A

PROJECT SCHOOL REPORT

on

TRANSFORMER FOR TIME SERIES ANOMALY DETECTION

Submitted by

LADELLA SIRIVALLI	245322733090
KALWA SAI SREEJA	245322733083
KUCHURU MITHILA	245322733030
KOUSTUB MAKTAL	245322733088
V SHIREESHA	245322733123
CHINTHA REDDY NITHIN REDDY	245322733139

Under the guidance of

Mr. K. RAMAKRISHNA

Professor,

Dept. of Humanities and Sciences, KMEC

NEIL GOGTE INSTITUTE OF TECHNOLOGY

Kachavanisingaram Village, Hyderabad, Telangana 500058.

DECEMBER, 2024

NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES)

Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

CERTIFICATE

This is to certify that the project work entitled **"TRANSFORMER FOR TIME SERIES ANOMALY DETECTION"** is a Bonafide work carried out by LADELLA SIRIVALLI (245322733090), KALWA SAI SREEJA (245322733083), KUCHURU MITHILA (245322733030), KOUSTUB MAKTAL (245322733088), V SHIREESHA (245322733123), CHINTHA REDDY NITHIN REDDY (245322733139) of III-year V semester, Bachelor of Engineering in CSE during the academic year 2024-2025 and is a record of Bonafide work carried out by them.

Mr. K. RAMAKRISHNA
Dept. of Humanities and Sciences,
KMEC

ABSTRACT

In recent years, the importance of understanding and analysing time series data has grown significantly due to its wide applications in finance, healthcare, manufacturing, and system monitoring. Time-series data consists of information recorded over time at regular intervals, such as stock prices, sensor readings or website traffic. Anomalies refer to unusual patterns or outliers in data that deviate from expected behaviour. These anomalies can indicate potential issues such as system failures, fraud, or unexpected events, making their detection crucial.

This project focuses on detecting anomalies in time-series data using a Transformer-based model. A transformer is a type of neural network architecture that's used to process sequential data like time-series or text. Transformers are widely used in areas like natural language processing, image processing, and time-series analysis. for anomaly detection in time-series data: the Anomaly Attention Mechanism, Association Discrepancy, and Minimax Strategy.

The Anomaly Attention Mechanism uses two types of associations Series Association and Prior Association to measure how data points relate to each other. Series Association captures the relationships based on the current data, while Prior Association represents predefined expectations based on the positions of the points. Association Discrepancy measures the difference between Series and Prior Associations for each point. The Anomaly Score for each data point is calculated by combining the Reconstruction Error and Association Discrepancy. Anomalies are identified by comparing anomaly scores with threshold and the Minimax Strategy is applied to refine the threshold for anomaly detection.

Contents

LIST OF FIGURES

1. INTRODUCTION

This project addresses the challenge of detecting unusual patterns or rare events in continuous data streams. It focuses on identifying anomalies in time-series data using a Transformer-based architecture. Transformers are highly efficient neural network models that process entire sequences simultaneously, capturing both short-term and long-term dependencies in data. This makes them ideal for analysing time-series data, where patterns and irregularities may span across time intervals.

The project employs advanced techniques such as the Anomaly Attention Mechanism, Association Discrepancy, and the Minimax Strategy to enhance the accuracy of anomaly detection. Anomaly scores are calculated by combining reconstruction errors with association discrepancies, and thresholds are refined using the minimax strategy for better performance. This unsupervised approach eliminates the need for labelled datasets, making it suitable for scenarios where labelled anomalies are scarce.

The model has wide-ranging applications, including industrial equipment monitoring, financial fraud detection, healthcare patient tracking, and space mission data analysis. By automating the monitoring process, it cuts down the need for manual checks and helps make better decisions. The project also provides useful tools, including a trained model to detect unusual patterns, easy-to-use interfaces to see the results, and clear documentation about how well it works.

Traditional methods for time series anomaly detection, such as statistical techniques (e.g., ARIMA and STL), distance-based methods (e.g., KNN and LOF), clustering approaches (e.g., DBSCAN and K-Means), and machine learning models (e.g., Isolation Forest and One-Class SVM), have several limitations like:

Statistical Methods

1. Assumption of Stationarity: Methods like ARIMA assume the data is stationary (constant mean and variance), which may not hold true for real-world time series data.
2. Limited Complexity Handling: They struggle with capturing complex patterns or relationships in multivariate time series data.
3. Sensitivity to Noise: High sensitivity to noise can lead to false anomaly detections.

Distance-Based Methods

1. Scalability: Algorithms like KNN and LOF become computationally expensive as the dataset size grows.
2. Global Perspective: These methods often assume a global structure, making them less effective in handling local anomalies or varying data densities.

3. Parameter Sensitivity: Performance depends heavily on parameter settings, such as the number of neighbours in KNN or LOF

Clustering-Based Methods

1. Predefined Clusters: Techniques like K-Means require the number of clusters to be predefined, which may not always be feasible.
2. Handling Noise: DBSCAN, while robust to noise, may struggle with datasets containing varying densities.
3. Dependence on Distance Metrics: Effectiveness depends on the choice of distance metrics, which might not align with the true data structure.

Transformer is known for its self-attention mechanism, allowing the model to focus on relevant parts of the time series sequence, regardless of their position. This is particularly useful in time series data where the relationships between distant data points are important. The model uses an encoder-decoder setup to reconstruct the time series data. The encoder learns to capture patterns from the input sequence, and the decoder tries to reconstruct it. Anomalies are detected by measuring the reconstruction error (the difference between the original and reconstructed sequences). Larger errors indicate anomalies. The self-attention mechanism helps the model consider long-range dependencies, making it suitable for capturing complex patterns and trends in the data.

Ultimately the Anomaly Transformer is a powerful deep learning model that leverages self-attention mechanisms to capture complex temporal dependencies, making it highly effective for detecting anomalies in large and non-linear time series data. Unlike traditional methods, it offers scalability and flexibility, particularly for intricate and evolving patterns

PROBLEM STATEMENT

Anomalies in time series data (unexpected deviations from typical patterns) can indicate critical events such as equipment failure, fraud, or health risks. The goal of this project is to develop an effective and scalable anomaly detection system for univariate and multivariate time series data using transformer architecture. The transformer’s ability to capture long-range dependencies and complex temporal patterns makes it a promising candidate for this task.

The project would be taken up in the following steps:

1. Gain knowledge on Transformer Architecture and uses.
2. Download the Time Series datasets.
3. Process the above downloaded dataset to remove any missing data, noisy data etc.
4. Develop a transformer architecture tailored to time series data.
5. Train the model with Optimized hyperparameters like attention heads, sequence length, and learning rate for time series data characteristics.
6. Implement anomaly scoring methods such as thresholding attention weights, reconstruction loss, or predictive confidence intervals and Flag anomalies.
7. Test robustness to noise, outliers, and non-stationarity.
8. Analyse model decisions to build trust in high-stakes applications.

2. LITERATURE SURVEY

1.TRANSFORMER model for time series anomaly detection

Paper: ***<u>ANOMALY TRANSFORMER: TIME SERIES ANOMAY DETECTION WITH ASSOCIATION DISCREPANCY.</u>***

→Authors: Jiehui Xu; Jianmin Wang; Mingsheng Long (School of software, BNRist, Tsinghua University, China).

→The document discusses the "Anomaly Transformer," for unsupervised time series anomaly detection. This method introduces an "Anomaly-Attention" mechanism, which models temporal associations to distinguish between normal and abnormal data points. By leveraging Association Discrepancy (a measure of the deviation between prior- and series-associations) this approach effectively identifies anomalies. Key innovations include a two-branch structure for attention and a minimax strategy to amplify distinctions between normal and abnormaldata.

2. Attention is All You Need

Paper**: *<u>Attention is All You Need.</u>***

→Authors: Ashish Vaswani, NoamShazeer, Niki Parmar, JakobUszkoreit, N. Gomez, Łukasz Kaiser.

→The abstract of the document outlines the introduction of the Transformer model, a novel architecture that eschews recurrence and instead relies entirely on an attention mechanism to draw global dependencies between input and output.

→This approach significantly improves efficiency by allowing for parallelization, which addresses some limitations of RNN-based models, such as long training times. The Transformer model achieves new state-of-the-art results in translation tasks and shows that attention mechanisms alone can produce high-quality results.

3.Artificial Neural Network

Paper***: <u>Build an Artificial Neural Network from Scratch: Part 1.</u>***

→Author: Nagesh Singh Chauhan.

→ An Artificial Neural Network (ANN) is a computational model inspired by the human brain, consisting of layers of interconnected nodes called neurons. Each neuron processes input data by applying a weighted sum followed by an activation function, enabling it to learn complex patterns and relationships. ANNs are structured into input, hidden, and output layers, with the hidden layers allowing the network to model non-linear functions.

→The article is about building of ANN from scratch using only the numpy Python library.part-1 builds a fairly easy ANN with just having 1 input layer and 1 output layer and no hidden layer.

4. Artificial Neural Network

Paper***:*** ***Build an Artificial Neural Network from Scratch: Part 2.***

→Author: Nagesh Singh Chauhan.

→creating a simple neural network with one input and one output layer, from scratch in Python. Such a neural network is called a perceptron. However, real-world neural networks, capable of performing complex tasks such as image classification and stock market analysis, contain multiple hidden layers in addition to the input and output layer.

5. Anomaly Detection in Time Series

Paper: ***Anomaly Detection in Time Series: A Comprehensive Evaluation.***

→Authors: Sebastian Schmidl, Phillip Wenig, Thorsten Papenbrock.

→scientific study carefully evaluates most state-of-the-art anomaly detection algorithms. The research team collected and re-implemented 71 anomaly detection algorithms from different domains and evaluated them on 976 time series datasets. The algorithms have been selected from different algorithm families and detection approaches to represent the entire spectrum of anomaly detection techniques. In the paper, concise overview of the techniques and their commonalities have been provided and evaluating their in correlation anomaly 1 0 400 450 500 550 600 650 700 750 800 (b) Synthetic multivariate time series with a correlation anomaly and the scoring of k-Means. Individual strengths and weaknesses and, thereby, consider factors, such as effectiveness, efficiency, and robustness.

6. ANOMALY DETECTION IN UNIVARIATE TIME-SERIES

Paper: ***ANOMALY DETECTION IN UNIVARIATE TIME-SERIES: A SURVEY ON THE STATE-OF-THE-ART.***

→Authors: MohammadBraei, Dr.-Ing. Sebastian Wagner.

→ This paper studies 20 univariate anomaly detection methods from the all three categories. The evaluation is conducted on publicly available datasets, which serve as benchmarks for time-series anomaly detection. By analysing the accuracy of each method as well as the computation time of the algorithms, and provide a thorough insight about the performance of these anomaly detection approaches, alongside some general notion of which method is suited for a certain type of data.

