Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics

Classifiers Explainable in the Latent Space

Bachelor's Thesis

Alina Daroshka

Prague, May 2025

Study programme: Open Informatics
Specialization: Artificial Intelligence and Computer Science

Supervisor: prof. Ing. Václav Šmídl, Ph.D.
Supervisor Specialist: Ing. Michaela Mašková

Acknowledgments

Firstly, I would like to express my profound gratitude to my supervisor, prof. Ing. Václav Šmídl, Ph.D., for his expert guidance, constructive feedback, and steady support throughout the development of this thesis. His perspective and encouragement were invaluable at every stage of the work. I am also deeply thankful to my Supervisor Specialist, Ing. Michaela Mašková, for the time and patience she generously offered, and for her consistent support and engagement throughout this process. I am sincerely grateful for the opportunity to work under their guidance and to learn from their experience. Finally, I am truly thankful to my family for their unwavering support and belief in me, which made this journey possible.

The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Daroshka Alina** Personal ID number: **516443**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Classifiers Explainable in the Latent Space

Bachelor’s thesis title in Czech:

Klasifikátory vysvětlitelné v latentním prostoru

Guidelines:

Deep neural networks are commonly used to design classifiers due to their superb accuracy. However, in high-stakes decisions, such as financial crime detection, the user requires an explanation of the verdict, e.g., in the form of identifying which part of the sample was critical for the classifier. The aim of this work is to review and analyze existing approaches to explain deep neural networks at the level of their latent spaces.

1. Create a review of recent methods for explaining deep neural network classifiers, such as prototypes and concept activation vectors. Pay special attention to the extensibility of these approaches to complex input structures such as sequences or graphs.
2. Implement selected methods for vector spaces and demonstrate their properties on selected benchmark datasets, preferably from financial transaction data.
3. Extend the problem formulation to a more complex setup, e.g., using raw transaction history as input to the classifier. Validate suitability of the tested methods for the proposed extension.

Bibliography / sources:

[1] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. "Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1. 2018.

[2] Frank Willard, Luke Moffett, Emmanuel Mokel, Jon Donnelly, Stark Guo, Julia Yang, Giyoung Kim, Alina Jade Barnett, and Cynthia Rudin. "This looks better than that: Better interpretable models with protopnext." arXiv preprint arXiv:2406.14675 (2024).

[3] Patrick Schramowski, Wolfgang Stammer, Stefano Teso, Anna Brugger, Franziska Herbert, Xiaoting Shao, Hans-Georg Luigs, Anne-Katrin Mahlein, and Kristian Kersting. "Making deep neural networks right for the right scientific reasons by interacting with their explanations." Nature Machine Intelligence 2, no. 8 (2020): 476-486.

Name and workplace of bachelor’s thesis supervisor:

prof. Ing. Václav Šmídl, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **13.02.2025** Deadline for bachelor thesis submission: **23.05.2025**

Assignment valid until: **20.09.2026**

prof. Dr. Ing. Jan Kybic
Head of department’s signature

prof. Mgr. Petr Páta, Ph.D.
Vice-dean´s signature on behalf of the Dean

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

DECLARATION

I, the undersigned

Student's surname, given name(s): Daroshka Alina
Personal number: 516443
Programme name: Open Informatics

declare that I have elaborated the bachelor’s thesis entitled

Classifiers Explainable in the Latent Space

independently, and have cited all information sources used in accordance with the Methodological Instruction on the Observance of Ethical Principles in the Preparation of University Theses and with the Framework Rules for the Use of Artificial Intelligence at CTU for Academic and Pedagogical Purposes in Bachelor’s and Continuing Master’s Programmes.

I declare that I used artificial intelligence tools during the preparation and writing of this thesis. I verified the generated content. I hereby confirm that I am aware of the fact that I am fully responsible for the contents of the thesis.

In Prague on 20.05.2025

Alina Daroshka
....
student's signature

Abstract

This work investigates explainable artificial intelligence (XAI) techniques for financial fraud detection, with a focus on interpretability in models that rely on latent representations. While machine learning models can effectively detect subtle patterns in complex transaction data, their lack of transparency — particularly in learned latent spaces — poses challenges for practical and regulatory use.

The study explores two primary explanation strategies grounded in latent space. The first combines local surrogate models with latent neighborhood selection to generate fine-grained, instance-specific explanations. SHAP-based feature attributions are employed as a baseline to benchmark the interpretability of this approach. The second strategy adapts a prototype-based method, where predictions are explained through similarity to learned latent prototypes, which are later anchored to representative training examples for interpretability.

Experiments on a large-scale transaction dataset highlight distinct strengths and limitations of each method. While latent neighborhood models can capture nuanced local behavior, their performance depends heavily on the structure of the latent space and the quality of selected neighbors. Prototype-based explanations, by contrast, offer consistent and self-contained justifications, though they may lack explanatory depth without further semantic structuring. The findings underscore the potential of latent-space interpretability techniques and point toward future improvements through refined latent structuring and expert-in-the-loop guidance.

Keywords Anomaly detection, Fraud detection, Interpretability, ProtoPNet, Triplet loss, Explainable AI, Financial fraud, Latent space, Autoencoder, SHAP, Lasso

Abstrakt

Tato práce se zabývá technikami vysvětlitelné umělé inteligence (XAI) pro detekci finančních podvodů, se zaměřením na interpretovatelnost modelů, které využívají latentní reprezentace. Ačkoli modely strojového učení dokážou účinně detekovat jemné vzory v komplexních transakčních datech, jejich netransparentnost — zejména v naučeném latentním prostoru — představuje překážku pro praktické využití i splnění regulatorních požadavků.

Studie zkoumá dva hlavní přístupy k vysvětlování, založené na latentním prostoru. První přístup kombinuje lokální náhradní modely s výběrem sousedství v latentním prostoru, aby generoval detailní, na instanci zaměřená vysvětlení. Jako referenční metoda pro srovnání interpretovatelnosti je využito přiřazování důležitosti vstupních atributů pomocí metody SHAP. Druhý přístup adaptuje metodu založenou na prototypech, kde jsou predikce vysvětlovány na základě podobnosti s naučenými latentními prototypy, které jsou následně přiřazeny k reprezentativním trénovacím příkladům pro zajištění interpretace.

Experimenty na rozsáhlém datovém souboru finančních transakcí ukazují specifické výhody i omezení obou přístupů. Modely s latentním sousedstvím dokáží zachytit jemné lokální chování, avšak jejich účinnost výrazně závisí na struktuře latentního prostoru a kvalitě zvolených sousedů. Naproti tomu vysvětlení založená na prototypech poskytují konzistentní a srozumitelná odůvodnění, i když mohou postrádat větší hloubku bez dodatečné sémantické strukturalizace. Výsledky poukazují na potenciál interpretačních technik v latentním prostoru a naznačují možnosti dalšího zlepšení prostřednictvím cílené strukturalizace latentního prostoru a zapojení odborníků do procesu vysvětlování.

Klíčová slova Detekce anomálií, Detekce podvodů, Interpretovatelnost, ProtoPNet, Triplet loss, Vysvětlitelná umělá inteligence, Finanční podvody, Latentní prostor, Autoenkodér, SHAP, Lasso

Contents

1 Introduction

Fraud detection is a critical issue in nancial transactions, impacting both consumers and nancial institutions. According to the Association of Certi ed Fraud Examiners, organizations lose an estimated 5% of their revenue to fraud annually, amounting to trillions of dollars globally [1]. These activities not only lead to substantial nancial losses but also undermine the trust in nancial systems. Fraud can manifest in various forms, such as identity theft, account takeovers and payment fraud. As technology evolves, so do the tactics employed by fraudsters. Detecting and preventing fraud on a large scale has become an increasingly complex challenge due to the sheer volume and diversity of nancial transactions that must be monitored in real time.

