A project report on

TRANSPARENT CREDIT CARD FRAUD DETECTION:INTEGRATING DEEP LEARNING WITH HUMAN-INTERPRETABLE KOLMOGOROV-ARNOLD NETWORKS

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Robotics

By

SUSHANT GARGI (21BRS1072)

SAKSHAM SHARMA (21BAI1223)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April,2025

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled “TRANSPARENT CREDIT CARD FRAUD DETECTION:INTEGRATING DEEP LEARNING WITH HUMAN-INTERPRETABLE KOLMOGOROV-ARNOLD NETWORKS” submitted by SUSHANT GARGI (21BRS1072), for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Dr. Suganeshwari G.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: XX-April-2025 **Signature of the Candidate**

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **“TRANSPARENT CREDIT CARD FRAUD DETECTION:INTEGRATING DEEP LEARNING WITH HUMAN-INTERPRETABLE KOLMOGOROV-ARNOLD NETWORKS”** is prepared and submitted by **SUSHANT GARGI (21BRS1072)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering with specialization in Artificial Intelligence and Robotics** is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:
Name: Dr. Suganeshwari G
Date: XX-Apr-2025

Signature of the Examiner	Signature of the Examiner
Name:	Name:
Date:	Date:

Approved by the Head of Department,
Computer Science and Engineering with specialization in Artificial Intelligence and Robotics
Name: Dr. Harini S
Date: XX-Apr-2025

ABSTRACT

Credit card fraud is a serious problem that requires sophisticated detection. Although deep learning can perform exceptionally well in fraud detection, the black-box nature of models is an issue for trust and effective human oversight. Conversely, although interpretable, traditional machine learning models may be unable to detect subtle fraud patterns in intricate transactional data. To overcome this, we introduce a new fraud detection framework based on the Linear Time Attention CNN (LTACNN). This black-box model is designed to capture anomalies and behavioural changes typical of fraudulent transactions over the short and long term. The LTACNN uses convolutional-based operations to identify local, immediate patterns within streams of transactions, effectively capturing sudden bursts of fraudulent transactions. Meanwhile, long short-term memory cells operate on the overall transactional history, detecting subtle, evolving fraud patterns. An attention mechanism then selectively integrates these short-term and long-term perspectives to generate a strong and informed fraud judgment. Our framework also incorporates a Kolmogorov-Arnold Network (KAN) to make interpretability open. The KAN makes the LTACNN's decision-making transparent, generating precise mathematical functions and feature importance scores. This explainable output provides human analysts with trust, enables informed intervention, and ultimately enhances the efficacy of fraud management.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Suganeshwari G, Assistant Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Deep Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Harini S, Head of the Department, B.Tech. Computer Science and Engineering with Specialization in Artificial Intelligence and Robotics and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: XX-Apr-2025 **Sushant Gargi**

CONTENTS

LIST OF FIGURES PAGE NO.

LIST OF ACRONYMS

ADASYN	– Adaptive Synthetic Sampling
LSTM	– Long Short-Term Memory
CNN	– Convolutional Neural Network
LIME	– Local Interpretable Model-Agnostic Explanations
SHAP	– SHapley Additive exPlanations
KAN	– Kolmogorov-Arnold Network
LTACNN	– Linear Time Attention CNN
SMOTE	– Synthetic Minority Over-sampling Technique
XAI	– Explainable Artificial Intelligence

Chapter 1

Introduction

1.1 OVERVIEW

Credit card fraud remains one of the most pressing and pervasive threats in the financial industry, causing billions of dollars in losses annually. As digital transactions continue to grow exponentially, financial institutions must deploy sophisticated fraud detection methods to keep pace with increasingly advanced fraudulent schemes. Fraudsters continuously evolve their tactics, leveraging advanced techniques such as synthetic identity fraud, transaction obfuscation, and adversarial machine learning to bypass conventional security mechanisms. The rise of mobile banking, e-commerce, and contactless payments has further amplified the need for robust fraud detection systems capable of identifying fraudulent behavior in real-time.

Traditional fraud detection techniques, including rule-based systems and basic machine learning models, have often struggled to effectively combat modern fraud due to their inability to capture complex transactional patterns. Rule-based methods rely on predefined thresholds and heuristics, which fraudsters can easily manipulate by slightly altering their behavior. Meanwhile, traditional machine learning models, such as decision trees and logistic regression, may lack the ability to generalize well to unseen fraudulent patterns, leading to increased false positives or false negatives.

Deep learning has emerged as a powerful tool for detecting fraudulent transactions, as it can uncover hidden patterns in high-dimensional data. However, a significant challenge with deep learning models is their "black-box" nature—while they provide high accuracy, they lack interpretability. This lack of transparency makes it difficult for regulators and financial analysts to trust and adopt these models in real-world, high-stakes financial settings. Conventional machine learning models, on the other hand, offer better interpretability but often fall short in capturing the evolving nature of fraud.

This trade-off between high accuracy and interpretability is the primary motivation for our research. We propose a novel fraud detection system that combines a highly accurate deep-learning model—the Linear Time Attention CNN (LTACNN)—with the Kolmogorov-Arnold Network (KAN), which enhances interpretability. Our system not only detects fraudulent transactions with high precision but also provides transparent explanations that help financial institutions understand why a transaction was flagged as fraudulent.

The LTACNN model integrates convolutional neural networks (CNNs) and long short-term memory (LSTM) networks to capture both short-term and long-term patterns in transactional data. It also employs Linear Attention, an efficient mechanism inspired by transformers, to highlight the most important features in a transaction sequence. Additionally, to further improve fraud detection, we use the ADASYN technique for handling class imbalance and RobustScaler for effective feature scaling.

To address the interpretability challenge, we incorporate the Kolmogorov-Arnold Network (KAN), which serves as an explanation layer. The KAN approximates the decision-making process of the LTACNN using smooth mathematical functions, making it possible to understand why a transaction is classified as fraudulent. It also provides feature importance scores, which help human analysts make informed decisions.

Our framework is designed to meet regulatory requirements, including the General Data Protection Regulation (GDPR), which mandates transparency in automated decision-making. By balancing accuracy with interpretability, our approach enhances fraud detection while ensuring compliance with financial regulations.

1.2 PROBLEM STATEMENT

Credit card fraud poses a significant challenge to financial institutions, consumers, and regulatory bodies. The dynamic nature of fraud tactics, the massive volume of daily transactions, and the increasing reliance on automated decision-making necessitate the development of advanced fraud detection models. However, current fraud detection approaches face several limitations:

1.2.1 Black-Box Nature of Deep Learning Models

Deep learning models, particularly those based on CNNs, LSTMs, and transformers, have demonstrated remarkable success in fraud detection. These models excel at identifying intricate patterns that are often missed by traditional rule-based systems. However, they operate as black boxes, providing no clear explanation for their decisions. This lack of transparency makes it difficult for financial institutions to justify their fraud detection decisions to customers and regulators. Without interpretability, it is challenging to gain trust, perform compliance audits, and improve fraud prevention strategies.

1.2.2 Limited Effectiveness of Conventional Machine Learning Models

While traditional machine learning models such as decision trees, support vector machines (SVMs), and logistic regression provide better interpretability, they fail to capture the complexity of modern fraud schemes. Fraudsters continuously evolve their tactics, using strategies such as transaction obfuscation, gradual changes in spending behavior, and multi-account fraud to evade detection. Conventional models struggle to detect these sophisticated fraud patterns, leading to high false positives and false negatives.

1.2.3 Handling Class Imbalance in Fraud Detection

One of the critical challenges in fraud detection is class imbalance. Fraudulent transactions make up a tiny fraction of all transactions, often less than 1%. Standard machine learning algorithms tend to be biased toward the majority class (legitimate transactions), resulting in poor fraud detection performance. Oversampling techniques such as the Adaptive Synthetic Sampling (ADASYN) algorithm can help address this issue by generating synthetic fraud samples to balance the dataset.

1.2.4 Regulatory Compliance and the Need for Explainability

Financial regulations, including the GDPR and various fair lending laws, require that automated decision-making systems provide explanations for their predictions. Institutions must justify why a particular transaction was classified as fraudulent, particularly when denying a transaction or blocking a user's account. Failure to comply with these regulations can lead to legal consequences and a loss of customer trust.

1.2.5 Computational Efficiency and Scalability

With billions of transactions occurring daily, fraud detection systems must be computationally efficient and scalable. Traditional attention mechanisms in deep learning, such as those used in transformers, have quadratic time complexity, making them unsuitable for real-time fraud detection. A more efficient approach, such as Linear Attention, is needed to process long transaction sequences quickly without compromising accuracy.

To address these challenges, our proposed fraud detection framework integrates LTACNN and KAN, combining high accuracy with transparency and efficiency.

1.3 SIGNIFICANCE AND RELEVANCE OF WORK

The proposed fraud detection system is highly relevant in today's financial landscape due to the increasing reliance on digital transactions. The significance of our work can be understood in the following key areas:

1.3.1 Bridging the Gap Between Accuracy and Interpretability

Our framework is one of the first to combine a deep-learning model (LTACNN) with an interpretable mathematical model (KAN). This combination ensures that financial institutions can deploy high-accuracy fraud detection models while maintaining transparency in their decision-making processes.

1.3.2 Enhancing Real-Time Fraud Detection

With the integration of Linear Attention, our model processes transaction sequences efficiently, reducing computational overhead and making real-time fraud detection feasible. The ability to detect fraud instantly is crucial in preventing unauthorized transactions before they occur.

1.3.3 Compliance with Financial Regulations

Our system aligns with regulatory requirements such as GDPR by providing transparent explanations for flagged transactions. The feature importance scores generated by KAN help financial analysts and customers understand why a transaction was deemed fraudulent.

1.3.4 Handling Complex and Evolving Fraud Tactics

By leveraging CNNs, LSTMs, and Linear Attention, our model can detect both short-term anomalies and long-term fraud patterns. This capability is essential for identifying sophisticated fraud schemes that unfold over extended periods.

1.3.5 Addressing Class Imbalance in Fraud Data

The use of ADASYN ensures that our model is not biased toward legitimate transactions. By generating synthetic fraudulent samples, we improve the model's ability to detect rare fraud cases without inflating false positive rates.

1.4 OBJECTIVES

Our study aims to achieve the following key objectives, ensuring that the fraud detection model is highly accurate, interpretable, scalable, and efficient. The integration of

LTACNN and KAN will enable financial institutions to identify fraudulent transactions effectively while ensuring regulatory compliance and trust.

1.4.1 Develop a Fraud Detection Model that Balances Accuracy and Interpretability

The primary objective is to develop a fraud detection model that combines deep learning accuracy with interpretability.

- Implement LTACNN, leveraging CNNs, LSTMs, and Linear Attention to detect fraudulent transaction patterns effectively.
- Integrate KAN as an interpretability layer that approximates the LTACNN's decision function, making it transparent and understandable to financial analysts and regulators.
- Ensure that the model provides not only accurate predictions but also clear justifications for why a transaction is considered fraudulent.

1.4.2 Ensure Computational Efficiency and Scalability

Fraud detection systems must be capable of processing large-scale financial transactions in real-time.

- Implement Linear Attention to improve computational efficiency, reducing memory consumption while maintaining predictive accuracy.
- Optimize the LTACNN-KAN framework to ensure that it can handle real-world deployment in banking and financial services.
- Improve processing speed to ensure that fraudulent transactions are detected before they are completed, allowing for real-time fraud prevention.

1.4.3 Improve Handling of Class Imbalance

Fraud detection models must overcome class imbalance, as fraudulent transactions represent only a small fraction of total transactions.

- Use ADASYN (Adaptive Synthetic Sampling) to generate synthetic fraud samples, improving the model's ability to learn fraudulent transaction patterns.

- Reduce bias toward the majority class (legitimate transactions), ensuring that fraudulent transactions are correctly identified without increasing false positives.
- Develop a robust training strategy that allows the model to adapt to evolving fraud patterns over time.

1.4.4 Enhance Explainability for Regulatory Compliance

Interpretability is a critical requirement for fraud detection systems due to GDPR and financial regulations.

- Provide feature importance scores using KAN, allowing financial institutions to understand which factors contributed to a fraud decision.
- Ensure transparency in automated fraud detection, making it easier for analysts to audit model decisions and provide justifications to customers.
- Enable human-in-the-loop decision-making, where analysts can review flagged transactions and intervene when necessary.

1.4.5 Validate the Model on Real-World Financial Data

To ensure the practical effectiveness of the fraud detection framework, rigorous validation is necessary.

- Conduct empirical testing on large-scale transaction datasets to compare the model's performance against industry-standard fraud detection systems.
- Evaluate key performance metrics, including precision, recall, F1-score, and ROC-AUC, to ensure that the model performs well across different scenarios.
- Perform comparative analysis with existing fraud detection models, demonstrating the superiority of LTACNN-KAN in both accuracy and interpretability.

By achieving these objectives, this research provides a scalable, transparent, and highly effective fraud detection system that meets modern financial security challenges.

1.5 CHALLENGES

Despite its advantages, the proposed fraud detection framework faces several challenges that must be addressed to ensure its real-world applicability. The integration of deep learning and interpretable AI presents several complexities, from computational costs to adversarial fraud tactics.

1.5.1 Computational Complexity of Training

Training a hybrid deep-learning and interpretable model requires significant computational resources.

- LTACNN involves multiple convolutional and LSTM layers, requiring high processing power and memory consumption during training.
- The KAN layer introduces additional complexity, as it must approximate LTACNN's decision function while maintaining interpretability.
- Implementing efficient parallel processing techniques and utilizing hardware accelerators (e.g., GPUs and TPUs) can mitigate computational constraints.

1.5.2 Interpretability vs. Accuracy Trade-Off

One of the primary challenges in interpretable AI is finding the right balance between model accuracy and interpretability.

- While KAN enhances explainability, it may introduce a slight reduction in predictive performance compared to purely deep-learning models.
- Ensuring that LTACNN's decisions remain highly interpretable without sacrificing accuracy requires careful optimization.
- Fine-tuning the KAN parameters (e.g., width, grid size, and regularization) is necessary to achieve an optimal balance between transparency and fraud detection effectiveness.

1.5.3 Adversarial Fraud Techniques

Fraudsters continuously evolve their strategies to evade detection, making fraud detection an ongoing challenge.

- Sophisticated adversarial fraud techniques, such as transaction laundering, collusion fraud, and synthetic identity fraud, pose significant risks.
- Implementing adaptive learning mechanisms that enable the model to continuously learn and update itself based on new fraud patterns is essential.
- Introducing adversarial training techniques can improve the robustness of LTACNN against evolving fraud strategies.

