THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Analysis air-quality monitoring data for anomaly detection: LSTM-AE

School of Information Technology and
Electrical Engineering, The University of Queensland

Submitted for the degree of Bachelor of Engineering (Honours) of Electrical engineering.

2th of November 2023

Acknowledgements

I extend my profound gratitude to my esteemed supervisor, Professor Marcus Gallagher, whose invaluable guidance and expertise have been the cornerstone of this research. His unwavering support and constructive feedback have been instrumental in elevating the quality of my work. I am also deeply thankful to my parents, whose constant encouragement and prayers have strengthened me throughout this academic journey. A special mention goes to my friends and classmates, who have been a beacon of support, making this research both enlightening and enjoyable. Their camaraderie and insights have enriched this experience, and I am genuinely grateful for that.

Keywords

LSTM-AE (Long Short-Term Memory Autoencoder), Anomaly Detection, Time-series Data, CO2 Dataset, Indoor Air Quality, Deep Learning, Temporal Dependencies, Reconstruction Loss, Encoder-Decoder Architecture, University of Queensland CO2 Dataset

Abstract

Indoor air quality, especially concerning CO2 levels, has become paramount due to its direct impact on human health and cognitive function. This research delves deep into this critical realm, emphasizing the importance of timely and accurate anomaly detection in CO2 levels to ensure a safe indoor environment.

Utilizing the wealthy University of Queensland CO2 Dataset, which encompasses data sourced from diverse real-world indoor environments, this study embarks on a journey to revolutionize anomaly detection in air quality data. While traditional techniques like k-Nearest Neighbour (KNN) and Autoregressive Integrated Moving Average (ARIMA) have provided foundational insights in the past, they often need to catch up on capturing the intricate temporal patterns inherent in air quality data.

The research introduces the Long Short-Term Memory Autoencoder (LSTM-AE) model to address this gap. This deep learning approach is uniquely tailored to handle time-series data, adeptly capturing temporal dependencies and nuances that conventional models might overlook. The driving force behind this innovative approach is to overcome the challenges posed by traditional machine learning methods, which often need help with the temporal dynamics of air quality data.

The results are compelling. The proposed LSTM-AE model effectively addresses the challenges and achieves a remarkable detection accuracy of over 99%. Benchmarking against other models in the same domain represents a significant leap forward in performance. A detailed comparative analysis with other LSTM and AE-based models further reinforces the superiority of the proposed approach, with evaluations grounded in crucial metrics like accuracy, precision, recall, and F1-score.

In wrapping up, this research illuminates the transformative potential of deep learning, with a spotlight on LSTM-AE, in reshaping the landscape of indoor air quality monitoring and anomaly detection. The insights gleaned from this study herald a new era of enhanced monitoring solutions, promising more robust, precise, and timely interventions that prioritize human health and well-being in indoor spaces.

Contents

1. Introduction

Indoor air quality, particularly the concentration of CO2, has garnered significant attention in recent years due to its profound impact on human health and well-being. Elevated levels of indoor CO2 can lead to various health issues, from minor discomforts like headaches and fatigue to more severe respiratory and cardiovascular problems [1]. As urbanization continues to rise and people spend more time indoors, monitoring and ensuring optimal indoor air quality becomes paramount.

The University of Queensland's CO2 Dataset offers a comprehensive collection of real-world indoor air quality readings, providing a valuable resource for researchers and scientists aiming to develop advanced anomaly detection techniques. Anomalies in such datasets can indicate various issues, from equipment malfunctions to sudden spikes in pollution levels. Detecting these anomalies promptly and accurately can lead to timely interventions, ensuring a healthier indoor environment.

While traditional machine learning techniques, such as k-nearest Neighbour (KNN) and Autoregressive Integrated Moving Average (ARIMA), have been employed for anomaly detection in various domains, they often need to catch up when dealing with time-series data like CO2. These methods overlook the temporal dependencies inherent in sequential data, leading to suboptimal performance [2].

This research introduces a novel approach to address this challenge: the Long Short-Term Memory Autoencoder (LSTM-AE) model. Designed to capture and understand temporal patterns in sequential data, LSTM-AE offers a promising solution for anomaly detection in time-series datasets [3]. This thesis delves deep into the development, implementation, and evaluation of the LSTM-AE model, comparing its performance with traditional methods and highlighting its advantages.

1.1 Background

The evolution of deep learning has been marked by the development of specialised architectures tailored to address unique challenges in data analysis. Among these architectures, Recurrent Neural Networks (RNNs) emerged as a promising solution for handling sequential data, given their inherent design to maintain a memory of previous inputs. However, traditional RNNs encountered a significant hurdle: the vanishing gradient problem [4]. This issue made it challenging for RNNs to learn and retain long-term dependencies in sequences, limiting their efficacy in applications requiring capturing extended temporal patterns.

To address this limitation, Long Short-Term Memory (LSTM) networks were introduced by Hochreiter Schmidhuber in 1997 [5]. Unlike traditional RNNs, LSTMs incorporated specialised structures known as "gates" – input, forget, and output gates. These gates regulate the flow of information, allowing the network to decide what to retain, discard, or update. This design enabled LSTMs to effectively capture both short-term and long-term dependencies in data, making them a preferred choice for many applications, from machine translation to stock price prediction.

Parallel to the advancements in RNNs, another deep learning architecture was making waves in data compression and reconstruction: the autoencoder (AE). Combining two primary components, an encoder, and a decoder, autoencoders were designed to compress input data into a lower-dimensional representation and reconstruct the original data from this compact form. The brilliance of autoencoders lies in their ability to highlight the most salient features of data, making them invaluable for tasks like dimensionality reduction and anomaly detection. The principle behind their use in anomaly detection is straightforward: Anomalies or outliers tend to have higher reconstruction errors than regular data points.

Recognising the strengths of LSTMs and autoencoders, researchers began to explore their combined potential, leading to conceptualising the LSTM-Autoencoder (LSTM-AE). This hybrid model married the sequence modelling capabilities of LSTMs with autoencoders' data compression and reconstruction expertise. In time-series data, the LSTM-AE emerged as a formidable tool. The model could effectively pinpoint anomalies based on reconstruction errors by capturing intricate temporal dependencies through the LSTM component and subsequently reconstructing sequences via the autoencoder.

The significance of such a model becomes even more pronounced when considering domains like indoor air quality monitoring. With the proliferation of sensors and the consequent explosion in time-series data, traditional anomaly detection methods often need help to capture the nuanced temporal patterns and relationships inherent in such datasets. The LSTM-AE, with its dual capabilities and deep learning foundations, offers a robust and sophisticated solution to this pressing challenge, laying the groundwork for the research presented in this thesis.

1.2 Objectives

The primary objective of this thesis is to design, implement, and evaluate the performance of a Long Short-Term Memory Autoencoder (LSTM-AE) model for predicting CO2 levels in air-quality monitoring data. The specific goals are as follows:

Model Development: To construct an LSTM-AE model tailored for time-series data, mainly focusing on the patterns and trends observed in CO2 levels over time.

Architectural Exploration: To investigate various architectures of the LSTM-AE model, analysing the impact of different layers, units, and configurations on the model's predictive performance.

Hyperparameter Tuning: To systematically tune the hyperparameters of the LSTM-AE model, identifying the optimal set of parameters that yield the best performance on our dataset.

Performance Evaluation: To rigorously evaluate the performance of the best-performing LSTM-AE model using appropriate metrics and validation techniques, ensuring its robustness and reliability.

Comparative Analysis: To benchmark the LSTM-AE model against other state-of-the-art approaches, highlighting its advantages and potential areas of improvement.

Through these objectives, this thesis aims to contribute to the field of air-quality prediction by offering a comprehensive study of the capabilities of the LSTM-AE model, providing insights into its effectiveness and potential for real-world applications.

2. Literature Review.

2.1 Time-series data analysis: Overview

Time-series data is a sequence of data points, typically consisting of successive measurements made over a time interval. Each data point in a time-series dataset is time-stamped, making the order of observations crucial. This type of data is ubiquitous and can be found in various domains, including finance (stock prices), meteorology (temperature readings), healthcare (heart rate monitoring), and environmental science (air quality measurements). [6]

Characteristics of Time-series Data:

1. Temporal Dependence: Unlike cross-sectional data, where observations are independent, time-series data points are often temporally dependent. This means the value at a particular time depends on its previous values.

2. Trend: Time-series data might exhibit an upward or downward movement over an extended period. This systematic, consistent increase or decrease is referred to as a trend.

3. Seasonality: Time-series data can also show patterns that repeat regularly, known as seasonality. For instance, retail sales might spike during the holiday season each year.

4. Noise refers to the random variations in the data that cannot be attributed to the trend or seasonality. Noise can result from random events or fluctuations not part of the underlying data pattern.

Analysis Techniques:

Analysing time-series data involves various techniques, each designed to extract meaningful insights from the data:

1. Descriptive Analysis: This involves visualizing the data to identify patterns, trends, and potential outliers. Tools like line plots and moving averages are commonly used.
2. Frequency Analysis: Techniques like Fourier Transform are used to study the cyclical patterns in the data.

3. Decomposition: This involves breaking down the time-series data into its constituent components: trend, seasonality, and noise.
4. Statistical Models: Models like ARIMA forecast future values based on past observations.

Challenges in Time-series Analysis:

Analysing time-series data has its challenges. The temporal dependence means that traditional data analysis techniques, which assume independence of observations, often need to be more suitable. Factors like missing values, irregular intervals, and external influences (like sudden, unexpected events) can complicate the analysis. [7]

Deep Learning in Time-series Analysis:

With the advent of deep learning [8], new avenues have opened in time-series analysis. Neural networks, especially architectures like LSTM, have shown promise in capturing long-term dependencies and intricate patterns in time-series data, making them valuable for forecasting and anomaly detection tasks.

In the context of this thesis, time-series data analysis is pivotal, given the focus on indoor air quality monitoring, especially the levels of CO2 over time. The subsequent sections will delve deeper into the specific deep learning techniques, like LSTM-AE, tailored for time-series data.

2.2 Introduction to Neural Networks for Time-series

Neural networks are a subset of machine learning models inspired by the structure of the human brain shown below in the Figure 1 and introduced in 1958 by Frank Rosenblatt [9]. Their adaptability and ability to learn intricate patterns have made them popular for various data-driven tasks, including time-series analysis.

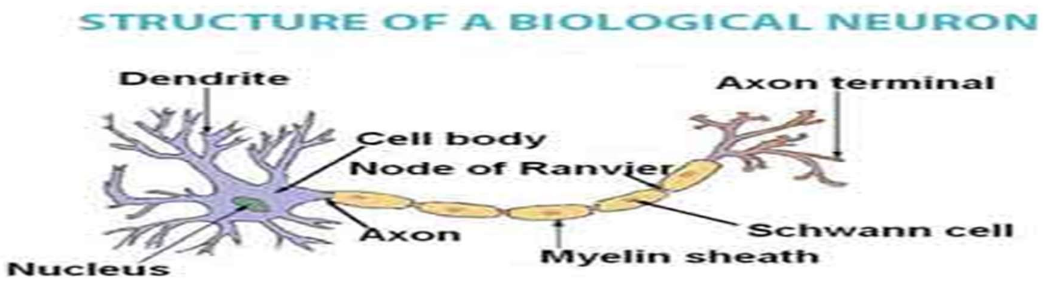


Figure 1. Structure of Biological Neuron

The fundamental units of a neural network are called neurons, nodes, or units. Each neuron receives input as shown in the figure 2, processes it using a transfer function, and produces an output. Neural networks are typically structured in layers comprising an input layer, one or more hidden layers, and an output layer [10]. Each layer consists of multiple neurons. The connections between these neurons have associated weights, which are adjusted during training. Biases are additional parameters that allow for more flexibility in the model. Mathematical functions, known as activation functions, are applied to a neuron's output to introduce non-linearity to the model [11].