Traditionally, fraud detection has relied on rule-based systems, manual intervention, and domain expert knowledge [2]. While e ective in well-understood scenarios, these approaches often require constant updates by domain experts | making them costly to maintain | and tend to struggle with adapting to new or evolving fraud patterns.

In contrast, machine learning (ML) models can automatically learn patterns and anomalies from large volumes of historical transaction data, enabling them to detect previously unseen or subtle forms of fraud [3]. These models, once trained, can operate at scale and adapt more readily to changing behaviors without the need for exhaustive manual reprogramming. However, this exibility comes with its own challenges | most notably, the lack of interpretability in many high-performing ML models. These models, especially deep learning architectures, operate as black boxes | providing little insight into how predictions are made. In sensitive domains like nance, this lack of transparency raises concerns around accountability, trust, and regulatory compliance [4], [5]. For example, regulators may require justi cation for model-based decisions, and nancial analysts must be able to trust and understand alerts produced by AI systems.

To address these issues, the eld of explainable arti cial intelligence (XAI) has emerged, aiming to make complex models more transparent and interpretable. Most existing approaches focus on explanations in the input space | such as identifying important features using methods like SHAP or LIME. While these methods have been used in fraud detection [6], [7], explanations based solely on input-space features may fail to capture complex interactions or abstract concepts learned by the model [8]{[10]. Methods using latent space representation overcome this problem by searching for explanations on the latent space [9]. The latent space is learned by a neural network, forming an internal compressed representation of the data, which is often more abstract but also more structured.

The aim of this thesis is to investigate the suitability of various explanation approaches of the classi ers for nancial fraud detection. Speci cally, we explore and compare the following strategies:

- SHAP-based explanations as a baseline approach, providing local feature attributions in input space.
- Latent-space neighborhood explanations , inspired by local surrogate methods (e.g., LIME), where explanations are generated by tting interpretable models around a neighborhood in latent space.
- Prototype-based explanations , adapted from ProtoPNet, which uses prototypical examples in the latent space to support decisions, allowing fraud cases to be interpreted

as similar to representative anomaly types.

These methods are implemented and evaluated on a large-scale nancial transaction dataset. The work aims to assess their e ectiveness in both detection and interpretability, and to contribute to the growing eld of explainable machine learning for critical applications.

2 Theory of Machine Learning

The task of fraud detection can be naturally formulated as a classification problem in machine learning. In this formulation, each financial transaction is represented by a feature vector and assigned a label indicating whether it is fraudulent or legitimate. Detecting fraudulent transactions involves identifying patterns that separate legitimate behavior from suspicious activity. Machine learning provides a data-driven way to approach this task by learning these patterns from past examples, rather than relying only on manually defined rules.

2.1 Classification and Learning

Classification is a fundamental task in machine learning where the goal is to assign each input sample to one of a finite set of predefined categories, known as classes [11]. Formally, let the input space be denoted by $X \subseteq \mathbb{R}^d$, where each $x \in X$ is a d-dimensional feature vector, and let the output space be $Y = \{1, 2, \ldots, K\}$, where K is the number of classes.

A classification model, or classifier, is a function

$$f : X \to Y$$

that maps each input x to a predicted label $y = f(x)$.

2.1.1 Types of Learning

Depending on the availability and structure of labeled data, different learning paradigms can be applied to the classification task:

- Supervised Learning: In supervised classification, the training dataset consists of input-label pairs (x_i, y_i), where the correct class label y_i is known for each sample. The model learns to generalize from these examples to predict labels for new, unseen data. This is the primary setup used when the anomaly type labels are available.
- Unsupervised Learning: In unsupervised learning, no labels are provided. Instead, the goal is often to discover patterns or clusters in the data. This approach is commonly used in anomaly detection, where rare or unusual patterns are identified without labeled examples. In the context of this thesis, unsupervised learning becomes relevant when analyzing the latent space structure of the model, especially to see if meaningful groupings emerge without using labels during training.
- Semi-Supervised Learning: This setting lies between supervised and unsupervised learning. A small subset of the data is labeled, while the rest remains unlabeled. Semi-supervised learning aims to leverage the structure of the unlabeled data to improve classification performance. In this thesis, semi-supervised techniques were used to guide prototype learning with minimal labeled examples for each anomaly type.

2.1.2 Types of Classification

Classification problems can be categorized based on the number of possible output labels:

- **Binary Classification:** The label space is $Y = \{0, 1\}$ or $\{-1, 1\}$, representing two mutually exclusive classes. Fraud detection typically falls into this category.
- **Multiclass Classification:** The label space includes more than two distinct categories ($K > 2$). In some fraud detection scenarios, including the one considered in this thesis, different types of fraud are labeled separately. This requires the model not only to identify fraudulent activity but also to classify it into one of several predefined fraud categories.
- **Multilabel Classification:** Each input x may belong to multiple classes simultaneously. This setting is not considered in this thesis but is common in tasks such as document tagging or image annotation.

2.1.3 Imbalanced Classification

Fraud detection is a typical example of an imbalanced classification problem. In most real-world financial datasets, legitimate transactions vastly outnumber fraudulent ones. This means that the distribution of classes is highly skewed, with the minority class — fraud — representing only a small fraction of the data [12]. Such imbalance poses significant challenges for training and evaluating machine learning models. Without careful handling, a classifier may appear to perform well while completely failing to detect fraud. Two primary issues arise in this context:

- **Misleading Metrics:** Traditional evaluation metrics like overall accuracy can be deceptive. For instance, in a dataset where 99.5% of transactions are legitimate, a model that predicts every transaction as non-fraudulent would achieve 99.5% accuracy — while detecting no fraud at all. This creates a false impression of effectiveness and fails to capture the model's inability to identify the critical minority class.
- **Bias in Learning:** Models like neural networks and decision trees may prioritize the majority class during training, leading to poor generalization on the minority class. These models often optimize for overall error minimization, which skews the learning process toward the majority class (e.g. decision trees may split early based on majority class features).

To address these challenges, various strategies are commonly used. These include resampling techniques (such as oversampling the minority class or undersampling the majority), class-weighted loss functions that penalize misclassification of the minority class more heavily, and the use of evaluation metrics tailored to imbalanced problems, such as precision, recall, F1-score, and the area under the precision-recall curve (AUPRC) [13]. These techniques are discussed in more detail in later chapters in the context of model training and evaluation.

2.2 Neural Networks for Classification

Neural networks are a widely used class of machine learning models designed to approximate complex, non-linear functions. In classification tasks, neural networks aim to map input features to class probabilities or labels, making them suitable for problems involving high-dimensional or non-linearly separable data.

2.2.1 Basic Structure of Feedforward Neural Networks

A feedforward neural network consists of an input layer, one or more hidden layers, and an output layer. Each layer contains units called neurons. A neuron computes a weighted sum of its inputs and applies a non-linear activation function to produce an output.