7.Deep Learning and Classical Methods

Paper: ***IS IT WORTH IT? COMPARING SIX DEEP AND CLASSICAL METHODS FOR UNSUPERVISED ANOMALY DETECTION IN TIME SERIES.***

→Authors: Ferdinand Rewicki, JoachimDenzler, ulia Niebling.

→ Each method has its strengths in detecting certain types of anomalies. In this study, we compare six unsupervised anomaly detection methods of varying complexity to determine whether more complex methods generally perform better and if certain methods are better suited to certain types of anomalies.

→ Evaluating the methods using the UCRanomaly archive, a recent benchmark dataset for anomaly detection and analysed the results on a dataset and anomaly type level after adjusting the necessary hyperparameters for each method.

3.TECHNOLOGY STACK

- **Programming Languages: Python**

  Python is the primary language used in our project due to its extensive support for data science and machine learning libraries. It's widely used for tasks like text processing, model training, and evaluation. Python is used for training and fine-tuning the Transformer model using PyTorch. Additionally, libraries like NumPy are used for numerical computations, while Matplotlib is employed for data visualization and plotting. Sklearn is also utilized for various machine learning tools, such as metrics and preprocessing functions.

- **Data Processing Libraries:**

  → Standard python for built-in-data structures (like list and dictionaries).

  →Csv Module for reading and writing CSV files.

  → JSON Module for working with JSON data.

  →Pandas for data manipulation and analysis (reading, cleaning, and transforming data).

- **REACT**:(Frontend)

  Frontend is build using React.React is a popular JavaScript library for building user interfaces. It allows developers to create reusable UI components that update dynamically based on data changes, making web applications faster and more interactive. React uses a virtual DOM for efficient rendering. React is a declarative, component-based JavaScript library designed for building scalable, interactive, and dynamic user interfaces, primarily for web applications.

- **EXPRESS.JS**(Backend)

  Express.js is a lightweight and flexible Node.js web application framework commonly used for building APIs and web servers. In the context of integration, it acts as a middleware layer between a frontend (e.g., React) and a backend (e.g., a database or AI model). Express.js simplifies handling HTTP requests, routing, and data validation.

- **Data Visualization:** Matplotlib is a powerful plotting library in Python that allows you to visualize data in various forms. Matplotlib is a versatile Python library for creating basic and advanced plots. Matplotlib is a plotting library for creating static, interactive, and animated visualizations. Used for visualizing model performance, such as confusion matrices, and precision-recall plots and to create graphs to analyse data distribution and feature importance.

- **Machine Learning:**

  →Manually implement the transformer architecture using only Python's built-in classes and functions.

  →We need to implement Attention mechanisms, Attention mechanisms are a powerful concept in machine learning that enable models to focus on specific parts of input data when making decisions.

  →We need to implement Feed Forward Neural Networks, It consists of layers of neurons that are connected in a directed manner, where information moves in one direction, from the input layer to the output layer, passing through one or more hidden layers.

  → We need to implement Backpropagation; it adjusts the weights of the network by minimizing the error (or loss) between the predicted output and the actual target using a method called gradient descent.

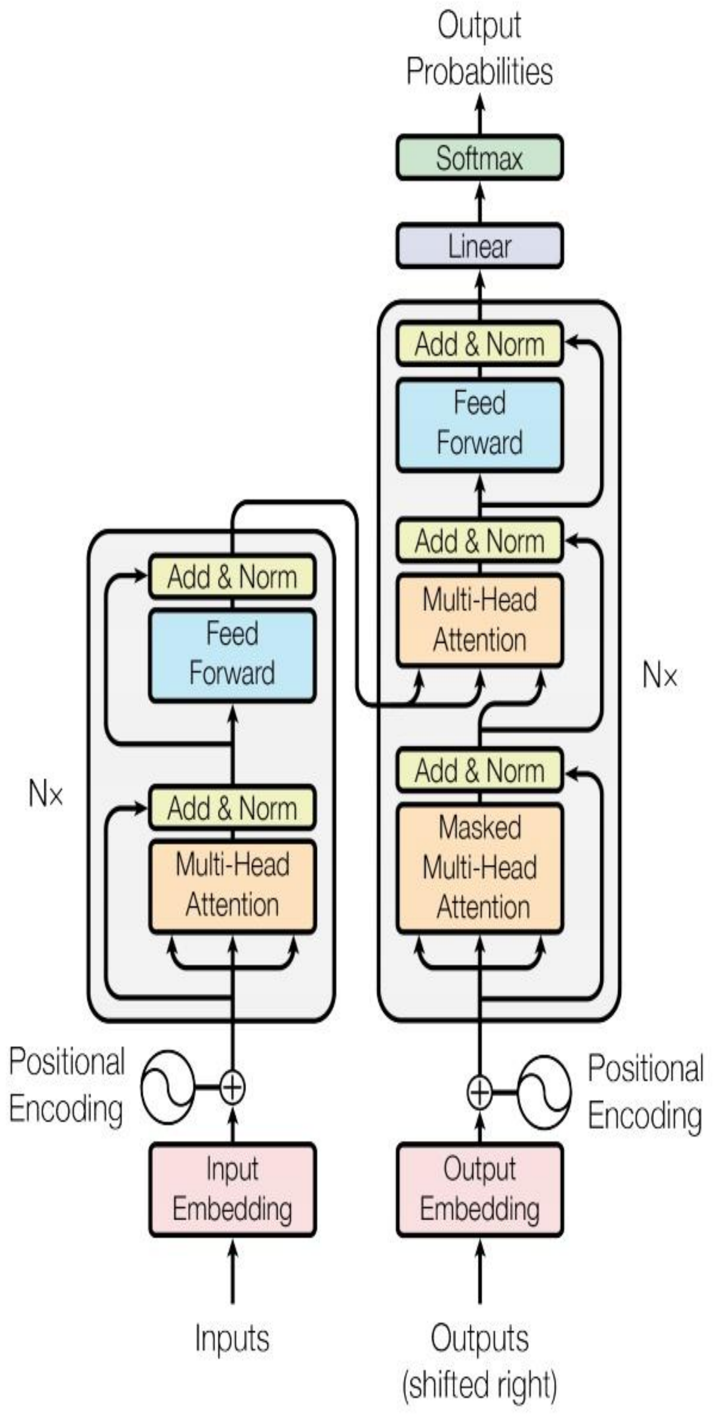
- **Anomaly Detection**: Custom Algorithms like Z-score method, which is a statistical measure that quantifies how far a data point is from the mean of the dataset, measured in standard deviations etc.

- **Data Storage:**

  →**File-based Storage**: Use plain text files for saving and loading data, manually implementing functions for file reading and writing.

- **Development Tools:** Uses standard IDE like Vs Code because VS Code is highly customizable and offers a rich ecosystem of extensions that enhance its functionality and it has features like Cross-Platform, Intelligent Code Completion, Integrated Terminal, Source Control Integration, Debugging, etc.

TRANSFORMER ARCHITECTURE



DESCRIPTION ABOUT THE ABOVE FIGURE:

The image depicts the architecture of the **Transformer model**, introduced in the seminal paper *"Attention Is All You Need"* by Vaswani et al. in 2017. The Transformer is a deep learning architecture widely used for tasks such as machine translation, text generation, and many other natural language processing (NLP) and vision applications. The above diagram illustrates the two main components of the Transformer. They are **Encoder** (on the left) which Processes the input sequence (e.g., words or tokens) and generates a sequence of contextualized representations for each input token. **Decoder** (on the right) which Takes the output sequence (shifted to avoid looking at future tokens) and uses both its own self-attention and the encoder's output to generate predictions token by token.

1.Encoder:

→**Inputs**: The raw input sequence is fed into the encoder. Each input element typically represents a token in a sentence.

→**Input Embedding**: Each token is converted into a continuous dense vector representation (embedding). This step maps discrete tokens into a high-dimensional space, where semantically similar tokens have closer vector representations.

→**Positional Encoding**: Since the Transformer architecture processes tokens in parallel (not sequentially), positional encodings are added to embeddings to encode order information. These encodings use sinusoidal functions, with each position assigned a unique encoding based on its index. The result is a sum of the token embedding and its positional encoding.

→**Add & Norm**: A **residual connection** is applied: the output of the previous layer is added to the input. This helps prevent the vanishing gradient problem and stabilizes learning. **Layer Normalization** (applied after addition) ensures normalized activations, leading to faster convergence and better performance.

→**Multi-Head Attention**: This mechanism computes how much focus (attention) each token should have on every other token in the sequence. **Multi-head** means multiple attention heads compute attention in parallel, focusing on different relationships or patterns in the data. It outputs a weighted sum of input embeddings, allowing the model to capture global context.

→**Add & Norm**: Another residual connection and normalization are applied after the attention mechanism to preserve gradients.

→**Feed Forward**: A position-wise feed-forward network (FFN) processes the attention output independently for each position. Typically consists of - A linear transformation to a higher-dimensional space, A ReLU (or other activation function) for non-linearity, A second linear transformation back to the original dimensionality.

→**Add & Norm**: Residual connection and normalization are applied again after the feed-forward layer.

→**N x (Encoder Layers)**: The above sequence of **Multi-Head Attention → Add & Norm → Feed Forward → Add & Norm** is repeated **N times** (where **N** is the number of encoder layers). Each layer allows the encoder to progressively build a richer representation of the input.

2.DECODER:

→**Outputs (Shifted Right):** The target sequence (e.g., a sentence in the output language) is shifted right to prevent the decoder from seeing future tokens during training.

→**Output Embedding**: Similar to the encoder, the shifted tokens are converted into dense vector embeddings.

→**Positional Encoding**: Positional information is added to the output embeddings.

→**Add & Norm**: Residual connections and normalization are applied to the embeddings with positional encodings.

→**Masked Multi-Head Attention**: This attention mechanism is restricted so that a token can only attend to earlier tokens in the sequence (self-attention). This ensures causal relationships are preserved during generation.

→**Add & Norm**: Residual connection and normalization are applied after masked multi-head attention.

→**Multi-Head Attention (Cross-Attention)**: The decoder attends to the encoder's outputs, allowing it to condition its generation on the encoded input sequence.

→**Add & Norm**: Another residual connection and normalization are applied after cross-attention.

→**Feed Forward**: A feed-forward neural network is applied to refine the representations.

→**Add & Norm**: Residual connection and normalization follow the feed-forward layer.

→N **x (Decoder Layers)**: The combination of Masked Multi-Head Attention, Add & Norm, Multi-Head Attention, Add & Norm, Feed Forward, and Add & Norm is repeated **N times**, similar to the encoder.