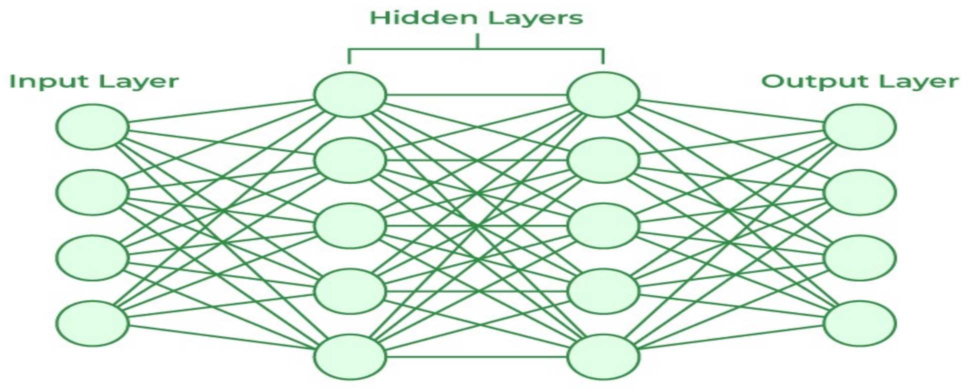


Figure 2. Structure of Neuron Network

According to the Singh (2021), feedforward neural networks are the simplest type of neural network where information flows in one direction, from input to output, without loops or cycles. In contrast, recurrent neural networks (RNNs) have connections that loop backward, allowing them to maintain a 'memory' of previous inputs. This characteristic makes RNNs especially suitable for sequential data like time series. However, traditional RNNs can suffer from the vanishing gradient problem, making it challenging to learn long-term dependencies.

Long Short-Term Memory (LSTM) networks were introduced to address this limitation [5]. LSTMs are a special kind of RNN designed to capture long-term dependencies in sequences. They use structures known as gates, specifically input, forget, and output gates, to regulate the flow of information, allowing the network to decide what information to retain or discard over time.

While convolutional neural networks (CNNs) are primarily known for their application in image processing, they have also found utility in time-series data analysis [13]. By using convolutional layers that scan input data for local patterns, CNNs can detect temporal patterns in sequences effectively. Furthermore, hybrid models combining different neural network architectures, such as CNNs and LSTMs, can leverage local pattern detection and sequence memory, often yielding superior results.

Neural networks offer several advantages for time-series analysis. Their flexibility allows them to model both linear and non-linear relationships. They can learn directly from raw data, eliminating the need for manual feature extraction, and are scalable to handle large datasets and complex architectures. However, they also come with challenges. Overfitting can be a concern if networks become too complex. Intense neural networks' interpretability can be challenging, as they are often considered "black boxes." Additionally, training deep neural networks can be computationally intensive [11].

In time-series analysis, neural networks present a powerful toolkit capable of capturing and predicting intricate temporal patterns. Their ability to learn from sequences and remember past observations positions them as valuable assets for forecasting and anomaly detection tasks.

2.3 Existing work on LSTM-AE for Time-series Data (Related works)

Given the significance of detecting anomalies, extensive research has been conducted to explore various methods and fields of application for anomaly identification.

We examine the leading-edge methods for identifying anomalies in indoor air quality datasets and related domains. Based on the study by Chandola, Banerjee, and Kumar (2009), they offer insights into various dimensions of the anomaly detection challenge. They categorize anomaly detection methods into several groups, including those rooted in Neural Networks, Bayesian Networks, Support Vector Machines, and Rule-Based systems. Also, [15] categorizes anomaly detection techniques into three primary methods: statistical methodologies, approaches based on neural networks, and methods grounded in nearest-neighbour techniques. According to research, the anomaly detection techniques are reviewed in detail [16, 17] In [18], the authors present an extensive overview of anomaly detection techniques that leverage deep learning. They highlight the latest advancements in anomaly detection, both in terms of the foundational methods and their specific application areas. Figure 3 illustrates the categorization of these methods. [18]

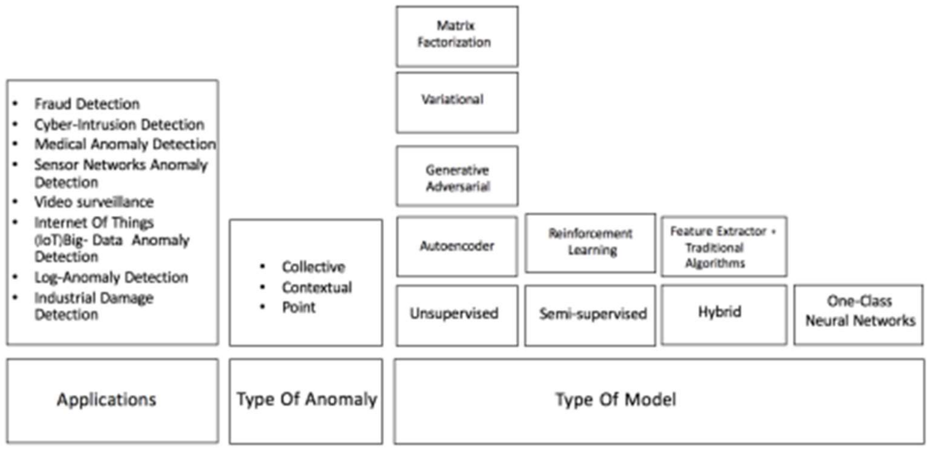


Figure 3. Machine/Deep learning methods, used for anomaly detection.

Numerous research endeavours have leveraged the encoder-decoder framework to understand input data representations in an unsupervised manner. For instance, the study by [22] employed an LSTM-based encoder-decoder structure to comprehend representations of video sequences, shedding light on the feature extraction capabilities of the embedded layer. Given the high-dimensional nature of video data, the encoder-decoder paradigm proved instrumental in discerning the essence of video content. Park and collaborators [19] proposed a hybrid model, LSTM-VAE, tailored to pinpoint anomalies in robot-assisted feeding mechanisms crucial for individuals needing physical support. Their analysis spanned data from 155 such systems, aiming to detect aberrant robot actions. Anomaly identification was based on a state threshold derived from data detailing the task execution state. Their model's performance, as indicated by the ROC curve, was notably superior to other analogous methods. Similarly, Liu and team [20] presented a combined VAE-LSTM model, amalgamating Variational Autoencoder and LSTM to spot irregularities across diverse time frames. Their methodology employed VAE for encapsulating short-term local data and LSTM for discerning patterns over extended periods, ensuring comprehensive anomaly detection. Jung and associates [21] employed LSTM to project indoor environmental conditions, drawing from IoT sensor datasets that monitored parameters like temperature, humidity, and luminosity. Their LSTM model was calibrated to spot irregularities in indoor conditions when the amalgamated dataset readings strayed from a benchmark set during the training phase.

3. Preliminaries

3.1 Long Short-Term Memory (LSTM) Network

Long Short-Term Memory, commonly known as LSTM, is an advanced Recurrent Neural Network (RNN) form. While RNNs introduced the concept of "short-term memory" by leveraging previous information up to a certain point for current computations, LSTMs extend this idea by incorporating "long-term memory." This means that LSTMs can remember and utilize a list of all previous information for the current neural node instead of just a specific point in time.

A typical LSTM unit consists of a cell, an input gate, an output gate, and a forget gate as shown below in the figure 4. The cell's primary function is to remember values over arbitrary time intervals, while the three gates control the flow of information into, within, and out of the cell.

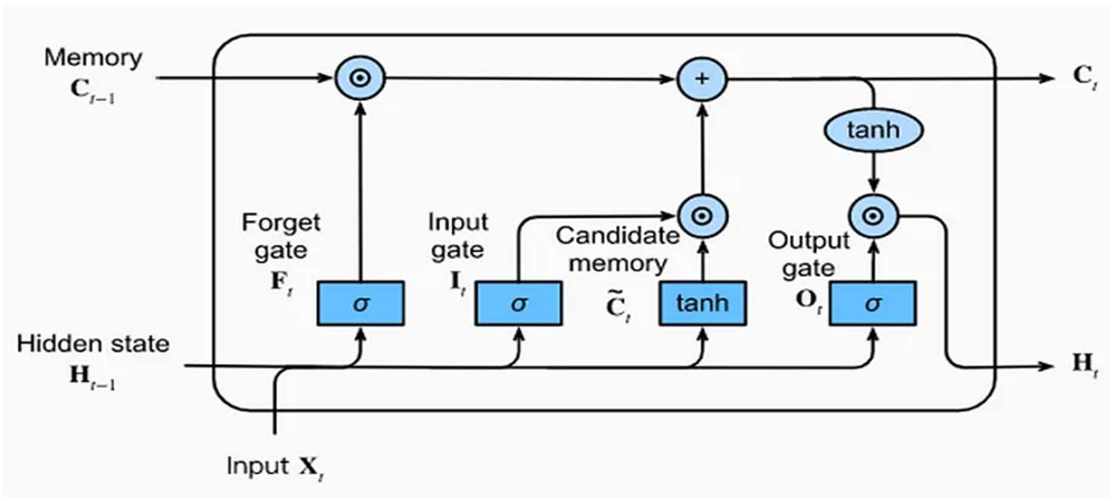

Memory $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Candidate memory $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
σ
tanh
σ
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$

Figure 4. Architecture of LSTM Unit

Inputs and Outputs of the LSTM Unit

Every LSTM cell is designed to process three distinct vectors at each time step. Two of these, namely the cell state (C) and the hidden state (H), are internally generated and represent the memory of the LSTM from the preceding time step (t - 1). The third vector, the input vector (X), is externally fed into the LSTM at the current time step (t).

Inside the LSTM, intricate gates control the flow of information, ensuring that the cell state and hidden state vectors are updated appropriately. The cell state is akin to the unit's long-term memory, while the hidden state resembles a short-term memory. The LSTM utilizes the information from the immediate past (short-term memory, H) combined with the current external input (X) to refresh its long-term memory (cell state, C). Subsequently, this updated long-term memory is employed to refresh the short-term memory.

The Role of Gates in LSTM

The LSTM architecture incorporates three pivotal gates: the forget, the input, and the output. These gates are information regulators, producing selector vectors with values oscillating between zero and one.

Each gate is constructed using a neural network that employs the sigmoid activation function for its output layer. This ensures that the output values are constrained between zero and one. The concatenated vector of the external input (X) and the previous hidden state (H_[t−1]) serves as the typical input for all these gates.

Forget Gate Dynamics

The forget gate governs the initial activity within the LSTM at any time step. It determines which portions of the information from the preceding cell state (C_[t−1]) should be retained or discarded. This decision is influenced by the current input vector (X_[t]) and the preceding hidden state (H_[t−1]).