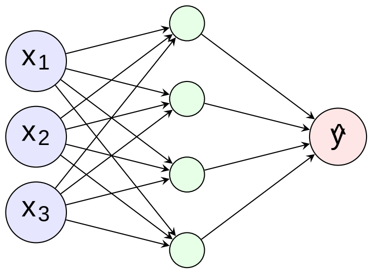


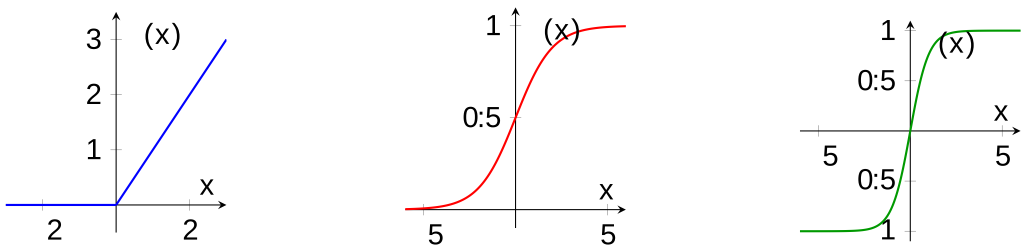
Figure 2.1: Simple feedforward neural network with one hidden layer. Each neuron computes a weighted sum of its inputs followed by a non-linear activation.

Given an input vector $x \in \mathbb{R}^d$, the transformation in a single layer is defined as:

$$z = Wx + b; \quad a = \sigma(z);$$

where W is the weight matrix, b is the bias vector, and σ is a non-linear activation function. Common activation functions include:

- ReLU (Rectified Linear Unit): $\sigma(z) = \max(0; z)$
- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- Tanh: $\sigma(z) = \tanh(z)$



(a) ReLU (b) Sigmoid (c) Tanh

Figure 2.2: Common activation functions used in neural networks.

The final output layer in a multiclass classification network typically uses the softmax activation function to convert the network outputs into class probabilities:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}};$$

where K is the number of classes.

This transformation ensures that each output value lies between 0 and 1, and the sum of all output values equals 1, making them interpretable as probabilities, which makes the network's predictions easier to understand and analyze.

2.2.2 Multilayer Perceptrons (MLPs)

A multilayer perceptron (MLP) [14] is a type of feedforward neural network composed of multiple layers of fully connected neurons. Each neuron in one layer is connected to every neuron in the next layer. MLPs are among the most commonly used neural architectures for classification tasks, especially on tabular data, due to their simplicity, flexibility, and general applicability.

MLPs are capable of approximating arbitrary functions, given enough hidden units and non-linear activations. This makes them effective for modeling complex, non-linear relationships between features and labels.

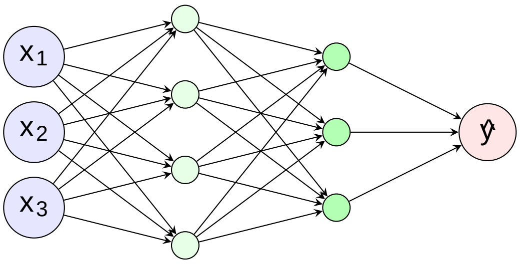


Figure 2.3: A simple multilayer perceptron (MLP) with two hidden layers.

Despite their strengths, MLPs also have limitations. They do not naturally account for spatial, sequential, or relational structure in the input. For example, they treat all features independently and are unaware of temporal order unless such information is explicitly included in the feature engineering process. As a result, they are not ideal for tasks like time series forecasting, natural language processing, or graph-based reasoning without adaptation.

Nonetheless, due to their general-purpose nature and ease of implementation, MLPs are frequently used as baseline models and are integral components in many larger neural architectures. In this thesis, MLPs are used both for direct classification and as downstream classifiers operating on representations produced by autoencoders.

2.2.3 Forward Pass and Backpropagation

The training of a neural network involves two key computational steps: the forward pass and the backward pass (backpropagation).

Forward Pass

In the forward pass, an input vector x is passed through the network layer by layer. Each layer performs a linear transformation followed by a non-linear activation function. The final layer produces class scores (logits), which are typically converted into probabilities via the softmax function. The forward pass computes the model's prediction $\hat{y}$ given current weights and biases.

For a network with L layers, the activations $a^{(l)}$ and linear transformations $z^{(l)}$ at each layer l are computed as:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}; \quad a^{(l)} = \sigma(z^{(l)})$$

for $l = 1, 2, \ldots, L$, with $a^{(0)} = x$ as the input.

Loss Function

Once the predicted output $\hat{y}$ is obtained (typically the result of a softmax activation), it is compared to the true label y using a loss function. In classification tasks, the cross-entropy loss is commonly used, defined as:

$$L(y, \hat{y}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i)$$

where y is the true label (one-hot encoded) and $\hat{y}_i$ is the predicted probability for class i. The loss function measures how far the predicted distribution is from the true distribution.

Backpropagation

To train the network, the model parameters $\{W^{(l)}, b^{(l)}\}$ are updated to minimize the loss. This is done using gradient-based optimization, such as stochastic gradient descent (SGD) or Adam[15]. Backpropagation is the algorithm used to compute the gradients of the loss with respect to each parameter in the network.

Backpropagation efficiently applies the chain rule of calculus to propagate the error signal backward through the network. At each layer, it computes:

$$\delta^{(l)} = \frac{\partial L}{\partial z^{(l)}} \tag{2.1}$$

which can then be used to compute gradients of the weights and biases:

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} (a^{(l-1)})^{\top}, \quad \frac{\partial L}{\partial b^{(l)}} = \delta^{(l)} \tag{2.2}$$

These gradients are used to update the parameters using:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}}, \quad b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}} \tag{2.3}$$

where η is the learning rate.

Cross-Entropy with Class Weights

In the case of imbalanced datasets, where some classes are significantly underrepresented, standard cross-entropy loss can lead to biased models that favor the majority class. To mitigate this, class weights can be introduced to penalize misclassification of minority classes more heavily. The weighted cross-entropy loss is defined as:

$$L_{\text{weighted}}(y, \hat{y}) = -\sum_{i=1}^{K} w_i \cdot y_i \log(\hat{y}_i),$$

where w_i is a positive weight assigned to class i, typically chosen inversely proportional to class frequency. This adjustment increases the contribution of rare classes to the total loss, encouraging the model to pay more attention to them during training. Weighted loss functions are commonly used in fraud detection and other anomaly detection tasks where class imbalance is significant[13], [16].

2.3 Evaluation

The performance of the classi cation models is assessed using several standard evaluation metrics, including the confusion matrix, precision, recall, F1-score, Receiver Operating Characteristic (ROC) curve, and Precision{Recall Curve (PRC).

In the context of imbalanced datasets, such as the one used in this thesis, the choice of evaluation metrics is particularly important. Metrics like accuracy may be misleading, as they can be dominated by the majority class. Therefore, greater emphasis is placed on metrics that more accurately re ect performance on the minority (anomalous) classes.

2.3.1 Confusion Matrix

Confusion matrix is a table used to evaluate the performance of a classi cation algorithm. It compares the predicted labels with the actual labels to show the number of correct and incorrect predictions made by the model. It is particularly useful for understanding the types of errors a classi er is making.

A confusion matrix for a binary classi cation problem is a 2x2 table that takes the following form:

$$\begin{matrix} TP & FP \\ FN & TN \end{matrix}$$

Each entry is de ned as follows:

- TP (True Positives): Number of instances correctly predicted as the positive class (e.g., fraudulent transactions that were correctly identi ed as fraud).
- FP (False Positives): Number of instances incorrectly predicted as the positive class (e.g., legitimate transactions wrongly agged as fraud).
- FN (False Negatives): Number of instances incorrectly predicted as the negative class (e.g., fraudulent transactions that the model missed and labeled as legitimate).
- TN (True Negatives): Number of instances correctly predicted as the negative class (e.g., legitimate transactions correctly identi ed as non-fraud).

Confusion matrix provides the basis for several important performance metrics, such as precision, recall, and F1-score, which are particularly informative in contexts with class imbalance.

