Deep Learning Project Report

on

INSIDER THREAT DETECTION

Submitted by

P. Lavanya 22501A05D4

T. Ramya 22501A05I0

T. Thanuja 22501A05I2

T. Yogitha 22501A05I3

Under the Esteemed Guidance of

Mrs. Y. SUREKHA, M.Tech.,(Ph.D.)

Department of Computer Science and Engineering

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

(Permanently affiliated to JNTU: Kakinada, Approved by AICTE)

(An NBA & NAAC accredited and ISO 9001:2015 certified institution)

Kanuru, Vijayawada-520007

2025-2026

Signature of the Guide

Mrs. Y. Surekha,

Assistant Professor

Signature of the HOD

Dr. A. Jaya Lakshmi,

Professor & HOD

TABLE OF CONTENTS

1. ABSTRACT

Insider threats, defined as security risks originating from within an organization by authorized users, pose a significant challenge due to their inherent ability to bypass external security measures. Detecting these threats relies on identifying subtle deviations from established normal user behaviour. This project proposes an automated **Insider Threat Detection System** leveraging **Deep Learning-based Anomaly Detection**. The proposed approach utilizes two main models: a **Standard Autoencoder** to learn normal aggregate user profiles and an **LSTM Autoencoder** to capture and detect anomalies in sequential (time-series) behavioural patterns, specifically focusing on user logon activities. The system processes extensive user activity logs, including logon events, email communication, and device usage, transforming them into comprehensive feature vectors. By training the models solely on normal behaviour, any activity resulting in a high reconstruction error is flagged as an anomaly or potential threat. This unsupervised methodology effectively overcomes the challenge of limited labelled threat data, providing a robust, scalable framework capable of identifying both instantaneous and temporal deviations indicative of insider risk.Additionally, the system integrates data preprocessing techniques to filter noise and normalize activity patterns, ensuring higher detection accuracy. The deep learning models continuously adapt to evolving user behaviours through periodic retraining, enhancing their ability to detect emerging threats. Visualization dashboards can be incorporated to display anomaly scores, risk trends, and user activity summaries for better interpretability by security analysts. This proactive approach minimizes false positives while strengthening the organization's defence against malicious insiders. Overall, the proposed system provides an intelligent, adaptive, and data-driven solution for safeguarding sensitive organizational assets.

Keywords: *Insider Threat Detection, Deep Learning, Anomaly Detection, Autoencoder, LSTM, Behavioural Analysis, Unsupervised Learning, Cybersecurity.*

2. INTRODUCTION

2.1. Problem Statement

The growing frequency of data breaches caused by insider activity, whether through deliberate actions or accidental errors, exposes a major weakness in enterprise cybersecurity. Organizations deal with enormous amounts of user activity data such as logon records, emails, and file transfers which makes manual monitoring impractical and time-consuming. Traditional rule-based detection systems struggle to keep pace with evolving threats, often generating numerous false alerts while missing unfamiliar attack methods. Similarly, supervised machine learning approaches depend heavily on labelled datasets, which are difficult to obtain and often unbalanced, reducing their accuracy and adaptability in real-world environments.

To address these limitations, this project proposes an intelligent and fully automated unsupervised insider threat detection framework. The system is designed to analyse and learn from complex, high-dimensional behavioural data to recognize deviations that may signal potential insider risks. By leveraging advanced deep learning models, it can detect unusual user activities without requiring predefined attack patterns or labelled threat data. This ensures faster, more accurate identification of both intentional and unintentional insider threats, strengthening an organization's overall data protection and resilience against internal security breaches.

In addition, the proposed system promotes continuous learning by adapting to changing user behaviour over time, ensuring long-term effectiveness. It can be integrated with existing security infrastructures to provide real-time alerts and detailed insights into suspicious activities. This proactive and scalable approach enables organizations to minimize insider risks and maintain a secure, trustworthy digital environment.

2.2. Objective

The primary objectives of the Insider Threat Detection project are as follows:

1. **Behavioural Modelling:** Develop comprehensive aggregate and sequential features from raw user activity logs (logon, email, device events).

2. **Unsupervised Anomaly Detection:** Implement a **Standard Autoencoder** to establish a baseline of "normal" static user behaviour.

3. **Temporal Anomaly Detection:** Implement an **LSTM Autoencoder** to model and detect anomalies within time-series user activity (e.g., daily logon patterns).

4. **Risk Scoring:** Use the reconstruction error from the trained models as a quantifiable measure of anomaly or risk score for each user and behavioural sequence.

5. **Threat Triage:** Identify and analyse the specific behavioural features contributing most significantly to the highest anomaly scores to assist security analysts.

6. **Real-Time Monitoring and Alerting**: Integrate the system with live data streams to enable continuous monitoring of user activities and generate instant alerts for suspicious or high-risk behaviours, allowing timely investigation and response by security teams.

