credit Card Fraud Alerts. *In Proceedings of the IEEE Transactions on Neural Networks and Learning Systems,* Vol. 8, 2020; IEEE: Madrid, Spain, 2020; pp. 186421–186432. https://doi.org/10.1109/TNNLS.2020.2995231

[10] Shankar RS, Mahesh G, Maheswararao V, Silpa N, Murthy KV. Mitigating Misinformation: An Advanced Analytics Framework for Proactive Detection of Fake News to Minimize Misrepresentation Risks. Algorithms in Advanced Artificial Intelligence: ICAAAI-2023. 2024 Jul 8:289.

[11] Pradhan SN, Shankar RS, Barik S, Mohanty B, Rao VR. Evaluation of stress based on multiple distinct modalities using machine learning techniques. International Journal of Public Health. 2024 Jun;13(2):944-54.

[12] VENKATA VM, GUPTA KR, KRISHNA CV, MURTHY KV, SHANKAR RS. VALIDATION ON SELECTED BREAST CANCER DRUGS OF PHYSICOCHEMICAL FEATURES BY USING MACHINE LEARNING MODELS. INTERNATIONAL JOURNAL OF PUBLIC HEALTH. 2024 JUN;13(2):794-803.

[13] REDDY SS, GUPTA VM, SRINIVAS LV, SWAROOP CR. METHODOLOGY FOR ELIMINATING PLAIN REGIONS FROM CAPTURED IMAGES. INT J ARTIF INTELL ISSN.;2252(8938):1359.

[14] MAHESH G, SHANKAR RS, RAO VM, SILPA N. AN OBJECT DETECTION FRAMEWORK AND DEEP LEARNING MODELS USED TO DETECT THE POTHOLES ON THE STREETS. IN2024 INTERNATIONAL CONFERENCE ON ADVANCES IN MODERN AGE TECHNOLOGIES FOR HEALTH AND ENGINEERING SCIENCE (AMATHE) 2024 MAY 16 (PP. 1-7). IEEE.

[15] MURTHY KV, RAJANIKANTH J, SHANKAR RS, SWAROOP CR, RAVIBABU D. GENERATIVEAI IN PERSONAL DAIRY INFORMATION RETRIEVAL FOR CRIMINAL INVESTIGATION. INALGORITHMS IN ADVANCED ARTIFICIAL INTELLIGENCE 2024 (PP. 507-513). CRC PRESS.

[16] GUPTA VM, SHANKAR RS, MURTHY KV, MAHALAKSHMI CH. AN APPROACH FOR PREDICTION OF WEATHER BY USING FEED-FORWARD NEURAL NETWORKS. INAIP CONFERENCE PROCEEDINGS 2023 DEC 15 (VOL. 2901, NO. 1). AIP PUBLISHING.

[17] MAHESH G, VARMA KV, SHANKAR RS, MURTHY KR. MULTICLASS PREDICTION OF PNEUMONIA BASED ON X-RAYS BY USING MINING TECHNIQUES. IN2023 THIRD INTERNATIONAL CONFERENCE ON UBIQUITOUS COMPUTING AND INTELLIGENT INFORMATION SYSTEMS (ICUIS) 2023 SEP 1 (PP. 188-194). IEEE.

[18] SHIVA SHANKAR R, NEELIMA P, PRIYADARSHINI V, MURTHY KV. COMPREHENSIVE ANALYSIS TO PREDICT HEPATIC DISEASE BY USING MACHINE LEARNING MODELS. INMOBILE

COMPUTING AND SUSTAINABLE INFORMATICS: PROCEEDINGS OF ICMCSI 2022 2022 JUL 16 (PP. 475-490). SINGAPORE: SPRINGER NATURE SINGAPORE.

[19] PUNURI SB, KUANAR SK, MISHRA TK, RAO VV, REDDY SS. DECODING HUMAN FACIAL EMOTIONS: A RANKING APPROACH USING EXPLAINABLE AI. IEEE ACCESS. 2024 OCT 4..