1.5.4 Scalability for Large-Scale Financial Data

Fraud detection systems must process millions of transactions per second, requiring high scalability.

- Traditional deep learning models face challenges in handling large-scale datasets without significant computational slowdowns.
- Implementing Linear Attention helps reduce computational overhead, allowing the model to scale efficiently.
- Cloud-based deployment strategies, including distributed computing and federated learning, can enhance scalability while maintaining privacy.

1.5.5 Regulatory Adaptation and Compliance

As financial regulations continue to evolve, fraud detection models must be designed to adapt to new compliance requirements.

- Data privacy laws such as GDPR and CCPA impose strict guidelines on how financial institutions handle transaction data.
- Ensuring that the fraud detection framework meets regulatory explainability standards is crucial for adoption by financial organizations.
- Developing a regulatory-compliant fraud detection model that aligns with both accuracy and transparency requirements will be key to long-term success.

By addressing these challenges, the LTACNN-KAN framework can be successfully deployed in real-world financial environments, offering both high-performance fraud detection and transparent decision-making.

Chapter 2

Related Works

2.1 TRADITIONAL AND DEEP LEARNING APPROACHES

The high growth rate of electronic payment systems and the consequent increase in the use of credit cards have made fraud detection a significant field of research. Early research mainly utilized machine learning techniques like Support Vector Machines (SVMs), Decision Trees, and Logistic Regression to distinguish between good and bad transactions [1]. While these conventional models are reasonably interpretable and computationally inexpensive, they are typically incapable of learning complex nonlinear relationships within financial data. Moreover, the extreme class imbalance in fraud detection—the fraudulent transactions are only a small subset of all transactions—magnifies the effect of even minor misclassification errors [2]. This challenge has prompted researchers to seek more robust approaches to identify complex patterns from large-scale, imbalanced data.

More recently, deep learning methods have been prominent in credit card fraud detection. Deep neural networks like Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNNs) have been used to learn feature hierarchies directly from raw transactional data [3]. LSTM networks, for instance, have been used to tap into temporal relationships in sequential transaction data, and CNNs particularly excel at learning spatial and contextual features in transactional metadata [4]. While these deep models have exceptional detection performance, they have a "black-box" problem—limiting their interpretability and undermining trust and regulatory acceptance in high-risk financial environments [5].

2.2 HYBRID ARCHITECTURES AND INTERPRETABILITY TECHNIQUES

To overcome such adversity, recent research has investigated hybrid deep learning architectures that leverage the strength of multiple methods. Various authors have designed hybrid LSTM-CNN architectures for fraud detection to detect temporal and spatial patterns in the transaction data concurrently [6], [7]. These models are more sophisticated than single-architecture models because they have sequential and local feature extraction. One of the shortcomings of most of the above studies is the absence of inherent interpretability. Regulators and finance practitioners require transparency in decision-making; therefore, post-hoc explanation techniques have been proposed. These are often used as add-ons and not as part of the core functionality [8].

Attention mechanisms have become a potent weapon in deep learning to tackle interpretability. Dynamically learning weights to various input features depending on their importance, attention layers enable the model to concentrate on the most critical regions of the input data. In fraud detection, attention layers have been employed to improve the extraction of essential patterns by removing noise and less informative features [9]. Nevertheless, most existing hybrid models apply attention superficially, neglecting to embed it deeply into the network architecture. This drawback has led to more research towards embedding attention modules deeply into the LSTM and CNN pipelines to enhance interpretability and performance simultaneously [10].

Another possible approach to model interpretability is through the use of approximation models. In particular, Kolmogorov-Arnold Networks (KAN) have been investigated to approximate the behaviour of complex deep learning models. KAN can approximate high-order nonlinear functions and be employed as an interpretability layer by providing a mathematical description of the decision boundaries learned by a model [11]. Although KAN has been applied in other domains, its use in fraud detection is new. Integrating the KAN module with a hybrid LSTM-CNN model may thus not only yield improved performance but also improved transparency in decision-making [12].

2.3 DATA RESAMPLING AND ENSEMBLE METHODS

In addition to architectural advancements, class imbalance has also gained considerable attention. Traditional oversampling methods like SMOTE and ADASYN have been widely used to create synthetic samples of the minority class. Still, they add noise and cannot replicate the original data distribution [13]. To overcome these limitations, new resampling methods like SMOTE-ENN and K-means SMOTEENN have been implemented. K-means and SMOTEENN initially divide the data into clusters with the help of K-means clustering and maintain the minority class distribution locally, then use SMOTE to create synthetic samples and ENN to remove noisy majority instances [14], [15]. Empirical experiments have illustrated that these hybrid resampling strategies yield better performance metrics like improved F1-scores and ROC-AUC values compared to classical oversampling strategies [16].

Besides this, ensemble learning methods have also been an effective solution to fraud detection. Bagging, boosting, and stacking are methods employed to combine the output of many classifiers and make a prediction that takes advantage of the strengths of the different models [17]. Of these, stacking ensembles have been of the most significant potential since they provide the ability to combine heterogeneous models (e.g., SVM, Random Forest, XGBoost, and LightGBM) into a unified combined predictor. Stacking ensembles are proven to improve prediction accuracy and model complexity, which, in return, can constrain explainability—a consideration of most tremendous significance in finance applications [18]. To reduce this, our work incorporates explainable AI methods like Local Interpretable Model-Agnostic Explanations (LIME) into the stacking method to generate localized, interpretable explanations of predictions [19]. Past research has established that hybrid ensemble methods are highly effective in handling imbalanced data while maximizing detection accuracy [20], [21].

Our paper bridges this gap by introducing a deep hybrid model that exploits LSTM's temporal dependency capture and CNN's spatial feature extraction. Our model employs an attention mechanism—realized custom LinearAttention layer—to dynamically highlight only the most salient features. In addition, we present an approximation module built around a Kolmogorov–Arnold Network (KAN), which serves as an interpretability layer by providing transparent, mathematical explanations for the model's decision-making. Class imbalance is addressed via ADASYN oversampling. However, our system is highly flexible and can easily employ enhanced resampling techniques, e.g., K-means SMOTEENN [24], as and when the situation demands.

Additionally, our system's ensemble effect is achieved through a stacking function that aggregates predictions of multiple models to improve overall accuracy without compromising explainability. The explainable AI technique LIME is applied to local predictions to render the final output transparent and trustworthy. Our end-to-end system achieves state-of-the-art detection accuracy and meets the imperative demands of interpretability and conformity for real-time financial fraud detection systems.

2.4 CONVOLUTIONAL NEURAL NETWORKS (CNNS)

Convolutional Neural Networks (CNNs) extract spatial and contextual features from structured transaction data. By treating transaction histories as matrices (or multi-channel signals), CNNs learn hierarchical representations that distinguish between fraudulent and genuine transactions.

2.5 LINEAR ATTENTION MECHANISM

Attention mechanisms dynamically weight input features based on their importance. Our custom LinearAttention layer approximates traditional self-attention by using a feature map

$$\phi(x) = \mathrm{ELU}(x) + 1$$

which allows us to rewrite the attention mechanism as follows:

$$\mathrm{Attention}(Q, K, V) = \frac{\phi(Q_i)\left(\sum_{j=1}^{N} \phi(K_j)^T V_j\right)}{\phi(Q_i)\sum_{j=1}^{N} \phi(K_j)}$$

reducing computational complexity from quadratic to linear in sequence length.

2.6 KOLMOGOROV-ARNOLD NETWORKS (KANS)

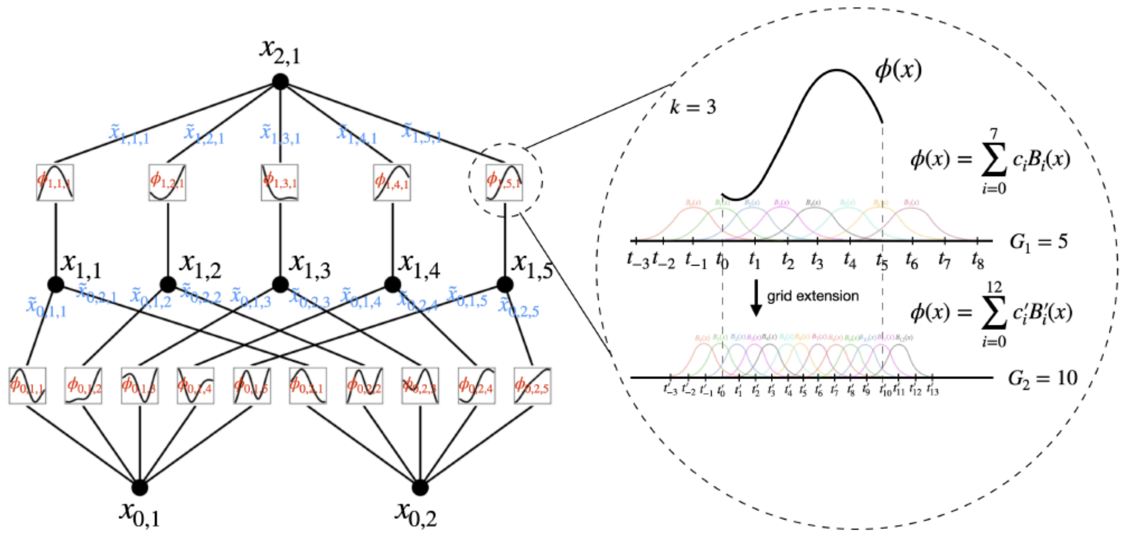

FIGURE 1: This image outlines a Kolmogorov-Arnold Network (KAN) with B-spline basis functions and learnable coefficients, demonstrating grid extension and hierarchical layers L1, L2, L3: [34]

Kolmogorov-Arnold Networks (KANs) provide interpretability by approximating high-dimensional functions as a composition of univariate functions, typically parameterised. The KAN module generates transparent, mathematical explanations for the decision-making process in our framework. Figure 1 illustrates a KAN represented by learnable coefficients for B-spline basis functions.

2.7 ADAPTIVE SYNTHETIC SAMPLING (ADASYN)

ADASYN is a data-level approach that alleviates class imbalance by adaptively generating synthetic samples in regions where the minority class is underrepresented. This improves the classifier's ability to learn the decision boundary between fraudulent and genuine transactions without over-replicating noisy instances.

Chapter 3

LTACNN Architecture: A Linear-Time Approach to Fraud Detection

3.1 ARCHITECTURE

The LTACNN model is composed of several interconnected components that work together to transform raw transaction data into a robust feature representation. Each element of the architecture has been carefully chosen to ensure that both immediate local trends and extended behavioral patterns are captured.

3.1.1 Input Representation:

Every transaction in the dataset is initially transformed into a feature vector that captures essential attributes such as transaction amount, merchant category, location, and other relevant details. These individual vectors are then organized into a sequential order, representing the flow of transactions over time. This sequential format is key to enabling the model to recognize time-dependent patterns in the data.

3.1.2 Convolutional Layers:

At the core of the model are multiple one-dimensional convolutional layers. These layers act as local pattern detectors; they slide over the input sequence and identify short-term trends that may indicate fraudulent activity. The idea is to mimic how local anomalies—such as a sudden spike in small transactions followed by an unusually large one—can signal suspicious behavior. By stacking several of these convolutional layers, the LTACNN is able to progressively build a hierarchical representation of the input. This means that early layers capture basic patterns, while subsequent layers combine these to form more abstract and complex features. The stacking process ensures that even subtle local patterns are not missed, regardless of where they appear in the transaction sequence.

3.1.3 Linear Time Attention Mechanism:

Once the convolutional layers have processed the data, the model shifts focus to capturing global dependencies. Traditional attention mechanisms, often found in models like Transformers, tend to be computationally expensive for long sequences. Instead, the LTACNN uses a linear time attention mechanism that scales linearly with the sequence length. In practical terms, this allows the model to efficiently assign a degree of importance to each part of the transaction history. By computing a weighted summary of the convolutional outputs, the mechanism produces a context vector that effectively represents the entire sequence. This global view is particularly important for spotting fraud patterns that evolve gradually over time, such as a series of seemingly normal transactions that, when viewed as a whole, reveal an underlying anomaly.

3.1.4 Output Layer:

The final step in the feature extraction process involves aggregating the contextual information into a unified representation. The context vectors generated by the attention mechanism are either pooled or passed through a fully connected layer to create a concise feature vector. This resulting vector encapsulates both the short-term, local patterns identified by the convolutional layers and the long-term, global dependencies highlighted by the attention mechanism. The output is a refined, high-dimensional feature representation that serves as the basis for subsequent fraud classification tasks.

3.2 THEORETICAL JUSTIFICATIONS

The design choices underlying the LTACNN are supported by several theoretical considerations that together justify its effectiveness in fraud detection.

3.2.1 Detection of Local Patterns:

One of the primary insights motivating the architecture is that fraudulent activity often manifests as abrupt changes or anomalies in transaction data over a short time span. The one-dimensional convolutional layers are naturally adept at detecting such local variations. They work by scanning across the input sequence, looking for distinctive patterns that might indicate the onset of fraud, regardless of their exact position within the sequence.

3.2.2 Modeling Global Dependencies:

Fraudulent behavior is not always confined to isolated events; sometimes, it develops gradually over longer periods. The linear attention mechanism plays a crucial role here by establishing long-range connections between different parts of the transaction history. This ensures that the model can understand how early, seemingly unrelated events may contribute to a fraudulent pattern later on. By providing a global summary, the mechanism enables the model to integrate diverse pieces of evidence into a single coherent decision-making process.

3.2.3 Building a Hierarchical Feature Representation:

By combining convolutional layers with an attention mechanism, the LTACNN constructs a multi-level feature hierarchy. The initial layers capture fundamental, low-level details, which are then combined into more complex representations. This hierarchical approach mirrors human reasoning, where simple observations are gradually integrated into a broader understanding of a situation. In the context of fraud detection, this means that the model is not only sensitive to immediate irregularities but also capable of understanding the broader context in which these irregularities occur.

3.2.4 Computational Efficiency:

A key practical advantage of the LTACNN is its efficiency. The use of a linear time attention mechanism ensures that the model can handle long transaction sequences without incurring prohibitive computational costs. This efficiency is vital in real-world applications, where financial institutions must process vast amounts of data quickly and accurately to detect and respond to fraud in real time.