2.3. Proposed Model Overview

The proposed system adopts a multi-faceted Deep Learning architecture specifically designed for unsupervised anomaly detection in insider threat scenarios. It focuses on modeling both static and dynamic aspects of user behavior to provide a comprehensive understanding of normal activity patterns. By combining multiple deep learning models, the system can effectively detect deviations that may indicate potential insider risks without relying on labeled threat data.

In the **Static Profile Analysis** phase, an aggregate behavioral profile is generated for each user by summarizing their overall activities across the dataset, such as total emails sent, average logon times, and device usage frequency. These aggregated features are then passed through a Standard Autoencoder built using Dense Neural Network layers. The Autoencoder learns to reproduce the normal activity profiles by minimizing reconstruction error during training. When a user's reconstructed profile exhibits a high error, it suggests that their overall behavior significantly deviates from the learned norm, signaling possible anomalous activity.

In the **Sequential Behavior Analysis** phase, the focus shifts to temporal modeling using an LSTM Autoencoder. Here, user logon data is organized into sequential time windows, such as seven-day patterns of logon frequency or unique device usage. The LSTM Autoencoder captures the temporal dependencies and rhythm in user behavior over time, making it capable of identifying subtle and evolving deviations in activity patterns. This enables the system to detect both sudden and gradual changes in user actions, improving the accuracy and reliability of insider threat detection.

3. MATERIALS AND METHODS

3.1 Requirements:

1. Software Requirements

- **Programming Language:** Python (for deep learning and data processing)
- **Frameworks & Libraries:** TensorFlow / Keras or PyTorch, NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn
- **Database:** MySQL or MongoDB for storing user activity logs and model outputs
- **Development Tools:** Jupyter Notebook or Google Colab for training and testing models
- **Version Control:** GitHub or GitLab for maintaining project versions and collaboration

2. Hardware Requirements

- **Processor:** Intel Core i5 / i7 or equivalent multi-core CPU
- **Memory (RAM):** Minimum 8 GB (16 GB recommended for model training)
- **Storage:** Minimum 500 GB for storing logs and datasets
- **Graphics Processing Unit (GPU):** NVIDIA CUDA-enabled GPU for efficient deep learning computations
- **System Environment:** Windows, Linux, or macOS with Python support

3. Dataset Requirements

- **Data Type:** User activity logs such as logon details, email records, device connections, and file transfer logs
- **Data Attributes:** User ID, timestamp, source/destination IP, device ID, event type, and session duration
- **Data Volume:** Large-scale, continuous data collected from enterprise environments for realistic modelling
- **Data Quality:** Clean, consistent, and noise-free data through preprocessing and normalization steps
- **Data Privacy:** Sensitive information must be anonymized to ensure compliance with privacy regulations

4. Model Training Requirements

- **Training Approach:** Unsupervised learning using normal (non-threat) user behaviour data
- **Static Model:** Standard Autoencoder for aggregate (non-sequential) behavioural profiles
- **Sequential Model:** LSTM Autoencoder for modelling time-dependent user activity sequences
- **Training Objective:** Minimize reconstruction error to capture normal activity patterns
- **Batch Size & Epochs:** Optimized based on dataset size and system performance

5. Model Evaluation Requirements

- **Evaluation Metrics:** Reconstruction error, anomaly detection accuracy, precision, recall, and F1-score
- **Testing Data:** Include both normal and synthetic anomalous behaviours for evaluation
- **Validation Technique:** Use cross-validation or hold-out methods for unbiased performance assessment
- **Performance Goal:** Achieve a balance between high detection accuracy and low false positives

3.2. Dataset

The project utilizes simulated, yet realistic, enterprise user activity data, typically provided across multiple CSV files (often based on CMU's CERT dataset structure):

- **users.csv:** Contains static user metadata (user_id, role, department, employee_name).
- **logon.csv:** Detailed log of user logon and logoff events, including PC identifier and timestamp.
- **email.csv:** Records of email communication, including sender, recipient, activity (Send/Receive), size, and external recipient indicators.
- **device.csv:** Records of external device connections (e.g., USB drives), including file path data.

3.3. Deep Learning Models

3.3.1. Standard Autoencoder (For Aggregate Data)

A basic autoencoder structure is used for the static user profile. It comprises an **Encoder** (which compresses the high-dimensional input vector into a low-dimensional latent space) and a **Decoder** (which attempts to reconstruct the original input from the latent representation).

- **Input Dimension:** 29 (Number of engineered aggregate features).
- **Encoding Dimension:** 16 (Latent space size).
- **Architecture:** Multiple Dense layers with ReLU activation, interspersed with BatchNormalization and Dropout (0.2) layers to prevent overfitting.
- **Training:** Optimized using the Adam optimizer with Mean Squared Error (MSE) loss.