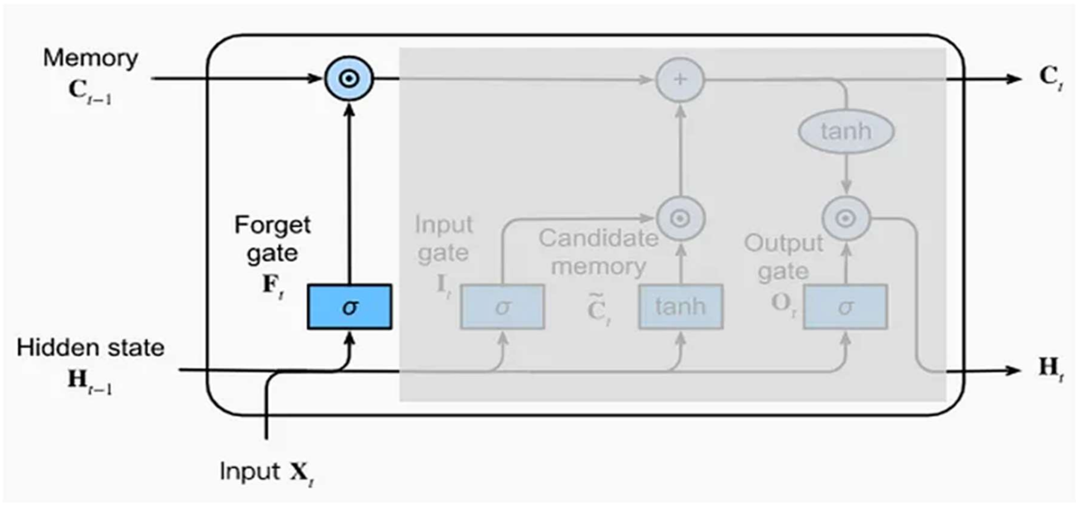


Figure 5. How forget gate works.

The mathematic equation (1) shown below represents how the forget gate works.

$$f_t = \sigma(W_f[H_{t-1}, x_t] + b_f \quad (1)$$

Where:

f_t : This is the forget gate's output. Each element of f_t is in the range [0,1]. It represents the amount of each component in the cell state C_{t-1} that should be kept.

σ : The sigmoid activation function, which squashes values to the range [0.1].

W_f : Weight matrix for the forget gate. It determines the importance of values in H_{t-1} and X_t for the forget gate's decision.

H_{t-1} : Hidden state from the previous time steps. It carries information from earlier time steps.

X_t : Input vector at the current time step.

b_f : Bias term for the forget gate. It offsets the gate's decision.

Input Gate and Memory Cell Operations

Once certain information from the preceding cell state is discarded, there is room to introduce new information. Two distinct neural networks orchestrate this task: the memory cell and the input gate. The memory cell generates a candidate vector, a set of information that might be added to the cell state. Concurrently, the input gate produces a selector vector that will decide which segments of the candidate vector should be integrated into the cell state.

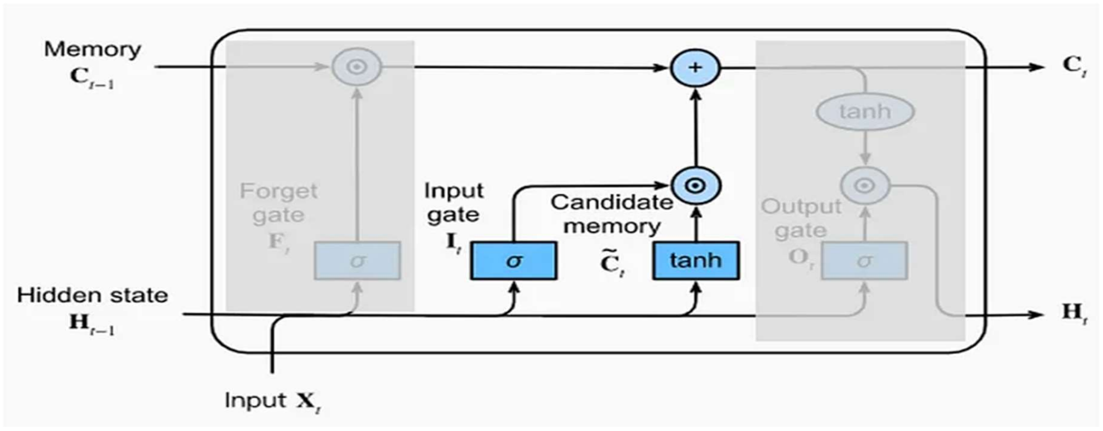


Figure 6. How input gate works.

The mathematic equation (2) and (3) shown below represents how the forget gate works.

$$I_t = \sigma(W_I[H_{t-1}, x_t] + b_I$$

$$\tilde{C}_t = tanh(W_c[H_{t-1}, x_t] + b_C$$

Where:

I_t *: Output of the input gate's sigmoid layer. It decides which parts of the cell state will be updated.*

$\tilde{C}_t$ *: New candidate values for the cell state. These values are a combination of the input and the previous hidden state.*

W_i *and* W_C *: Weight matrices for the input gate. They determine how much importance to give to the input and the previous hidden state.*

b_i *: Bias term for the input gate*

The cell stat is updated by forgetting certain values (using the forget gate's output) and adding new values (using the input gate's output)

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

Where:

f_t *and* I_t *: Outputs from the forget and input gates, respectively. They decide which information to forget and which new information to add.*

C_t *: Updated cell state at time t*

$\tilde{C}_t$ *: New candidate values for the cell state.*

Output Gate Mechanism

The output gate executes the final act within the LSTM during any time step. It determines the value of the hidden state for that particular time step (t). This is achieved by a multiplication operation between a selector vector (originating from the output gate) and a candidate vector derived from the cell state. The resulting hidden state serves as the LSTM's output for that time step and becomes a part of the input for the subsequent time step (t + 1).

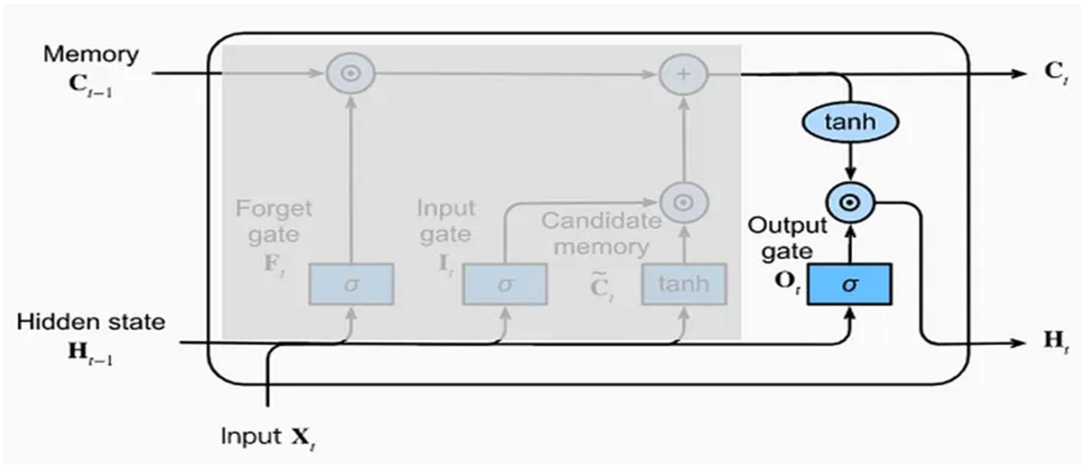


Figure 7. How output gate works.

$$O_t = \sigma(W_o \cdot [H_{t-1}, x_t] + b_o$$

$$H_t = tanh(C_t) + O_t$$

Where:

O_t *: Output gate's decision on which parts of the cell state will be exposed to the next hidden state.*

h_t *: Updated hidden state at time t.*

C_t *: Updated cell state at time t.*

W_o *: Weight matrix for the output gate.*

b_o *: Bias term for the output gate.*

In essence, the LSTM unit uses these gates to regulate the flow of information, ensuring that the cell state retains long-term dependencies. In contrast, the hidden state captures short-term dependencies. The intricate interplay of these gates allows the LSTM to model complex temporal sequences effectively.

3.2 Autoencoders (AE)

Autoencoders are specialized neural network architectures tailored to learn compact and efficient data representations, often referred to as encodings. These encodings are derived from the input data, and the process doesn't typically require labeled data. The foundational structure of an autoencoder is bifurcated into two primary segments: the encoding mechanism and the decoding mechanism, which can be visualized as mirror images in terms of functionality.

For a graphical depiction of the AE's operation, refer to Fig.8.

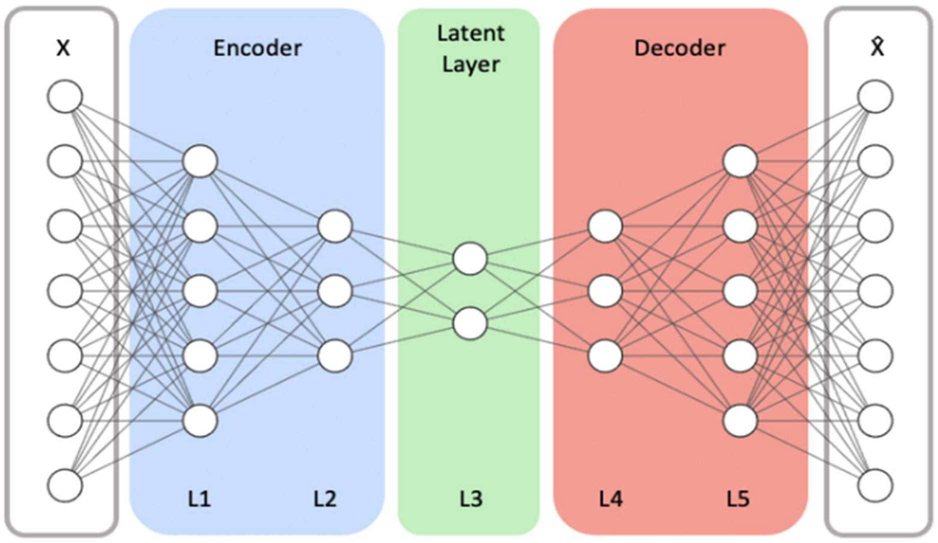


Figure 8. Architecture of Autoencoder

Encoding Mechanism:

This segment, frequently constructed using LSTM layers, ingests the high-dimensional input time-series data, symbolized as x. Its primary function is to distil this data into a more concise representation, termed h, or the "bottleneck". This compression process ensures that the data's most salient features and patterns are retained while less pertinent nuances are filtered out.

The mathematical representation of this transformation is:

$$h = \emptyset_1(\alpha_i x + \beta_i)$$

Here, $\emptyset_1$ denotes an activation function, α_i represents the weight matrix, and β_i is the encoder's bias term.

Decoding Mechanism:

Acting as the counterpart to the encoder, the decoder, often realized using LSTM structures, endeavors to regenerate the original data from its compressed form. It interprets the encoded data, h, and attempts to project it back to the space of the original data, resulting in an approximation, $\hat{x}$, of the initial time-series data, x.

The transformation can be mathematically articulated as follows:

$$\hat{x} = \emptyset_2(\alpha_j h + \beta_j)$$

In this expression, $\emptyset_2$ signifies another activation function, α_j is the weight matrix associated with the decoder, and β_j is its corresponding bias.