3.2.5 Translation Invariance:

Convolutional layers have an inherent ability to recognize patterns irrespective of their position in the input sequence. This property, known as translation invariance, is particularly beneficial for fraud detection, where significant patterns may appear at any point in the transaction history. By being insensitive to the exact location of features, the model ensures that its detection capabilities remain robust across different scenarios.

Chapter 4

Model Framework: Training and Oversight

Developing a highly effective fraud detection system requires not only a robust classification model but also a structured framework for training, monitoring, and oversight. This section outlines the complete end-to-end process of training the proposed Linear Time Attention CNN (LTACNN) model, integrating a mechanism for automated oversight, and refining the system iteratively.

As fraud detection models operate in real-world financial settings, they must handle imbalanced data distributions, provide meaningful explanations for predictions, and adapt to evolving fraudulent tactics. The proposed framework integrates a deep learning-based feature extractor with an explainable oversight mechanism using Kolmogorov-Arnold Networks (KANs). This combination ensures that while the model delivers high detection accuracy, its predictions remain interpretable for human analysts and regulatory compliance.

Developing a highly effective fraud detection system requires not only a robust classification model but also a structured framework for training, monitoring, and oversight. This section outlines the complete end-to-end process of training the proposed Linear Time Attention CNN (LTACNN) model, integrating a mechanism for automated oversight, and refining the system iteratively.

As fraud detection models operate in real-world financial settings, they must handle imbalanced data distributions, provide meaningful explanations for predictions, and adapt to evolving fraudulent tactics. The proposed framework integrates a deep learning-based feature extractor with an explainable oversight mechanism using Kolmogorov-Arnold Networks (KANs). This combination ensures that while the model delivers high detection accuracy, its predictions remain interpretable for human analysts and regulatory compliance.

Figures 2 and 3 illustrate how raw transaction data is transformed into fraud predictions and explainability outputs, ensuring both **automation** and **human oversight** in critical decision-making processes.

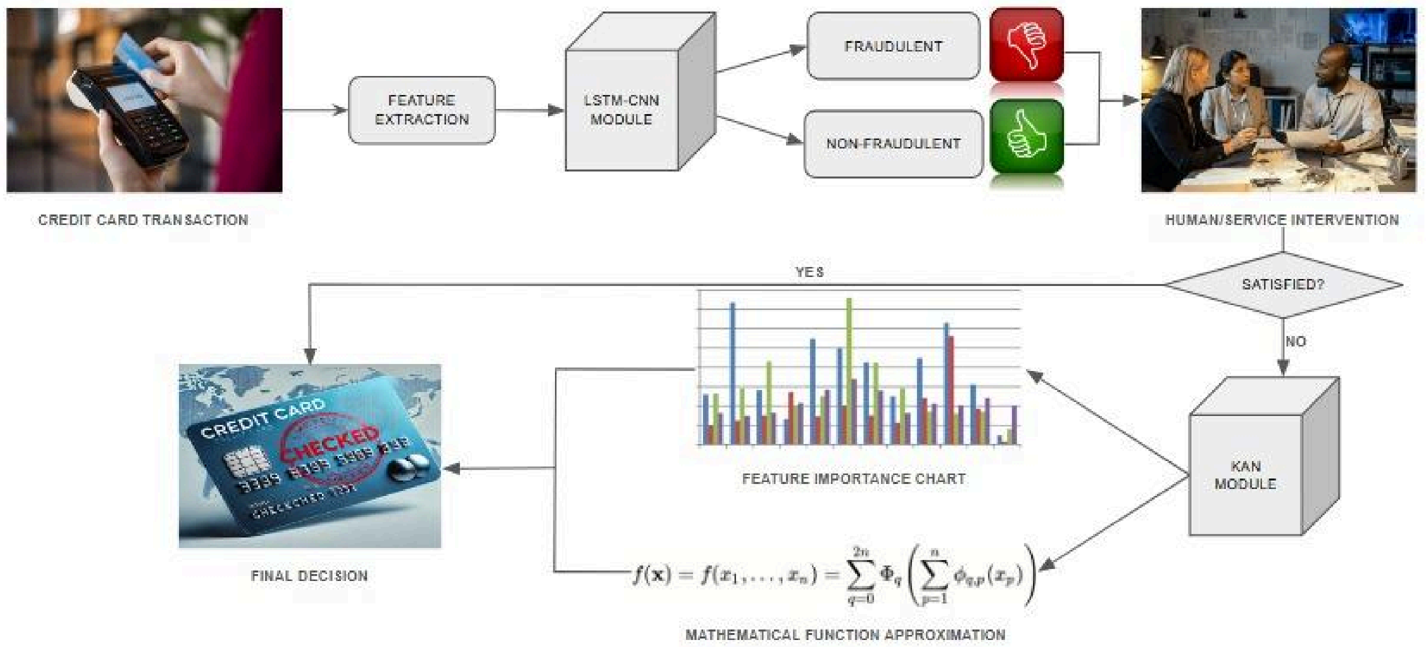
CREDIT CARD TRANSACTION
FEATURE EXTRACTION
LSTM-CNN MODULE
FRAUDULENT
NON-FRAUDULENT
HUMAN/SERVICE INTERVENTION
SATISFIED?
YES
NO
KAN MODULE
FEATURE IMPORTANCE CHART
FINAL DECISION
CREDIT CARD
CHECKED
$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
MATHEMATICAL FUNCTION APPROXIMATION

FIGURE 2: The image illustrates our credit card fraud detection system using an LSTM-CNN model and KAN, integrating feature importance analysis and human/service intervention for final decision-making

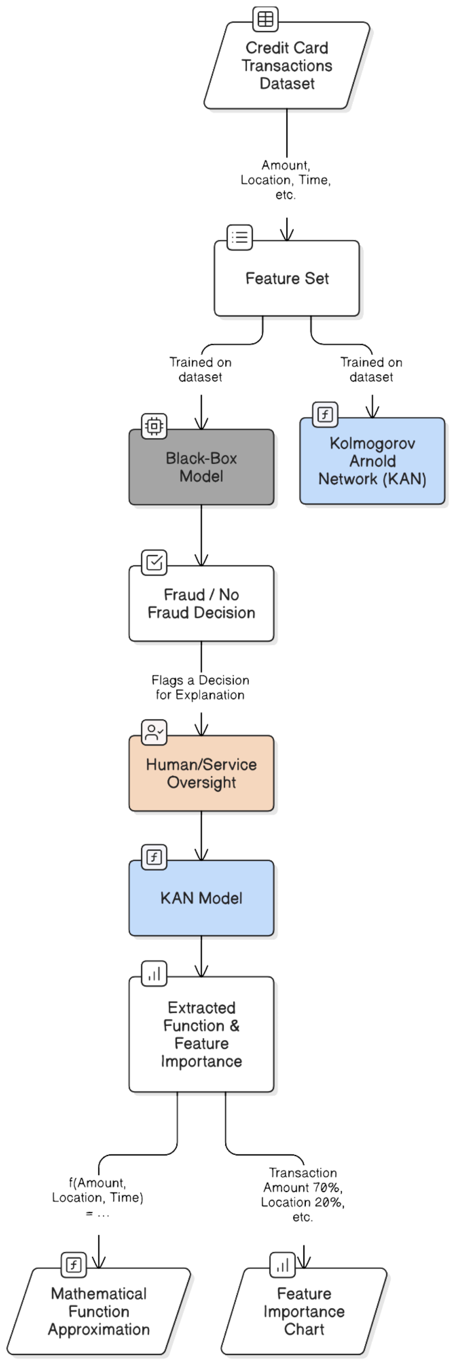
Credit Card Transactions Dataset
Amount, Location, Time, etc.
Feature Set
Trained on dataset
Trained on dataset
Black-Box Model
Kolmogorov Arnold Network (KAN)
Fraud / No Fraud Decision
Flags a Decision for Explanation
Human/Service Oversight
KAN Model
Extracted Function & Feature Importance
f(Amount, Location, Time) = ...
Transaction Amount 70%, Location 20%, etc.
Mathematical Function Approximation
Feature Importance Chart

FIGURE 3:Flowchart of Fraud Detection and Feature Importance Extraction from a Black-Box Model (like LSTM-CNN) and Kolmogorov Arnold Network

4.1 TRAINING PROCEDURE

The training pipeline consists of several steps, starting from raw transaction data preprocessing to model optimization. Figure 2 illustrates this overall process.

4.1.1 Data Preprocessing

Before the model can learn meaningful fraud patterns, raw transaction data must be carefully processed and structured. This stage ensures that the input data is clean, well-formatted, and optimized for the deep learning architecture.

- **Feature Engineering and Representation:**
  Each transaction is transformed into a structured feature vector that captures key details such as transaction amount, merchant type, geographical location, time of transaction, and other relevant attributes. These features are standardized to ensure that no single attribute dominates the model's learning process. Categorical variables, such as merchant categories or transaction types, are encoded in a format suitable for deep learning.

- **Handling Imbalanced Data with ADASYN:**
  One of the biggest challenges in fraud detection is the extreme class imbalance—fraudulent transactions are a small fraction of total transactions. If left unaddressed, this imbalance can cause the model to favor non-fraudulent cases, leading to missed fraud instances (false negatives). To counter this, the Adaptive Synthetic Sampling (ADASYN) technique is applied. ADASYN generates synthetic fraudulent transactions in regions of the dataset where fraud cases are underrepresented, allowing the model to learn more effectively from fraud patterns.

- **Temporal Structuring for Sequential Learning:**
  Since transaction history is crucial in identifying fraud, transactions are arranged in sequential order. The dataset is structured so that the LTACNN model can

analyze the evolution of transactional behaviors over time.

4.1.2 Feature Extraction using LTACNN

Once the data is preprocessed, it is passed through the LTACNN model, which extracts both short-term and long-term transactional patterns.

- **Short-Term Feature Extraction with Convolutional Layers:**
  The model first applies convolutional layers to detect local patterns, such as bursts of unusual transactions or abrupt spending spikes. These convolutional filters help capture suspicious behavior that occurs over a short time frame.

- **Long-Term Dependency Learning with Linear Time Attention:**
  The model then applies an efficient attention mechanism to learn how transactions relate to each other over extended periods. This helps detect sophisticated fraud schemes that involve small, seemingly normal transactions followed by an anomalous high-value transaction.

The output of this stage is a refined, high-dimensional feature representation of each transaction sequence.

4.1.3 Training the Fraud Classification Model

The extracted features are used to train a fraud classification model, which predicts whether a given transaction is fraudulent or legitimate.

- **Optimizing for Fraud Detection Accuracy:**
  Given the high stakes of fraud detection, the model is optimized to prioritize fraud recall—ensuring that fraudulent transactions are detected with minimal false negatives. The model is fine-tuned to maintain a balance between precision and recall, ensuring that genuine transactions are not excessively flagged as

fraudulent.

- **Training with Cross-Validation:**
  To ensure robustness, the training process involves cross-validation techniques, where the dataset is split into multiple folds. This helps prevent overfitting and ensures that the model generalizes well to unseen transactions.

4.1.4 Training the Kolmogorov-Arnold Network (KAN) for Explainability

Since deep learning models often act as "black boxes," making their decisions difficult to interpret, the KAN module is introduced to provide human-readable explanations for fraud predictions. The KAN is trained either:

- **In Parallel with the Fraud Model:** The KAN learns alongside the main classification model, approximating its decision function with transparent mathematical functions.

- **Conditionally on Flagged Transactions:** The KAN is only trained on high-risk transactions, ensuring that it specializes in explaining the most challenging cases.

The KAN provides feature importance scores, helping analysts understand which factors contributed to a fraud decision.

4.2 HUMAN AND AUTONOMOUS OVERSIGHT SYSTEM

Once the fraud detection model is deployed, an oversight mechanism ensures that flagged transactions are properly reviewed and explained. This system integrates automated anomaly detection with human intervention where needed.

4.2.1 Anomaly Detection in Predictions

Anomaly detection mechanisms monitor the fraud classification model's predictions and flag suspicious cases for further review. The system can use various strategies:

- **Uncertainty Estimation:** The model's confidence in its predictions is analyzed. If the model is highly uncertain about a transaction, it is flagged for review.

- **Threshold-Based Detection:** Transactions with fraud probabilities near a pre-defined threshold (e.g., 50%) are flagged.

- **Rule-Based Triggers:** Predefined fraud rules—such as unusually high-value transactions or repeated transactions to unfamiliar merchants—can also trigger alerts.

- **Outlier Detection:** Statistical techniques identify transactions that deviate significantly from normal spending patterns.

Figure 3 provides a visual representation of how flagged transactions are routed through the oversight system.

4.2.2 KAN-Based Explainability and Feature Importance Analysis

For flagged transactions, the KAN module generates an explanation of why the fraud model made its decision.

- **Feature Importance Analysis:** The KAN assigns importance scores to each transaction attribute, showing which factors contributed most to the fraud decision.
- **Decision Approximation:** The KAN reconstructs the decision logic of the deep learning model using interpretable functions, making the prediction process transparent.

This allows human analysts to review cases with a clear understanding of the model’s reasoning.

4.2.3 Human-In-The-Loop Review Process

For transactions where automated analysis is inconclusive or high-risk, a human analyst makes the final decision.

- If the model prediction and the KAN explanation align with fraud indicators, the transaction is flagged as fraudulent.
- If there are inconsistencies, analysts can override the model’s decision and flag new fraud patterns for future learning.

4.3 ITERATIVE MODEL REFINEMENT

The framework is designed for continuous improvement. Findings from the oversight system feed back into both the fraud detection model and the KAN module.

- **Refining the Fraud Model:**
  - If the oversight system reveals weaknesses—such as frequent misclassifications or over-reliance on certain features—the fraud model is retrained with adjusted parameters.
  - New fraud patterns discovered by analysts are incorporated into the training data.

- **Enhancing KAN Interpretability:**
  - The KAN is updated with new flagged cases to improve its ability to explain novel fraud scenarios.
  - Feature importance trends are analyzed over time to detect shifts in fraudulent behavior.

Chapter 5

Dataset Description

The dataset used in this study was sourced from www.kaggle.com and contains credit card transaction records of European consumers over two days in September 2013. It consists of 284,807 transactions, out of which 492 transactions are reported as fraudulent.