3.3.2. LSTM Autoencoder (For Sequential Data)

An LSTM Autoencoder is used to process the time-series data of user behavior, capturing the temporal dynamics (e.g., how daily logon count fluctuates over a week).

- **Input Shape:** (Timesteps, Features) = (7 days, 2 daily logon features).
- **Architecture:**
    - **Encoder:** Stacked LSTM layers (return_sequences=True for intermediate layers, return_sequences=False for the final encoder layer).
    - **Decoder:** A RepeatVector layer expands the final encoder output back to the required timesteps, followed by stacked LSTM layers and a TimeDistributed(Dense) output layer.
- **Training:** Also uses Adam optimizer and MSE loss.

3.4. Evaluation Parameters

Since this is an unsupervised anomaly detection project, evaluation focuses on clustering quality and error analysis:

- **Reconstruction Error (MSE/MAE):** The primary metric. Anomalies are defined as data points with a reconstruction error exceeding a threshold (typically the 95th or 99th percentile of all errors).
- **Clustering Metrics (Post-Hoc on Encoded Features):**
    - **Silhouette Score:** Measures how similar a data point is to its own cluster compared to other clusters. A higher score is better.
    - **Davies-Bouldin Index:** Measures the average similarity between clusters. A lower score is better.
    - **Calinski-Harabasz Score:** Measures the ratio of between-cluster variance to within-cluster variance. A higher score is better.
- **Feature Importance:** Measures the contribution of each original feature to the overall reconstruction error, indicating *why* a user was flagged.

4. METHODOLOGY

The methodology involved a structured, multi-stage pipeline, starting with data acquisition and ending with a comprehensive risk report.

1. **Data Loading & Preprocessing:** Load raw user logs (logon, email, device) and user metadata (users). Parse date/time fields.
2. **Feature Engineering (Static):**
    - **Logon:** Calculate total events, unique PCs used, after-hours logons, and the standard deviation of logon hour (std_logon_hour), a key indicator of routine disruption.
    - **Email:** Calculate total emails, emails sent externally, average/max/total size, and external_email_ratio.
    - **Device:** Calculate total device events, unique PCs used, and file path depth features.
    - **Categorical Encoding:** Convert user metadata (Role, Department) into numerical encodings.
3. **Feature Engineering (Sequential/Temporal):** Aggregate logon data to a daily level, counting daily_logons and daily_unique_pcs. Create sliding 7-day windows (lookback=7) of this scaled daily data for the LSTM model.
4. **Data Scaling:** Apply StandardScaler to all feature sets to normalize distributions and ensure fair contribution during model training.
5. **Model Training:** Train the Standard Autoencoder and the LSTM Autoencoder. Crucially, the models are trained to accurately reconstruct **normal** behavioral patterns.
6. **Anomaly Detection:**
    - Calculate the reconstruction error for all data points using both trained models.
    - Set the anomaly threshold (e.g., 95th percentile of errors).
    - Flag any user(static model) or sequence(temporal model) whose error exceeds threshold.

5. IMPLEMENTATION

Feature Engineering Details

A total of 29 aggregate features were created. Key risk-indicator features calculated include:

- after_hours_ratio: Percentage of logons occurring outside standard business hours (7 AM - 7 PM).
- email_to_logon_ratio: Ratio of total email events to total logon events.
- external_email_ratio: Ratio of external emails sent to total emails sent.
- max_file_depth: Maximum depth of the file tree accessed via connected devices.

Autoencoder Training

The static autoencoder model was trained on the scaled aggregate features (X_scaled). The purpose was to identify users whose *overall profile* deviates from the norm.

Metric	**Training Result**
Final Loss	~0.1073
Final Validation MAE	~0.2002

LSTM Autoencoder Training

The LSTM model was trained on the scaled 7-day sequential data (X_seq_scaled). This model is vital for detecting temporal anomalies, such as a user who suddenly starts logging in at erratic times or on consecutive days when they were typically absent.

Metric	**Training Result**
Final Loss	~0.0326
Final Validation MAE	~0.0275

Code:

Step 1: Setup and Imports

```
import pandas as pd
import numpy as np
import os
import warnings
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, Model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import joblib

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')
# Set the plot style
sns.set_style('whitegrid')
```

Step 2: Data Loading

```
print("="*70)
print(" 🔒 INSIDER THREAT DETECTION - DEEP LEARNING")
print("="*70)

# Define data directory and file paths
data_dir = "datasets"
files = {
    "users": os.path.join(data_dir, "users.csv"),
    "logon": os.path.join(data_dir, "logon.csv"),
    "email": os.path.join(data_dir, "email.csv"),
    "device": os.path.join(data_dir, "device.csv")
}

# Load all datasets
print("\n📂 Loading datasets...")
data = {}
for name, path in files.items():
    try:
        df = pd.read_csv(path)
        data[name] = df
```

```
        print(f"✅ Loaded {name}: {df.shape}")
    except Exception as e:
        print(f"❌ Error loading {name}: {e}")
        data[name] = None
```