2.3.2 Accuracy

Accuracy is de ned as the ratio of correctly identi ed occurrences comparative to the overall number of instances. The formula for quantifying binary accuracy is:

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

Despite being the most widely used metric, it is extremely misleading when it comes to highly imbalanced dataset like fraud detection in transactional data. The ratio of fraudulent transactions is usually less than 1 percent of the total data. Therefore, even if the model misclassi es all fraudulent cases, it will still achieve high accuracy of approximately 99%.

2.3.3 Precision

Precision is the proportion of all the model's positive classi cations that are actually positive.

$$\text{Precision} = \frac{\text{Correctly Classi ed Actual Positives}}{\text{Everything Classi ed as Positive}} = \frac{TP}{TP + FP}$$

In the fraud detection example, precision measures the fraction of transactions classi ed as fraud that were actually fraud.

2.3.4 Recall

Recall measures the model's ability to accurately predict the positive class. Intuitively, it re ects how e ectively the model identi es all positive instances in the dataset.

$$\text{Recall} = \frac{\text{Correctly Classi ed Actual Positives}}{\text{Total Actual Positives}} = \frac{TP}{TP + FN}$$

A high recall value means that the model successfully identi es most of the fraudulent transactions, which is crucial in fraud detection to avoid missing potential fraudulent activity.

2.3.5 F1-score

The F1 score is the harmonic mean of the precision and recall. It is used to evaluate the balance between these two metrics in a classi cation model.

$$F1 = 2 \quad \frac{\text{Precision} \quad \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.3.6 ROC Curve and AUC (Area Under Curve)

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a classi cation model's performance across di erent decision thresholds. In probabilistic classi- ers, the model outputs a con dence score (typically between 0 and 1), and a prediction is considered positive if the score exceeds a chosen threshold (e.g., 0.5 by default). By varying this threshold, one can observe how the trade-o between the true positive rate (TPR) and the false positive rate (FPR) evolves. The ROC curve plots TPR on the y-axis and FPR on the x-axis, where TPR is equivalent to recall and FPR is de ned as:

$$\text{FPR} = \frac{FP}{FP + TN}$$

A classi er that achieves a curve closer to the top-left corner of the plot is considered to perform better.

The Area Under the Curve (AUC) is a single number summarizing the overall shape of the ROC curve. It ranges from 0 to 1. AUC close to 1 indicates perfect classi cation | the model correctly ranks all positive instances above all negative ones. AUC around 0.5 indicates that the model is making random predictions.

While the ROC curve is widely used, it can be misleading in the case of highly imbalanced datasets, such as in fraud detection. This is because the False Positive Rate may appear low even when the absolute number of false positives is high, simply due to the large number of true negatives. As a result, the ROC curve might suggest good performance even if the model fails to adequately detect fraud. In such cases, precision-recall curves and metrics like the Area Under the Precision-Recall Curve (AUPRC) often provide more meaningful insights.

2.3.7 Precision-Recall Curve

The Precision-Recall (PR) curve is an alternative to the ROC curve, particularly useful for evaluating models on imbalanced datasets. It plots precision on the y-axis against recall on the x-axis, across varying classification thresholds.

The PR curve is especially valuable when false positives and false negatives carry different costs, as it emphasizes the model's performance on the positive class. Unlike the ROC curve — which can appear overly optimistic in imbalanced settings due to the abundance of true negatives — the PR curve provides a more informative view by focusing on the proportion of true positives among all predicted positives. This makes it especially relevant for fraud detection, where the minority class (fraudulent transactions) is of primary interest.

2.4 Autoencoders and Latent Representations

Autoencoders are unsupervised neural network architectures used for learning compressed representations of input data [17]. The main goal of an autoencoder is to map high-dimensional input data into a lower-dimensional latent space and then reconstruct the original input from that compressed representation. This process allows the network to learn meaningful internal features, which can be useful for tasks such as dimensionality reduction, anomaly detection, or representation learning.

2.4.1 Autoencoder Architecture

An autoencoder consists of two main components:

- Encoder: A neural network that transforms the input vector $x \in \mathbb{R}^d$ into a lower-dimensional representation $z \in \mathbb{R}^k$ where $k < d$. The encoder learns to capture the most relevant information about the input.
- Decoder: A neural network that attempts to reconstruct the original input x from the latent vector z. The decoder produces $\hat{x} \in \mathbb{R}^d$, an approximation of the original input.

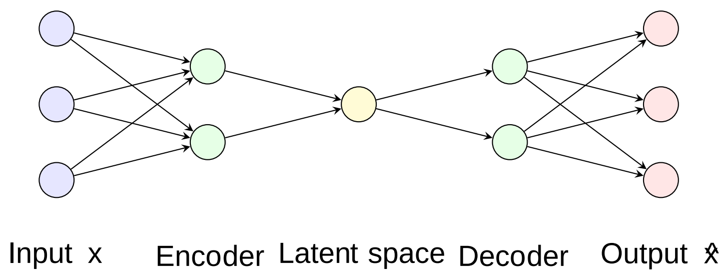


Figure 2.4: Autoencoder architecture. The encoder compresses the input x into a latent representation z, which the decoder uses to reconstruct the original input as $\hat{x}$.

The model is trained to minimize the reconstruction loss between the input x and its reconstruction $g(f(x))$, where f is the encoder and g is the decoder:

$$L(x, g(f(x)))$$

Here, L is a loss function that penalizes dissimilarity between the input and its reconstruction. A common choice is the mean squared error (MSE):

$$L_{\mathrm{MSE}}(x, \hat{x}) = \frac{1}{d}\sum_{i=1}^{d}(x_i - \hat{x}_i)^2$$

In this thesis, MSE is used as the reconstruction loss. It is particularly suitable for continuous, real-valued inputs and emphasizes larger errors more heavily due to squaring.

Depending on the nature of the input data, other reconstruction losses may be more appropriate. For instance, mean absolute error (MAE) may be preferred when robustness to outliers is desired, as it penalizes large deviations linearly. For binary or normalized inputs, binary cross-entropy is commonly used, especially when the output activation is sigmoid.

2.4.2 Using Latent Space for Classification

The latent space z produced by the encoder serves as a lower-dimensional representation of the input data. It is designed to capture the most salient features necessary for reconstructing the input while filtering out noise or redundancy [17].

In this thesis, the encoder output is not only used for reconstruction, but also as the input to a downstream classifier (e.g., MLP or prototype-based model). This setup enables classification to operate in a compressed, semantically rich space and facilitates the application of explainability techniques that analyze the internal structure of that latent space.

Formally, let $f_{\mathrm{enc}} : \mathbb{R}^d \to \mathbb{R}^k$ denote the encoder, and let $f_{\mathrm{clf}} : \mathbb{R}^k \to \mathbb{R}^C$ be the classifier mapping the latent space to C class logits. The overall classification model is thus:

$$\hat{y} = f_{\mathrm{clf}}(f_{\mathrm{enc}}(x))$$

To jointly learn useful representations and enable accurate classification, the model is trained by minimizing a combined loss:

$$L_{\mathrm{total}} = \lambda_{\mathrm{rec}} L_{\mathrm{rec}} + \lambda_{\mathrm{clf}} L_{\mathrm{clf}},$$

where L_{rec} is the reconstruction loss, and L_{clf} is the classification loss. The weighting factors λ_{rec} and λ_{clf} control the trade-off between preserving input fidelity and optimizing predictive performance.

This approach combines representation learning and classification, allowing the encoder to learn important features and produce a compact, structured and denoised input space for downstream tasks.