The overarching aim of an autoencoder is to ensure that the reconstructed data, $\hat{x}$, is as close as possible to the original input, x. The measure of this closeness or similarity is termed the "reconstruction loss" (L). When autoencoders are harnessed for anomaly detection tasks, any data point with a reconstruction loss surpassing a set threshold is flagged as an outlier or anomaly.

$$L(x, \hat{x}) = \frac{1}{n}\sum_{t=1}^{n} | x_t - \hat{x}_t |$$

In the equation (9), x is the original dataset, $\hat{x}$ denotes its reconstructed counterpart, and n is the total sample count in the dataset being trained on.

3.3 LSTM-AE: A Fusion of Sequence processing and Data Compression

The LSTM-AE architecture harmoniously integrates LSTM's sequence processing capabilities with AE's data compression and reconstruction strengths. This combination is particularly potent for time-series data, where both temporal dependencies and data dimensionality play crucial roles.

Sequential Encoding:

Temporal Dynamics: The LSTM layers at the onset of the encoder are not just sequence processors; they are temporal dynamics capturers. They understand the ebb and flow of time-series data, ensuring that the temporal dependencies, seasonality, and trends are adequately captured.

Memory Retention: The LSTM's ability to selectively remember and forget ensures that the encoder captures not just the immediate past but also significant events from the distant past, making the encoding rich and informative.

Dimensionality Reduction and Reconstruction:

Feature Compression: The LSTM layers feed their output into the dense layers of the encoder, which further compresses the data, ensuring that only the most salient features are retained in the bottleneck vector.

Data Regeneration: The decoder, taking cues from traditional AE, uses this bottleneck vector to regenerate the sequence. This reconstruction is not just a mere replication but a testament to how well the LSTM-AE has understood and internalized the input data's dynamics.

4. Data Description

4.1 Overview of Air-Quality Monitoring Data

Dataset Overview and Significance

The dataset in focus is derived from a series of air quality monitoring instruments, each embedded with a diverse set of sensors. These instruments are strategically dispersed across various locations within the UQ ITEE building, with notable placements at GP South and Axon. For readers seeking a granular understanding of the dataset's specifics, Appendix A offers a detailed exposition.

Temporal Coverage and Feature Set:

The dataset chronicles a series of readings over a time interval of 30 seconds from '2022-01-10 04:11:58' to '2023-04-05 05:27:03'. It encapsulates myriad features pertinent to air quality, including PM1, PM10, PM2.5, PM4, VOC, Humidity, Pressure, Dewpoint, Carbon Dioxide (CO2), and Temperature.

Emphasis on Carbon Dioxide Analysis:

In the face of such a diverse and expansive dataset, it became essential to narrow the focus to a specific feature that would be both relevant and impactful. Carbon Dioxide (CO2) emerged as the prime candidate. Its pivotal role in assessing air quality and the availability of an impressive 4,273,138 readings made it the ideal choice. The primary objective then shifted to leveraging the capabilities of the LSTM-AE model to detect anomalies specifically within the CO2 readings.

Challenges with Data Volume and Selection Criteria:

While the extensive CO2 data provided a rich resource, its sheer size also presented computational challenges. Given the limitations of the experimental environment and the need for efficient processing, it was clear that the entire dataset could not be utilized. This led to a meticulous search for a subset of the CO2 readings that would be comprehensive and computationally feasible. This search culminated in the identification of data from the 'gs116_794B44' device, which had logged a more digestible 507,570 CO2 readings. This curated subset was then earmarked for further analysis, representing the CO2 levels within the UQ ITEE environment.

4.2 Data Preprocessing and cleaning and normalization

Data from real-world scenarios frequently contain noise discrepancies and may lack complete values. Before leveraging machine learning models on such datasets, it is essential to undergo preprocessing. Most machine learning algorithms are optimized for comprehensive datasets without gaps. In this project, the preprocessing phase posed significant challenges and was crucial to the overall success of the analysis.

To ensure the quality and relevance of our data, zero readings, which might represent sensor errors or periods of inactivity, were identified, and subsequently removed. The dataset was then reset to maintain continuity in indexing.

Any duplicate timestamps within this subset were identified and removed to prevent redundancy and potential skewing of results.

For easy analysis and to prepare the data for modelling, the dataset was structured to include only the timestamp ('ts') and the corresponding value ('v'). The timestamp was then set as the index, ensuring a chronological order for our time-series data. This step is crucial for any time-series analysis as the sequence and timing of data points can significantly influence results.

Before splitting the dataset into testing and training, the number of anomalies in the dataset must be figured out to compare with results from our model. To calculate the number of anomalies, we must check if the dataset follows a normal distribution or not. If it does, the 2-sigmoid rule can be applied to the dataset to remove and count the outliers. The "2-sigma rule" refers explicitly to the fact that about 95% of the data is expected to lie within two standard deviations from the mean in a normally distributed dataset. This rule is often used in statistics and quality control to identify outliers or to determine intervals in which data points are expected to fall with a certain probability. In anomaly detection or quality control, if a data point falls outside the 2-sigma range, it might be considered unusual or an outlier. However, the specific threshold for considering a point as an outlier can vary based on the application. After applying the 2-sigma rule, we can know number of potential anomalies, about 11% (54,835) and figure 9 represents overall projection of co2 dataset and potential outliers

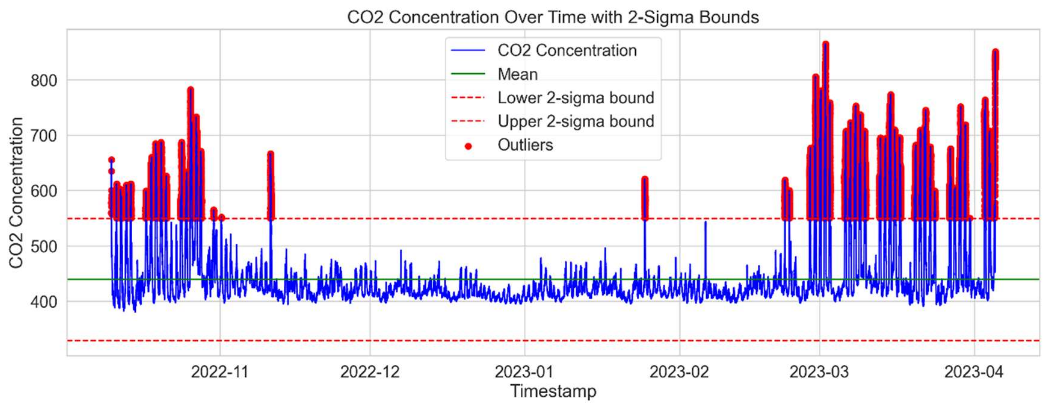


Figure 9. Projection of dataset of CO2 dataset with potential outliers' visualization.

In machine learning and deep learning, how data is divided is crucial for model evaluation. Properly segmenting the dataset ensures that the model is trained on a diverse subset of the data and evaluated on unseen data to determine its real-world applicability. For our study, given the dataset's substantial size of 507,570 samples, we opted for a 10:90 test-train split (Train: '2022-10-10 04:11:58':'2023-03-18 23:59:59', Test: '2023-03-19 00:00:00':'2023-04-05 05:27:03'). The size of our dataset influenced this decision. Even though it does not run into millions, it is sizable enough that a 10% subset, which translates to approximately 50,757 samples, can provide a comprehensive evaluation of the model's capabilities.

Another factor influencing this choice is the temporal nature of our data. As our dataset is a time series, it is essential to ensure that the model is correctly trained on future data. By chronologically dividing the data, we maintain its temporal integrity, ensuring the model learns from past events and is tested on future occurrences.

The 10% test set is designed to capture the overall characteristics of the data, ensuring that the model's evaluation is rooted in real-world scenarios. This reflects the model's potential performance when it is eventually deployed.

After training, the model's performance on the test set will be a crucial indicator of its effectiveness. If it does not perform as expected, it might be necessary to reconsider the split ratio or delve deeper into other potential influencing factors, such as the preprocessing steps or the chosen model architecture.

Before proceeding with training, it is crucial to ensure that the training dataset primarily consists of "normal" data. This means removing potential anomalies from the training set. By doing so, the model learns the characteristics of typical data points, making it more adept at recognizing anomalies as deviations from this norm during the testing phase. Conversely, the test set should retain its anomalies, allowing us to evaluate the model's effectiveness in detecting these outliers. To remove all potential anomalies from the training dataset, we do have to apply 2-sigma rule again to training datasets. As shown in Figure 10 below, most training datasets lie within two standard deviations from the mean in a normally distributed dataset. Even though we do not remove potential outliers from the test dataset, we labelled the potential outliers as 1. The number of potential anomalies in the test dataset is 8,218 which is 16% and it is close value to 11% in the total dataset.

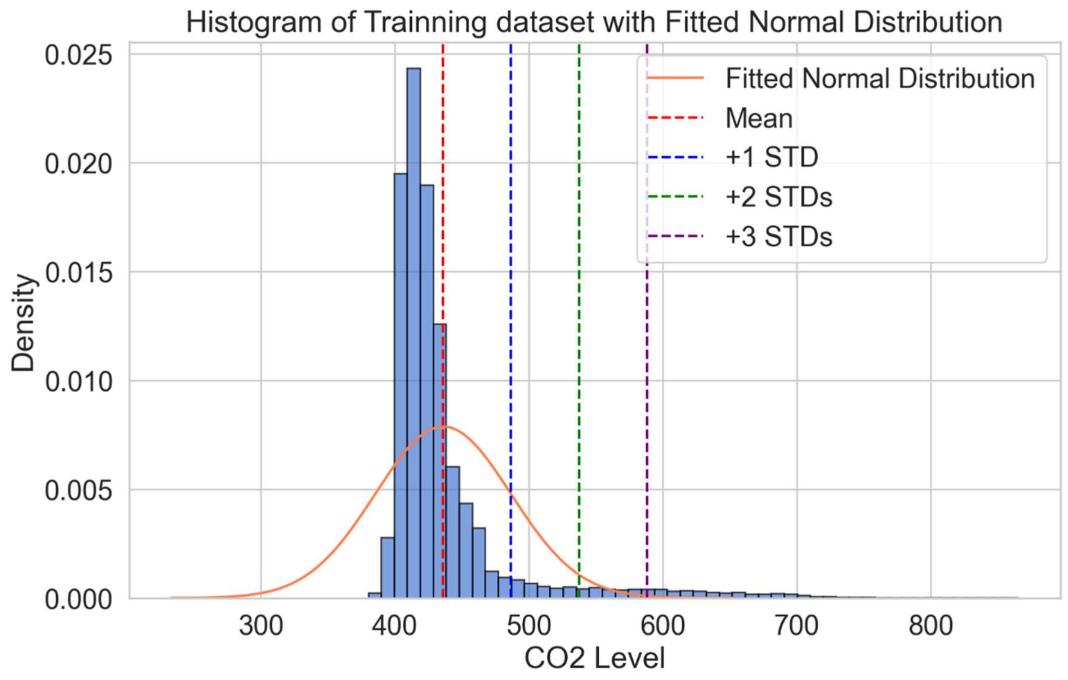


Figure 10. Histogram of training dataset with 2-sigma rules

Once we remove potential anomalies based on the 2-sigma rule, we can proceed to the next step, the normalization of the whole dataset. In our study, we employed the StandardScaler from the sklearn—preprocessing module to address this need.