Step 3.1 : Logon Feature

```
print("\n🔧 Engineering behavioral features...")
features = {}

# Process logon data if available
if data['logon'] is not None:
    logon_df = data['logon'].copy()
    logon_df['date'] = pd.to_datetime(logon_df['date'], format='%m/%d/%Y %H:%M:%S',
errors='coerce')

    # Aggregate logon patterns
    logon_features = logon_df.groupby('user').agg({
        'id': 'count',
        'pc': 'nunique',
        'activity': lambda x: (x == 'Logon').sum()
    }).rename(columns={
        'id': 'total_logon_events',
        'pc': 'unique_pcs_logon',
        'activity': 'logon_count'
    })

    # Create time-based features
    logon_df['hour'] = logon_df['date'].dt.hour
    logon_df['day_of_week'] = logon_df['date'].dt.dayofweek
    logon_df['is_weekend'] = logon_df['day_of_week'].isin([5, 6]).astype(int)
    logon_df['is_after_hours'] = ((logon_df['hour'] < 7) | (logon_df['hour'] > 19)).astype(int)

    time_features = logon_df.groupby('user').agg({
        'is_weekend': 'sum',
        'is_after_hours': 'sum',
        'hour': ['mean', 'std']
    })
    time_features.columns = ['weekend_logons', 'after_hours_logons', 'avg_logon_hour',
'std_logon_hour']

    # Combine logon features
    logon_features = logon_features.join(time_features)
    features['logon'] = logon_features
```

Step 3.2 : Logon Feature

```
if data['device'] is not None:
    device_df = data['device'].copy()
    device_df['date'] = pd.to_datetime(device_df['date'], format='%m/%d/%Y %H:%M:%S',
errors='coerce')

    # Aggregate device usage
    device_features = device_df.groupby('user').agg({
        'id': 'count',
        'pc': 'nunique',
        'activity': lambda x: (x == 'Connect').sum()
    }).rename(columns={
        'id': 'total_device_events',
        'pc': 'unique_pcs_device',
        'activity': 'device_connect_count'
    })

    # Analyze file tree depth
    device_df['file_depth'] = device_df['file_tree'].fillna('').apply(
        lambda x: len(x.split(';')) if x else 0
    )

    file_features = device_df.groupby('user')['file_depth'].agg(['mean', 'max',
'std']).rename(columns={
        'mean': 'avg_file_depth',
        'max': 'max_file_depth',
        'std': 'std_file_depth'
    })

    # Combine device features
    device_features = device_features.join(file_features)
    features['device'] = device_features
```

Step 3.3 : Logon Feature

```
if data['email'] is not None:
    email_df = data['email'].copy()
    email_df['date'] = pd.to_datetime(email_df['date'], format='%m/%d/%Y %H:%M:%S',
errors='coerce')

    # Aggregate email statistics
    email_features = email_df.groupby('user').agg({
        'id': 'count',
        'pc': 'nunique',
        'activity': lambda x: (x == 'Send').sum(),
```

```
    'size': ['mean', 'sum', 'max', 'std']
  })
  email_features.columns = [
    'total_emails', 'unique_pcs_email', 'emails_sent',
    'avg_email_size', 'total_email_size', 'max_email_size', 'std_email_size'
  ]

  # Detect external emails and attachments
  email_df['is_external'] = ~email_df['to'].fillna('').str.contains('@dtaa.com')
  email_df['has_attachments'] = email_df['attachments'].notna().astype(int)

  external_features = email_df.groupby('user').agg({
    'is_external': 'sum',
    'has_attachments': 'sum'
  }).rename(columns={
    'is_external': 'external_emails',
    'has_attachments': 'emails_with_attachments'
  })

  # Combine email features
  email_features = email_features.join(external_features)
  features['email'] = email_features
```