2.4.3 Dimensionality Reduction for Visualization

The latent space learned by the encoder is often lower-dimensional than the original input space, but still too high-dimensional to interpret directly. To analyze and visualize the internal structure of this space, dimensionality reduction techniques can be used to project the latent vectors into two or three dimensions.

Two commonly used methods for this purpose are:

- t-SNE (t-distributed Stochastic Neighbor Embedding)[18]: A non-linear technique that preserves local structure by modeling pairwise similarities between samples. It is e ective for visualizing clusters but can distort global geometry.
- UMAP (Uniform Manifold Approximation and Projection)[19]: Another non-linear method that preserves both local and global structure better than t-SNE in many cases. UMAP is also typically faster and more scalable.

To illustrate the behavior of these methods, Figure 2.5 shows a side-by-side application of t-SNE and UMAP on the MNIST dataset, which consists of images of ten handwritten digits. Both methods reveal distinct clusters for each digit, but their layouts di er. t-SNE creates tight clusters that are well-separated, but the overall structure looks somewhat scattered, with no clear logic in how groups are arranged relative to each other. In contrast, UMAP produces a more organized and structured layout: the digit groups are still distinct, but they are positioned in a way that suggests relationships between them | for example, similar digits (9, 7, 4) tend to appear closer together. UMAP also appears to show clearer separation between some of the digit classes, with less overlap between clusters.

Figure 2.5: Comparison of t-SNE (left) and UMAP (right) applied to the MNIST dataset. Each point corresponds to a digit, colored by class label.

In this thesis, both t-SNE and UMAP are applied to the latent vectors to visualize how di erent types of transactions, including anomalies, are arranged in the learned space. These visualizations help assess whether meaningful patterns or clusters are formed and whether anomalies occupy distinguishable regions.

2.5 Post-hoc Explanation Methods

Post-hoc explanation methods aim to interpret the behavior of trained machine learning models after training has been completed. These methods are especially important for complex models such as neural networks, which are often considered "black boxes" due to their high dimensionality and non-linear decision boundaries. This section presents techniques that explain model predictions in terms of input features using local surrogate or attribution-based methods.

2.5.1 Motivation for Explainability

In many real-world applications, especially those involving high-stakes decision-making, interpretability is as important as predictive performance. This is particularly true in the domain of financial fraud detection, where automated decisions can have legal, financial, and ethical consequences. Stakeholders — including analysts, auditors, and regulators — must be able to understand and justify the model's decisions.

For example, if a model flags a transaction as potentially fraudulent, it is often necessary to answer questions such as:

- Which features were most influential in the model's decision?
- Does the decision align with known patterns of fraudulent behavior?
- Is the same reasoning applied consistently across similar cases?

Answering such questions requires interpretability methods that go beyond prediction accuracy and provide insight into the model's internal reasoning. In addition to helping humans understand individual decisions, explainability can also be used to audit models for potential biases. For example, if a model disproportionately flags transactions associated with certain demographic groups or behaviors, XAI methods can help identify whether the model is making decisions based on irrelevant or discriminatory patterns. This concern has been highlighted in other high-stakes domains, such as criminal justice, where a lack of transparency in predictive models has led to fairness and accountability issues [20]. Interpretability thus serves not only as a tool for understanding but also for promoting fairness and trust in automated systems.

2.5.2 Local Interpretable Model-agnostic Explanations (LIME)

LIME is a post-hoc explanation technique introduced in the paper “Why Should I Trust You?” by Ribeiro et al. [21] that aims to explain the predictions of any black-box classifier by approximating it locally with an interpretable surrogate model. Although a model may be complex globally, LIME assumes that its behavior in the vicinity of a specific input instance can be approximated by a simpler model, such as a sparse linear model.

Formally, given a complex classifier $f : \mathcal{X} \to \mathbb{R}$, and a specific input sample $x \in \mathcal{X}$, LIME constructs an explanation model $g \in G$ (e.g., a linear model or decision tree) that approximates f in a local neighborhood of x. The explanation is obtained by solving the following optimization problem:

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g),$$

where:

- $\mathcal{L}(f, g, \pi_x)$ is a loss function measuring how closely the surrogate model g approximates f near x, weighted by a proximity function π_x,
- $\Omega(g)$ is a regularization term that encourages interpretability (e.g., sparsity in the number of non-zero coefficients),
- $\pi_x(z)$ defines the locality around x, typically using an exponential kernel over some distance metric (e.g., cosine or Euclidean distance).

To fit g, LIME generates a set of perturbed samples around x, queries the black-box model f to obtain predictions for those samples, and then trains g to approximate f in the local region. The resulting explanation consists of the feature weights of g, which describe each input feature's contribution to the prediction at x.

LIME is widely used due to its simplicity and flexibility across different model architectures and data formats [22], [23]. However, as it focuses only on local fidelity, it may produce inconsistent explanations if the global model behavior is highly non-linear or irregular [10].

2.5.3 SHapley Additive exPlanations (SHAP)

SHAP [24] is a post-hoc explanation framework that assigns each input feature an importance value for a given prediction. It is grounded in cooperative game theory and builds on the concept of Shapley values, which provide a theoretically justified method for attributing a model's output to its input features.

SHAP interprets a model's prediction as a game, where each input feature acts as a "player" contributing to the final outcome. The goal is to fairly distribute the model's prediction among the input features, based on their individual contributions. This is formalized using Shapley values from cooperative game theory.

Given a model f and input $x \in \mathbb{R}^d$, the Shapley value ϕ_i for feature i is defined as:

$$\phi_i(f;x) = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S) \right],$$

where

- F is the set of all input features,
- S is any subset of features that does not include i,
- $f_S(x_S)$ is the model's output when only the features in S are present,
- $f_{S\cup\{i\}}(x_{S\cup\{i\}})$ is the output when feature i is added to that subset.

The expression inside the brackets represents the marginal contribution of feature i to subset S, and the sum averages this over all possible subsets.

Although exact Shapley values are computationally expensive to calculate, SHAP has efficient approximations and retains key theoretical properties. These include:

- Local accuracy (additivity): The explanation model matches the original model prediction for each input:

  $$f(x) = \phi_0 + \sum_{i=1}^{d} \phi_i,$$

  where ϕ_0 is the expected value of the model over the dataset.
- Consistency: If a model changes such that the marginal contribution of a feature increases (for all subsets), its Shapley value will not decrease.
- Missingness: Features not present in a model input get zero attribution.

These theoretical guarantees make SHAP values uniquely reliable among feature attribution methods.

Kernel SHAP. Kernel SHAP is a model-agnostic method for approximating SHAP values, combining ideas from LIME and Shapley values. Instead of evaluating the exact Shapley formula, which requires computing contributions from all 2^d subsets (where d is the number of features), Kernel SHAP samples a manageable number of subsets and uses their evaluations to estimate the Shapley values through weighted linear regression.

The approach operates on simplified binary input vectors $z' \in \{0, 1\}^d$, which indicate the presence or absence of features. A local surrogate model is trained to match the original model's behavior over these perturbed samples. The key insight is that by using a specific kernel to weight the importance of each sampled subset, the resulting regression coefficients correspond to the SHAP values.

The Shapley kernel used to weight each sample is:

$$\pi_x(z') = \frac{(d-1)}{\binom{d}{|z'|}\,|z'|\,(d-|z'|)},$$

where $|z'|$ is the number of present features in the perturbed input z'. This kernel ensures that the regression solution uniquely satisfies the Shapley value axioms (local accuracy, consistency, and missingness).

In the SHAP paper [24], the authors prove that this specific kernel, combined with a locally weighted linear model, is the only formulation that recovers Shapley values as additive feature attributions for any model f.