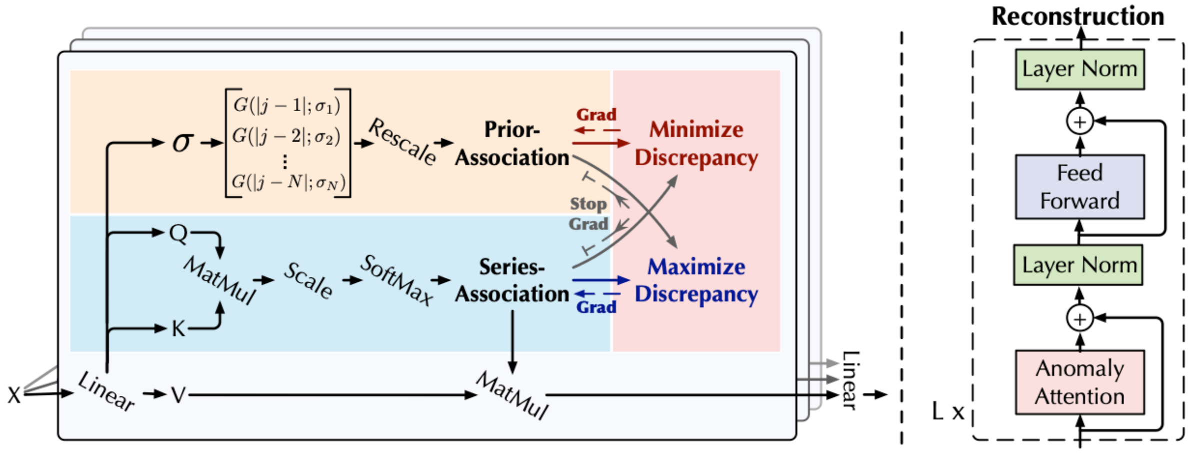
3.Output Layer:

→**Linear**: The final decoder representations are passed through a linear layer that maps them to the vocabulary size.

→**SoftMax**: A SoftMax function converts the scores into probabilities, indicating the likelihood of each token in the vocabulary as the next output.

→**Output Probabilities**: The final output is the predicted probability distribution over the vocabulary for each position in the target sequence.

Gaussian Kernal:



A **Gaussian kernel** in the context of transformers refers to a mathematical approach to model interactions or relationships in the attention mechanism using a Gaussian function. This technique can replace or augment traditional attention mechanisms to achieve specific advantages, such as computational efficiency or improved alignment with certain types of data.

A Gaussian kernel is a function defined as:

$$k(x, y) = exp\left(\frac{-|x - y|^2}{2\sigma^2}\right)$$

Where:

- x, yx, yx, y are input vectors.
- ∥x−y∥\|x - y\|∥x−y∥ is the Euclidean distance between the vectors.
- σ\sigmaσ is a parameter that controls the width of the Gaussian (or the kernel's sensitivity).

The Gaussian kernel is a measure of similarity:

- It outputs values close to 1 for similar inputs (x≈yx \approx yx≈y).
- It outputs values close to 0 for dissimilar inputs.

Transformers typically use dot-product attention, where attention scores are calculated as:

$$\text{Attention (Q, K, V) =softmax} \left(\frac{Qk^T}{\sqrt{d_k}}\right) v$$

Here:

- Q, K, V Q, K, VQ, K, V are query, key, and value matrices.
- The dot product QKTQK^TQKT captures pairwise similarities between queries and keys.

Using a Gaussian kernel instead introduces a non-linear similarity function. This has several potential advantages:

1. **Smooth Attention**: Gaussian functions can smooth attention distributions, reducing noise and focusing on relevant interactions.
2. **Scalability**: Gaussian kernels can approximate attention with fewer computations, improving efficiency for large datasets.

3. **Better Locality Bias**: Particularly for sequential data, Gaussian kernels naturally emphasize local relationships, which can improve performance on tasks like text or speech processing.

Benefits of Using Gaussian Kernels:

1. **Reduced Computational Complexity**: For large inputs, Gaussian kernels can reduce the complexity from O(n2) O(n^2) O(n2) (dot-product attention) to O(n)O(n)O(n) using efficient approximations.
2. **Improved Interpretability**: Gaussian similarity provides a more interpretable way to measure relationships between tokens.
3. **Flexibility**: The σ\sigmaσ parameter allows tuning the model to balance between local and global attention.

Association Discrepancy:

Association Discrepancy refers to the difference between the attention distribution of a time-series data point (based on observed relationships) and its contextual associations (expected relationships).

- **Normal Data**: In regular patterns, the associations are consistent, and the discrepancy is low. Attention weights align with expected relationships in the time-series context.
- **Anomalous Data**: For anomalies, the associations deviate significantly from expectations, leading to high discrepancy.

In the Anomaly Transformer, the attention mechanism computes two distributions for each data point:

1. **Actual Association (Actual_{actual}Actual)**: Represents observed attention patterns based on the input data.
2. **Expected Association (AexpectedA_{expected}Aexpected)**: Represents the theoretical or contextual association based on learned temporal or spatial patterns.

The **Association Discrepancy** is calculated as:

$$D(x_t) = A(x_t) - A_i(x_t)$$

Where:

- D(xt)D(x_t) D(xt): Association discrepancy for data point xtx_txt.
- ∥·∥p\|\cdot\|_p∥·∥p: Norm used to quantify the difference (commonly L2L_2L2-norm or L1L_1L1-norm).
- Actual_{actual}Actual: Observed attention distribution derived from the input sequence.
- AexpectedA_{expected}Aexpected: Expected attention distribution derived from contextual embeddings.

Benefits of Association Discrepancy:

Enhanced Interpretability:

- By isolating anomalous patterns as high discrepancies, it becomes easier to identify and understand anomalies in complex time-series data.

Context-Aware Detection:

- Incorporating expected associations ensures that anomalies are evaluated relative to the contextual behaviour of the time series.

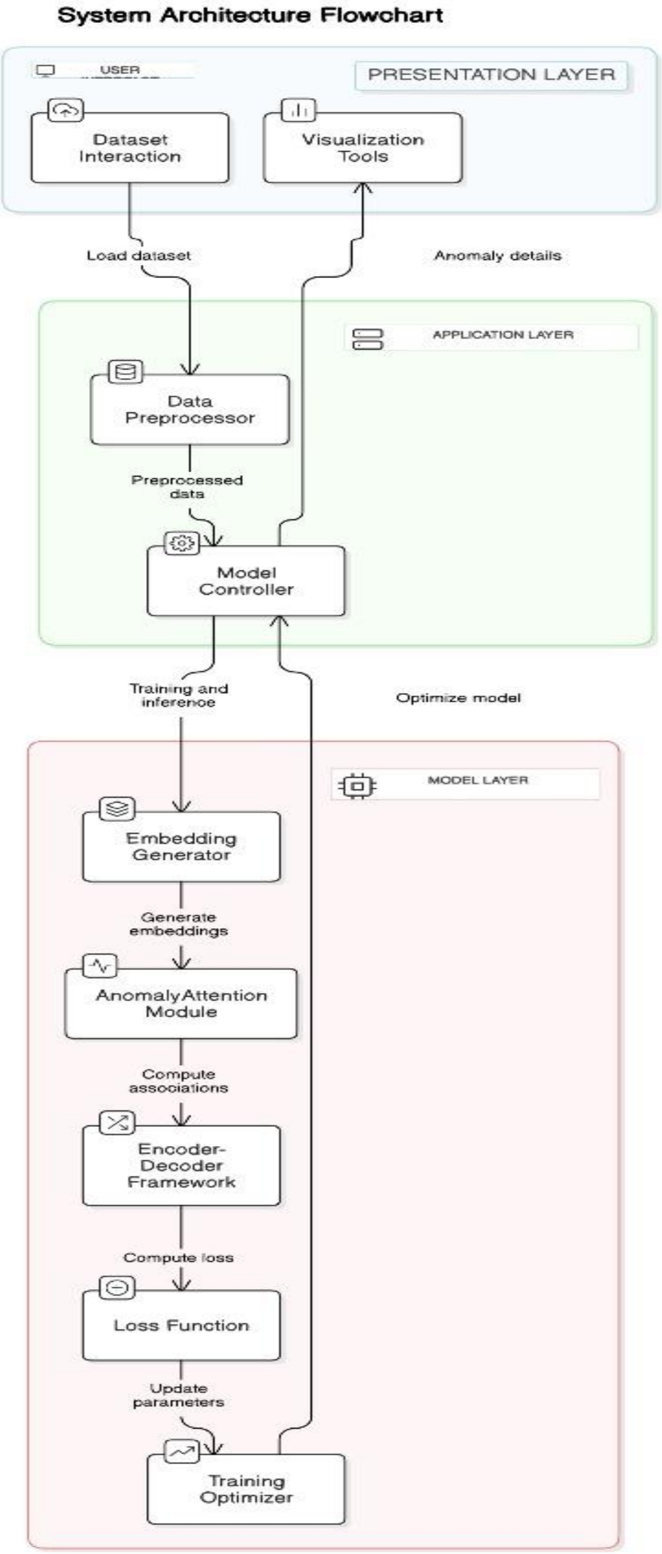
Robustness:

- The discrepancy mechanism is less sensitive to noise and focuses on meaningful deviations, reducing false positives.

Scalability:

- The transformer architecture enables efficient computation of discrepancies even for large-scale datasets.

ARCHITECTURE DIAGRAM



DESCRIPTION ABOUT ARCHITECTURE DIAGRAM:

→USER INTERFACE

- **Dataset Interaction:**

  Description: This is the entry point where users interact with the system to upload and manage datasets. It supports operations like dataset selection, format validation, and initial inspection of the dataset structure (e.g., headers, columns, etc.).

  Objective: Allow users to select and load datasets into the system for analysis and model training.

- **Visualization Tools**:

  Description: Tools for displaying data and anomaly details using graphs, charts, or other visual representations. These tools allow for detailed inspection of trends, distributions, and detected anomalies.

  Objective: Help users visually understand the dataset and identify anomalies or patterns effectively.

→ APPLICATION LAYER

- **Data Preprocessor:**

  Description: This module ensures the dataset is clean and formatted correctly. Key steps include handling missing values, scaling numerical values, encoding categorical variables, and transforming the data into a consistent format. It may also include techniques like time-series normalization or feature extraction.

  Objective: Ensure the data is clean and structured to improve the performance of downstream models.

- **Model Controller**:

  Description: Acts as a coordinator for the application. It handles communication between the pre-processed data and the model. It manages training, testing, and inference workflows, ensuring that the appropriate model configurations and parameters are used.