Step 3.4 : Logon Feature

```
# Merge all feature sets
user_features = pd.concat(features.values(), axis=1, join='outer')
user_features = user_features.fillna(0)

# Add user metadata from users.csv
if data['users'] is not None:
  users_df = data['users'].copy().set_index('user_id')

  # Encode categorical features
  le = LabelEncoder()
  for col in ['role', 'functional_unit', 'department']:
    if col in users_df.columns:
      users_df[f'{col}_encoded'] = le.fit_transform(users_df[col].fillna('Unknown'))

  user_features = user_features.join(
    users_df[['role_encoded', 'functional_unit_encoded', 'department_encoded']],
    how='left'
  )

user_features = user_features.fillna(0)

# Create derived risk indicator ratios
```

```
user_features['email_to_logon_ratio'] = user_features['total_emails'] /
user_features['total_logon_events'].replace(0, 1)
user_features['device_to_logon_ratio'] = user_features['total_device_events'] /
user_features['total_logon_events'].replace(0, 1)
user_features['external_email_ratio'] = user_features['external_emails'] /
user_features['total_emails'].replace(0, 1)
user_features['after_hours_ratio'] = user_features['after_hours_logons'] /
user_features['total_logon_events'].replace(0, 1)

print(f"✅ Created {user_features.shape[1]} features for {user_features.shape[0]} users")
```

Step 4. Data Preparation for Autoencoder

```
print("\n🔧 Preparing data for training...")
feature_names = user_features.columns.tolist()
user_ids = user_features.index.tolist()
X = user_features.values

# Scale features using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print(f"✅ Data prepared: {X_scaled.shape}")
```

Step 5.1. Build and Train the Model

```
print("\n" + "="*70)
print("MODEL 1: AUTOENCODER")
print("="*70)

print("\n🏗️ Building Autoencoder...")

# Define model architecture
input_dim = X_scaled.shape[1]
encoding_dim = 16

input_layer = layers.Input(shape=(input_dim,))
encoded = layers.Dense(64, activation='relu')(input_layer)
encoded = layers.BatchNormalization()(encoded)
encoded = layers.Dropout(0.2)(encoded)
encoded = layers.Dense(32, activation='relu')(encoded)
encoded = layers.BatchNormalization()(encoded)
encoded = layers.Dropout(0.2)(encoded)
encoded = layers.Dense(encoding_dim, activation='relu', name='encoding')(encoded)

decoded = layers.Dense(32, activation='relu')(encoded)
```

```
decoded = layers.BatchNormalization()(decoded)
decoded = layers.Dropout(0.2)(decoded)
decoded = layers.Dense(64, activation='relu')(decoded)
decoded = layers.BatchNormalization()(decoded)
decoded = layers.Dropout(0.2)(decoded)
decoded = layers.Dense(input_dim, activation='linear')(decoded)
autoencoder_model = Model(inputs=input_layer, outputs=decoded)
encoder_model = Model(inputs=input_layer, outputs=encoded)

autoencoder_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss='mse',
metrics=['mae'])
print(f"✅ Autoencoder built - Input: {input_dim}, Encoding: {encoding_dim}")

# Train the model
print("\n🎓 Training Autoencoder...")
callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
]
history_ae = autoencoder_model.fit(
    X_scaled, X_scaled,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=callbacks,
    verbose=1
)
print("✅ Training complete")
```

Step 5.2. Visualize Training History

```
# Plot training and validation loss & MAE
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))

ax1.plot(history_ae.history['loss'], label='Training Loss')
ax1.plot(history_ae.history['val_loss'], label='Validation Loss')
ax1.set(xlabel='Epoch', ylabel='Loss', title='Autoencoder - Loss')
ax1.legend()
ax1.grid(True)

ax2.plot(history_ae.history['mae'], label='Training MAE')
ax2.plot(history_ae.history['val_mae'], label='Validation MAE')
ax2.set(xlabel='Epoch', ylabel='MAE', title='Autoencoder - MAE')
ax2.legend()
ax2.grid(True)
plt.tight_layout()
plt.show()
```

Step 5.3. Detect and Analyze Anomalies

```
# Detect anomalies based on reconstruction error
reconstructed_ae = autoencoder_model.predict(X_scaled, verbose=0)
errors_ae = np.mean(np.square(X_scaled - reconstructed_ae), axis=1)
threshold_ae = np.percentile(errors_ae, 95)
anomalies_ae = errors_ae > threshold_ae

# Analyze anomaly distribution
print("\n📊 Anomaly Detection Analysis:")
n_total = len(anomalies_ae)
n_anomalies = np.sum(anomalies_ae)
anomaly_rate = n_anomalies / n_total * 100

print(f"  • Total samples: {n_total}")
print(f"  • Detected anomalies: {n_anomalies} ({anomaly_rate:.2f}%)")
print(f"  • Normal samples: {n_total - n_anomalies} ({100 - anomaly_rate:.2f}%)")
print("\n  Reconstruction Error Statistics:")
print(f"    - Mean: {np.mean(errors_ae):.4f}")
print(f"    - Std: {np.std(errors_ae):.4f}")
print(f"    - 95th percentile: {np.percentile(errors_ae, 95):.4f}")
```