2.5.4 Sparse Linear Models (Lasso)

Sparse linear models are a simple but powerful approach for interpreting high-dimensional data, especially when only a subset of features is expected to be relevant for each prediction. The Lasso (Least Absolute Shrinkage and Selection Operator) [25] is a linear regression model that includes an ℓ_1-penalty term to encourage sparsity in the feature coefficients.

The Lasso optimization problem is defined as:

$$\hat{\beta} = \arg\min_{\beta} \; \|y - X\beta\|_2^2 + \lambda\|\beta\|_1 \;,$$

where:

- $X \in \mathbb{R}^{n \times d}$ is the feature matrix,
- $y \in \mathbb{R}^n$ is the target vector (e.g., black-box predictions),
- $\beta \in \mathbb{R}^d$ is the vector of model coefficients,
- $\lambda \geq 0$ controls the degree of sparsity.

This objective is equivalent to minimizing the average squared error plus an ℓ_1-penalty, a formulation commonly used in practical implementations. The ℓ_1 penalty encourages many coefficients in β to become exactly zero, which makes the resulting model more interpretable and resistant to overfitting.

2.6 Prototype-based Methods

Prototype-based methods offer an interpretable approach to machine learning by associating predictions with representative examples. Unlike black-box models that rely on abstract, high-dimensional decision boundaries, these models make decisions by comparing new inputs to a set of learned reference points called prototypes. These prototypes are not selected directly from the training data but are learned in the latent space and later projected onto the closest actual training instances to enhance interpretability.

A prominent example of this approach is ProtoPNet, introduced by Chen et al. in the paper "This Looks Like That: Deep Learning for Interpretable Image Recognition" [9]. The method enhances interpretability in image classification by learning a set of representative image patches, which act as prototypes for different classes. During inference, an input image is classified based on how similar its features are to these prototypes. The method promotes interpretability by explicitly linking predictions to real training examples, making it possible to explain why a given input belongs to a particular class.

The original ProtoPNet architecture involves the following components:

- A convolutional encoder that transforms input images into spatial feature maps.
- A prototype layer consisting of m learnable prototype vectors, each associated with a class and corresponding to a specific spatial region in the feature map.
- A similarity-based mechanism: distances between input features and prototypes are converted into similarity scores used for classification.
- A linear classifier that combines similarity scores to produce class logits.
- A specialized loss function composed of:
  - Classification loss (Class): standard cross-entropy to ensure predictive accuracy.
  - Clustering loss (Clst): encourages prototypes to move closer to latent patches from the same class.
  - Separation loss (Sep): pushes prototypes away from patches belonging to other classes.

The similarity score between a prototype p_j and an input x is computed based on the minimum distance between p_j and all spatial patches z_ℓ of the encoded feature map $f(x)$. A decreasing function is applied to obtain a similarity measure:

$$s_j(x) = \max_{\ell} \log\left(\frac{\|z_\ell - p_j\|_2^2 + 1}{\|z_\ell - p_j\|_2^2 + \varepsilon}\right);$$

where ε is a small constant added for numerical stability.

While the original method was designed for image classification with spatial feature maps, this thesis adapts the idea to vector-based latent representations obtained from tabular transaction data. Instead of spatial feature maps, each input x is mapped to a fixed-length latent vector using an encoder. Prototypes are learned directly in this latent space, with each prototype representing a latent pattern typical for a class. While the original ProtoPNet compares spatial patches, the adapted model compares the entire latent vector to each prototype.

The adapted model architecture consists of:

- An encoder f that maps input vectors to a latent space,
- A prototype layer g_P containing m learnable prototype vectors $p_j \in \mathbb{R}^D$.
- A similarity function that compares the latent representation to each prototype.
- A final linear classifier l that maps similarity scores to class logits.

Given an input vector x, the encoder produces a latent representation:

$$z = f(x) \in \mathbb{R}^D: \tag{2.4}$$

In this setting, each prototype $p_j \in \mathbb{R}^D$ is compared to the full latent vector. The prototype layer computes squared Euclidean distances:

$$d = g_P(z) = \left(\|z - p_1\|_2^2, \dots, \|z - p_m\|_2^2\right). \tag{2.5}$$

To transform distances into similarity scores, the following function is applied:

$$s = \log\left(\frac{1 + d}{d + \varepsilon}\right),$$

where the transformation yields high similarity for small distances. These scores are then passed to a linear classifier:

Each class $k \in \{1, \dots, C\}$ is associated with a subset of prototypes $P_k \subseteq \{p_1, \dots, p_m\}$.

Optimization Process. Each training epoch consists of three phases:

- Joint optimization of encoder and prototypes.
- Projection of prototypes onto training data to enhance interpretability.
- Optimization of the final classifier with fixed encoder and prototype parameters.

(i) **Joint Optimization of Encoder and Prototypes:** The model minimizes the following objective:

$$\min_{f,P} \; \text{Class} + \lambda_1 \text{Clst} + \lambda_2 \text{Sep} \tag{2.6}$$

where:

$$\text{Class} = \frac{1}{n}\sum_{i=1}^{n} \text{CrossEntropy}(\hat{y}_i, y_i), \tag{2.7}$$

$$\text{Clst} = \frac{1}{n}\sum_{i=1}^{n} \min_{p_j \in P_{y_i}} \|z_i - p_j\|_2^2, \tag{2.8}$$

$$\text{Sep} = -\frac{1}{n}\sum_{i=1}^{n} \min_{p_j \notin P_{y_i}} \|z_i - p_j\|_2^2. \tag{2.9}$$

(ii) **Prototype Projection:** Instead of selecting representative image patches, as done in image-based ProtoPNet, each prototype is updated to match the nearest training latent vector from its assigned class:

$$p_j \leftarrow \arg\min_{z_i : y_i = k} \|p_j - z_i\|_2^2. \tag{2.10}$$

(iii) **Final Layer Optimization:** The classifier is fine-tuned with fixed encoder and prototype parameters. Initial weights are set as:

$$w_l^{(k,j)} = \begin{cases} +1.0, & \text{if } p_j \in P_k, \\ -0.5, & \text{otherwise.} \end{cases} \tag{2.11}$$

This setting encourages the classifier to increase the logit of class k when prototypes from P_k are highly activated, and to suppress the logits for other classes. The values are based on the original ProtoPNet formulation [9], designed to guide the latent space toward discriminative, class-specific structure.

Interpretability. The model inherently supports interpretability, as each prediction can be traced to its proximity to class-specific prototypes. In this context, prototypes represent latent transaction patterns, enabling post-hoc analysis of classifier decisions based on similarity to learned, representative examples.

2.7 Metric Learning with Triplet Loss

The prototype approach relies on good representation of the latent space, where the Euclidean distance serves as a meaningful similarity metric. However, achieving such a representation may be too difficult in an unsupervised manner. Therefore, we investigate the use of metric learning methods to better organize the latent space. Specifically, we focus on the triplet loss that is widely used in metric learning and face recognition to encourage separation between classes in the latent space[26].

Triplet Loss Formulation. The triplet loss operates on three samples:

- An anchor z_a — a sample of a particular class,
- A positive z_p — another sample of the same class,
- A negative z_n — a sample from a different class.

The loss encourages the distance between the anchor and the positive to be smaller than the distance between the anchor and the negative, by a margin $\alpha > 0$:

$$\mathcal{L}_{\text{triplet}}(z_a, z_p, z_n; \theta) = \max\{0, D_\theta(z_a, z_p) - D_\theta(z_a, z_n) + \alpha\} \tag{2.12}$$

where D_θ is a general distance with parameter θ. The metric is typically used to learn the parameter

This objective enforces local clustering of samples from the same class while repelling samples from different classes.