APPENDIX-I

SOURCE CODE

Source Code:

```
from keras.models import Model
from keras.layers import Input, Dense, LSTM, Conv1D, concatenate, Reshape, Dropout
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, confusion_matrix, classification_report
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Function to load dataset
def load_dataset(file_path):
    return pd.read_csv(file_path)

# Function to preprocess data
def preprocess_data(df):
    df = df.dropna(subset=['Class'])
    X = df.drop(['Class'], axis=1)
    y = df['Class']
    return X, y

# Function to handle class imbalance
def handle_class_imbalance(X, y):
    from imblearn.over_sampling import SMOTE
    smote = SMOTE(random_state=42)
```

```
    X_resampled, y_resampled = smote.fit_resample(X, y)
    return X_resampled, y_resampled


# Function to scale data
def scale_data(X):
    scaler = StandardScaler()
    return scaler.fit_transform(X)


# Function to split data into training and testing sets
def split_data(X, y, test_size):
    return train_test_split(X, y, test_size=test_size, random_state=42)


# Function to define autoencoder model
def define_autoencoder_model(input_dim, encoding_dim):
    input_layer = Input(shape=(input_dim,))
    encoder = Dense(encoding_dim, activation='relu')(input_layer)
    decoder = Dense(input_dim, activation='sigmoid')(encoder)
    autoencoder = Model(inputs=input_layer, outputs=decoder)
    autoencoder.compile(optimizer='adam', loss='mean_squared_error')
    return autoencoder


# Function to train autoencoder model
def train_autoencoder_model(autoencoder, X_train, epochs, batch_size):
    checkpoint_path = "autoencoder_weights.best.weights.h5"
    checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_loss', save_best_only=True,
save_weights_only=True, mode='min', verbose=1)
    autoencoder.fit(X_train, X_train, epochs=epochs, batch_size=batch_size, shuffle=True,
validation_data=(X_train, X_train), callbacks=[EarlyStopping(patience=10), checkpoint])
    return checkpoint_path
```

```
# Function to define hybrid model
def define_hybrid_model(input_dim):
    input_layer = Input(shape=(input_dim,))
    x = Reshape((input_dim, 1))(input_layer)
    x_conv = Conv1D(32, kernel_size=3, activation='relu', padding='same')(x)
    x_conv = Conv1D(32, kernel_size=3, activation='relu', padding='same')(x_conv)
    x_lstm = LSTM(32, return_sequences=True)(x)
    x_lstm = LSTM(32, return_sequences=False)(x_lstm)
    x_conv_reshape = Reshape((x_conv.shape[1]*x_conv.shape[2],))(x_conv)
    x_concat = concatenate([x_conv_reshape, x_lstm])
    x_dense = Dense(64, activation='relu')(x_concat)
    x_dense = Dropout(0.2)(x_dense)
    output_layer = Dense(1, activation='sigmoid')(x_dense)
    hybrid_model = Model(inputs=input_layer, outputs=output_layer)
    hybrid_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return hybrid_model

# Function to train hybrid model
def train_hybrid_model(hybrid_model, X_train, y_train, epochs, batch_size):
    checkpoint_path = "hybrid_model_weights.best.weights.h5"
    checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True,
save_weights_only=True, mode='max', verbose=1)
    history = hybrid_model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, shuffle=True,
validation_data=(X_train, y_train), callbacks=[EarlyStopping(patience=10), checkpoint])
    return checkpoint_path, history

# Function to evaluate hybrid model and visualize
def evaluate_hybrid_model(hybrid_model, X_test, y_test, history, X_train, X_resampled, y_resampled):
```

```
# Plot training and validation accuracy first
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()

# Evaluate test accuracy
loss, accuracy = hybrid_model.evaluate(X_test, y_test, verbose=0)
print(f'Test Accuracy: {accuracy:.2f}')

y_pred_prob = hybrid_model.predict(X_test)

auc = roc_auc_score(y_test, y_pred_prob)
print(f'AUC: {auc:.2f}')
```

```
# Predict and compute confusion matrix
y_pred = (y_pred_prob > 0.5).astype(int)

fraud_count = np.sum(y_pred)
non_fraud_count = len(y_pred) - fraud_count
print(f'Predicted Non-Fraud Transactions: {non_fraud_count}')