  Objective: Manage the workflow between data preprocessing, model training, and anomaly detection.

→MODEL LAYER:

- **Embedding Generator:**

  Description: Converts raw input data into low-dimensional embeddings. These embeddings are dense, numerical representations that retain the essential characteristics of the data, making them suitable for machine learning tasks.

  Objective: Generate compact and informative representations of the data for anomaly detection.

- **Anomaly Attention Module**:
  Description: Focuses on specific regions or aspects of the data that are more likely to contain anomalies, using attention mechanisms.
  Objective: To improve the precision and robustness of anomaly detection by emphasizing critical data regions or attributes that exhibit irregular behavior.
- **Encoder-Decoder Framework**:
  Description: This architecture is designed for learning compact data representations and reconstructing the original data. The encoder compresses the input data into a latent space, while the decoder reconstructs it. Reconstruction errors are used as an indicator of anomalies.
  Objective: Measure reconstruction errors to detect anomalies, as anomalies typically have higher errors.
- **Loss Function**:
  Description: A mathematical function that evaluates how well the model is performing. It quantifies the difference between the predicted outputs and actual values or reconstruction errors.
  Objective: Optimize the model by minimizing the error and improving anomaly detection accuracy.
- **Training Optimizer**:
  Description: This component updates the model's weights and parameters based on the loss function using optimization algorithms like stochastic gradient descent (SGD), Adam, or RMSprop. It ensures efficient convergence to an optimal solution.
  Objective: To iteratively improve the model’s ability to detect anomalies by minimizing the loss function, ensuring better performance with each training epoch.

→FLOW:

Objective: The system ensures seamless interaction between user input, data processing, model training, and anomaly detection to provide accurate and interpretable results.

PROPOSED WORK AND IMPLEMENTATION

To begin the project, it's crucial to plan and set up the environment effectively. This should include formation of a clear objective, scope, and deliverables. In addition, the setup of the development environment is critical; this includes setting the programming language as Python because Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It is widely used in fields like web development, data analysis, artificial intelligence, and scientific computing due to its extensive libraries and community support.

The User Interface of Time series analysis using Transformer is developed as follows:

We use React as our project Frontend because React is a popular JavaScript library for building user interfaces. It allows developers to create reusable UI components that update dynamically based on data changes, making web applications faster and more interactive. React uses a virtual DOM for efficient rendering. React is a declarative, component-based JavaScript library designed for building scalable, interactive, and dynamic user interfaces, primarily for web applications.

React Frontend Components:

1. Dataset Uploader Component

- Purpose: This component allows users to upload a dataset file (e.g., CSV) to the backend. It interacts with the Flask backend's /upload endpoint.
- Key Features:
    - Provides a file selection interface.
    - Sends the selected file to the backend using a POST request.
    - Displays a success or error message based on the upload status.

2. Visualization Component

- Purpose: This component retrieves processed data or anomaly detection results from the backend and visualizes it. The backend serves these results via an API endpoint like /get-visualization-data.
- Key Features:
    - Fetches visualization data (e.g., detected anomalies or pre-processed dataset) from the backend.
    - Renders dynamic charts (e.g., line plots, bar charts) using libraries like Chart.js or Recharts.
    - Offers interactivity for the user to analyse results.

3. App Component

- Purpose: The App component acts as the main entry point and container for all other components. It integrates the dataset uploader and Visualization components to create a cohesive user experience.
- Key Features:
  - Manages routing and layout.
  - Connects the upload and visualization workflows.

Component Workflow:

1. ***Dataset Uploader:***
   - User selects a dataset file using a file input field.
   - The file is sent to the backend using a POST request to /upload.
   - Displays the status of the upload (success or failure).
2. ***Visualization:***
   - After processing, the user triggers anomaly detection (optional, via a button or automatic).
   - Fetches processed results from the /get-visualization-data API endpoint.
   - Renders results dynamically as charts.
3. ***App:***
   - *Manages the layout and combines the upload and visualization components for a seamless workflow.*

The Backend of Time series analysis using Transformer is developed as follows:

Thus, User Interface is developed which is required for loading the data.

Next comes building the Model that ensures the detection of anomalies in the time series data which is the most important aspect of Transformer.

A Transformer is a deep learning model which is a neural network architecture that relies on self-attention mechanisms to process sequential data. While Transformers have become foundational for many LLMs (such as GPT and BERT), they are a general-purpose deep learning architecture that can be used for various tasks beyond language processing, including time series analysis, image recognition, and more.

Transformers process data **in parallel** during training. Unlike traditional sequential models like RNNs or LSTMs, which process data step-by-step, Transformers use self-attention mechanisms that allow them to analyse all time steps or tokens in the input sequence simultaneously. This parallel processing significantly speeds up training and makes Transformers more efficient for handling long sequences, as they do not rely on the sequential nature of the data.

Why Transformer?

- **Long-Range Dependencies**: Transformers excel at capturing long-range dependencies within data, making them ideal for tasks like time series analysis or language modeling, where relationships across distant elements are crucial.
- **Parallel Processing**: Unlike sequential models, Transformers process input data in parallel, significantly speeding up training and making them scalable for large datasets and complex tasks.
- **Flexibility**: Transformer models can be adapted for various applications beyond their initial use in NLP, including time series anomaly detection, image processing, and more, offering versatility across domains.
- **Scalability**: Due to their ability to handle large amounts of data efficiently, Transformer models can scale up to handle massive datasets, improving performance for complex real-world tasks.

Here's a detailed explanation of each step followed.

1. *Importing Necessary Libraries:*

The environment is set up by importing various libraries essential for data analysis, manipulation, and machine learning. The libraries include pandas which is a powerful library for data manipulation and analysis and it provides data structures like DataFrame and Series that make it easy to load, clean, and transform data; Numpy A core library for numerical computing in Python. It supports large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. It is essential for performing numerical operations, linear algebra, and scientific computing; matplotlib a widely-used plotting library that provides tools for creating static, animated, and interactive visualizations. It supports a wide range of chart types, including line plots, bar charts, scatter plots, and histograms, allowing users to visualize data and analysis results.

2. Loading the Dataset:

The **Bitcoin Historical Dataset** is loaded, containing Bitcoin price data from 2018 to 2024. This dataset has over, 15,000 rows with columns such as timestamp, open, high, low, close, and volume. The data is pre-processed, ensuring the timestamp and price data are consistently formatted for time series analysis.

Dataset Overview:

This dataset contains historical price data for Bitcoin (BTC/USDT) from January 1, 2018, to December 10, 2024. The data was sourced using both the Binance API and Yahoo Finance, providing granular candlestick data in four timeframes:

- 1-hour (1H) - 2536 rows
- 4-hour (4H) - 60.7k rows
- 1-day (1D) - 15.2k rows
- 14-minutes (15M) - 243k rows

General Description:

- **Time Frame**: 2018 through October 10, 2024.
- **Granularity**: Data is aggregated at 15-minute intervals.
- **Currency**: Typically, in USD (United States Dollar).

This dataset includes the following fields for each timeframe:

- **Open time:** The timestamp for when the interval began.
- **Open price:** The price of Bitcoin at the beginning of the interval.
- **High price:** The highest price during the interval.
- **Low price:** The lowest price during the interval.
- **Close price:** The price of Bitcoin at the end of the interval.
- **Volume:** The trading volume during the interval.
- **Close time: The timestamp for when the interval closed.**

3. Dropping Rows with Null Values and Converting Columns:

Rows with missing values in essential columns like timestamp or price are removed. This cleaning step ensures that only complete and valid data is used in model training. Once cleaned, all data is converted into suitable formats: timestamps are converted to **datetime** objects, and price data is converted to **float** for further numerical operations.

4. Removing Invalid Values:

The dataset is filtered to ensure that the columns, such as price (open, close, high, low), contain valid numeric values. Invalid or erroneous entries, such as negative values in price or zero trading volumes, are removed to avoid skewing the model's training.

5. Converting Columns to Appropriate Formats:

After filtering out invalid values, columns like the timestamp (converted to datetime) and price (converted to float) are processed to ensure compatibility with machine learning algorithms.

For time series modelling, it is crucial to maintain proper formats for indexing, sequencing, and feature extraction.

6. Resetting the Index:

After removing rows and cleaning the dataset, the index might become disordered or contain gaps. The index is reset to maintain continuity, which helps in ensuring proper time series analysis and avoids any indexing errors during model training.

7. Defining Features for Transformer Model:

Transformers require sequential input data. For time series forecasting or anomaly detection, features like:

- **Price differences** (e.g., between consecutive time points),
- **Moving averages** (e.g., 5-day, 10-day, etc.),
- **Volatility indicators** (e.g., rolling standard deviation), are often engineered. These features capture temporal trends and volatility, which help the model identify patterns and anomalies in Bitcoin price movements.

8. Data Normalization and Scaling:

Before feeding data into the transformer model, the features are normalized or standardized. Scaling methods like **Min-Max Scaling** or **Standardization** (z-score) are applied to ensure all features are on the same scale, which helps in improving the model's performance and *convergence speed.*

9. Preparing Data for Transformer Model:

The data is then prepared in a way suitable for transformer input. This involves:

- **Sequence creation**: Time series data is broken into sequences (sliding window approach) of a fixed length (e.g., 30 days).
- **Input and output creation**: For forecasting, the input might be past prices, and the output is the next day's price or sequence prediction.
- **Anomaly labels (if available)**: For semi-supervised anomaly detection, the dataset may include a small set of labelled anomalies (e.g., sudden price spikes or drops).

10. Building the Transformer Model:

A **Transformer model** is built to process the time series data. The model typically includes:

- **Input embedding layer**: To transform input data into embeddings suitable for the transformer.
- **Positional Encoding**: To capture the temporal ordering of the data.
- **Encoder layers**: A stack of self-attention and feed-forward layers that help the model learn temporal dependencies.

- **Output layer**: Depending on the task (forecasting or anomaly detection), this layer predicts the next value or classifies anomalies.

11. Training the Model:

The model is trained using the training dataset. For **anomaly detection**, the model can be trained to predict normal price patterns, and anomalies are detected based on large deviations from predictions. For **forecasting**, the model learns the temporal dependencies to predict future Bitcoin prices.

Training might use:

- For anomaly detection, a reconstruction loss (for autoencoder-based transformers) or classification loss (binary cross-entropy for anomaly detection) can be used.
- **Adam optimizer** for efficient training.
- The model is trained using various datasets of different sizes and rows in same and different domain.
- ***12. Evaluation:***

The trained model is evaluated metrics like Precision, Recall, F1-Score, Accuracy and the AUC-ROC curve are computed to evaluate how well the model identifies anomalies in Bitcoin prices.