The StandardScaler standardizes features by removing the mean and scaling them to unit variance. This process is often referred to as z-score normalization. It is mathematically represented as $\frac{x-\mu}{\sigma}$, where x is the original feature vector, μ is the mean of the feature vector, and σ is its standard deviation [23]. Feature scaling is vital for several reasons. Many machine learning algorithms, especially those that compute distances between data points or rely on gradient descent optimization, require features to be on a consistent scale for optimal performance. When features are appropriately scaled, the convergence rate of algorithms that use gradient descent is notably faster. The optimizer needs fewer steps to find the minimum or maximum. Furthermore, some algorithms operate under the assumption that all features are centred around zero with a variance in the same order. Meeting these assumptions can lead to more accurate and reliable models.

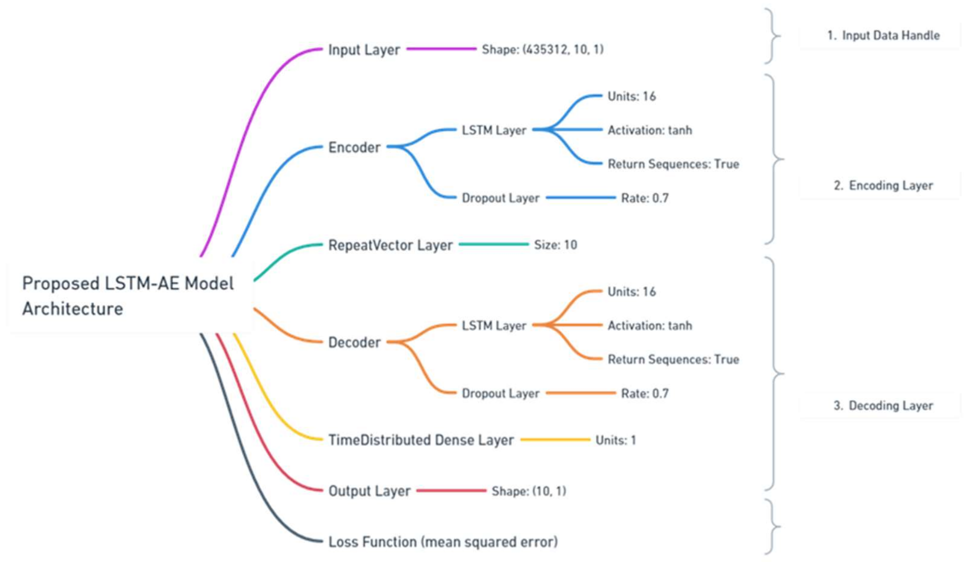
In our study's context, the StandardScaler was explicitly applied. First, the scaler was initialized and then fitted only to the training data. This approach ensures that the scaling mirrors real-world conditions, as typically, future data (similar to test data) is not accessible during the training phase. After fitting, the training data was transformed using the scaler to produce a standardized version. The same transformation was then extended to the test data. It is essential to highlight that the scaler was not fitted to the test data to prevent data leakage. This ensures that no information from the test set influences the training set, a practice vital for maintaining the model's integrity and generalizability.

5. Methodology

The methodology section elucidates the systematic approach adopted to develop, fine-tune, and validate the LSTM-AE model for anomaly detection in indoor air quality datasets. The process is segmented into three primary stages: designing the model architecture, selecting, and tuning hyperparameters, and devising training and validation strategies.

5.1 Model Architecture Design

The architecture of our LSTM-AE model is designed to cater specifically to time-series data, like the indoor air quality dataset in our study. The model comprises a few main steps: Input data handle, the LSTM-based encoder, the decoder, and Anomaly detection. The detailed architecture can be seen in the figure 11 below.



c

Figure 11. Architecture of proposed LSTM-AE model

Input Data Handling

The primary dataset is structed as a chronological time series, represented by [X_1, $X_2, X_3 \ldots, X_n$]. Each segment denotes X, encompasses data points within a consistent time window of length T. Illustrated as [$x_1, x_2, x_3, \ldots, x_n$]. Here each x_t being a vector in R^m, indicating its composition of m distinct features corresponding to the specific time point t.

To make this data more structured and suitable for analysis, it's transformed into a two-dimensional matrix. In this matrix format, the first axis corresponds to the individual segments or samples, while the second axis represents the consecutive positions or timesteps within each segment. As an illustration, when processing our CO2 concentration data, each segment is mapped into this 2D matrix format. Here, every row of the matrix signifies a distinct segment, and each column within that row corresponds to one of the 10 consecutive positions or timesteps of that segment. The figure 12 shown below briefly shows how the time-series data converts into input for LSTM-AE model.

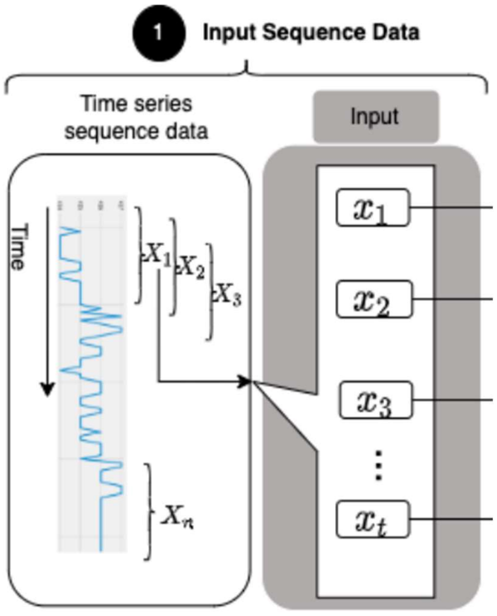


Figure 12. Time-series data converts into input for LSTM-AE model.

Encoder

The second step involves the LSTM Encoder. The encoder layer, termed LSTM, takes in data with a shape of (449112, 10, 1). However, within its structure, it primarily focuses on the last two dimensions, making its effective input shape (10, 1). This represents a time sequence spanning 10 steps, with each step characterized by a 1D vector. The output from this layer is a 16-dimensional vector, given its configuration of 16 units. As the LSTM layer typically returns the output from its final timestep, the resulting dimension is (None, 16), where 'None' can represent any batch size. This encoder's primary role is to compress the input data into a more concise, lower-dimensional representation.

The encoder follows the dropout layer, which takes in the output from the preceding LSTM layer, maintaining the shape (None, 16). This Dropout layer retains the input shape but occasionally sets specific neuron values to zero during the training phase. This sporadic dropping of inputs ensures the model doesn't become overly dependent on particular inputs, thereby minimizing the risk of overfitting.

Subsequently, the repeat vector layer accepts the output from the dropout layer. It then repeats the input 10 times, matching the timesteps in the initial data. This repetition transforms the 2D input into a 3D output with a shape of (None, 10, 16). Serving as a bridge between the encoder and decoder sections of the LSTM-AE, this layer replicates the encoder's compressed output across the time dimension, producing a 3D tensor suitable for input into the LSTM decoder. All the processing is shown in the figure below.

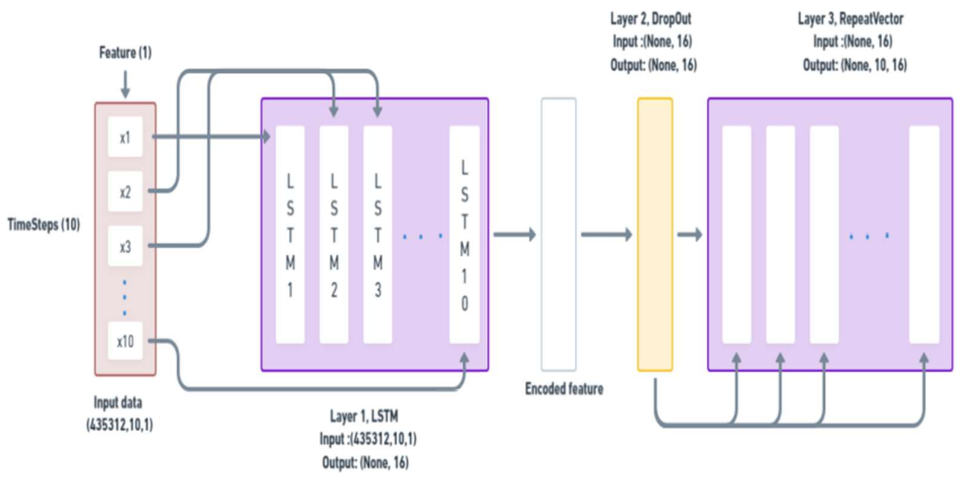

Feature (1)
TimeSteps (10)
x1
x2
x3
x10
Input data (435312,10,1)
LSTM 1
LSTM 2
LSTM 3
LSTM 10
Layer 1, LSTM Input :(435312,10,1) Output: (None, 16)
Encoded feature
Layer 2, DropOut Input :(None, 16) Output: (None, 16)
Layer 3, RepeatVector Input :(None, 16) Output: (None, 10, 16)

Figure 13. Encoder steps

Decoder

The third step encompasses the LSTM Decoder. The decoder layer, labelled as LSTM, ingests the output from the prior RepeatVector layer, which has a shape of (None, 10, 16). Given its configuration of 16 units, it produces an output for each of the 10 timesteps, retaining the shape (None, 10, 16). This decoder's primary function is to attempt the reconstruction of the input sequence based on the condensed representation.

After the decoder is the dropout layer, which accepts the output from the earlier LSTM layer, preserving the shape (None, 10, 16), this Dropout layer, akin to its predecessor, ensures the model's resilience against overfitting, especially within the decoder segment.

Following this is the time distributed layer, which takes in the output from the Dropout layer with a shape of (None, 10, 16). It then transforms this input into an output shape of (None, 10, 1). The TimeDistributed layer consistently applies a Dense operation to every timestep within a 3D tensor. As a result, it yields a sequence spanning 10 steps, with each step characterized by a 1D vector. This layer serves as the LSTM-AE's output layer, tasked with reconstructing the initial input sequence based on the encoded representation.

To encapsulate, this LSTM-Autoencoder model adeptly condenses an input sequence with a shape of (10,1) into a more streamlined representation with a shape of (16,).

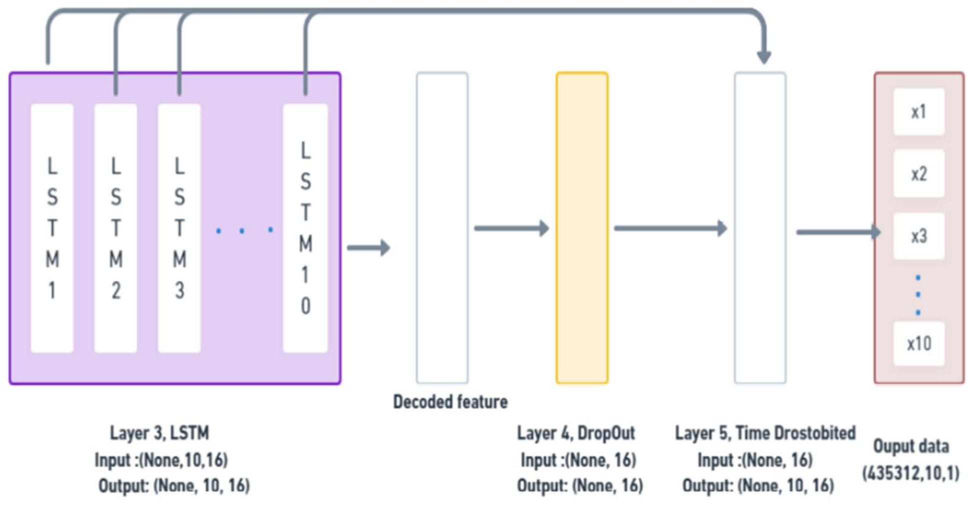


Figure 14. Decoder Steps

Anomaly Detection

Anomalies are often defined as data points that diverge considerably from the expected pattern. To pinpoint these deviations, a specific boundary or threshold is set. This boundary acts as a benchmark to gauge the degree of deviation. Data points that breach this boundary are labelled as anomalies.