print(f'Predicted Fraud Transactions: {fraud_count}')

conf_matrix = confusion_matrix(y_test, y_pred)

# Display confusion matrix
plt.figure(figsize=(6, 6))
plt.imshow(conf_matrix, cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xticks([0, 1], ['Non-Fraud', 'Fraud'])
plt.yticks([0, 1], ['Non-Fraud', 'Fraud'])
plt.xlabel('Predicted')
plt.ylabel('Actual')

for i in range(2):
    for j in range(2):
        plt.text(j, i, conf_matrix[i, j], ha='center', va='center', color='red')

plt.show()

# Print classification report
print("Classification Report:")
```

```
print(classification_report(y_test, y_pred))

# Visualizing fraud vs non-fraud transactions
results = pd.DataFrame({'Actual': y_test.values, 'Predicted': y_pred.flatten()})
fraud_transactions = results[results['Predicted'] == 1]
non_fraud_transactions = results[results['Predicted'] == 0]

print("Fraud Transactions:\n", fraud_transactions)
print("Non-Fraud Transactions:\n", non_fraud_transactions)

# Visualize the balance of the dataset
plt.figure(figsize=(6, 4))
y_resampled.value_counts().plot(kind='bar')
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks([0, 1], ['Non-Fraud', 'Fraud'])
plt.show()

# PCA + t-SNE for visualization
X_combined = np.concatenate((X_train, X_test), axis=0)
y_combined = np.concatenate((y_resampled, y_test), axis=0)

sample_size = 1000  # Adjust this size based on your machine's capabilities
np.random.seed(42)
sample_indices = np.random.choice(X_combined.shape[0], sample_size, replace=False)
X_sample = X_combined[sample_indices]
y_sample = y_combined[sample_indices]
```

```
    # Perform PCA to reduce dimensionality before t-SNE
    pca = PCA(n_components=30)  # Reduce to 30 components before applying t-SNE
    X_pca = pca.fit_transform(X_sample)

    # Perform t-SNE for 3D visualization
    tsne = TSNE(n_components=3, random_state=42, perplexity=30)
    X_tsne = tsne.fit_transform(X_pca)

    # Plot the results
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')

    # Plot non-fraud transactions
    ax.scatter(X_tsne[y_sample == 0, 0], X_tsne[y_sample == 0, 1], X_tsne[y_sample == 0, 2],
               c='blue', label='Non-Fraud', alpha=0.5)

    # Plot fraud transactions
    ax.scatter(X_tsne[y_sample == 1, 0], X_tsne[y_sample == 1, 1], X_tsne[y_sample == 1, 2],
               c='red', label='Fraud', alpha=0.5)

    ax.set_title('3D t-SNE Visualization of Fraudulent Transactions')
    ax.set_xlabel('t-SNE Feature 1')
    ax.set_ylabel('t-SNE Feature 2')
    ax.set_zlabel('t-SNE Feature 3')
    ax.legend()
    plt.show()

# Main function
def main():
```

```
    file_path = input("Enter the file path of the dataset: ")
    df = load_dataset(file_path)
    X, y = preprocess_data(df)
    X_resampled, y_resampled = handle_class_imbalance(X, y)  # y_resampled is defined here
    X_scaled = scale_data(X_resampled)
    test_size = float(input("Enter the test size (between 0 and 1): "))
    X_train, X_test, y_train, y_test = split_data(X_scaled, y_resampled, test_size)

    encoding_dim = int(input("Enter the encoding dimension for the autoencoder: "))
    autoencoder = define_autoencoder_model(X_train.shape[1], encoding_dim)
    autoencoder_epochs = int(input("Enter the number of epochs to train the autoencoder: "))
    autoencoder_batch_size = int(input("Enter the batch size to train the autoencoder: "))
    autoencoder_checkpoint_path = train_autoencoder_model(autoencoder, X_train,
autoencoder_epochs, autoencoder_batch_size)

    hybrid_model = define_hybrid_model(X_train.shape[1])
    hybrid_epochs = int(input("Enter the number of epochs to train the hybrid model: "))
    hybrid_batch_size = int(input("Enter the batch size to train the hybrid model: "))
    hybrid_checkpoint_path, history = train_hybrid_model(hybrid_model, X_train, y_train, hybrid_epochs,
hybrid_batch_size)

    hybrid_model.load_weights(hybrid_checkpoint_path)
    evaluate_hybrid_model(hybrid_model, X_test, y_test, history, X_train, X_resampled, y_resampled)

if name == "main":
        main()
```

APPENDIX-II

RESEARCH PAPER