13. Visualization:

Visualizations are created to help interpret the results:

-
- **Anomaly Detection Plots**: For anomaly detection, visualizations show where the anomalies occurred (e.g., large price deviations or sudden spikes).

COMMUNICATION BETWEEN USER INTERFACE AND TRANSFORMER MODEL

Express.js is a minimalist web framework for Node.js that facilitates building APIs and web servers. When integrating a React frontend with a Transformer model backend, Express.js serves as the middleware, enabling smooth communication between these two components.

Role of Express.js in Integration:

Express.js acts as the intermediary layer that:

- Handles requests from the React frontend.
- Processes and validates the data.
- Communicates with the Transformer model backend.
- Sends responses back to the React frontend in a structured format.

Workflow of Integration:

1. **User Input**: A user submits input through the React app (e.g., a sentence for text summarization).
2. **Frontend Request**: React sends the input data to the Express.js server via an HTTP request (e.g., POST).
3. **Request Handling**: Express.js receives the request, validates and preprocesses the input.
4. **Model Invocation**: The Express.js server forwards the processed input to the Transformer model (e.g., via an API call or direct Python execution).
5. **Response Handling**: The model processes the input and sends a response back to Express.js.
6. **Frontend Update**: Express.js formats the response and sends it back to the React app, where the result is displayed to the user.

Working Flow in a nutshell:

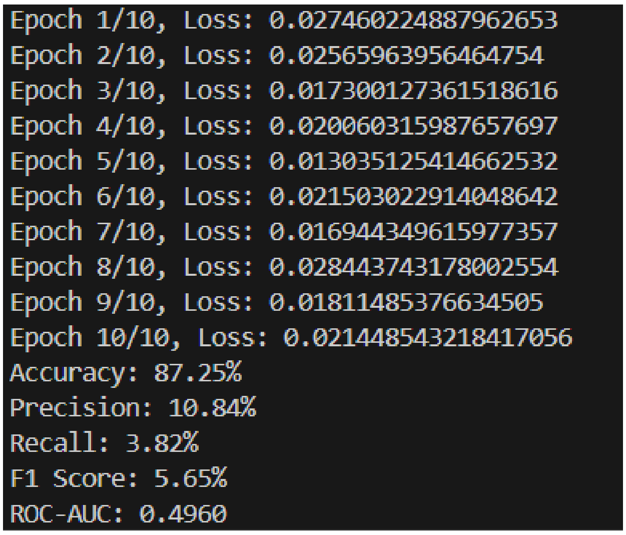
1. User Action: A user submits a time series dataset or a sequence of data points through the user interface.
2. Data Transmission: The time series data is transmitted to the backend server from the UI.
3. Pre-processing: The backend pre-processes the time series data, including Normalization, Tokenization etc.
4. Model Interaction: The pre-processed time series data is fed into a fine-tuned transformer model designed for anomaly detection.
5. Model Output: The transformer model outputs a probability score or a classification indicating whether the data point (or the sequence) is normal or an anomaly.
6. Result Display: The backend sends the model's prediction back to the UI. The result is displayed to the user, such as showing whether a particular time series data point is normal or anomalous.

RESULTS AND DISCUSSIONS

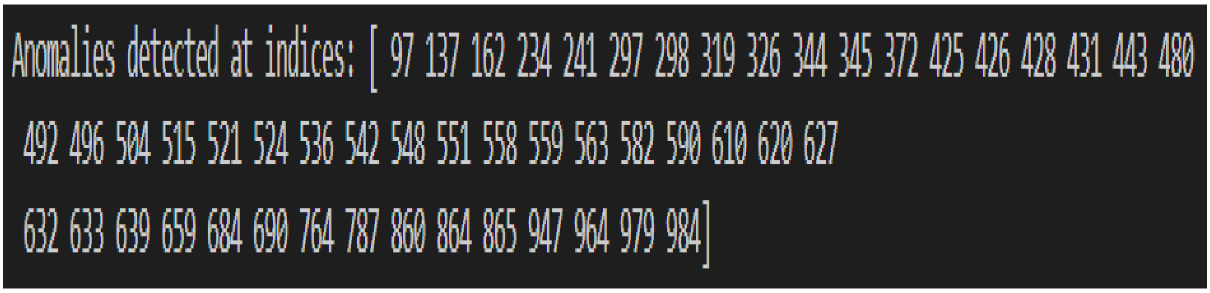
The Transformer-based model was trained on the Bitcoin historical dataset (2018-2024) to detect anomalies in Bitcoin prices and to predict future price trends. The dataset was split into training and testing sets, and various preprocessing steps.

Types of Anomalies We Can Detect are Point Anomalies (Sudden spikes or drops in Bitcoin prices (e.g., flash crashes)); Contextual Anomalies (Price fluctuations that are unusual for a specific time of day or season); Collective Anomalies (Sequences of events such as price trends deviating significantly from expected behavior).We mainly focused on sudden spikes or drops in bitcoin prices.

1)A basic Transformer architecture was implemented for anomaly detection on the Bitcoin 4-hour dataset. The model was built without using backpropagation or any third-party libraries and followed simple anomaly attention mechanism and following are the outputs of my model.

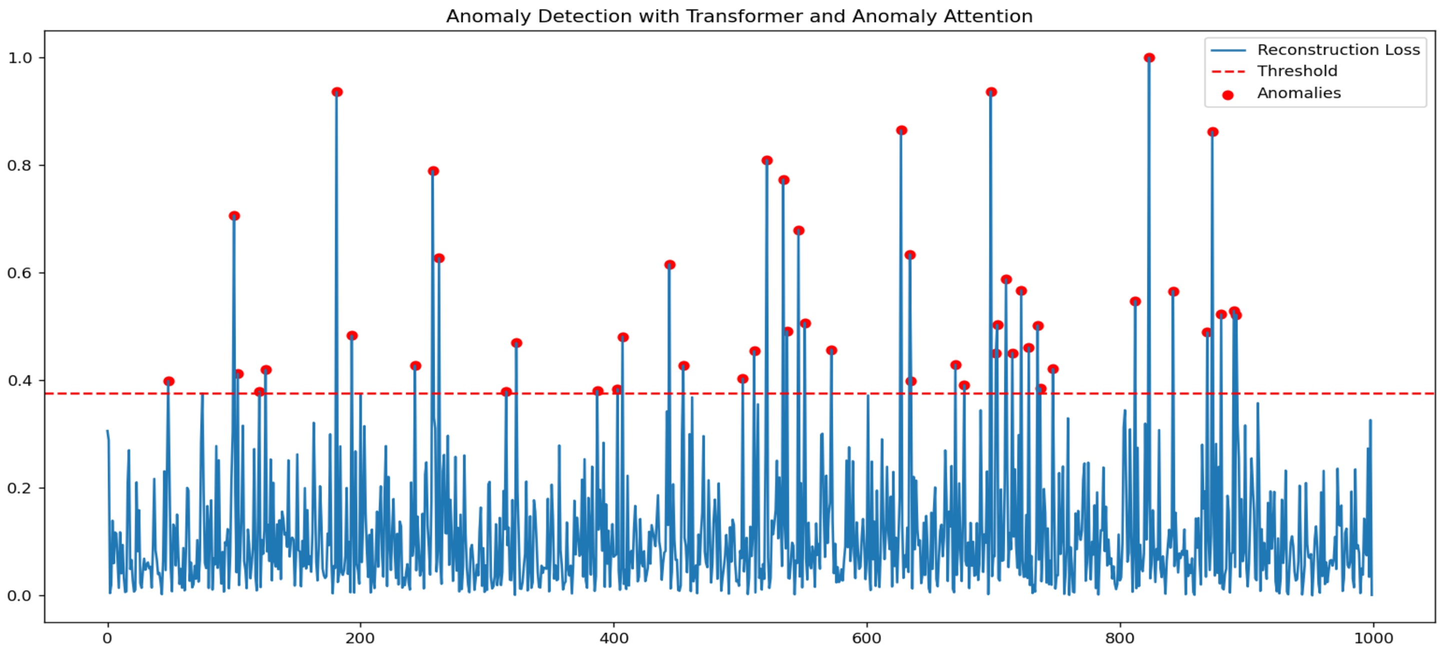


```
Epoch 1/10, Loss: 0.027460224887962653
Epoch 2/10, Loss: 0.02565963956464754
Epoch 3/10, Loss: 0.017300127361518616
Epoch 4/10, Loss: 0.020060315987657697
Epoch 5/10, Loss: 0.013035125414662532
Epoch 6/10, Loss: 0.021503022914048642
Epoch 7/10, Loss: 0.016944349615977357
Epoch 8/10, Loss: 0.028443743178002554
Epoch 9/10, Loss: 0.01811485376634505
Epoch 10/10, Loss: 0.021448543218417056
Accuracy: 87.25%
Precision: 10.84%
Recall: 3.82%
F1 Score: 5.65%
ROC-AUC: 0.4960
```



```
Anomalies detected at indices: [ 97 137 162 234 241 297 298 319 326 344 345 372 425 426 428 431 443 480
 492 496 504 515 521 524 536 542 548 551 558 559 563 582 590 610 620 627
 632 633 639 659 684 690 764 787 860 864 865 947 964 979 984]
```

Graph:



X-Axis: Represents the index or time step of the data points, ranging from 0 to 1000.

Y-Axis: Represents the reconstruction loss values, which measure how well the model reconstructs data points. Higher values typically indicate anomalies.