Step 5.4. Visualize Anomaly Results

```
# Plot reconstruction error distribution
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

ax1.hist(errors_ae[~anomalies_ae], bins=50, alpha=0.7, label='Normal', color='green')
ax1.hist(errors_ae[anomalies_ae], bins=50, alpha=0.7, label='Anomaly', color='red')
ax1.axvline(threshold_ae, color='black', linestyle='--', linewidth=2, label='Threshold')
ax1.set(xlabel='Reconstruction Error', ylabel='Frequency', title='Autoencoder - Distribution')
ax1.legend()
ax1.grid(True, alpha=0.3)

indices = np.arange(len(errors_ae))
ax2.scatter(indices[~anomalies_ae], errors_ae[~anomalies_ae], c='green', alpha=0.5, s=10,
label='Normal')
ax2.scatter(indices[anomalies_ae], errors_ae[anomalies_ae], c='red', alpha=0.7, s=30,
label='Anomaly')
ax2.axhline(threshold_ae, color='black', linestyle='--', linewidth=2, label='Threshold')
ax2.set(xlabel='Sample Index', ylabel='Reconstruction Error', title='Autoencoder - Sample-wise')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

```
# Plot PCA projection
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(10, 6))
plt.scatter(X_pca[~anomalies_ae, 0], X_pca[~anomalies_ae, 1], c='green', alpha=0.5, s=20,
label='Normal')
plt.scatter(X_pca[anomalies_ae, 0], X_pca[anomalies_ae, 1], c='red', alpha=0.7, s=50,
label='Anomaly', edgecolors='black')
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.2%} variance)')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.2%} variance)')
plt.title("Autoencoder - PCA Projection")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Step 5.5. Evaluate Clustering Quality and Identify Top Anomalies

```
# Evaluate clustering quality using encoded features
encoded_features = encoder_model.predict(X_scaled, verbose=0)
cluster_labels = (errors_ae > threshold_ae).astype(int)

print("\n📊 Clustering Quality Metrics:")
sil_score = silhouette_score(encoded_features, cluster_labels)
db_score = davies_bouldin_score(encoded_features, cluster_labels)
ch_score = calinski_harabasz_score(encoded_features, cluster_labels)
print(f"  • Silhouette Score: {sil_score:.4f}")
print(f"  • Davies-Bouldin Index: {db_score:.4f}")
print(f"  • Calinski-Harabasz Score: {ch_score:.2f}")

# Identify and display top anomalous users
print("\n🚨 Top 20 Anomalous Users (Autoencoder):")
top_anomaly_indices = np.argsort(errors_ae)[-20:][::-1]
results_ae = pd.DataFrame({
    'user_id': [user_ids[i] for i in top_anomaly_indices],
    'reconstruction_error': errors_ae[top_anomaly_indices],
    'is_anomaly': anomalies_ae[top_anomaly_indices]
})
print(results_ae.to_string(index=False))
print("\n" + "="*70)
```

Step 6.1 LSTM Autoencoder for Sequential Anomaly Detection

```
print("MODEL 2: LSTM AUTOENCODER")
print("="*70)

print("\n📊 Creating daily sequences (lookback=7 days)...")
lookback = 7
lstm_sequences = {}
X_seq_scaled = None # Initialize to handle cases with no data

if data['logon'] is not None:
    logon_df = data['logon'].copy()
    logon_df['date'] = pd.to_datetime(logon_df['date'], format='%m/%d/%Y %H:%M:%S',
errors='coerce')
    logon_df['date_only'] = logon_df['date'].dt.date

    daily_logon = logon_df.groupby(['user', 'date_only']).agg({
        'id': 'count',
        'pc': 'nunique'
    }).rename(columns={'id': 'daily_logons', 'pc': 'daily_unique_pcs'})

    # Helper logic to create sequences
    sequences = []
    users = []
    for user in daily_logon.index.get_level_values(0).unique():
        user_data = daily_logon.xs(user, level=0).sort_index()
        if len(user_data) >= lookback:
            values = user_data.values
            for i in range(len(values) - lookback + 1):
                sequences.append(values[i:i+lookback])
                users.append(user)

    lstm_sequences['logon'] = (np.array(sequences), users)

# Check if sequences were created
if lstm_sequences and 'logon' in lstm_sequences and len(lstm_sequences['logon'][0]) > 0:
    X_seq, users_seq = lstm_sequences['logon']
    print(f"✅ Sequences created: {X_seq.shape}")

    # Scale sequences
    n_samples, timesteps, features = X_seq.shape
    X_seq_reshaped = X_seq.reshape(-1, features)
    X_seq_scaled = scaler.fit_transform(X_seq_reshaped).reshape(n_samples, timesteps, features)
else:
    print("❌ Not enough sequential data to build the LSTM model.")
```