Benefits for Anomaly Type Differentiation. In the anomaly detection context, where each anomaly type reflects a distinct behavioral pattern, prototype collapse can occur without additional constraints. Triplet loss can be used to encourage better separation between anomaly-type prototypes by anchoring them to class-consistent examples. This improves the spatial organization of the latent space and enhances interpretability by aligning each prototype more clearly with its intended class.

3 Data

3.1 Introduction to the Dataset

This thesis makes use of a synthetic dataset developed for the Amaretto framework, an active learning system designed to detect money laundering in nancial transactions[27]. Due to legal, regulatory, and con dentiality constraints, access to real-world anti-money laundering (AML) data is extremely limited. As a result, the dataset was provided as part of the Amaretto framework, developed in collaboration with an industry partner to replicate the structure, complexity, and behavioral patterns of real nancial markets.

The dataset simulates client activity in international capital markets and includes both legitimate and anomalous transaction behaviors. Anomalous patterns are based on established anti-money laundering typologies and are embedded in a high-volume, highly imbalanced transaction stream. The dataset has been publicly released to support reproducibility in AML research and has been used to evaluate several machine learning-based detection approaches.

In the context of this thesis, the dataset serves as a challenging and realistic benchmark for evaluating classi er behavior and explainability in latent space. Its multi-dimensional structure and subtle anomalous patterns provide a strong testbed for interpretability methods applied to high-dimensional representations.

3.2 Dataset Composition

3.2.1 Dataset Statistics

The dataset consists of synthetic nancial transactions simulating the behavior of institutional clients operating in international capital markets. It was generated using a custom-built tool designed to mimic real-world transaction patterns, volumes, and client behaviors. The result is a rich and complex dataset that closely resembles the structure and statistical properties of actual nancial transaction logs encountered in the anti-money laundering (AML) domain.

Volume and Scope. The dataset includes a total of 29,704,090 transactions executed over 60 business days , corresponding to 12 weeks of market activity (excluding weekends). These transactions were generated for 400 distinct clients , referred to as end clients, each acting as a transaction originator.

Client Activity Distribution. The transaction activity is highly skewed among the clients:

- Approximately 90% of the clients conducted at least 50,000 transactions over the 60-day period.
- The top 10% of clients were responsible for approximately 50% of the total transaction volume , each generating around 400,000 transactions .

Transaction Amount Distribution. The dataset captures a realistic spread of transaction sizes:

- 98.43% of transactions involve amounts less than $1,000,000.
- Approximately 40% of all transactions are below $10,000.

This long-tailed distribution re ects typical capital market behaviors, where the majority of trades are moderate in size, with a minority involving large capital movements.

Figure 3.1: Distribution of normalized transaction amounts. The x-axis is log-scaled to reveal long-tailed behavior, where a majority of transactions involve relatively small amounts.

Temporal Structure. Transactions are evenly distributed across the 12-week period. A typical week includes 5 business days, re ecting market closures on weekends. Most transactions occur during standard market hours, while only a small fraction takes place during early morning or late-day windows.

Figure 3.2: Distribution of transactions by hour of day (log-scaled y-axis). Most transactions occur during standard market hours, while early morning and late-day windows show significantly fewer activities.

Currency and Market Diversity. Transactions involve multiple currencies and span several market types, which adds to the dataset's realism and complexity. This is essential for simulating laundering behavior, which often involves cross-currency trades and atypical market interactions.

Class Imbalance. The dataset is **heavily imbalanced**, with less than 1% of transactions labeled as anomalous. This aligns with real-world AML challenges, where suspicious activity is both rare and subtle. This imbalance makes the dataset particularly well-suited for evaluating both detection performance and classifier behavior under skewed distributions.

3.2.2 Features

The dataset contains a set of features that represent transactional and contextual information typical of capital markets. These raw features form the foundational layer of the dataset and are used for subsequent preprocessing, aggregation, and modeling. Below is a description of each feature included in the original dataset:

- id: A unique transaction identifier, represented as a 32-bit unsigned integer. It serves only as a technical reference and carries no analytical value.
- Originator: A categorical string identifier representing the client (e.g., Client_051, Client_367). There are 400 unique clients in total, each acting as the originator of one or more transactions.
- InputOutput: A categorical variable indicating the direction of the transaction. It takes one of two values: Buy or Sell.
- Market: A numerical identifier for the market where the transaction was executed. It takes integer values ranging from 1 to 4, representing different financial markets or exchanges. The specific market mappings are anonymized.

- Product Type : A categorical feature describing the nancial instrument involved in the transaction. The dataset includes 17 distinct types, such as FX, Repo, Bond, Future, Equity , and others. These re ect the diversity of assets traded in the capital markets.
- Product Class : A higher-level categorization of the transaction type. There are four main classes: Trade, External Fee , Cash in/out , and ADR Conversion. This eld provides a coarse grouping for analyzing transaction intent or purpose.
- Normalized Amount : A oating-point variable representing the monetary amount of the transaction. All values have been normalized, though they still preserve meaningful scale (e.g., values can exceed one million). This is a key numerical feature for identifying anomalous transaction sizes.
- Currency : An integer-encoded categorical variable denoting the currency in which the transaction occurred. The dataset includes two currency identi ers (1 and 2), representing anonymized at currencies.
- Anomaly : A label indicating whether a transaction is part of a known anomalous pattern. The values range from 0 to 5, where 0 indicates a legitimate transaction and values 1 to 5 correspond to di erent injected anomaly classes (see Section 3.3).
- datetime : A timestamp indicating the exact time the transaction was executed, with microsecond resolution. This feature is essential for temporal aggregation and for detecting behaviors such as activity burst or o -hour trading.

Table 3.1: Summary of original dataset features

Feature Name	Type	Description	Domain / Example
id	Identi er (u32)	Unique transaction ID	12821854, 3125856
Originator	Categorical (str)	Client identi er	Client _051, Client _367
InputOutput	Categorical (str)	Direction of transaction	Buy, Sell
Market	Categorical (int)	Market identi er	1, 2, 3, 4
Product Type	Categorical (str)	Financial instrument type	FX, Bond, Repo
Product Class	Categorical (str)	High-level transaction class	Trade, Cash in/out
Normalized Amount	Numerical (f64)	Transaction amount	67474.37, 3.3e6
Currency	Categorical (int)	Currency identi er	1, 2
Anomaly	Categorical (int)	Anomaly label	0 (legit), 1{5 (anomalies)
datetime	Timestamp (s)	Transaction time	2019-01-17 19:44:27

3.3 Anomaly Classes

In order to simulate realistic anti-money laundering (AML) scenarios, the dataset includes a small proportion of synthetic anomalies. These anomalies were injected by domain experts to re ect common suspicious transaction patterns as de ned by the Financial Action Task Force (FATF). Each anomaly is assigned a unique integer label from 1 to 5 in the Anomaly column, while legitimate transactions are labeled as 0.

The anomalies are designed to be subtle and well-hidden, often mimicking the statistical properties of legitimate transactions to make detection more challenging. Below is a description of each anomaly class:

(i) Small but highly frequent transactions generated within a short timeframe:
A pattern that contains multiple transactions below applicable reporting thresholds.

(ii) Transactions with rounded normalized amounts bought or sold within an account: It is unusual for transactions in capital markets to have rounded amounts (unless they occur in markets where foreign exchange conversion causes rounding errors).

(iii) Security bought or sold at an unusual time: It is unusual for clients to trade specific securities outside of a specific timeframe (for example, outside of the opening and closing times of a stock exchange).