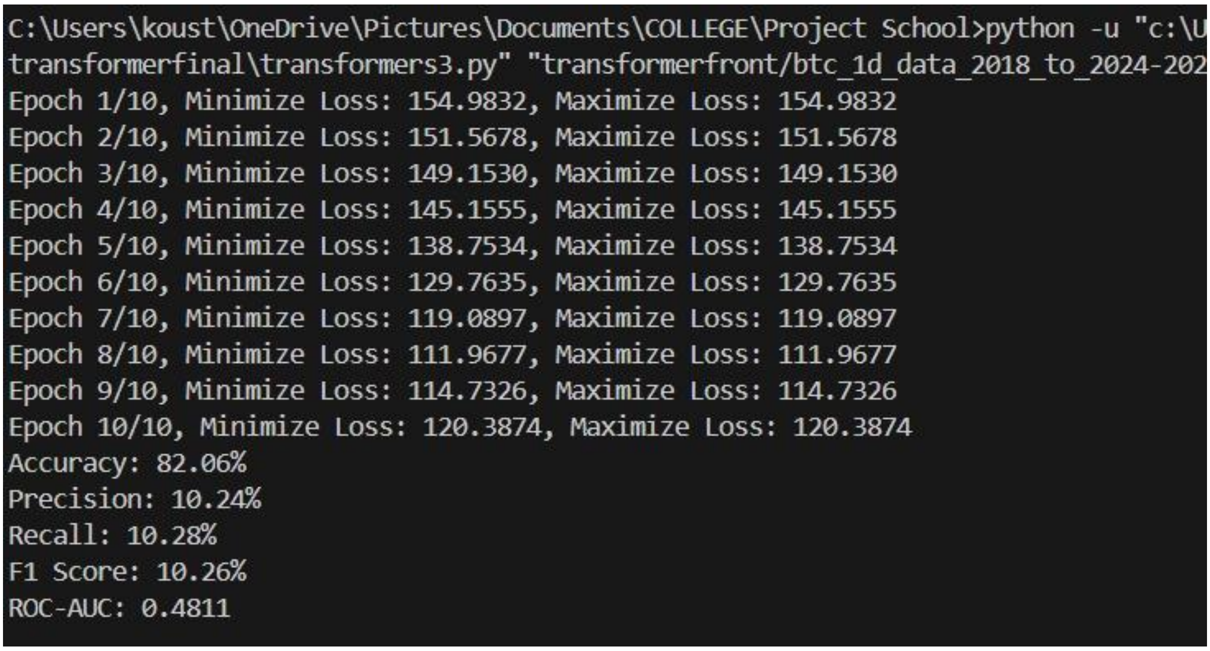
Observations:

→**Anomalies**: Several peaks in the reconstruction loss exceed the threshold, marked with red dots. These are distributed across the dataset, with clusters around indices 200–300, 500–700, and a few others.

→**Threshold**: The threshold appears to be constant, set around a reconstruction loss of 0.4, serving as the cutoff for anomaly detection.

→**Model Performance**: The use of transformer and anomaly attention effectively identifies spikes that deviate significantly from the baseline reconstruction loss.

2) An Anomaly Attention Transformer model was built using PyTorch, incorporating key concepts such as association discrepancy, backpropagation, the minimax strategy, and KL divergence. The Bitcoin 1-day dataset was loaded, and the following output was generated



```
C:\Users\koust\OneDrive\Pictures\Documents\COLLEGE\Project School>python -u "c:\U
transformerfinal\transformers3.py" "transformerfront/btc_1d_data_2018_to_2024-202
Epoch 1/10, Minimize Loss: 154.9832, Maximize Loss: 154.9832
Epoch 2/10, Minimize Loss: 151.5678, Maximize Loss: 151.5678
Epoch 3/10, Minimize Loss: 149.1530, Maximize Loss: 149.1530
Epoch 4/10, Minimize Loss: 145.1555, Maximize Loss: 145.1555
Epoch 5/10, Minimize Loss: 138.7534, Maximize Loss: 138.7534
Epoch 6/10, Minimize Loss: 129.7635, Maximize Loss: 129.7635
Epoch 7/10, Minimize Loss: 119.0897, Maximize Loss: 119.0897
Epoch 8/10, Minimize Loss: 111.9677, Maximize Loss: 111.9677
Epoch 9/10, Minimize Loss: 114.7326, Maximize Loss: 114.7326
Epoch 10/10, Minimize Loss: 120.3874, Maximize Loss: 120.3874
Accuracy: 82.06%
Precision: 10.24%
Recall: 10.28%
F1 Score: 10.26%
ROC-AUC: 0.4811
```

Observations:

Loss Reduction:

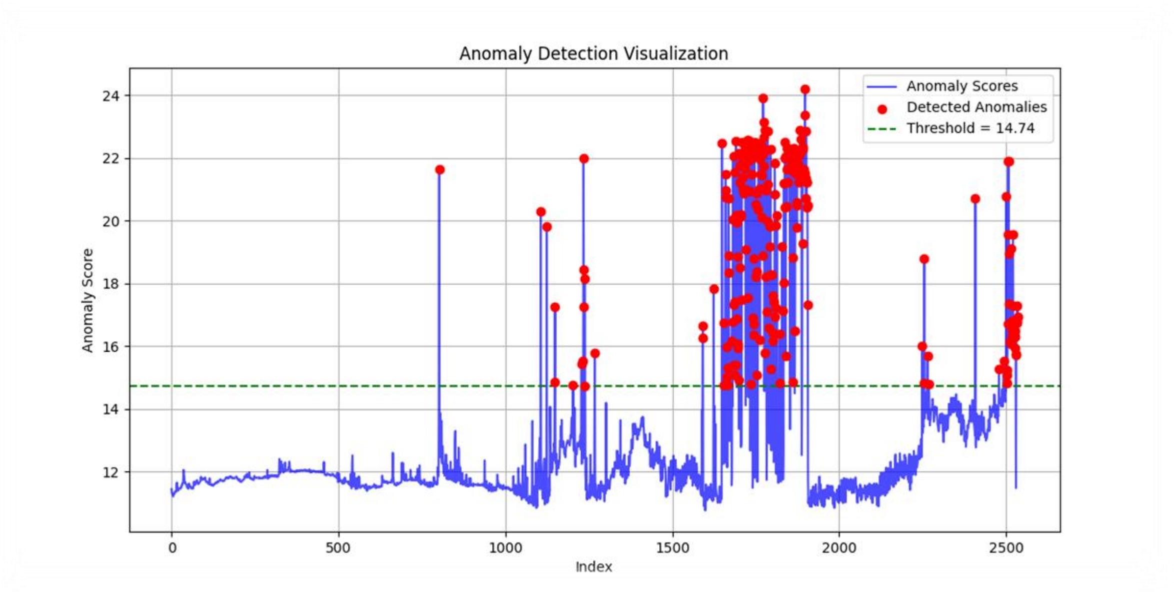
Loss is steadily reducing across the 10 epochs, which suggests that the model is learning and optimizing.

Metrics:

Accuracy: 82.06% indicates the percentage of correct predictions out of the total predictions.

Precision, Recall, F1 Score: These values are low (~10%), which may suggest class imbalance or difficulty in identifying the positive class.

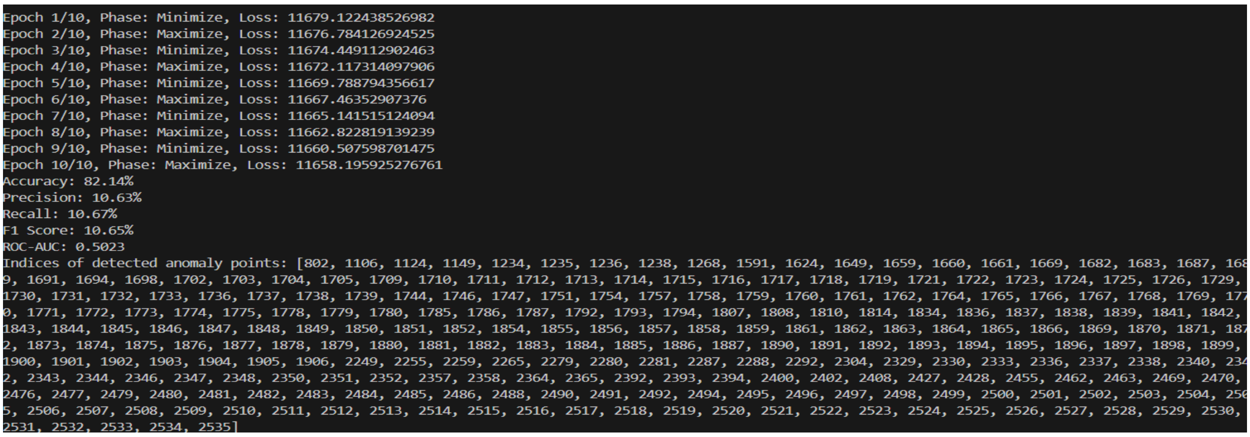
ROC-AUC: 0.4811 indicates the model is not performing well in terms of separating positive and negative classes.



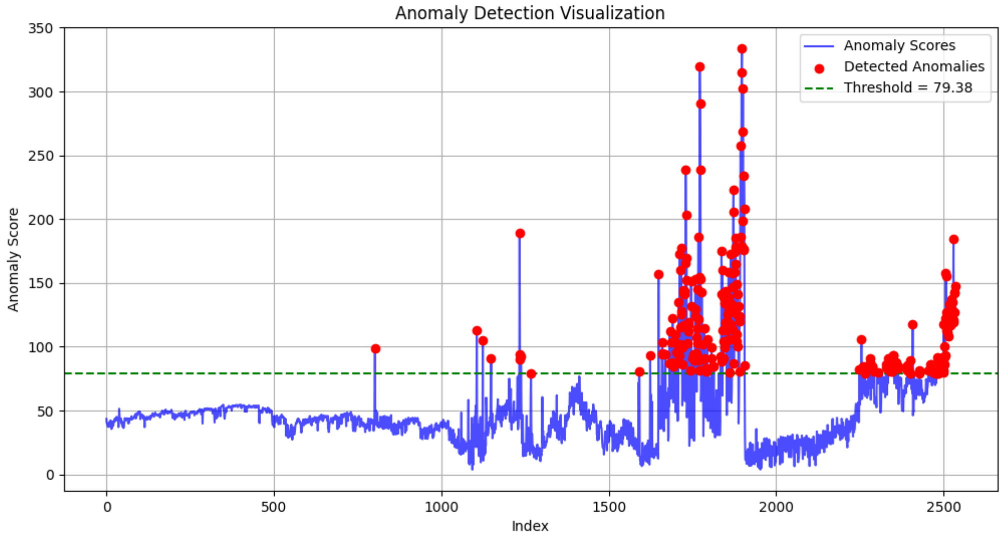
X-Axis: Represents the index or time step of the data points, ranging from 0 to 1000.

Y-Axis: Represents the reconstruction loss values, which measure how well the model reconstructs data points. Higher values typically indicate anomalies.

3) An anomaly attention transformer model was build without using any 3rd party libraries(by using Numpy and mathplotlib) incorporating key concepts such as association discrepancy and backpropagation. These are the results obtained.