In our methodology, the model is trained on a dataset that predominantly contains CO2 values within a standard range. This training ensures the model becomes familiar with the reconstruction error rates typical of normal CO2 readings. Once the training concludes, the peak reconstruction error rate observed across all training samples is chosen as the threshold. The model is then tested using a CO2 dataset that spans a wider range of readings. A reconstruction error rate is computed for each data point in this test set. Data points whose error rate exceeds the set threshold are marked as anomalous.

For illustration, consider five data points: [x1, x2, x3, x4, x5]. These are grouped into three time-series sequences: [X1, X2, X3]. Each sequence consists of three data points spread over three distinct time intervals. For instance, X1 encompasses [x1, x2, x3], X2 includes [x2, x3, x4], and X3 consists of [x3, x4, x5]. When the model processes these sequences, it produces corresponding reconstructed outputs.

Figure 15 shows how we calculate Reconstruction loss. Assuming the original sequence values are: X1 as [x1 = 1, x2 = 2, x3 = 3], X2 as [x2 = 2, x3 = 3, x4 = 4], and X3 as [x3 = 3, x4 = 4, x5 = 5]. The model might reconstruct these sequences as:

X1 being [x1 = 1.05, x2 = 2.03, x3 = 3.02],

X2 as [x2 = 2.02, x3 = 3.03, x4 = 4.01], and

X3 as [x3 = 3.02, x4 = 4.03, x5 = 5.01].

The reconstruction loss for each data point is computed as follows:

x1's loss is 0.05, determined by |1.05 - 1|.

x2's loss is 0.015, the average of the differences |2.03 - 2| and |2.02 - 2|.

x3's loss is 0.02, the mean difference of |3.02 - 3|, |3.03 - 3|, and |3.02 - 3|.

x4's loss is 0.015, averaged from the differences |4.01 - 4| and |4.03 - 4|.

x5's loss is 0.01, derived from |5.01 - 5|.

In this scenario, the highest reconstruction loss, 0.05, is the threshold. During the evaluation phase, any data point with a reconstruction loss that surpasses 0.05 is flagged as an anomaly.

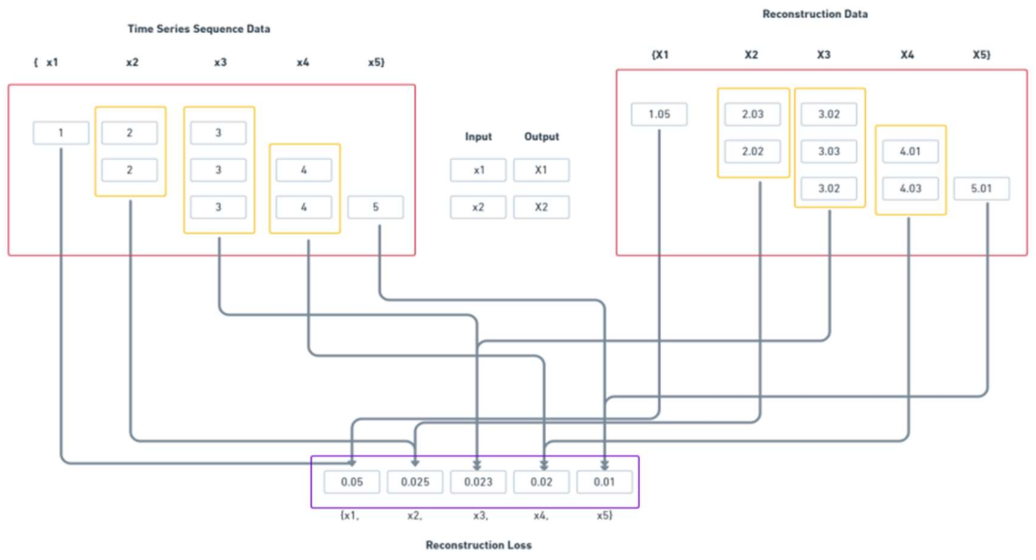


Figure 15, How to calculate Reconstruction loss from Time-series Data.

5.2 Hyperparameter Selection and Tunning

As the purpose of this project is to figure out the best-performing LSTM-AE model for a given dataset, hyperparameters play a pivotal role in determining the performance of deep learning models, including LSTM-Autoencoders. Unlike model parameters, which are learned during training, hyperparameters are set before the training begins. The right combination of hyperparameters can significantly enhance the model's ability to capture patterns in the data, while suboptimal settings can hinder its performance.

Importance in LSTM-AE:

In the context of LSTM-AE, hyperparameters dictate the architecture of the network (like the number of layers and units in each layer), the learning process, and the regularization techniques applied. They influence the encoding and decoding phases, affecting how well the model compresses the input data and how accurately it reconstructs it.

Key Hyperparameters in LSTM-AE:

Number of Layers and Units: The depth and width of the LSTM-AE determine its capacity. While a larger model might capture more complex patterns, it is also more prone to overfitting, especially with limited data. Conversely, a smaller model might only capture some of the nuances in the data.

Learning Rate: This hyperparameter controls the step size during optimization. A high learning rate might cause the model to converge quickly, but it can overshoot the optimal solution. A low learning rate ensures more precise convergence but can be time-consuming.

Dropout Rate: Dropout is a regularization technique where random neurons are "dropped" or turned off during training. It helps prevent overfitting by ensuring the model only relies somewhat on one neuron.

Batch Size: It determines the number of samples used to update the model's weights in each iteration. Larger batches offer more stable gradient estimates, while smaller batches can make the optimization process noisier, potentially allowing the model to escape local minima.

Sequence Length: In LSTM-AE, the length of the input sequences can be a crucial hyperparameter, especially when dealing with time-series data. It determines how many time steps the model considers when encoding and decoding the input.

6. Experimental Setup and Result

6.1 Experimental Setup

GPU	NVIDIA GeForce RTX 3060
CPU	Intel® Core™ i5- 10400 CPU @ 2.90Hz (12 CPUs), ~2.9Hz
OS	Window 11 PRO 64 bit
RAM	16384 MB RAM
TensorFlow	2.10.1
Sklearn	1.2.2

Table 1. Hardware/Software setup for training

Model Architecture	1 Hidden layer with 16 Units
Learning rate	1e-05
Dropout rate	0.7
Batch Size	64
Time window Size	10

Table 2. Hyperparameter setup for the best-performing model

6.2 Evaluation Matrix

The evaluation of an anomaly detection model is crucial to understand its effectiveness and reliability. In our study, we employed the confusion matrix as the primary evaluation matrix. The confusion matrix is a widely used tool in machine learning and statistics to visualize and understand the performance of an algorithm, especially in classification problems. A confusion matrix provides a breakdown of predictions into a table showing correct predictions and the types of incorrect predictions made. It is essentially a table with two dimensions ("Actual" and "Predicted"), and each dimension has two possible outcomes (Anomaly or Normal). This results in a 2x2 matrix structure, where each matrix cell represents a specific type of prediction described below,

True Positives (TP): These are the cases in which the model correctly predicted the anomaly.

True Negatives (TN): These are the cases in which the model correctly predicted normal behaviour.

False Positives (FP): These are the cases in which the model incorrectly predicted an anomaly.

False Negatives (FN): These are the cases in which the model failed to detect an anomaly.

The confusion matrix provides a more detailed breakdown of the model's performance than just the accuracy metric. It allows for calculating several other metrics like precision, recall, and F1-score, which can give a more comprehensive view of the model's performance.

Precision: It is the ratio of correctly predicted anomalies to the total predicted anomalies. It is a measure of the model's accuracy in predicting anomalies.

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

Recall: The ratio of correctly predicted anomalies to the total actual anomalies. It measures the model's ability to detect all potential anomalies.

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

F1-Score: It is the harmonic mean of precision and recall and balances the two. An F1 score reaches its best value at 1 (perfect precision and recall) and the worst at 0.

$$F1 = \ 2\, X\ \frac{Pecision\ \ x\ Recall}{Precision + Recall}$$

Accuracy: It measures the proportion of all correct predictions (anomalies and normal) in the dataset. While it is a general measure of the model's performance, in imbalanced datasets, like many anomaly detection scenarios, relying solely on accuracy can be misleading.

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

In the context of anomaly detection, especially when anomalies are rare, the confusion matrix and the derived metrics are fundamental. They provide a nuanced understanding of how well the model is performing, ensuring that it not only detects anomalies but also reduces the number of false alarms.

6.3 Results of LSTM-AE Models with Different Architecture and Hyperparameters

In the pursuit of optimizing the LSTM-Autoencoder (LSTM-AE) for anomaly detection, various model architectures and hyperparameters were experimented with. The objective was to discern the combination that yields the highest anomaly detection accuracy and efficiency.

Variations in Model Architecture

The architecture of the LSTM-AE was a primary focus of our experimentation. We explored:

Number of Hidden Layers: Models with varying depths were constructed, ranging from a single hidden layer to more complex architectures with up to three hidden layers. The depth of a model can profoundly influence its capacity to learn and reconstruct sequences, which subsequently impacts its anomaly detection capability.

Number of Units: For each of these architectures, the number of units in the LSTM layers was varied, starting from 16 and extending up to 128 units. The number of units can dictate the granularity at which the model captures patterns in the data.

In Figure 16, we observe subtle variations in performance across different model architectures. Notably, the model with a single hidden layer containing 16 units, as well as all two-hidden-layer models, achieves the highest scores across all metrics.

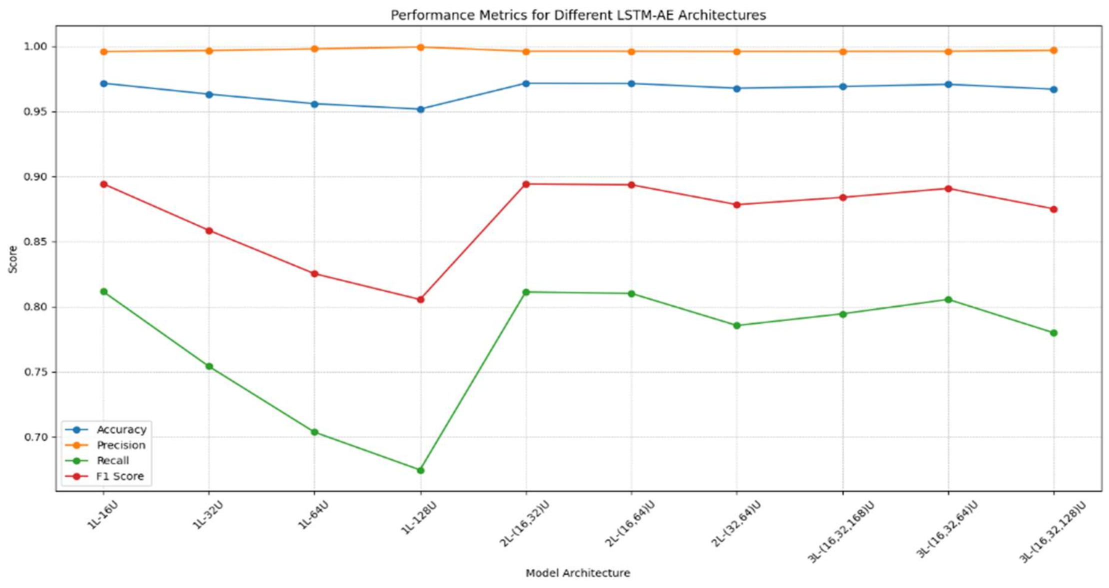


Figure 16. Performance comparison with Different model architectures.