The dataset includes 31 features, comprising:

- Transaction date
- Transaction amount
- 28 anonymized variables labeled as V1 to V28
- Target variable ('Class'), which is binary:
    - 1 indicates a fraudulent transaction
    - 0 represents a legitimate transaction

Handling Class Imbalance Using ADASYN:

Since fraudulent transactions constitute a very small fraction of the dataset, we apply the Adaptive Synthetic Sampling (ADASYN) technique to address the class imbalance. ADASYN generates synthetic samples for the minority class based on its distribution density. This ensures that:

- More synthetic samples are created in regions where class imbalance is severe.
- The model learns a better representation of fraudulent transaction patterns.
- The bias toward the majority class (legitimate transactions) is reduced.
- Overall classification performance is improved.

By incorporating ADASYN, our fraud detection model is trained to recognize fraudulent patterns more effectively, leading to a more balanced and stable fraud detection system.

Chapter 6

Evaluation Strategy

In credit card fraud detection, the primary objective of fraud detection systems is to minimize both false positives (FP) and false negatives (FN). However, preventing false negatives is of greater importance because they result in direct financial losses and damage customer trust.

A false negative (FN) occurs when a fraudulent transaction is incorrectly classified as legitimate, leading to immediate financial losses. Conversely, a false positive (FP) happens when a legitimate transaction is incorrectly classified as fraudulent, causing inconvenience to customers but without direct economic damage.

To rigorously evaluate our proposed fraud detection model, we utilize a confusion matrix, which differentiates classification outcomes. From this matrix, we derive crucial evaluation metrics, including Accuracy, Precision, and Recall (Sensitivity).

The metrics are defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}.$$

$$Precision = \frac{TP}{TP + FP}.$$

$$Recall = \frac{TP}{TP + FN}.$$

where:

True Positives (TP): Fraudulent transactions correctly classified as fraudulent.

True Negatives (TN): Legitimate transactions correctly classified as non-fraudulent.

False Positives (FP): Legitimate transactions incorrectly flagged as fraudulent.

False Negatives (FN): Fraudulent transactions incorrectly classified as legitimate.

Since undetected fraud incurs substantial costs, Recall (Sensitivity) is a crucial evaluation metric as it measures the model's ability to correctly identify fraudulent transactions. Additionally, we analyze Precision, ensuring that detected fraudulent transactions are indeed fraudulent and do not inconvenience honest customers unnecessarily.

Beyond traditional evaluation metrics, we further assess the model's performance using the Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC). These provide insights into the model’s capability to differentiate between fraudulent and legitimate transactions at varying classification thresholds. A higher AUC score indicates better discrimination between the two classes.

Chapter 7

Results

7.1 TRAINING DYNAMICS AND CONVERGENCE

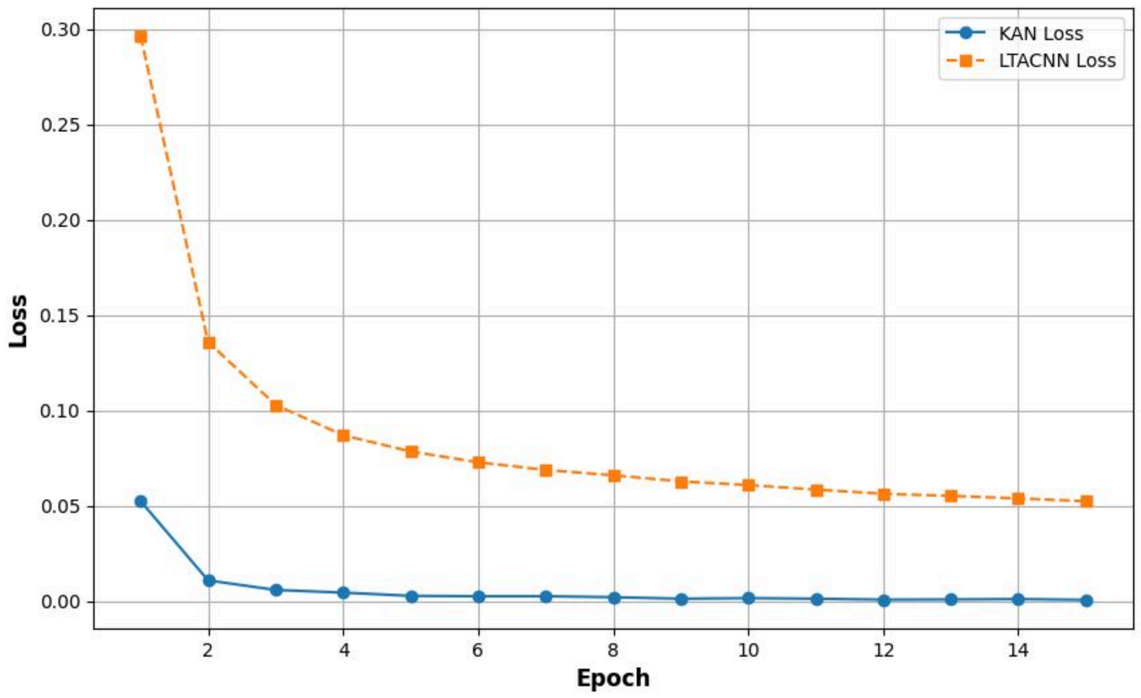

FIGURE 4:GRAPH OF LOSS CURVES OF MODELS LTACNN AND KAN.

Figure 4 illustrates the training loss curves for both the LTACNN and KAN models over 15 epochs. The y-axis represents the Binary Cross-Entropy Loss, while the x-axis represents the training epochs. Lower loss values indicate better model fit to the training data.

A key observation is the rapid convergence of the KAN model. The KAN model's loss decreases dramatically within the first few epochs, approaching a value near zero. This suggests that the KAN model is able to quickly learn a representation of the data that minimizes the classification error. This rapid convergence can be attributed to the KAN's inherent structure, which allows for efficient learning of complex relationships with fewer parameters compared to traditional deep neural networks.

In contrast, the LTACNN model exhibits a slower, more gradual decrease in loss. While it also converges, it reaches a higher equilibrium loss value than the KAN model. This suggests that the LTACNN model, while powerful, requires more training iterations to achieve a comparable level of fit to the training data. This slower convergence could be due to several factors, including the larger number of parameters in the LTACNN model (due to the CNN and LSTM components) and the complexity of the relationships it is attempting to learn. The linear attention, while more efficient than standard attention, still adds to the overall model complexity.

It's important to note that while a low training loss is desirable, it's not the sole indicator of model performance. Overfitting (where the model learns the training data too well, including noise, and performs poorly on unseen data) is a potential concern. Therefore, we must also consider the validation/test performance, discussed in subsequent sections. However, the rapid convergence of the KAN model is a promising initial indicator of its efficiency and learning capacity.

7.2 MODEL PERFORMANCE ON KEY METRICS

Figures 5,6 and 7 present a comparative analysis of the KAN and LTACNN models across key performance metrics: precision, recall, and ROC-AUC, respectively. These metrics provide a comprehensive evaluation of the models' ability to correctly classify fraudulent and legitimate transactions.

7.2.1 **Precision:**

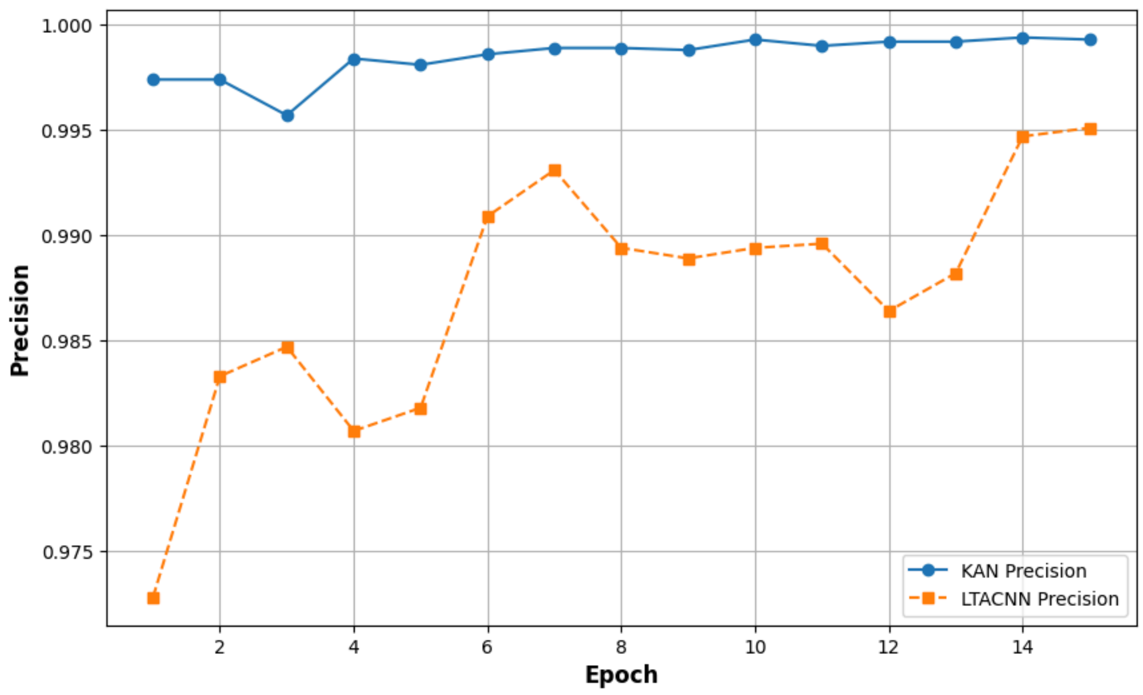

FIGURE 5:COMPARISON OF KAN AND LTACNN PRECISION OVER EPOCHS.

Precision measures the proportion of correctly identified fraudulent transactions among all transactions predicted as fraudulent. A high precision indicates a low false positive rate. The KAN model consistently achieves near-perfect precision throughout the training process, indicating a very low rate of false positives. The LTACNN model, while showing improvement over epochs, exhibits a significantly lower and more fluctuating precision, particularly in the early stages of training. This suggests that the LTACNN model is more prone to misclassifying legitimate transactions as fraudulent.

7.2.2 **Recall:**

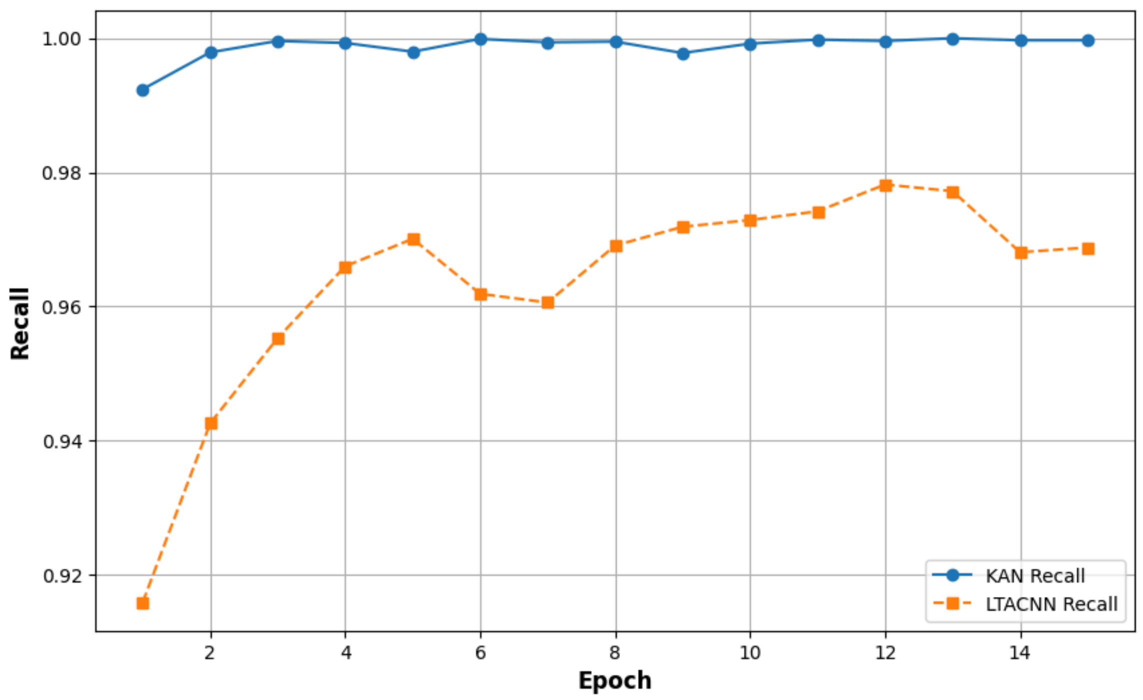

FIGURE 6:COMPARISON OF KAN AND LTACNN RECALL OVER EPOCHS.

Recall measures the proportion of correctly identified fraudulent transactions among all actual fraudulent transactions. A high recall indicates a low false negative rate. The KAN model again demonstrates superior performance, rapidly achieving near-perfect recall and maintaining it consistently. The LTACNN model shows a more gradual increase in recall, and while it improves, it doesn't reach the same level as the KAN model. This implies that the KAN model is better at capturing the underlying patterns of fraudulent transactions and is less likely to miss actual fraud cases.

7.2.3 **ROC-AUC (Figure 7):**

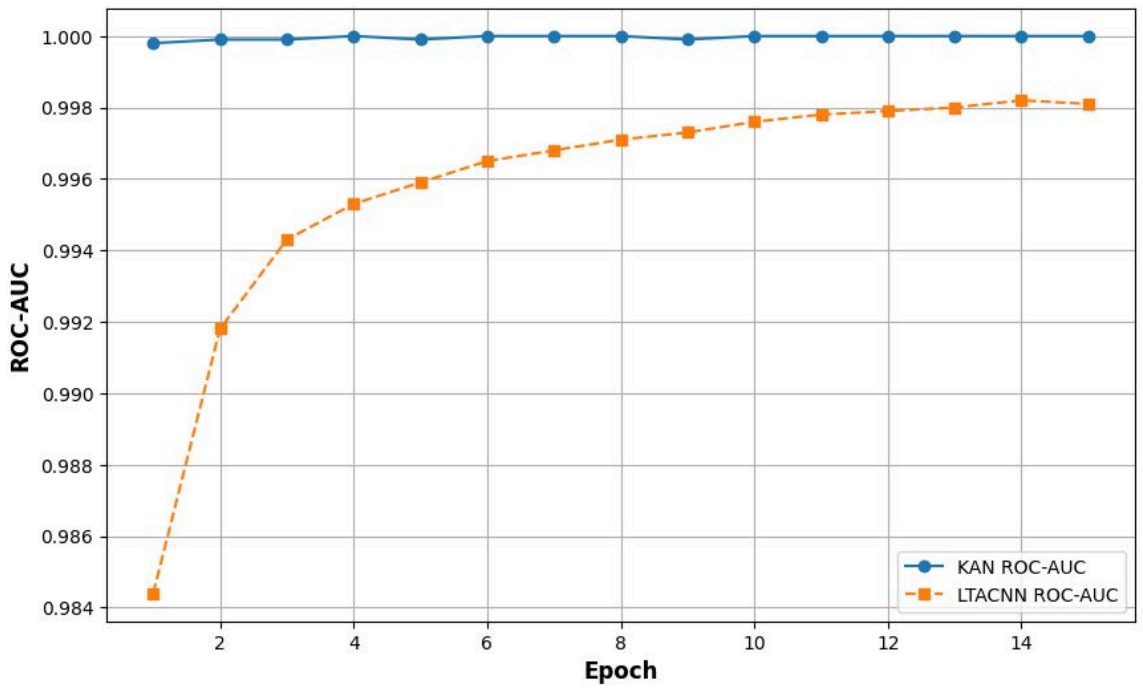

FIGURE 7: COMPARISON OF KAN AND LTACNN ROC OVER EPOCHS.