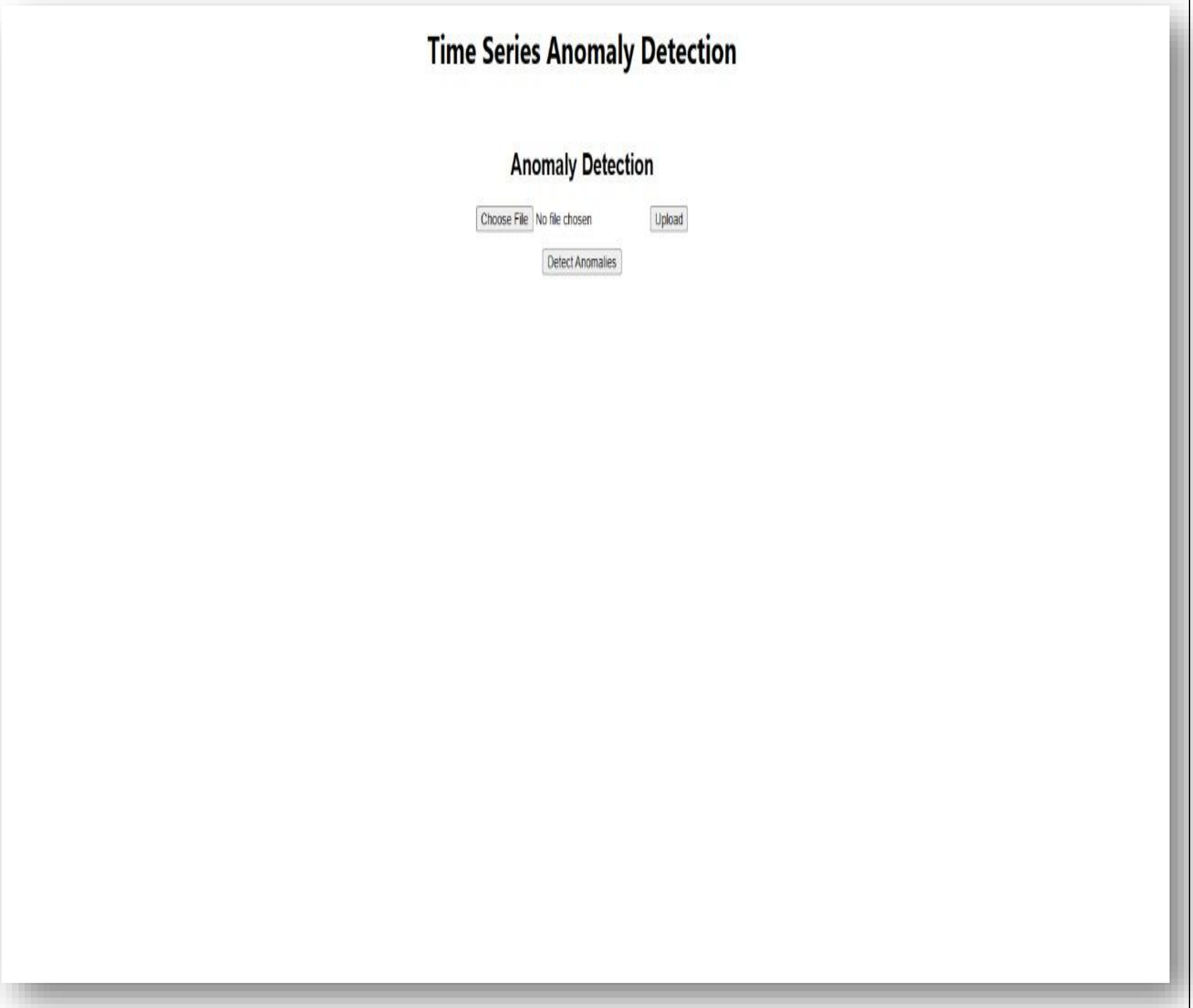
```
Epoch 1/10, Phase: Minimize, Loss: 11679.122438526982
Epoch 2/10, Phase: Maximize, Loss: 11676.784126924525
Epoch 3/10, Phase: Minimize, Loss: 11674.449112902463
Epoch 4/10, Phase: Maximize, Loss: 11672.117314097906
Epoch 5/10, Phase: Minimize, Loss: 11669.788794356617
Epoch 6/10, Phase: Maximize, Loss: 11667.46352907376
Epoch 7/10, Phase: Minimize, Loss: 11665.141515124094
Epoch 8/10, Phase: Maximize, Loss: 11662.822819139239
Epoch 9/10, Phase: Minimize, Loss: 11660.507598701475
Epoch 10/10, Phase: Maximize, Loss: 11658.195925276761
Accuracy: 82.14%
Precision: 10.63%
Recall: 10.67%
F1 Score: 10.65%
ROC-AUC: 0.5023
Indices of detected anomaly points: [802, 1106, 1124, 1149, 1234, 1235, 1236, 1238, 1268, 1591, 1624, 1649, 1659, 1660, 1661, 1669, 1682, 1683, 1687, 168
9, 1691, 1694, 1698, 1702, 1703, 1704, 1705, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 1721, 1722, 1723, 1724, 1725, 1726, 1729,
1730, 1731, 1732, 1733, 1736, 1737, 1738, 1739, 1744, 1746, 1747, 1751, 1754, 1757, 1758, 1759, 1760, 1761, 1762, 1764, 1765, 1766, 1767, 1768, 1769, 177
0, 1771, 1772, 1773, 1774, 1775, 1778, 1779, 1780, 1785, 1786, 1787, 1792, 1793, 1794, 1807, 1808, 1810, 1814, 1834, 1836, 1837, 1838, 1839, 1841, 1842,
1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1854, 1855, 1856, 1857, 1858, 1859, 1861, 1862, 1863, 1864, 1865, 1866, 1869, 1870, 1871, 187
2, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899,
1900, 1901, 1902, 1903, 1904, 1905, 1906, 2249, 2255, 2259, 2265, 2279, 2280, 2281, 2287, 2288, 2292, 2304, 2329, 2330, 2333, 2336, 2337, 2338, 2340, 234
2, 2343, 2344, 2346, 2347, 2348, 2350, 2351, 2352, 2357, 2358, 2364, 2365, 2392, 2393, 2394, 2400, 2402, 2408, 2427, 2428, 2455, 2462, 2463, 2469, 2470,
2476, 2477, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2488, 2490, 2491, 2492, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 250
5, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530,
2531, 2532, 2533, 2534, 2535]
```



Code	Simple Transformer	Transformer with Pytorch	Transformer without 3rd party libraries
Accuracy	**87.25%**	**82.06%**	**82.14%**

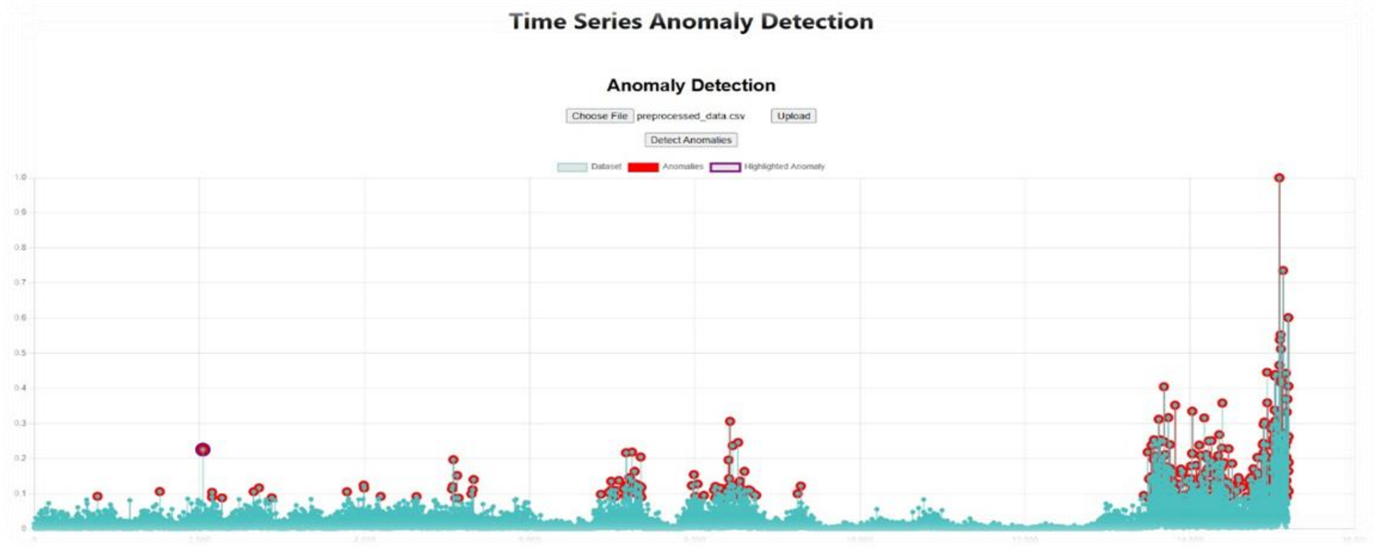
OUTPUT SCREENS:

This is the **home page** of our UI where a user can upload a dataste which is in the form of .csv to use our services.



Time Series Anomaly Detection

Anomaly Detection



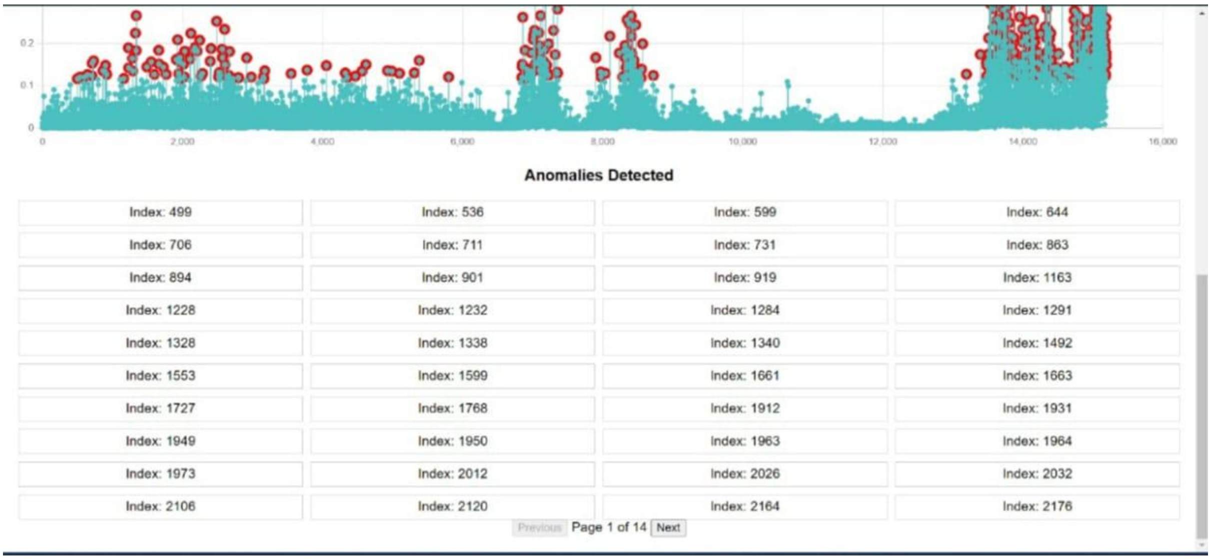
This graph displays the anomalies in the Time series data. Here's what the key elements represent:

Dataset (cyan dots): The main time-series data being analysed.