However, as depicted in Figure 17, the time taken per epoch escalates significantly with adding more hidden layers. Thus, **the single hidden layer model with 16 units offers comparable performance** to architectures with multiple hidden layers but is considerably more time-efficient per epoch.

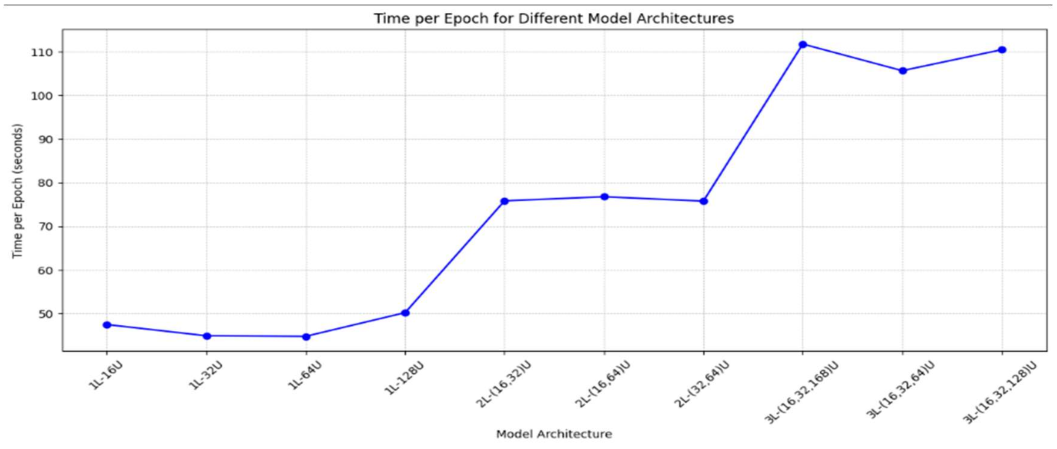


Figure 17. Comparison time taken per epoch with different model architectures.

After determining the optimal architecture for our LSTM-AE model, the next step was to fine-tune the hyperparameters to optimize the model's performance. This tuning involved adjusting the hyperparameters outlined in the Methodology section. The initial aspect I evaluated was the influence of the time window size on the LSTM-AE model. By incrementally adjusting the window size from 5 to 40 in step 5, it was observed that while the time per epoch experienced a slight increase, it was not substantial. Notably, a window size 10 yielded competitive scores across all evaluation metrics. Figure 18 shows performance in terms of time window size.

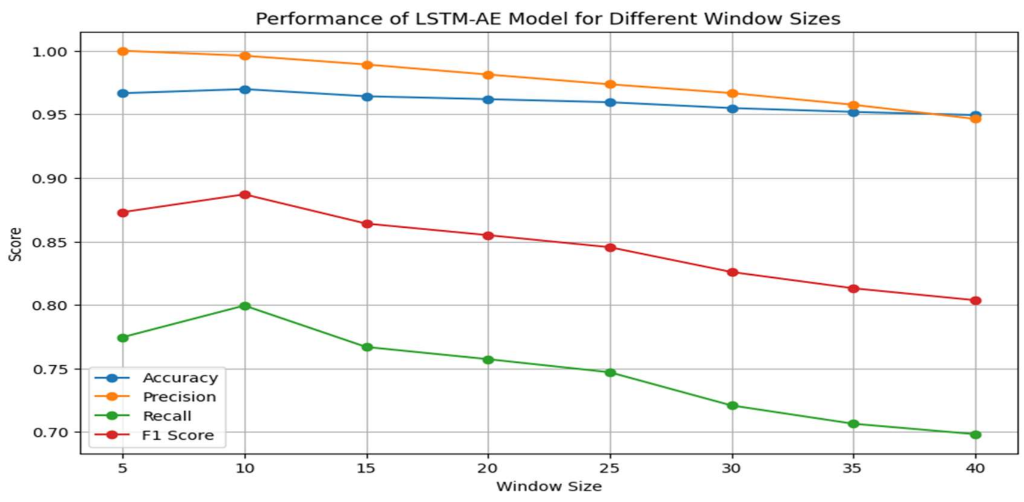


Figure 18. Comparison models' performance with different time window size.

Following the adjustment of the time window size, the focus shifted to fine-tuning the batch size and learning rate. As depicted in Table 3, there is a clear variation in performance and time per epoch contingent on the batch size and learning rate. After a thorough evaluation, it was ascertained that a learning rate of 1e-5 combined with a batch size of 128 yielded the most optimal results.

Learning Rate	Batch Size	Accuracy	Precision	Recall	F1 Score	Time/Epoch (sec)
0.01	**64**	0.9610	0.9947	0.7410	0.8493	44.93
0.001	**64**	0.9739	0.9952	0.8277	0.9037	44.50
0.0001	**64**	0.9636	0.9975	0.7566	0.8605	49.13
0.00001	**64**	0.9740	0.9965	0.8277	0.9043	44.83
0.00001	**32**	0.9675	0.9971	0.7830	0.8771	83.67
0.00001	**128**	**0.9900**	**09910**	**0.9409**	**0.9653**	**21.47**

Table 3 Performance comparison with different Learning rate and Batch size.

The final adjustment made was to the dropout rate. As illustrated in Figure 19, the performance of the LSTM-AE model was examined across various dropout rates. The rates were incrementally adjusted from 0.2 to 0.7 in steps of 0.1. It was observed that setting the dropout rate beyond 0.7 led to overfitting of the model. This analysis was crucial in pinpointing the ideal dropout rate that not only bolstered the model's resilience but also ensured it did not compromise performance due to overfitting.

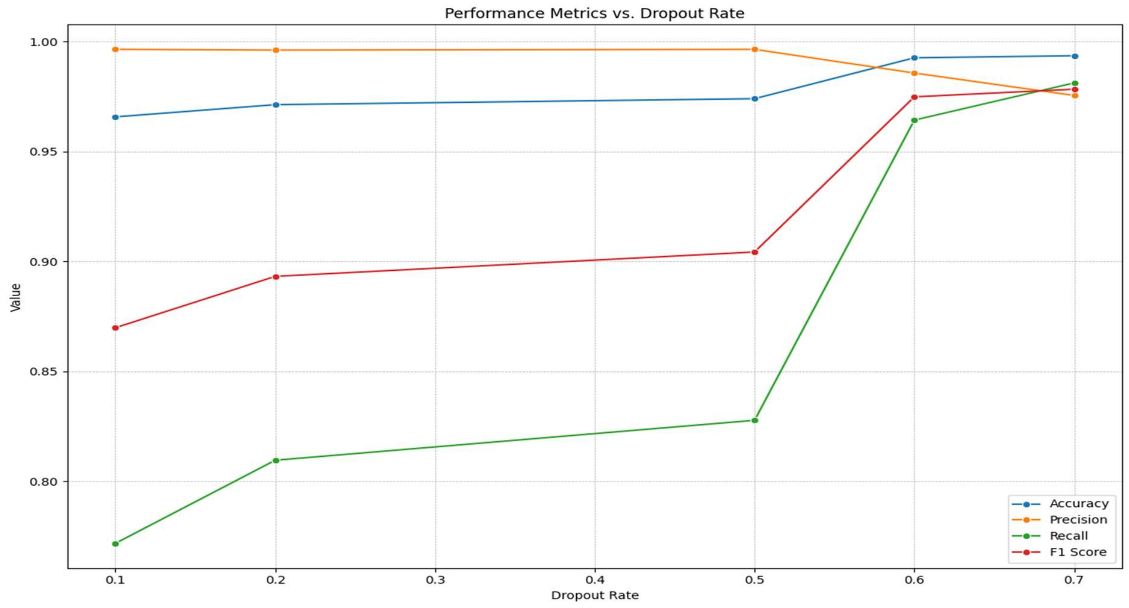


Figure 19. Performance metrics vs Dropout Rate

Performance of the Best-Performing LSTM-AE model

Figure 20 illustrates the progression of the loss across various epoch intervals. The training loss, represented by the blue curve, gauges the model's error rate during its training phase. The training loss reaches a steady state roughly by the tenth epoch. Contrary to expectations, the validation loss only stabilizes after the tenth epoch. Nevertheless, post the tenth epoch, the validation loss mirrors the training loss, with an average rate close to 0.15%. This indicates a well-fitted model, suggesting that our designed model neither overfits nor underfits.

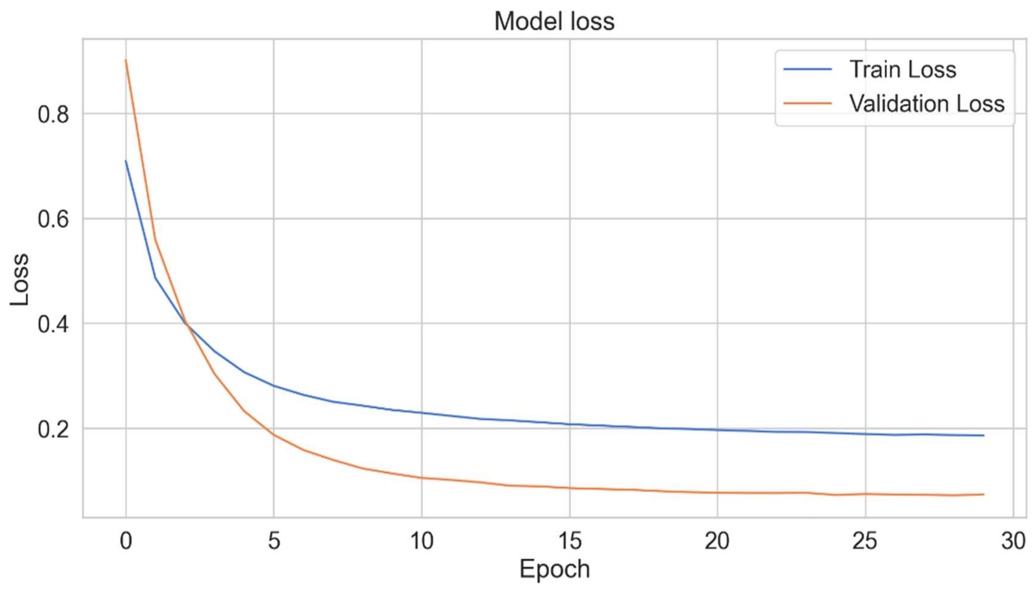


Figure 20. Training loss vs Validation loss.