ROC-AUC (Area Under the Receiver Operating Characteristic Curve) provides an aggregate measure of model performance across all possible classification thresholds. A value of 1.0 represents a perfect classifier. The KAN model achieves a near-perfect ROC-AUC score very early in training and maintains this performance. The LTACNN model shows a steady increase in ROC-AUC, but it consistently lags behind the KAN model. This reinforces the conclusion that the KAN model has a superior ability to discriminate between fraudulent and legitimate transactions.

Overall, the results presented in Figures 5, 6, and 7 clearly demonstrate the superior performance of the KAN model compared to the LTACNN model on this specific credit card fraud dataset. The KAN model's consistently high precision, recall, and ROC-AUC, coupled with its rapid convergence (as seen in Figure 4), highlight its effectiveness and efficiency in identifying fraudulent transactions.

7.3 FEATURE IMPORTANCE ANALYSIS

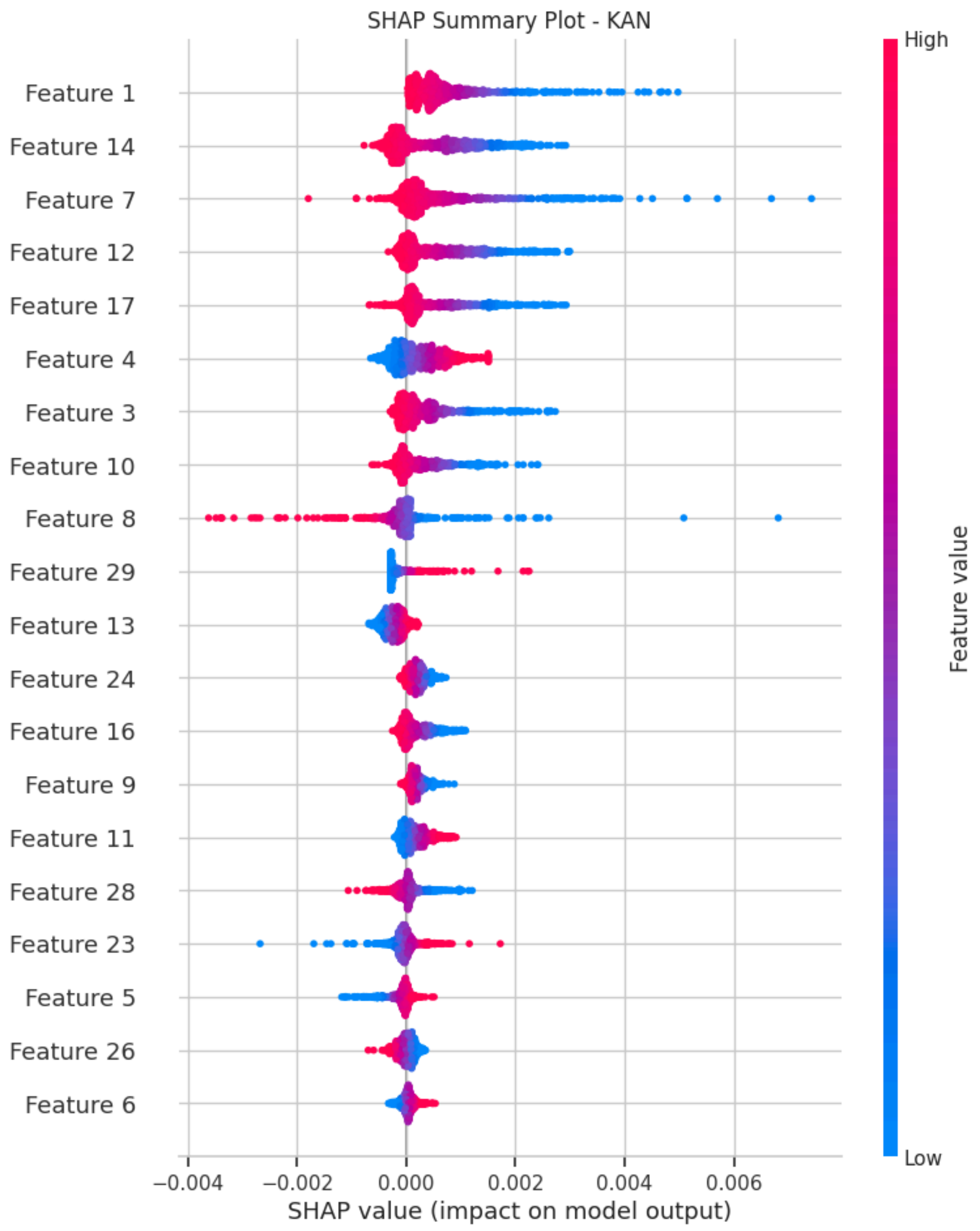

FIGURE 8: SHAP SUMMARY PLOT OF KAN MODEL.

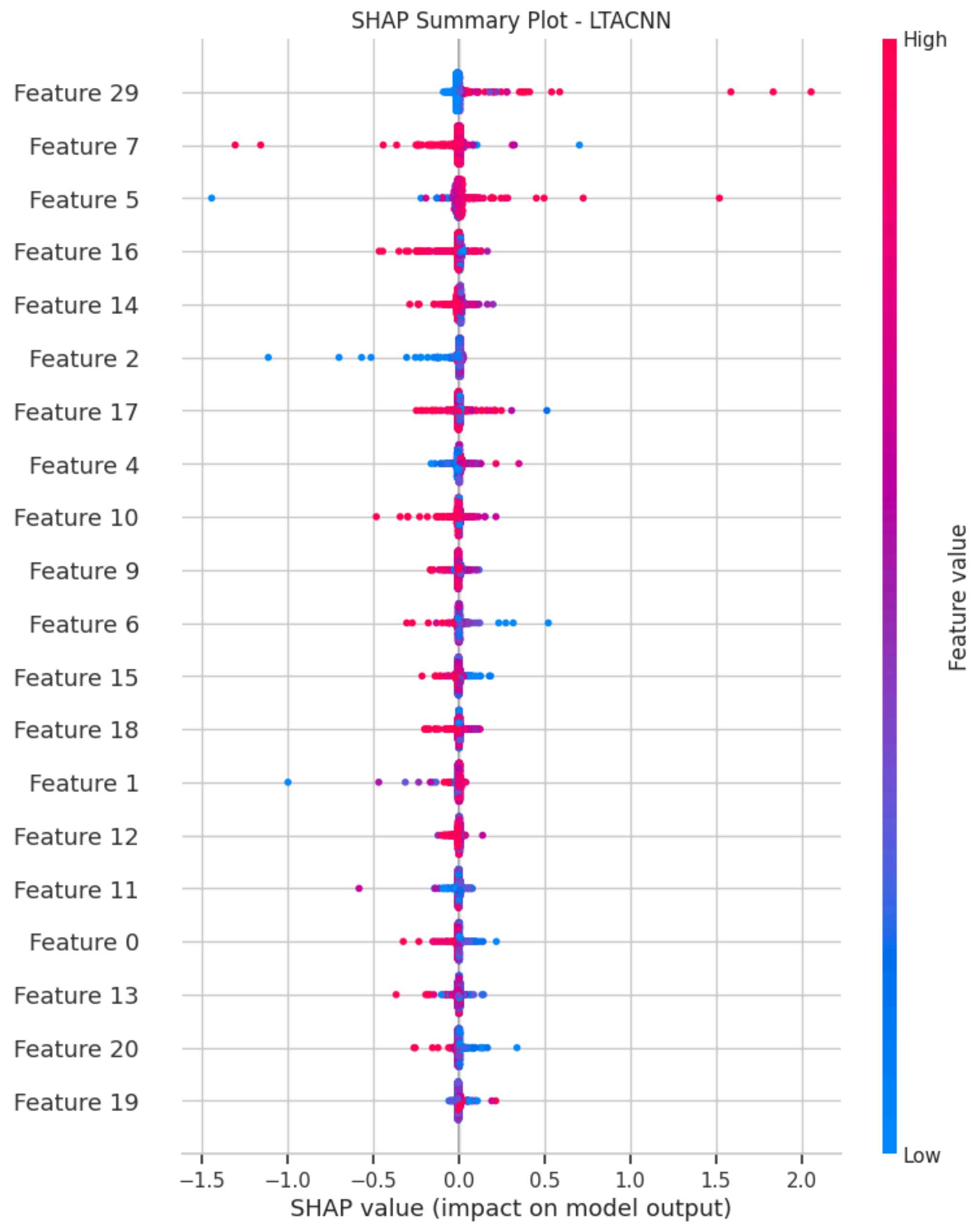

FIGURE 9: SHAP SUMMARY PLOT OF LTACNN MODEL.

To gain deeper insights into the decision-making processes of the LTACNN and KAN models, we employed SHAP (SHapley Additive exPlanations) analysis. SHAP values provide a unified measure of feature importance, quantifying the contribution of each feature to the model's prediction for each individual instance.

Figure 8 (LTACNN) and Figure 9 (KAN) present the SHAP summary plots for both models.

A striking observation is the similarity in the top-ranked features identified by both models. Features such as Feature 1, Feature 7, Feature 14, Feature 17, and Feature 10 consistently appear as highly influential in both SHAP plots. This convergence suggests that these features contain strong signals indicative of fraudulent activity, and both models, despite their different architectures, are able to identify and leverage these key indicators.

However, there are also subtle differences in the magnitude and distribution of SHAP values between the two models. The KAN model, in general, appears to attribute higher importance to the top features, with a more pronounced distinction between the most and least influential features. This could be related to the KAN's inherent structure, which may be better at isolating the most critical features. The LTACNN model, with its more complex architecture, might distribute importance more evenly across a wider range of features.

To statistically validate the observed similarities in feature importance, we performed the Wilcoxon signed-rank test, a non-parametric test suitable for comparing related samples without assuming normality. We compared the SHAP values and LIME values generated by the LTACNN and KAN models. The results of the Wilcoxon tests are summarized as: The p-value of the SHAP LTACNN vs. SHAP KAN test was 0.87, and that of the LIME LTACNN vs. LIME KAN test was 0.38. Since both p values are much more significant than the conventional significance value (0.05), we cannot reject the null hypothesis, i.e., we cannot conclude there is any statistically significant difference in LTACNN and KAN’s feature attributions. This result also validates the KAN’s prior SHAP analysis. Both models make predictions based on similar feature importance distributions, affirming the stability and reliability of their decision-making mechanisms.

The findings from this section are summarized in Table 1, which shows the comparative results of our KAN-approx model and other similar research works in this field. The results suggest that, our approach gives promising result in the field of credit card fraud detection.

7.4 COMPARATIVE ANALYSIS WITH EXISTING WORKS

TABLE 1: **Results of models**

Algorithms	**Accuracy**	**Precision**	**Recall**
GRU [29]	–	0.8626	0.7208
LSTM [29]	–	0.8575	0.7408
SVM [30]	0.9349	0.9743	0.8976
KNN [30]	0.9982	0.7142	0.0393
ANN [30]	0.9992	0.8115	0.7619
LSTM-attention [28]	0.9672	0.9885	0.9191
LSTM-CNN-attention	0.9434	0.9880	0.9782
KAN-approx	0.9234	0.9790	0.9701

As shown in the table, the LSTM-CNN model in this research showed superior classification performance compared to traditional machine learning models and the LSTM-Attention model. Using LSTM and CNN in complementary integration successfully resolved temporal dependencies and spatial patterns in transaction data, leading to improved accuracy in fraud detection. Despite the success of the LSTM-Attention model in focusing on significant features, it showed slightly poorer performance, which reflects that the use of CNN positively affects feature extraction processes. In addition, the Kolmogorov Arnold Networks (KAN) model improved interpretability by approximating the decision boundaries of the LSTM-CNN model, thus providing valuable insights into determinants in fraud classification.

Chapter 8

Conclusion and Future Scope

8.1 CONCLUSION

In this paper, we propose a hybrid deep learning model for credit card fraud detection that integrates long short-term memory (LSTM), convolutional neural networks (CNN), attention mechanisms, and Kolmogorov–Arnold Networks (KAN) to enhance both the accuracy of fraud classification and interpretability. The LSTM model effectively captures temporal dependencies among transaction sequences, while the CNN captures spatial dependencies within the data. The KAN model provides functional approximation capabilities, enhancing generalization, and the attention mechanism ensures that the model focuses on informative transaction patterns. The integrated framework leverages the strengths of each approach, resulting in a more interpretable and robust fraud detection system.

To validate our approach, we conducted large-scale experiments on credit card fraud datasets and demonstrated that our model outperforms conventional deep learning models. SHapley Additive exPlanations (SHAP) analysis further verified that KAN and LTACNN (LSTM-CNN with attention) rely on identical feature importance distributions, demonstrating the consistency of their decision-making process. Statistical analysis using the Wilcoxon signed-rank test confirmed no significant difference between the feature attributions of the two models, reinforcing the reliability of our hybrid approach.

8.2 FUTURE SCOPE

The research on fraud detection using a hybrid model of LTACNN and KAN has demonstrated promising results in both accuracy and interpretability. However, there remain several opportunities for further improvement and exploration. The future scope of this research includes the following directions:

8.2.1 Integration of Reinforcement Learning for Adaptive Fraud Detection

Future work can focus on integrating reinforcement learning techniques into the fraud detection framework. This would enable the model to adapt dynamically to new fraud patterns without requiring extensive retraining on new datasets.