Anomalies (red circles): Points in the dataset identified as anomalies or deviations from the expected pattern.

Highlighted Anomaly (purple circle): A specific anomaly that might be emphasized for further analysis.

The x-axis likely represents the time or sequence index, and the y-axis represents the value of the time-series. Let me know if you'd like assistance in processing this data or interpreting it further.

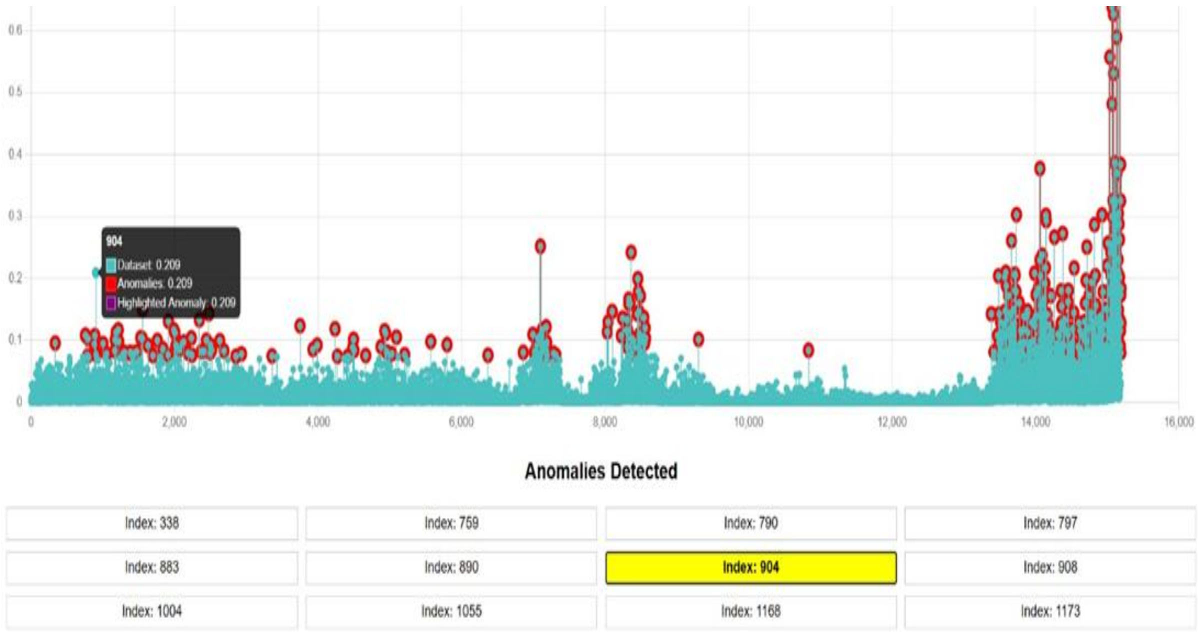


Anomalies Detected

Index: 499	Index: 536	Index: 599	Index: 644
Index: 706	Index: 711	Index: 731	Index: 863
Index: 894	Index: 901	Index: 919	Index: 1163
Index: 1228	Index: 1232	Index: 1284	Index: 1291
Index: 1328	Index: 1338	Index: 1340	Index: 1492
Index: 1553	Index: 1599	Index: 1661	Index: 1663
Index: 1727	Index: 1768	Index: 1912	Index: 1931
Index: 1949	Index: 1950	Index: 1963	Index: 1964
Index: 1973	Index: 2012	Index: 2026	Index: 2032
Index: 2106	Index: 2120	Index: 2164	Index: 2176

Anomalies List (Bottom Section):

A table showing indices of the detected anomalies. Each index corresponds to the position of an anomalous point in the time series.



Anomalies Detected

Index: 338	Index: 759	Index: 790	Index: 797
Index: 883	Index: 890	Index: 904	Index: 908
Index: 1004	Index: 1055	Index: 1168	Index: 1173

If an index of a particular anomaly is clicked then exact location of that particular anomaly is highlighted and shows the location as shown in the picture. For example, as shown in the figure if index 904 is clicked then the location is showed in the form of y- axis that is 0.209.

Business Use Cases:

Time series anomaly detection using transformer models has a wide range of applications across various industries. Few of them are:

1.Financial Services: Fraud Detection: Transformers can detect anomalies in transactional data, such as unusual spending behaviours or market manipulation. This helps in identifying fraudulent activities like credit card fraud or suspicious trading behaviour in real-time. Its benefit from transformer is Real-time fraud detection with high accuracy, minimizing losses.

2. E-commerce: Inventory Management and Demand Forecasting: By analysing sales and customer data, transformers can detect anomalies in demand, helping businesses manage inventory efficiently. They also improve sales forecasting, especially during high-demand events. Its benefit from transformer is Optimized inventory management, reducing stockouts or overstocking.

3. Manufacturing: Predictive Maintenance: Transformers can analyse sensor data from machines to detect patterns that may indicate equipment failure. This enables predictive maintenance, reducing downtime and unexpected repairs. Benefit is Cost savings and improved operational efficiency by preventing breakdowns.

4. Healthcare: Patient Monitoring: Transformers can monitor patient vitals to detect anomalies indicating potential health issues like heart attacks or respiratory failure, enabling faster intervention. Its benefit is Early detection of critical health issues, improving patient outcomes.

5. Retail: Customer Behaviour Monitoring: Transformers can analyse customer behaviour patterns, detecting anomalies in purchasing or browsing activities, which can inform personalized marketing strategies or inventory adjustments.

6. Energy: Power Grid Monitoring: By analysing power consumption and grid data, transformers can detect anomalies such as sudden demand spikes or equipment failures, helping to prevent outages and optimize energy distribution.

7. IoT: Smart City Monitoring: In smart cities, transformers can analyse sensor data from traffic, air quality, and waste management systems to detect anomalies, improving city services and resource allocation.

Transformer models in time series anomaly detection provide significant value across various sectors. By identifying patterns and detecting anomalies in real-time, they help businesses optimize operations, improve customer experience, and reduce costs.

CONCLUSION & FUTURE SCOPE

1. Conclusion:

The use of transformer models for time series anomaly detection represents a significant advancement in handling sequential data. With their self-attention mechanism, transformers are uniquely capable of capturing long-term dependencies and complex temporal patterns that traditional methods may struggle to identify. This ability to process and analyse large sequences of data efficiently makes transformers highly effective in detecting anomalies in various domains, including financial forecasting, sensor monitoring, and network security.

The model's strength lies in its capacity to not only classify anomalies accurately but also to adapt to the underlying structure of the data, identifying subtle deviations from expected behaviour. By leveraging techniques like autoencoders or sequence-based classification heads, transformers can output precise anomaly scores or classifications, which are essential for real-time decision-making.

Furthermore, the transformer model's ability to handle variable-length sequences and its robustness in detecting both global and local anomalies contribute to its versatility and reliability. This makes it an excellent choice for applications where timely detection of outliers or abnormal patterns is critical.

Overall, the combination of transformers' advanced architecture and the application of proper pre-processing techniques for time series data provides an effective and scalable solution for anomaly detection, making it well-suited for deployment in production environments where high accuracy and fast response times are necessary.

2. Future Scope:

- **Enhancements for Real-Time Applications:**

  Investigating the deployment of the Anomaly Transformer in real-time systems where latency is critical, such as financial fraud detection, industrial IoT monitoring, or health diagnostics.

- **Scalability Improvements**:

  Optimizing the computational efficiency and memory usage of the Anomaly Transformer to handle larger-scale time series data or higher-dimensional datasets effectively.

- **Adaptation to Multivariate Scenarios**:

  Extending the model's capabilities to explore and better handle highly correlated multivariate datasets, such as those found in sensor networks or climate monitoring.

- **Domain-Specific Fine-Tuning**:

  Tailoring the model for specific domains like genomics, neuroscience, or personalized medicine by incorporating domain knowledge into the anomaly detection process.

- **Incorporating Semi-Supervised Learning**:

  Developing hybrid approaches that combine unsupervised anomaly detection with limited labelled data to improve detection accuracy and reduce false positives.

- **Robustness Against Noisy Data**:

  Addressing the challenges of noisy or incomplete time series data, which is common in real-world applications, to enhance the model's robustness.

- **Cross-Domain Transfer Learning**:

  Enabling the model to transfer knowledge learned from one domain or dataset to another, enhancing its adaptability and reducing the need for extensive retraining.

REFERENCES

- Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. "**Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy**." *ICLR 2022*. URL: https://arxiv.org/pdf/2110.02642

- Vaswani et al., "**Attention is All You Need**," *NeurIPS 2017*. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

- SMAP Hands-On Tutorial by Jet Propulsion Laboratory California Institute of Technology.

  URL: 315_smap_hands_on_v1[1].pdf

- Anomaly Detection In Time Series Data a practical implementation for pulp and paper industry Master's thesis in Engineering Mathematics and Computational Science by Department of Mathematical Sciences CHALMERS UNIVERSITY OF TECHNOLOGY Gothenburg, Sweden 2016. URL: Anomaly_Detection_In_Time_Series_Data[1].pdf

- Solving Transformer by Hand: A Step-by-Step Math Example by "Fareed Khan". URL: https://levelup.gitconnected.com/understanding-transformers-from-start-to-end-a-step-by-step-math-example-16d4e64e6eb1

- Introduction to Artificial Neural System by JACEK M. ZURADA Professor of Electrical Engineering and of Computer Science and Engineering. URL: Introduction_to_Artificial_Neural_Systems[1].pdf

- Build an Artificial Neural Network from Scratch by "Nagesh Singh Chauhan".

- Anomaly Detection in Time Series: A Comprehensive Evaluation by Sebastian Schmidl , Phillip Wenig , Thorsten Papenbrock". URL: Anomaly_detection_in_time_series_-_a_comprehensive_evaluation[1].pdf

- IS IT WORTH IT? COMPARING SIX DEEP AND CLASSICAL METHODS FOR UNSUPERVISED ANOMALY DETECTION IN TIME SERIES by "Ferdinand Rewicki , JoachimDenzler , Julia Niebling ". URL: Is_it_worth_it_-_Comparing_six_deep_and_classical_methods_for_unsupervised_anomaly_detection_in_time[1].pdf

- ANOMALY DETECTION IN UNIVARIATE TIME-SERIES: A SURVEY ON THE STATE-OF-THE-ART by "MohammadBraei and Dr.-Ing. Sebastian Wagner". URL: ANOMALY_DETECTION_IN_UNIVARIATE_TIME-SERIES[1].pdf