Step 6.2. Build and Train the LSTM Model

```
# Proceed only if sequences were created
if X_seq_scaled is not None:
    print("\n📐 Building LSTM Autoencoder...")

    # Define model architecture
    n_samples, timesteps, features = X_seq_scaled.shape
    encoding_dim_lstm = 8

    input_layer_lstm = layers.Input(shape=(timesteps, features))
    encoded_lstm = layers.LSTM(64, activation='relu', return_sequences=True)(input_layer_lstm)
    encoded_lstm = layers.Dropout(0.2)(encoded_lstm)
    encoded_lstm = layers.LSTM(32, activation='relu', return_sequences=False)(encoded_lstm)
    encoded_lstm = layers.Dropout(0.2)(encoded_lstm)
    encoded_lstm = layers.Dense(encoding_dim_lstm, activation='relu')(encoded_lstm)
    encoded_lstm = layers.RepeatVector(timesteps)(encoded_lstm)

    decoded_lstm = layers.LSTM(32, activation='relu', return_sequences=True)(encoded_lstm)
    decoded_lstm = layers.Dropout(0.2)(decoded_lstm)
    decoded_lstm = layers.LSTM(64, activation='relu', return_sequences=True)(decoded_lstm)
    decoded_lstm = layers.Dropout(0.2)(decoded_lstm)
    decoded_lstm = layers.TimeDistributed(layers.Dense(features))(decoded_lstm)

    lstm_autoencoder_model = Model(inputs=input_layer_lstm, outputs=decoded_lstm)
    lstm_autoencoder_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
loss='mse', metrics=['mae'])
    print(f"✅ LSTM Autoencoder built - Timesteps: {timesteps}, Features: {features}")

    # Train the model
    print("\n🎓 Training LSTM Autoencoder...")
    history_lstm = lstm_autoencoder_model.fit(
        X_seq_scaled, X_seq_scaled,
        epochs=50,
        batch_size=32,
        validation_split=0.2,
        callbacks=callbacks, # Re-using callbacks from AE
        verbose=1
    )
    print("✅ Training complete")
```

Step 6.3. Evaluate and Visualize LSTM Results

```
if X_seq_scaled is not None:
    # Plot training history
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))
    ax1.plot(history_lstm.history['loss'], label='Training Loss')
```

```
    ax1.plot(history_lstm.history['val_loss'], label='Validation Loss')
    ax1.set(xlabel='Epoch', ylabel='Loss', title='LSTM Autoencoder - Loss')
    ax1.legend()
    ax2.plot(history_lstm.history['mae'], label='Training MAE')
    ax2.plot(history_lstm.history['val_mae'], label='Validation MAE')
    ax2.set(xlabel='Epoch', ylabel='MAE', title='LSTM Autoencoder - MAE')
    ax2.legend()
    plt.tight_layout()
    plt.show()

    # Detect anomalies in sequences
    reconstructed_lstm = lstm_autoencoder_model.predict(X_seq_scaled, verbose=0)
    errors_lstm = np.mean(np.square(X_seq_scaled - reconstructed_lstm), axis=(1, 2))
    threshold_lstm = np.percentile(errors_lstm, 95)
    anomalies_lstm = errors_lstm > threshold_lstm

    # Analyze anomaly distribution
    print("\n📊 Anomaly Detection Analysis (LSTM):")
    n_total_lstm = len(anomalies_lstm)
    n_anomalies_lstm = np.sum(anomalies_lstm)
    anomaly_rate_lstm = n_anomalies_lstm / n_total_lstm * 100
    print(f"  • Detected anomalies: {n_anomalies_lstm} ({anomaly_rate_lstm:.2f}%)")

    # Visualize reconstruction errors
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
    ax1.hist(errors_lstm[~anomalies_lstm], bins=50, alpha=0.7, label='Normal', color='green')
    ax1.hist(errors_lstm[anomalies_lstm], bins=50, alpha=0.7, label='Anomaly', color='red')
    ax1.axvline(threshold_lstm, color='black', linestyle='--', label='Threshold')
    ax1.set(xlabel='Reconstruction Error', title='LSTM Errors - Distribution')
    ax1.legend()
    indices_lstm = np.arange(len(errors_lstm))
    ax2.scatter(indices_lstm[~anomalies_lstm], errors_lstm[~anomalies_lstm], c='green', s=10,
label='Normal')
    ax2.scatter(indices_lstm[anomalies_lstm], errors_lstm[anomalies_lstm], c='red', s=30,
label='Anomaly')
    ax2.axhline(threshold_lstm, color='black', linestyle='--')
    ax2.set(xlabel='Sample Index', title='LSTM Errors - Sample-wise')
    ax2.legend()
    plt.tight_layout()
    plt.show()

    # Top anomalies
    print("\n🚨 Top 20 Anomalous Sequences (LSTM):")
    top_lstm_indices = np.argsort(errors_lstm)[-20:][::-1]
    results_lstm = pd.DataFrame({
        'user_id': [users_seq[i] for i in top_lstm_indices],
        'reconstruction_error': errors_lstm[top_lstm_indices],
```

```
    'is_anomaly': anomalies_lstm[top_lstm_indices]
  })
  print(results_lstm.to_string(index=False))
```