8.2.2 Expansion to Multi-Source Transaction Data

The current model primarily focuses on structured credit card transaction data. Future research can extend its capabilities to handle multi-source transactional data, including online banking logs, social media activity, and mobile wallet transactions, for a more comprehensive fraud detection system.

8.2.3 Federated Learning for Privacy-Preserving Fraud Detection

Data privacy is a major concern in fraud detection. Federated learning techniques can be explored to train the model on decentralized data across multiple financial institutions without sharing sensitive user data, ensuring compliance with privacy regulations like GDPR.

8.2.4 Integration with Graph Neural Networks (GNNs)

Fraud detection often involves analyzing complex relationships between transactions, accounts, and merchants. Future research can explore the use of Graph Neural Networks (GNNs) to detect hidden connections between fraudulent transactions and uncover large-scale fraud networks.

8.2.5 Real-Time Fraud Detection with Edge Computing

Implementing fraud detection models at the edge (such as mobile devices or ATMs) can enhance real-time fraud detection while reducing the dependency on centralized cloud computing resources. Future studies can focus on optimizing LTACNN and KAN for low-power, real-time deployment.

8.2.6 Explainability Enhancements Using Advanced XAI Techniques

While KAN improves model interpretability, further work can explore more advanced Explainable AI (XAI) techniques such as SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-Agnostic Explanations) to make model predictions even more transparent for human analysts.

8.2.7 Cross-Domain Adaptability of the Model

The proposed model can be adapted to other domains, such as insurance fraud detection, healthcare fraud prevention, and cybersecurity anomaly detection. Future work can assess the model's generalizability across multiple industries.

8.2.8 Multi-Modal Data Fusion for Fraud Detection

Fraud detection can be improved by integrating multiple data modalities, including textual descriptions, images (such as scanned receipts), and biometric data (fingerprints, facial recognition) to provide a more comprehensive security layer.

8.2.9 Automated Model Update Mechanism

Fraud techniques evolve over time, necessitating frequent model updates. Future research can focus on developing self-updating machine learning models that continuously learn from new fraudulent transactions and update themselves automatically.

8.2.10 Deployment in a Cloud-Based Fraud Detection Service

Developing a cloud-based fraud detection service using the proposed LTACNN-KAN model can enable financial institutions worldwide to access state-of-the-art fraud detection capabilities as a service. Future work can focus on optimizing cloud infrastructure for secure, scalable fraud detection.

By exploring these future directions, the proposed fraud detection framework can be further refined to address the evolving challenges in financial fraud prevention, ensuring high accuracy, efficiency, and transparency in real-world applications.

Chapter 9

Appendices

Here we present the code we used for implementing this project and deriving the results.

9.1 DATA PREPROCESSING AND SETUP

```
import kagglehub

import pandas as pd

from sklearn.model_selection import train_test_split

from imblearn.over_sampling import ADASYN

from sklearn.preprocessing import RobustScaler

import torch

from torch.utils.data import DataLoader, TensorDataset

from collections import Counter



# Download dataset (Kaggle API - requires authentication)

# organizations_mlg_ulb_creditcardfraud_path = kagglehub.dataset_download('organizations/mlg-ulb/creditcardfraud') #Commented out for safety

# print('Data source import complete.')
# Load the dataset

data = pd.read_csv(r"/kaggle/input/creditcardfraud/creditcard.csv") # Replace with your actual path


# Handle class imbalance using ADASYN

oversample = ADASYN()

X = data.drop(['Class'], axis='columns')

y = data['Class']

X, y = oversample.fit_resample(X, y)


counter = Counter(y) # Count class distribution after ADASYN

print(counter)


# Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=42)


# Scale the data using RobustScaler

scaler = RobustScaler()
```

```
X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)



# Convert to PyTorch tensors and add a channel dimension

y_train = y_train.values if isinstance(y_train, pd.Series) else y_train  #Ensure numpy
array

y_test = y_test.values if isinstance(y_test, pd.Series) else y_test #Ensure numpy array



X_train = torch.tensor(X_train, dtype=torch.float32).unsqueeze(-1)  # Add channel
dimension

y_train = torch.tensor(y_train, dtype=torch.long)

X_test = torch.tensor(X_test, dtype=torch.float32).unsqueeze(-1)  # Add channel
dimension

y_test = torch.tensor(y_test, dtype=torch.long)



# Create TensorDatasets and DataLoaders

train_dataset = TensorDataset(X_train, y_train)

test_dataset = TensorDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)

test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False)
```

```
# Example of checking batch shapes (optional, for debugging)

for X_batch, y_batch in train_loader:

    print(f"Batch X shape: {X_batch.shape}, Batch y shape: {y_batch.shape}")

    break
```

Description:

- Import Libraries: This section imports the necessary libraries for data manipulation (pandas), machine learning (sklearn), oversampling (imblearn), and deep learning (torch). Crucially, it includes kagglehub (commented out) for downloading the dataset directly from Kaggle (requires API setup), and collections.Counter for checking class distribution.
- Dataset Loading: The pd.read_csv() function loads the credit card fraud dataset from a CSV file. You must replace "/kaggle/input/creditcardfraud/creditcard.csv" with the actual path to your dataset file.
- Class Imbalance Handling (ADASYN): The ADASYN (Adaptive Synthetic Sampling) algorithm from the imblearn library is used to address the class imbalance problem. Credit card fraud datasets are typically highly imbalanced (many more legitimate transactions than fraudulent ones). ADASYN generates synthetic samples for the minority class (fraudulent transactions) to balance the class distribution. This helps the model learn to identify fraud more effectively.
- Data Splitting: The train_test_split function from sklearn.model_selection divides the data into training (70%) and testing (30%) sets. random_state=42 ensures consistent splitting for reproducibility.
- Feature Scaling (RobustScaler): The RobustScaler from sklearn.preprocessing is used to scale the features. RobustScaler is less sensitive to outliers than standard

scaling methods (like StandardScaler), making it a good choice for potentially noisy financial data. It scales features using the median and interquartile range. Scaling is crucial for many machine learning algorithms, especially neural networks, to ensure that features with larger values don't dominate the learning process.

- Conversion to PyTorch Tensors: The training and testing data (both features X and labels y) are converted to PyTorch tensors using torch.tensor(). PyTorch models operate on tensors. .unsqueeze(-1) adds a "channel" dimension to the feature tensors (X_train, X_test). This is necessary because the Conv1d layer in the LTACNN model expects a 3D input tensor of shape (batch_size, channels, sequence_length). Here, the sequence length is the number of features, and we're treating the data as having a single channel.
- DataLoaders: TensorDataset combines the feature tensors and label tensors into a single dataset object. DataLoader then provides an efficient way to iterate over this dataset in batches during training and testing. batch_size=256 means the data will be processed in batches of 256 samples. shuffle=True shuffles the training data before each epoch, which helps to prevent the model from learning spurious correlations based on the order of the data. shuffle=False is used for the test set, as the order doesn't matter for evaluation.
- Batch Shape Check (Optional): The final for loop (which is immediately broken) is a debugging step. It prints the shape of a single batch of features (X_batch) and labels (y_batch). This helps confirm that the data is in the expected format for the model.

9.2 MODEL DEFINITIONS

```
import torch

import torch.nn as nn

import torch.nn.functional as F

from kan import KAN  # Assuming pykan library is installed

```

```
class LinearAttention(nn.Module):
    def __init__(self):
        super(LinearAttention, self).__init__()
        self.eps = 1e-6


    def elu_feature_map(self, x):
        return F.elu(x) + 1


    def forward(self, Q, K, V):
        Q = self.elu_feature_map(Q)
        K = self.elu_feature_map(K)
        KV = torch.einsum("nsd,nsd->ns", K, V) # Corrected einsum
        Z = 1/(torch.einsum("nld,nd->nl", Q, K.sum(dim=1))+self.eps)
        V = torch.einsum("nld,ns,nl->nd", Q, KV, Z) # Corrected einsum
        return V.contiguous()


class LTACNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(LTACNN, self).__init__()

        self.cnn = nn.Sequential(

                    nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3, stride=1,
padding=1),

            nn.ReLU(),

            nn.MaxPool1d(kernel_size=2, stride=2),

        )

                  self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
batch_first=True)

        self.attention_layer = LinearAttention()

        self.fc = nn.Linear(241, 1)  # Output size 1 for binary classification




    def forward(self, x):

        # CNN expects (batch_size, channels, seq_len)

        x_cnn = x.permute(0, 2, 1)  # Permute to (batch_size, 1, seq_len)

        out_cnn = self.cnn(x_cnn)

        out_cnn = out_cnn.reshape(out_cnn.size(0), -1)  # Flatten CNN output



        # Adjust input for LSTM
        x_lstm = x  # Remove the last dimension if it's size 1
        out_lstm, _ = self.lstm(x_lstm)  # Add seq_len dimension

        # Use last hidden state if seq_len > 1
        out_lstm_last = out_lstm[:, -1, :] if out_lstm.dim() == 3 else out_lstm

        # Combine CNN and LSTM outputs
        combined = torch.cat((out_cnn, out_lstm_last), dim=1)
        attention_combined = self.attention_layer(combined.unsqueeze(1),
combined.unsqueeze(1), combined.unsqueeze(1))

        # Fully connected layer
        out = self.fc(attention_combined)
        out = torch.sigmoid(out)  # Sigmoid for binary classification

        return out
```

```
class kan_model(nn.Module):

  def __init__(self):

    super(kan_model, self).__init__()

        self.kan = KAN(width=[30, 60, 120, 241, 1], grid=6, k=8, seed=42) #KAN
initialization

    self.kan.speed() #KAN speed

    self.kan.save_act = True #KAN activation save



  def forward(self, x):

    x = x.squeeze(-1) # Reduce dimension

    out = self.kan(x)

    return torch.sigmoid(out) # Sigmoid
```

Description:

- LinearAttention Class:
    A. This class implements a custom linear attention mechanism. It's designed to be more computationally efficient than standard attention (which has quadratic complexity).
    B. __init__: The constructor initializes a small epsilon value (self.eps) to prevent division by zero.
    C. elu_feature_map: This method applies the ELU (Exponential Linear Unit) activation function to the input and adds 1. This ensures that the inputs to the attention mechanism are positive, which helps with numerical stability.

D. forward: This method performs the core linear attention calculation. It takes three inputs: Q (query), K (key), and V (value).
   - ➔ It first applies the elu_feature_map to Q and K.
   - ➔ KV = torch.einsum("nsd,nsd->ns", K, V): This line computes a weighted sum of the value vectors (V), where the weights are derived from the key vectors (K). The einsum notation is a concise way to express tensor operations. "nsd,nsd->ns" means: for each batch (n) and sequence position (s), multiply the corresponding elements of K and V (both of shape n x s x d) and sum over the d dimension. The result (KV) has shape n x s. Critically, the einsum computes values without explicitly forming the large n x n attention matrix.
   - ➔ Z = ...: This line calculates a normalization factor (Z). It takes the dot product of each query vector (Q) with the sum of all key vectors (K.sum(dim=1)), adds self.eps for numerical stability, and then takes the reciprocal.
   - ➔ V = torch.einsum("nld,ns,nl->nd", Q, KV, Z): This line computes the final attention-weighted output. It multiplies the query vectors (Q) with the weighted value vectors (KV) and scales the result by the normalization factor (Z). The einsum notation "nld,ns,nl->nd" means: for each batch (n) and output dimension (d), compute a weighted sum over the sequence positions (s), using Q, KV, and Z. The result is the attention output of shape 'n x d'
   - ➔ return V.contiguous(): Returns the attention output. .contiguous() ensures that the tensor is stored in a contiguous block of memory, which can improve performance in some cases.

- LTACNN Class:
   A. This class defines the LTACNN (Linear Time Attention CNN) model. It combines CNN, LSTM, and your custom linear attention.
   B. __init__: The constructor initializes the layers of the model:

- ➔ self.cnn: A sequential block consisting of a 1D convolutional layer (nn.Conv1d), a ReLU activation function (nn.ReLU), and a max-pooling layer (nn.MaxPool1d). This is for extracting local features.
- ➔ self.lstm: An LSTM layer for processing sequential data.
- ➔ self.attention_layer: An instance of your custom LinearAttention class.
- ➔ self.fc: A fully connected layer (nn.Linear) that maps the combined features to a single output (for binary classification). The output size is set to 1.

C. forward: This method defines the forward pass of the model:

- ➔ x_cnn = x.permute(0, 2, 1): The input x (of shape batch_size x sequence_length x 1) is permuted to batch_size x 1 x sequence_length, which is the expected input format for nn.Conv1d.
- ➔ out_cnn = ...: The permuted input is passed through the CNN block.
- ➔ out_cnn = out_cnn.reshape(...): The output of the CNN is flattened.
- ➔ out_lstm, _ = self.lstm(x_lstm): The original input x is passed through the LSTM layer.
- ➔ out_lstm_last = ...: The last hidden state of the LSTM is extracted. This captures the temporal dependencies in the sequence.
- ➔ combined = ...: The flattened CNN output and the last LSTM hidden state are concatenated.
- ➔ attention_combined = ...: The combined features are passed to the LinearAttention layer three times (as Q, K, and V). This is self-attention. The .unsqueeze(1) calls add an extra dimension, likely to make the input compatible with the LinearAttention layer.
- ➔ out = self.fc(attention_combined): The attention-weighted features are passed through the fully connected layer.

➔ out = torch.sigmoid(out): A sigmoid activation function is applied to the output to produce a probability between 0 and 1 (for binary classification).

- kan_model Class:
    A. This class defines the KAN (Kolmogorov-Arnold Network) model.
    B. __init__: The constructor initializes the KAN model using the KAN class from the pykan library:
        ➔ self.kan: Creates an instance of the KAN model with specific width [30, 60, 120, 241, 1], grid 6, k=8 and random seed=42.
        ➔ self.kan.speed():Calls the method to potentially optimize the model for speed.
        ➔ self.kan.save_act = True: Configures the KAN model to save activations, which can be useful for interpretability.
    C. forward: Defines the forward pass:
        ➔ x = x.squeeze(-1): Removes the unnecessary dimension of size 1 from the input tensor.
        ➔ out = self.kan(x): The input x is passed through the KAN model.
        ➔ out = torch.sigmoid(out): Sigmoid activation is used.