(iv) Large asset withdrawal: A sudden spike in transaction amount withdrawn from an account or transferred out, which deviates from the previous transactional activity and is absent of any commercial rationale or related corporate action event.

(v) An unusually large amount of collateral transferred in and out of an account within a short period of time: This behavior is unusual as a client would not be able to invest by simply trading collateral, or at least such a strategy would be unusual.

Each anomaly type is designed to challenge different aspects of anomaly detection algorithms, such as temporal analysis, behavioral profiling, and statistical rarity. Their presence supports evaluation of models not only in detection accuracy but also in robustness and explainability under diverse threat scenarios.

Anomaly Type	Number of Transactions	Proportion in Dataset
1	5,884	7.24%
2	2,576	3.17%
3	12,088	14.87%
4	30,640	37.70%
5	30,080	37.01%

Table 3.2: Distribution of different anomalous transaction types.

3.4 Data Preprocessing and Feature Aggregation

Before applying machine learning models to the transaction data, a comprehensive preprocessing and feature engineering pipeline was applied to convert the original raw fields into a rich and structured set of numerical features. The preprocessing aimed to extract temporal, behavioral, and statistical properties from transaction activity that are relevant for anomaly detection, especially in the context of explainable models.

Preprocessing Overview

The dataset initially consists of raw transactional attributes such as the originator ID, transaction direction, market and product identifiers, currency, amount, and timestamp. To make this data suitable for machine learning, several key preprocessing steps were performed:

- Timestamp normalization and periodic encoding: The datetime field was used to extract hour, day, and weekday components, which were then encoded using sine and cosine transformations to capture the cyclical nature of time-based patterns.
- Amount transformation: Transaction amounts were log-transformed (log1p) to reduce the impact of extreme values and better reflect human-scale differences between small and large amounts.
- Categorical encoding: Categorical features such as market, product type, product class, and currency were one-hot encoded. Additionally, cumulative counts and relative frequencies of these categories per originator were computed to capture client-specific preferences or biases.

- Boolean feature derivation: A binary feature was created to indicate whether a transaction had a perfectly rounded amount (a common trait in anomalous behavior), as well as a binary ag for transactions occurring during unusual trading hours (e.g., between 2:00 AM and 5:00 AM).

Temporal and Behavioral Aggregation

To capture temporal dynamics and transaction context, a variety of time-based and sequence-based features were derived:

- Lag-based features: For each transaction, timestamp and amount di erences to previous transactions (lags of 1{5) were computed within various groupings: per originator, per market, per product type, and per product class. This helps detect irregular timing or sudden changes in behavior.
- Rolling window statistics: Using xed time windows of 1 hour, 12 hours, 1 day, and 7 days, rolling means, sums, standard deviations, and transaction counts were computed globally and per originator. These features capture short- and long-term trends or volatility in behavior.
- Frequency-based behavior modeling: The relative frequency of each categorical value (e.g., how often an originator uses a certain market or product type) was tracked cumulatively and normalized to estimate evolving client pro les.

Final Feature Set

The result of this process was a high-dimensional, fully numerical feature matrix consisting of 118 features (excluding the target anomaly label). These features encompass:

- One-hot encoded transaction properties (e.g., product and market types)
- Log-transformed and derived amount statistics
- Cyclical and binary time features
- Lagged timestamp and amount di erences
- Rolling window statistics over multiple time scales
- Client-level cumulative counts and relative frequencies

This diverse feature set captures both short-term and long-term behavioral patterns, enabling the model to identify immediate irregularities as well as more subtle deviations over time. Because the input features are derived from meaningful temporal, categorical, and behavioral properties, they provide a solid foundation for analyzing patterns in the latent space and performing post-hoc explanations of model decisions.

3.5 Limitations of the Dataset

While the dataset used in this thesis o ers a valuable and realistic approximation of transaction activity in capital markets, several limitations must be acknowledged regarding its origin, design, and generalizability.

Synthetic Nature of the Data. The dataset is entirely synthetic and was generated using a rule-based simulator designed in collaboration with industry experts. While it approximates the structure and statistical properties of real nancial transactions, synthetic data inherently lacks the unpredictability, noise, and adversarial behavior that characterize real-world nancial systems. As such, models trained and evaluated on this dataset may perform differently when deployed in operational environments with truly adversarial actors.

Bias from Predefined Anomaly Patterns. Anomalies in the dataset were manually injected based on predefined patterns inspired by FATF typologies. While these reflect common forms of money laundering, they represent a limited subset of possible suspicious behaviors. Consequently, the dataset may bias models toward detecting only those specific patterns and not generalize well to more subtle or novel laundering strategies.

Domain-Specific Design. The dataset simulates institutional client activity in capital markets. It does not include data from other financial sectors such as retail banking, fin-tech platforms, or cryptocurrency exchanges. Therefore, results obtained using this dataset may not generalize to other transaction environments with different user behavior, regulatory constraints, or risk profiles.

Feature Engineering Based on Known Anomalies. The feature engineering process is designed to capture the characteristics of the specific anomaly types present in the data. As a result, the feature space may not include attributes that are useful for detecting other forms of suspicious behavior that were not anticipated during dataset construction.

Absence of Feedback Dynamics. In practice, anti-money laundering systems are part of an interactive process that includes analyst reviews, feedback loops, and evolving criminal tactics. The dataset does not model any of these dynamics. All clients and transactions follow fixed patterns, and no adaptation occurs in response to detection or changes in strategy.

In summary, while the dataset is useful for benchmarking detection models and studying explainability, it does not fully represent the complexity of real-world financial crime detection scenarios. These limitations should be considered when evaluating the general applicability of the findings.

4 Experiments

4.1 Unsupervised Latent Representation

To explore whether meaningful structure emerges in the absence of label supervision, an autoencoder was trained solely to reconstruct input transactions. The goal of this experiment is to assess whether the latent space learned by the encoder naturally separates anomalous transactions from normal ones.

4.1.1 Autoencoder Setup.

The autoencoder consists of a fully connected encoder and decoder, with the latent space dimension set to either 2 or 16. The 2-dimensional variant is used to directly visualize the encoded space, while the 16-dimensional version provides a more expressive representation that preserves additional information. Both models are trained using a mean squared reconstruction loss on normalized features.

Parameter	Latent Dim = 2	Latent Dim = 16
Input / Output Dimension	118	
Hidden Layers (Encoder)	[32, 16]	[64, 32]
Hidden Layers (Decoder)	[16, 32]	[32, 64]
Activation Function	Tanh	
Optimizer	AdamW (lr = 0.001, weight decay = 0.001)	
Loss Function	MSELoss	
Epochs	7	

Table 4.1: Autoencoder con gurations used in unsupervised experiments.

4.1.2 Latent Space Visualization

To assess whether meaningful structure emerges in the latent space without supervision, the encoded representations were visualized for both autoencoder variants. For the model with a 16-dimensional latent space, the embeddings were projected to two dimensions using Uniform Manifold Approximation and Projection (UMAP). In contrast, the autoencoder with a 2-dimensional latent space required no additional dimensionality reduction.

Two types of visualizations are presented for each model. The plot on the left shows a random subsample of approximately 70,000 transactions, with 10% of them labeled as anomalies. The sampling process preserves the original class distribution among anomaly types. The plot on the right displays only the anomalous transactions, allowing for a clearer view of how di erent anomaly types are positioned relative to one another in the latent space.

It is important to note that dimensionality reduction methods such as UMAP and t-SNE t their projection based on the data provided. As a result, the resulting structure can vary depending on whether normal transactions are included or not. This should be kept in mind when interpreting apparent cluster shapes or distances between points.