Step 7. Generate Final Risk Report

```
print("\n" + "="*70)
print(" 📋 RISK ASSESSMENT REPORT")
print("="*70)

report = pd.DataFrame()
if data['users'] is not None:
  users_df = data['users'].set_index('user_id')
  report = results_ae.merge(
    users_df[['employee_name', 'role', 'department']],
    left_on='user_id',
    right_index=True,
    how='left'
  )

  behavior_cols = ['total_emails', 'external_emails', 'after_hours_logons']
  report = report.merge(
    user_features[behavior_cols],
    left_on='user_id',
    right_index=True,
    how='left'
  )

  report = report.sort_values('reconstruction_error', ascending=False)

  print("\n 🚨 HIGH RISK USERS (Top 20):\n")
  print(report.head(20).to_string(index=False))

  report.to_csv('risk_assessment_report.csv', index=False)
  print("\n ✅ Full report saved to 'risk_assessment_report.csv'")
```

6. RESULT AND ANALYSIS

6.1. Anomaly Detection Performance

Both models successfully isolated a small percentage of anomalous users/sequences based on a fixed 95th percentile reconstruction error threshold.

Model	Anomaly Rate	Mean Reconstruction Error (Normal)
Standard Autoencoder	5.00%	0.1382
LSTM Autoencoder	5.00%	0.0116

The small mean reconstruction error for normal samples confirms that both deep learning models effectively learned the core patterns of routine user behavior.

6.2. Feature Importance and Comparison

The top features contributing to the high reconstruction error of the static anomalies provide actionable insights into the nature of the detected risk.

Feature	Importance (MAE Score)	Implication
max_email_size	0.445	Extremely large, unusual data transfer attempts via email.
external_email_ratio	0.369	High propensity to send data outside the organization.
std_logon_hour	0.359	Erratic or highly varied logon times, breaking routine.
functional_unit_encoded	0.354	High error on a categorical feature suggests a rare combination of activity for that specific organizational unit.

Comparison of Normal vs. Anomalous Users (Static Model)

Metric	Normal Mean	Anomalous Mean	Difference (%)
total_logon_events	867	1172	+35.2%
after_hours_logons	48	305	**+538.8%**
unique_pcs_logon	32	164	**+409.1%**
external_emails	953	1228	+28.8%

Key Finding: Anomalous users were primarily flagged due to activities outside their established routine, characterized by **significantly higher after-hours logons** and use of an **unusual number of unique computers**.

7. CONCLUSION

This project successfully implemented an unsupervised Deep Learning framework for Insider Threat Detection. By employing a **Standard Autoencoder** for baseline behavioral profiling and an **LSTM Autoencoder** for temporal pattern recognition, the system demonstrated a strong capability to distinguish anomalous user activity from normal operations. The analysis revealed that the most high-risk users exhibit disproportionately high levels of activity (logons, emails) and severe deviations from routine, such as logging in many times after business hours and accessing systems from multiple unique PCs. The reconstruction error serves as a robust, explainable risk score, enabling security teams to prioritize investigations on users demonstrating the most non-conforming behavior.

8. FUTURE SCOPE

To enhance the robustness and practical applicability of this system, the following areas can be explored:

1. **Model Optimization:** Explore alternative architectures like **Variational Autoencoders (VAEs)** or **Isolation Forests** for comparison and improved latent space representation.
2. **Multivariate Sequential Analysis:** Extend the LSTM model to include sequential patterns from other data sources (email and device usage) alongside logon data to capture more complex, cross-domain temporal risks.
3. **Real-Time Scoring:** Develop a streaming pipeline capable of calculating reconstruction errors and risk scores in near real-time, moving the system from a batch processing model to a continuous monitoring solution.
4. **Supervised Integration:** If known breach data becomes available, the latent features extracted by the Autoencoder could be used as inputs for a final, highly accurate supervised classification model (Transfer Learning).

9. REFERENCES

- **Dataset:** https://kilthub.cmu.edu/articles/dataset/Insider_Threat_Test_Dataset/12841247?file=24856979
- **Auto encoders :** https://www.geeksforgeeks.org/machine-learning/auto-encoders/
- **LSTM :** https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/
- **Silhouette Score :** https://www.geeksforgeeks.org/machine-learning/what-is-silhouette-score/
- **Insider Threat Detection: https://www.cisa.gov/topics/physical-security/insider-threat-mitigation/detecting-and-identifying-insider-threats**