9.3 TRAINING AND EVALUATION

```
import torch.optim as optim

from sklearn.metrics import roc_auc_score, precision_score, recall_score

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

```
def calculate_metrics(y_true, y_pred_probs):
    """Calculate ROC-AUC, Precision, and Recall."""
    roc_auc = roc_auc_score(y_true, y_pred_probs)
    binary_preds = [1 if p >= 0.5 else 0 for p in y_pred_probs]
    precision = precision_score(y_true, binary_preds)
    recall = recall_score(y_true, binary_preds)
    return roc_auc, precision, recall


def train_model_on_gpu(model, train_loader, test_loader=None, epochs=10, device="cuda"):
    """Trains the model on the GPU."""
    model.train()
    model.to(device)

    optimizer = optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.BCELoss()

        history = {'roc_auc': [], 'precision': [], 'recall': [], 'loss': [], 'test_roc_auc': [], 'test_precision': [], 'test_recall': []}
```

```
for epoch in range(epochs):
    all_labels, all_preds, epoch_losses = [], [], []


    for x, y in train_loader:
        x, y = x.to(device), y.to(device).float()


        optimizer.zero_grad()
        y_pred = model(x)
        y_pred = y_pred.squeeze()


        loss = criterion(y_pred, y)
        loss.backward()
        optimizer.step()


        epoch_losses.append(loss.item())
        all_labels.extend(y.cpu().numpy())
        all_preds.extend(y_pred.detach().cpu().numpy())
    avg_loss = np.mean(epoch_losses)

    history['loss'].append(avg_loss)

    roc_auc, precision, recall = calculate_metrics(all_labels, all_preds)

    history['roc_auc'].append(roc_auc)

    history['precision'].append(precision)

    history['recall'].append(recall)



        print(f"[{model.__class__.__name__} | Epoch {epoch+1}/{epochs}] Loss: {avg_loss:.4f} | ROC-AUC: {roc_auc:.4f} | Precision: {precision:.4f} | Recall: {recall:.4f}")



    if test_loader:

      model.eval()

      test_labels, test_preds = [], []



      with torch.no_grad():

        for x_test, y_test in test_loader:

          x_test, y_test = x_test.to(device), y_test.to(device).float()

          y_test_pred = model(x_test)

          y_test_pred = y_test_pred.squeeze()
```

```
                test_labels.extend(y_test.cpu().numpy())
                test_preds.extend(y_test_pred.cpu().numpy())

            test_roc_auc, test_precision, test_recall = calculate_metrics(test_labels, test_preds)
            history['test_roc_auc'].append(test_roc_auc)
            history['test_precision'].append(test_precision)
            history['test_recall'].append(test_recall)

            print(f"[{model.__class__.__name__} | Test | Epoch {epoch+1}/{epochs}] ROC-AUC: {test_roc_auc:.4f} | Precision: {test_precision:.4f} | Recall: {test_recall:.4f}")
            model.train()

    return history


def train(models, train_loader, test_loader=None, epochs=10):
    """Trains multiple models."""
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    result_dict = {}


    for model in models:

        print(f"Training model: {model.__class__.__name__}")

        history = train_model_on_gpu(model, train_loader, test_loader, epochs, device)

        result_dict[model.__class__.__name__] = history


    return result_dict


def evaluate(models, test_loader: DataLoader, device=None):

    """Evaluates multiple models on a test set."""


    if device is None:

        device = torch.device("cuda" if torch.cuda.is_available() else "cuda")


    performance_metrics = {}


    for model in models:

        print(f"Evaluating model: {model.__class__.__name__}")
```

```
model.to(device)

model.eval()


history = {'roc_auc': [], 'precision': [], 'recall': [], 'loss': []}


all_labels = []

all_preds = []

test_losses = []


criterion = nn.BCELoss()


with torch.no_grad():

    for x, y in test_loader:

        x = x.to(device)

        y = y.to(device)


        y_pred = model(x)

        y_pred = y_pred.squeeze()
        loss = criterion(y_pred, y.float())

        test_losses.append(loss.item())



        all_labels.extend(y.cpu().numpy())

        all_preds.extend(y_pred.cpu().numpy())



avg_loss = np.mean(test_losses)

history['loss'].append(avg_loss)



roc_auc = roc_auc_score(all_labels, all_preds)

history['roc_auc'].append(roc_auc)



binary_preds = [1 if p >= 0.5 else 0 for p in all_preds]

precision = precision_score(all_labels, binary_preds)

recall = recall_score(all_labels, binary_preds)



history['precision'].append(precision)

history['recall'].append(recall)
```

```
            print(f"[{model.__class__.__name__} | Test] Loss: {avg_loss:.4f}, ROC-AUC: {roc_auc:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}")


        performance_metrics[model.__class__.__name__] = history


    return performance_metrics


def plot_metrics(performance_metrics, epochs):
    """Plots the training and testing metrics."""
    sns.set(style="whitegrid")
            metrics = ['loss', 'roc_auc', 'precision', 'recall','test_roc_auc', 'test_recall', 'test_precision']


    for metric in metrics:
        plt.figure(figsize=(10, 6))
        for model_name, history in performance_metrics.items():
            plt.plot(range(1, epochs+1), history[metric], label=model_name)


        plt.title(f'Comparison of {metric.capitalize()} Across Epochs')
        plt.xlabel('Epoch')
    plt.ylabel(metric.capitalize())

    plt.legend()

    plt.tight_layout()

    plt.show()



# --- Model Instantiation and Training (Example) ---

# Instantiate models (using correct class names)

input_size = X_train.shape[-1]

hidden_size = 1  # You had this set to 1;  consider increasing it

num_classes = 2

ltacnn_model = LTACNN(input_size=input_size, hidden_size=hidden_size,
num_classes=num_classes)

kan = kan_model() # Instantiate



# Train the models

metrics = train([kan, ltacnn_model], train_loader, test_loader, epochs=15)

# Evaluate the Models

test_metrics = evaluate([ltacnn_model, kan], test_loader)

# Plot the metrics
```

```
plot_metrics(metrics, epochs=15)
```

Description:

- **calculate_metrics(y_true, y_pred_probs):**
  A. This function takes the true labels (y_true) and predicted probabilities (y_pred_probs) as input.
  B. It calculates the ROC AUC score using roc_auc_score.
  C. It converts the predicted probabilities to binary predictions (binary_preds) using a threshold of 0.5.
  D. It calculates precision and recall using precision_score and recall_score, respectively.
  E. It returns all three metrics.
- **train_model_on_gpu(model, train_loader, test_loader, epochs, device):**
  A. This function trains a given PyTorch model (model) on the provided training data (train_loader).
  B. model.train(): Sets the model to training mode (important for layers like dropout and batch normalization).
  C. model.to(device): Moves the model to the specified device (GPU or CPU).
  D. optimizer = optim.Adam(...): Initializes the Adam optimizer with a learning rate of 0.001. Adam is a commonly used and effective optimization algorithm.
  E. criterion = nn.BCELoss(): Sets the loss function to Binary Cross-Entropy Loss, which is appropriate for binary classification.
  F. history = ...: Initializes a dictionary to store training and testing metrics (loss, ROC-AUC, precision, recall) for each epoch.
  G. Training Loop:
    ➔ The code iterates over the specified number of epochs.
    ➔ Inside each epoch, it iterates over the batches of data provided by train_loader.

- ➔ x, y = x.to(device), y.to(device).float(): Moves the input data (x) and labels (y) to the specified device (GPU or CPU). The labels are also converted to float type, which is required by BCELoss.
- ➔ optimizer.zero_grad(): Clears the gradients from the previous iteration. This is essential in PyTorch.
- ➔ y_pred = model(x): Performs the forward pass (gets predictions from the model).
- ➔ y_pred = y_pred.squeeze(): Removes extra dimension.
- ➔ loss = criterion(y_pred, y): Calculates the loss.
- ➔ loss.backward(): Performs the backward pass (calculates gradients).
- ➔ optimizer.step(): Updates the model's parameters based on the gradients.
- ➔ The code keeps track of the loss, true labels, and predicted probabilities for each batch.
- ➔ After processing all batches, it calculates and prints the average loss, ROC-AUC, precision, and recall for the epoch.
- ➔ Test Evaluation (Optional): If a test_loader is provided, the code evaluates the model on the test data after each epoch.
    - ★ model.eval(): Sets the model to evaluation mode (disables dropout, etc.).
    - ★ with torch.no_grad():: Disables gradient calculation during testing (saves memory and computation). This is very important.
    - ★ The code iterates over the test data, performs a forward pass, and stores the predicted probabilities and true labels.
    - ★ It then calculates and prints the ROC-AUC, precision, and recall on the test set.
    - ★ model.train(): Sets the model to train mode.

- **train(models, train_loader, test_loader, epochs):**

A. This function takes a list of models (models), the training data loader, the optional test data loader, and the number of epochs.
B. It determines the device (GPU or CPU) to use.
C. It iterates over the provided models and calls train_model_on_gpu for each one.
D. It stores the training history returned by train_model_on_gpu in a dictionary (result_dict), using the model's class name as the key.
E. It returns the result_dict.

- **evaluate(models, test_loader, device):**
  A. This function takes a list of models and the testing data.
  B. Determines the device to be used.
  C. It iterates over the provided models and evaluates them on the test data.
  D. Sets the model for evaluation mode.
  E. Calculates and returns performance metrics, specifically ROC-AUC, precision, recall, and loss.
- **plot_metrics(performance_metrics, epochs):**
  A. This function takes the performance_metrics dictionary (returned by train) and the number of epochs.
  B. It uses matplotlib and seaborn to create plots of the training and testing metrics (loss, ROC-AUC, precision, recall) over time (epochs).
  C. It creates a separate plot for each metric.
  D. It plots the training and the testing metrics.
- **Model Instantiation and Training:**
  A. This section instantiates both the LTACNN and the kan_model.
  B. It then calls the train function to train the instantiated models and storing the training results.
  C. Finally, the plots for the metrics are shown by calling the plot_metrics function.

9.4 INTERPRETABILITY ANALYSIS

```
import torch

import shap

import lime.lime_tabular

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from scipy.stats import spearmanr, wilcoxon



# Assuming LTACNN and kan_model are already trained and available



models = {

    "LTACNN": LTACNN,

    "KAN": kan_model,

}



# Sample data for SHAP and LIME

num_samples = 1000
```

```
random_indices_train = torch.randperm(X_train.shape[0])[:num_samples]

random_indices_test = torch.randperm(X_test.shape[0])[:num_samples]



X_train_sample = X_train[random_indices_train].clone().detach().cpu()

X_test_sample = X_test[random_indices_test].clone().detach().cpu()



X_train_np = X_train_sample.numpy()

X_test_np = X_test_sample.numpy()



shap_values_dict = {}



for name, model in models.items():

    model.to("cpu").eval()  # Ensure model is on CPU and in eval mode



    explainer = shap.DeepExplainer(model, X_train_sample)

    shap_values = explainer.shap_values(X_test_sample, check_additivity=False)

    X_test_flattened = X_test_sample.reshape(X_test_sample.shape[0], -1)



    shap_values_flattened = np.array(shap_values).reshape(shap_values.shape[0], -1)
    shap_values_dict[name] = shap_values_flattened


    shap.summary_plot(shap_values_flattened, X_test_flattened.numpy(), show=False)

    plt.title(f"SHAP Summary Plot - {name}")

    plt.show()


shap_df = pd.DataFrame({

    "Feature": [f"F{i}" for i in range(X_test_np.shape[1])]

})


for name in models.keys():

    shap_df[name] = np.mean(np.abs(shap_values_dict[name]), axis=0)


print(shap_df.head())


# --- LIME ---

print("X_train_np shape:", X_train_np.shape)

num_features = X_train_np.shape[1]

feature_names = [f"F{i}" for i in range(num_features)]
```

```
print("Number of features:", num_features)

print("Feature names:", feature_names)

X_train_np = X_train_np.reshape(num_samples, -1)

print("X_train_np shape:", X_train_np.shape)



explainer = lime.lime_tabular.LimeTabularExplainer(

    X_train_np, feature_names=shap_df["Feature"].values, mode="regression"

)



lime_values_dict = {}

X_test_np = X_test_np.reshape(num_samples, -1)

for name, model in models.items():

    exp = explainer.explain_instance(

        X_test_np[0],

        lambda x: model(

            torch.tensor(x, dtype=torch.float32)

            .unsqueeze(-1)  # Keep channel dimension

        ).detach().cpu().numpy(),

        num_features=X_train_np.shape[1]
    )

    lime_values_dict[name] = dict(exp.as_list())


lime_df = pd.DataFrame(lime_values_dict).fillna(0)

lime_df["Feature"] = shap_df["Feature"]


print(lime_df.head())


# --- Dataframe Melting and Plotting ---

shap_melted = shap_df.melt(id_vars="Feature", var_name="Model", value_name="SHAP Value")

lime_melted = lime_df.melt(id_vars="Feature", var_name="Model", value_name="LIME Value")


lime_df = lime_df.reset_index().rename(columns={"index": "Rule"}) #Renamed

lime_melted_rules = lime_df.melt(id_vars="Rule", var_name="Model", value_name="LIME Value") #Renamed


plt.figure(figsize=(12, 6))

sns.barplot(x="Feature", y="SHAP Value", hue="Model", data=shap_melted)
```

```
plt.xticks(rotation=45)

plt.title("SHAP Feature Importance Across Models")

plt.show()



print("shap_df:")

print(shap_df.head())

print("lime_df:")

print(lime_df.head())

print("lime_melted:")

print(lime_melted.head())

print("lime_melted columns:", lime_melted.columns)

print("lime_melted shape:", lime_melted.shape)

plt.figure(figsize=(24, 6))

sns.barplot(x="Rule", y="LIME Value", hue="Model", data=lime_melted_rules)

plt.xticks(rotation=45)

plt.title("LIME Rule Importance Across Models")

plt.show()



# --- Correlation Analysis ---
```

```
import re

def extract_feature(rule):

    m = re.search(r'(F\d+)', rule)

    return m.group(1) if m else None



lime_df['Feature'] = lime_df['Rule'].apply(extract_feature)



print("lime_df after extracting 'Feature':")

print(lime_df.head())



numeric_cols = ['LTACNN', 'KAN']

lime_df_grouped = lime_df.groupby('Feature')[numeric_cols].mean().reset_index()



print("\nAggregated LIME DataFrame (lime_df_grouped):")

print(lime_df_grouped.head())



merged_df = shap_df.set_index("Feature").join(

    lime_df_grouped.set_index("Feature"), lsuffix="_SHAP", rsuffix="_LIME"

)
```

```
merged_df_numeric = merged_df.select_dtypes(include=["number"])


print("\nMerged DataFrame (numeric only):")

print(merged_df_numeric.head())


corr_matrix = merged_df_numeric.corr(method="spearman")


plt.figure(figsize=(8, 6))

sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Spearman Correlation of SHAP & LIME Across Models")

plt.xticks(rotation=45)

plt.yticks(rotation=0)

plt.show()

# --- Statistical Tests ---


for m1, m2 in [("LTACNN", "KAN")]:

   print(f"Wilcoxon Test (SHAP {m1} vs {m2}):", wilcoxon(shap_df[m1], shap_df[m2]))

   print(f"Wilcoxon Test (LIME {m1} vs {m2}):", wilcoxon(lime_df[m1], lime_df[m2]))
```

Description:

- **Model Dictionary:**

  A dictionary models is created to store the trained LTACNN and kan_model instances, keyed by their names. This makes it easy to iterate over the models for analysis.

- **Data Sampling:**
  - **A.** num_samples = 1000: 1000 samples are randomly selected from both the training (X_train) and testing (X_test) sets. This is done for computational efficiency, as SHAP and LIME can be expensive on large datasets.
  - **B.** torch.randperm(...): Generates random permutations of indices.
  - **C.** [:num_samples]: Selects the first num_samples indices.
  - **D.** X_train_sample, X_test_sample: Create the sampled subsets.
  - **E.** .clone().detach().cpu(): This is crucial.
    - ➔ .clone(): Creates a copy of the tensor. This is important because we'll be modifying the data (reshaping, converting to NumPy). We don't want to change the original X_train and X_test.
    - ➔ .detach(): Detaches the tensor from the computation graph. This prevents any unintended gradient calculations during the interpretability analysis.
    - ➔ .cpu(): Moves the tensor to the CPU. SHAP and LIME often work with NumPy arrays, which reside on the CPU.
- **SHAP Analysis:**
  - ➔ **Loop through Models:**
    - A. model.to("cpu").eval(): Moves the current model to the CPU and sets it to evaluation mode (important for consistency).
  - ➔ **SHAP Explainer:**

A. shap.DeepExplainer(model, X_train_sample): Creates a DeepExplainer object. DeepExplainer is a SHAP explainer specifically designed for deep learning models. It uses the training data (X_train_sample) to approximate the expected values of the model's output.
B. shap_values = explainer.shap_values(...): Calculates the SHAP values for the test samples (X_test_sample). SHAP values explain the contribution of each feature to the model's prediction for each individual instance. The check_additivity=False is added.
C. X_test_flattened: Reshape the data.
D. shap_values_flattened: Reshape the shap values also.
E. shap_values_dict[name] = shap_values_flattened: Stores the computed SHAP values in the shap_values_dict, keyed by the model name.
F. shap.summary_plot(...): Generates a SHAP summary plot. This plot visualizes the global feature importance (average absolute SHAP values) and the distribution of SHAP values for each feature. show=False prevents the plot from being displayed immediately (we'll display it later using plt.show()).

➔ **SHAP DataFrame:**

shap_df = pd.DataFrame(...): Creates a Pandas DataFrame (shap_df) to store the average absolute SHAP values for each feature. This makes it easier to compare feature importance across models.

- **LIME Analysis:**
  - ➔ Reshaping Data (Again):
  - ➔ LIME Explainer:
  - ➔ LIME Explanations:

A. The code iterates through the models dictionary (again).

B. exp = explainer.explain_instance(...): Generates a LIME explanation for the first test instance (X_test_np[0]). This is a key point: LIME explains individual predictions.

    - ❖ lambda x: ...: This is a lambda function (an anonymous function) that acts as a wrapper around the model's prediction. LIME needs a function that takes a NumPy array as input and returns the model's output. The lambda function handles the necessary conversions (NumPy array to PyTorch tensor, adding the channel dimension, moving to the CPU, and detaching from the computation graph).
    - ❖ num_features=X_train_np.shape[1]: Specifies the number of features to use in the explanation.
    - ❖ lime_values_dict[name] = dict(exp.as_list()): Converts the LIME explanation to a dictionary and stores it in lime_values_dict.

➔ LIME DataFrame:

lime_df = pd.DataFrame(...): Creates a Pandas DataFrame (lime_df) to store the LIME values.

- **Dataframe Melting and Plotting:**
    - ➔ Melting: The melt function is used to transform the shap_df and lime_df DataFrames from a "wide" format to a "long" format. This is necessary for creating bar plots with seaborn. The id_vars argument specifies the column(s) that should remain as identifiers, and the var_name and value_name arguments specify the names of the new columns that will be created.
    - ➔ Renaming: Rename the columns as required
    - ➔ Bar Plots: sns.barplot(...): Creates bar plots to visualize the SHAP and LIME values.
- **Correlation Analysis:**
    - ➔ Extracting the features.

- ➔ Data Aggregation: Calculate the mean of each model's LIME values across all features by grouping.
- ➔ DataFrame Joining: The SHAP DataFrame and aggregated LIME data are combined into a single merged_df for correlation analysis.
- ➔ Numeric Selection: Selects only numeric columns to prepare for correlation.
- ➔ Correlation Calculation: Computes the Spearman rank correlation matrix.
- ➔ Heatmap Visualization: Generates a heatmap visualizing the correlations, offering insights into how consistently SHAP and LIME identify important features across different models.
- ➔ Spearman Correlation: merged_df_numeric.corr(method="spearman"): Calculates the Spearman rank correlation matrix. Spearman correlation is used because it's less sensitive to outliers and non-linear relationships than Pearson correlation.
- ➔ Heatmap: sns.heatmap(...): Creates a heatmap to visualize the correlation matrix.

- **Statistical Tests:**

Wilcoxon Signed-Rank Test: wilcoxon(shap_df[m1], shap_df[m2]): Performs the Wilcoxon signed-rank test to compare the SHAP values between pairs of models. The Wilcoxon test is a non-parametric test that checks whether two related samples come from the same distribution. It's used here to see if the feature importances assigned by different models are significantly different. The same is done for the LIME values.

Chapter 10

References

1. C. Yu, Y. Xu, J. Cao, Y. Zhang, Y. Jin and M. Zhu, "Credit Card Fraud Detection Using Advanced Transformer Model," in 2024 IEEE International Conference on Metaverse Computing, Networking, and Applications (MetaCom), Hong Kong, China, 2024, pp. 343–350, doi: 10.1109/MetaCom62920.2024.00064.
2. Y. Tang and Z. Liu, "A Credit Card Fraud Detection Algorithm Based on SDT and Federated Learning," in IEEE Access, vol. 12, pp. 182547–182560, 2024, doi: 10.1109/ACCESS.2024.3491175.
3. P. Singh, K. Singla, P. Piyush and B. Chugh, "Anomaly Detection Classifiers for Detecting Credit Card Fraudulent Transactions," in 2024 Fourth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), Bhilai, India, 2024, pp. 1–6, doi: 10.1109/ICAECT60202.2024.10469194.
4. P. Kumari and S. Mittal, "Fraud Detection System for Financial System Using Machine Learning Techniques: A Review," in 2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2024, pp. 1–6, doi: 10.1109/ICRITO61523.2024.10522197.
5. A. Rawat, S. S. Aswal, S. Gupta, A. P. Singh, S. P. Singh and K. C. Purohit, "Performance Analysis of Algorithms for Credit Card Fraud Detection," in 2024 2nd International Conference on Disruptive Technologies (ICDT), Greater Noida, India, 2024, pp. 567–570, doi: 10.1109/ICDT61202.2024.10489771.
6. M. Thilagavathi, R. Saranyadevi, N. Vijayakumar, K. Selvi, L. Anitha and K. Sudharson, "AI-Driven Fraud Detection in Financial Transactions with Graph Neural Networks and Anomaly Detection," in 2024 International Conference on Science Technology Engineering and Management (ICSTEM), Coimbatore, India, 2024, pp. 1–6, doi: 10.1109/ICSTEM61137.2024.10560838.
7. V. R. Adhegaonkar, A. R. Thakur and N. Varghese, "Advancing Credit Card Fraud Detection Through Explainable Machine Learning Methods," in 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), Bengaluru, India, 2024, pp. 792–796, doi: 10.1109/IDCIoT59759.2024.10467999.
8. N. R. S. Jebaraj, J. Shekhawat and R. Gupta, "An Overview of Clustering Algorithms for Credit Card Fraud Detection," in 2024 International Conference on Optimization Computing and Wireless Communication (ICOCWC), Debre Tabor, Ethiopia, 2024, pp. 1–6, doi: 10.1109/ICOCWC60930.2024.10470724.

9. R. Raut, A. B. Chandanshive, P. N. Gadkar and E. Govardhan, "Credit Card Fraud Detection Using Ensemble Modeling," in 2024 OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 4.0, Raigarh, India, 2024, pp. 1–6, doi: 10.1109/OTCON60325.2024.10687633.
10. J. G. Sherwin Akshay, T. Vinusha, R. Sharon Bianca, C. K. Sarath Krishna and G. Radhika, "Enhancing Credit Card Fraud Detection with Deep Learning and Graph Neural Networks," in 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1–6, doi: 10.1109/ICCCNT61001.2024.10725042.
11. G. Bharath and P. S. Uma Priyadarsini, "A Novel Accuracy Analysis of Credit Card Fraud Detection Using ResNet50 Over Linear Regression," in 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1–5, doi: 10.1109/ICCCNT61001.2024.10725812.
12. S. Sukruth, M. Haripriya, S. Deepa, J. Jayapriya and M. Vinay, "Comparative Study on GANs and VAEs in Credit Card Fraud Detection," in 2024 First International Conference on Software, Systems and Information Technology (SSITCON), Tumkur, India, 2024, pp. 1–6, doi: 10.1109/SSITCON62437.2024.10796972.
13. V. Suganthi and J. Jebathangam, "A Novel Approach for Credit Card Fraud Detection using Gated Recurrent Unit (GRU) Networks," in 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Kirtipur, Nepal, 2024, pp. 1716–1721, doi: 10.1109/I-SMAC61858.2024.10714795.
14. T.-T.-H. Le, Y. Hwang, H. Kang and H. Kim, "Robust Credit Card Fraud Detection Based on Efficient Kolmogorov-Arnold Network Models," in IEEE Access, vol. 12, pp. 157006–157020, 2024, doi: 10.1109/ACCESS.2024.3485200.
15. S. Jhansi Ida, K. Balasubadra, S. R R and L. N. T, "Enhancing Credit Card Fraud Detection through LSTM-Based Sequential Analysis with Early Stopping," in 2024 2nd International Conference on Networking and Communications (ICNWC), Chennai, India, 2024, pp. 1–6, doi: 10.1109/ICNWC60771.2024.10537550.
16. S. A, N. V and S. S. Pandi, "A Novel Approach for Credit Card Fraud Detection Using Deep Learning Algorithms," in 2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT), Kollam, India, 2024, pp. 1870–1875, doi: 10.1109/ICCPCT61902.2024.10672824.
17. K. G. Dastidar, O. Caelen and M. Granitzer, "Machine Learning Methods for Credit Card Fraud Detection: A Survey," in IEEE Access, vol. 12, pp. 158939–158965, 2024, doi: 10.1109/ACCESS.2024.3487298.

18. E. Ileberi and Y. Sun, "A Hybrid Deep Learning Ensemble Model for Credit Card Fraud Detection," in IEEE Access, vol. 12, pp. 175829–175838, 2024, doi: 10.1109/ACCESS.2024.3502542.
19. I. D. Mienye and N. Jere, "Deep Learning for Credit Card Fraud Detection: A Review of Algorithms, Challenges, and Solutions," in IEEE Access, vol. 12, pp. 96893–96910, 2024, doi: 10.1109/ACCESS.2024.3426955.
20. Y. Xie, G. Liu, C. Yan, C. Jiang, M. Zhou and M. Li, "Learning Transactional Behavioral Representations for Credit Card Fraud Detection," in IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 4, pp. 5735–5748, Apr. 2024, doi: 10.1109/TNNLS.2022.3208967.
21. O. Imran and A. Yakoob, "Leveraging LSTM and Attention for High-Accuracy Credit Card Fraud Detection," in Fusion: Practice and Applications, 2025, pp. 209–220, doi: 10.54216/FPA.170115.
22. P. Pandey and K. K. Garg, "Credit Card Fraud Detection Using KNC, SVC, and Decision Tree Machine Learning Algorithms," in 2025 IEEE 4th International Conference on AI in Cybersecurity (ICAIC), Houston, TX, USA, 2025, pp. 1–3, doi: 10.1109/ICAIC63015.2025.10848573.
23. B. Dharma and D. Latha, "Fraud Detection in Credit Card Transactional Data Using Hybrid Machine Learning Algorithm," in 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI), Goathgaun, Nepal, 2025, pp. 213–218, doi: 10.1109/ICMCSI64620.2025.10883549.
24. F. Khaled Alarfaj and S. Shahzadi, "Enhancing Fraud Detection in Banking With Deep Learning: Graph Neural Networks and Autoencoders for Real-Time Credit Card Fraud Prevention," in IEEE Access, vol. 13, pp. 20633–20646, 2025, doi: 10.1109/ACCESS.2024.3466288.
25. N. Damanik and C. -M. Liu, "Advanced Fraud Detection: Leveraging K-SMOTEENN and Stacking Ensemble to Tackle Data Imbalance and Extract Insights," in IEEE Access, vol. 13, pp. 10356–10370, 2025, doi: 10.1109/ACCESS.2025.3528079.
26. A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention," in Proceedings of the 37th International Conference on Machine Learning (ICML), Online, PMLR 119, 2020, pp. 5650–5659, doi: 10.48550/arXiv.2006.16236.
27. Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, "KAN: Kolmogorov–Arnold Networks," in Proceedings of the International Conference on Learning Representations (ICLR), 2025, [Online]. Available: https://arxiv.org/abs/2404.19756.

28. I. Benchaji, S. Douzi, B. El Ouahidi, and J. Jaafari, "Enhanced credit card fraud detection based on attention mechanism and LSTM deep model," Journal of Big Data, vol. 8, article 151, 2021, doi: 10.1186/s40537-021-00541-8.
29. J. Forough and S. Momtazi, "Ensemble of deep sequential models for credit card fraud detection," Applied Soft Computing, vol. 99, p. 106883, 2021, doi: 10.1016/j.asoc.2020.106883.
30. A. R. Asha and S. K. Kumar, "Credit card fraud detection using artificial neural network," Global Transitions Proceedings, vol. 2, no. 1, pp. 35–41, 2021, doi: 10.1016/j.gltp.2021.01.006.