Figure 21 displays the efficacy of our model using a confusion matrix. The test set comprised 49,536 samples, with 41,179 labelled as normal and 8,218 as abnormal. From the 8,218 abnormal samples, our model correctly identified 7,975, translating to an accuracy of 97.04%. Additionally, it accurately recognized all 41,179 normal samples, achieving 99.66% accuracy in this category. The model did not produce false positives by mislabeling normal samples as abnormal. However, it did have 243 false negatives where it mislabeled abnormal samples as normal. Taking all factors into account, the model achieved an overall accuracy of 99.23%, with a precision of 98.29%, a recall rate of 97.04%, and an F1-score of 97.66%

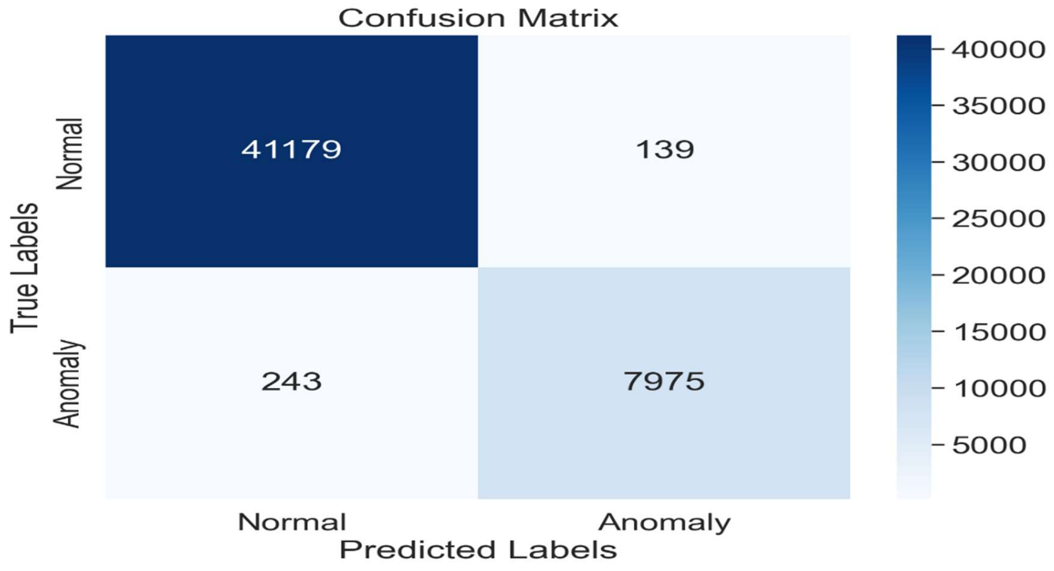


Figure 21. Confusion matrix of the best-performing model

Figure 22 below also represents detected anomalies based on the results from our proposed LSTM-AE model as red dots in the figure. The points over the threshold (1.5396) are detected as anomalies.

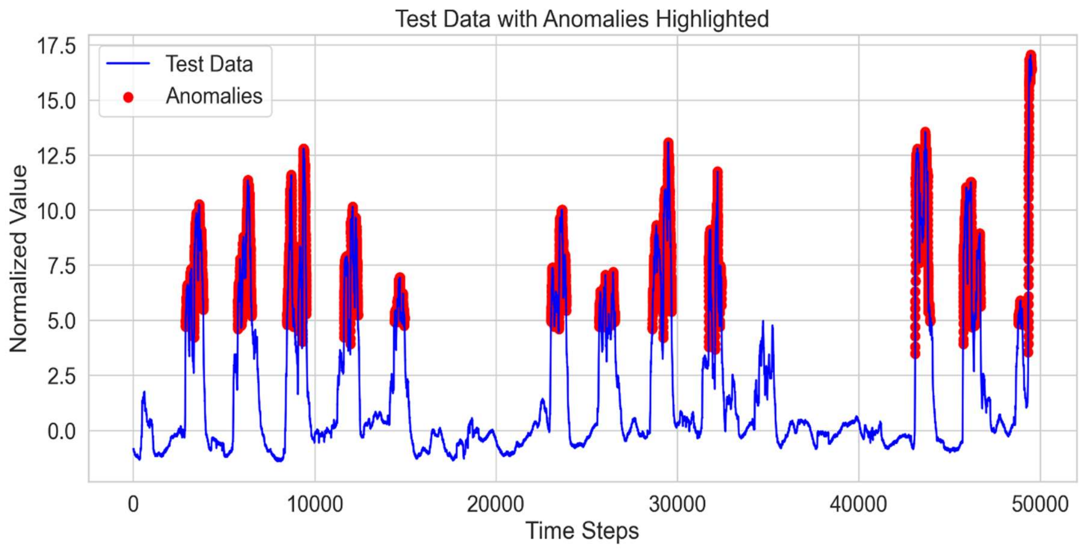


Figure 22. anomalies in the test dataset

6.4 Comparative Analysis with Other Approaches

In this section, we aim to provide a comparative analysis of our proposed model against several existing methods in the domain. This comparison is essential to gauge the effectiveness and efficiency of our approach in the broader context of the problem domain. We have selected LSTM-AE methods widely acknowledged in the literature for their performance and relevance to the problem as shown in the table 4.

Author	Datasets	Accuracy	Precision	Recall	F1 Score
Yin et al. [24]	Yahoo Web scope S5	99.25	97.84	94.16	95.97
Liu et al. [2]	ECG	98.57	97.55	97.55	-
Kang et al. [25]	Break Operating Unit (BOU) data	94.44	97.94	85.77	91.45
Junlin Yin	UQ ITEE $CO2$ data	99.11	98.72	95.22	96.94
Sejin Son	UQ ITEE $CO2$ data	99.23	98.29	97.04	97.66

Table 4. Comparison with existing LSTM-AE models

7. Conclusion and Future Work

In our research, we delved deep into the intricacies of the LSTM-AE model, aiming to pinpoint anomalies in air quality data, with a special emphasis on irregular $CO2$ concentrations. We meticulously analysed various elements that could sway the model's outcomes, such as the duration of the time window, the model's structure, and specific parameters. Through rigorous testing, the LSTM-AE model proved its mettle in consistently spotting anomalies across diverse setups. A comparative study with other established models further underscored the superior capabilities of our LSTM-AE approach in monitoring air quality anomalies.

Having established the LSTM-AE model's prowess in detecting anomalies, the logical progression is to unearth the underlying causes of these irregularities. Identifying the root causes can provide actionable insights, enabling timely interventions and preventive measures. This could involve analysing external factors, such as weather patterns, industrial activities, or traffic density, influencing air quality fluctuations.

Furthermore, expanding the scope of our research to encompass a broader range of data sources can offer a more holistic view. Diverse datasets, perhaps from different geographical regions (different buildings) or collected during varying seasons, can provide a richer context. This expanded data horizon can help in understanding the universality and adaptability of our model.

To further validate the superiority of our LSTM-AE model, it is imperative to juxtapose it with other anomaly detection techniques. By benchmarking against different methodologies, we can ascertain their relative strengths and areas of improvement. Such a comparative analysis can also shed light on scenarios where our model excels or where alternative techniques might be more apt.

Lastly, transitioning from mere detection to forecasting anomalies can be a game-changer. Predictive capabilities can empower stakeholders to take pre-emptive actions, minimizing the adverse effects of potential air quality deteriorations. Integrating forecasting mechanisms with our LSTM-AE model can thus be a pivotal step in our ongoing research journey.

8. Bibliography

[1] Manisalidis, I., Stavropoulou, E., Stavropoulos, A. and Bezirtzoglou, E. (2020). Environmental and Health Impacts of Air pollution: a Review. *Frontiers in Public Health*, [online] 8(14), pp.1–13. doi:https://doi.org/10.3389/fpubh.2020.00014.

[2] Liu, Y., Pang, Z., Karlsson, M. and Gong, S. (2020). Anomaly Detection Based on Machine Learning in IoT-based Vertical Plant Wall for Indoor Climate Control. *Building and Environment*, 183, p.107212. doi:https://doi.org/10.1016/j.buildenv.2020.107212.

[3] Melichov, M. (2022). *Time Series Anomaly Detection With LSTM AutoEncoder*. [online] Medium. Available at: https://medium.com/@maxme006/time-series-anomaly-detection-with-lstm-autoencoder-b13a4177e241 [Accessed 24 Oct. 2023].

[4] Lechner, M. and Hasani, R. (2020). *Learning Long-Term Dependencies in Irregularly-Sampled Time Series*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2006.04418.

[5] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780. doi:https://doi.org/10.1162/neco.1997.9.8.1735.

[6] Influxdata (2022). *What is time series data? | Definition and Discussion*. [online] InfluxData. Available at: https://www.influxdata.com/what-is-time-series-data/.

[7] Locascio, J.J. and Atri, A. (2011). An Overview of Longitudinal Data Analysis Methods for Neurological Research. *Dementia and Geriatric Cognitive Disorders Extra*, 1(1), pp.330–357. doi:https://doi.org/10.1159/000330228.

[8] Lara-Benítez, P., Carranza-García, M. and Riquelme, J.C. (2021). An Experimental Review on Deep Learning Architectures for Time Series Forecasting. *International Journal of Neural Systems*, 31(03), p.2130001. doi:https://doi.org/10.1142/s0129065721300011.

[9] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408. https://doi.org/10.1037/h0042519

[10] Singh, H. (2021). *Neural Network | Introduction to Neural Network | Neural Network for DL*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/.

[11] Montesinos López, O.A., Montesinos López, A. and Crossa, J. (2022). Fundamentals of Artificial Neural Networks and Deep Learning. *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, pp.379–425. doi:https://doi.org/10.1007/978-3-030-89010-0_10.

[12] Singh, H. (2021). *Neural Network | Introduction to Neural Network | Neural Network for DL*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/.

[13] Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4), pp.611–629. doi:https://doi.org/10.1007/s13244-018-0639-9.

[14] Chandola, V., Banerjee, A. and Kumar, V. (2009). Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3), pp.1–58. doi:https://doi.org/10.1145/1541880.1541882.

[15] Mingyan Teng. (2010) "Anomaly detection on time series". In: 2010 IEEE International Conference on Progress in Informatics and Computing. Vol. 1. IEEE. , pp. 603–608.

[16] Victoria Hodge, Jim Austin. (2004). "A survey of outlier detection methodologies". In: Artificial intelligence review 22.2 ,pp. 85–126

[17] Yang Zhang, Nirvana Meratnia, and Paul JM Havinga. (2010). "Outlier detection techniques for wireless sensor networks: A survey." In: IEEE Communications Surveys and Tutorials 12.2 , pp. 159–170.

[18] Chalapathy, R. and Chawla, S. (2019). Deep Learning for Anomaly Detection: A Survey. *arXiv:1901.03407 [cs, stat]*. [online] Available at: https://arxiv.org/abs/1901.03407.

[19] Park, D., Hoshi, Y. and Kemp, C.C. (2017). *A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-based Variational Autoencoder*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.1711.00614.

[20] S. Lin, R. Clark, R. Birke, S. Schonborn, N. Trigoni, and S. Roberts, (2020) ¨ "Anomaly detection for time series using vae-lstm hybrid model," in ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 4322–4326.

[21] Y. Jung, T. Kang, and C. Chun, (2021) "Anomaly analysis on indoor office spaces for facility management using deep learning methods," Journal of Building Engineering, vol. 43, p. 10313

[22] Srivastava, N., Mansimov, E. and Salakhutdinov, R. (2016). *Unsupervised Learning of Video Representations using LSTMs*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.1502.04681.

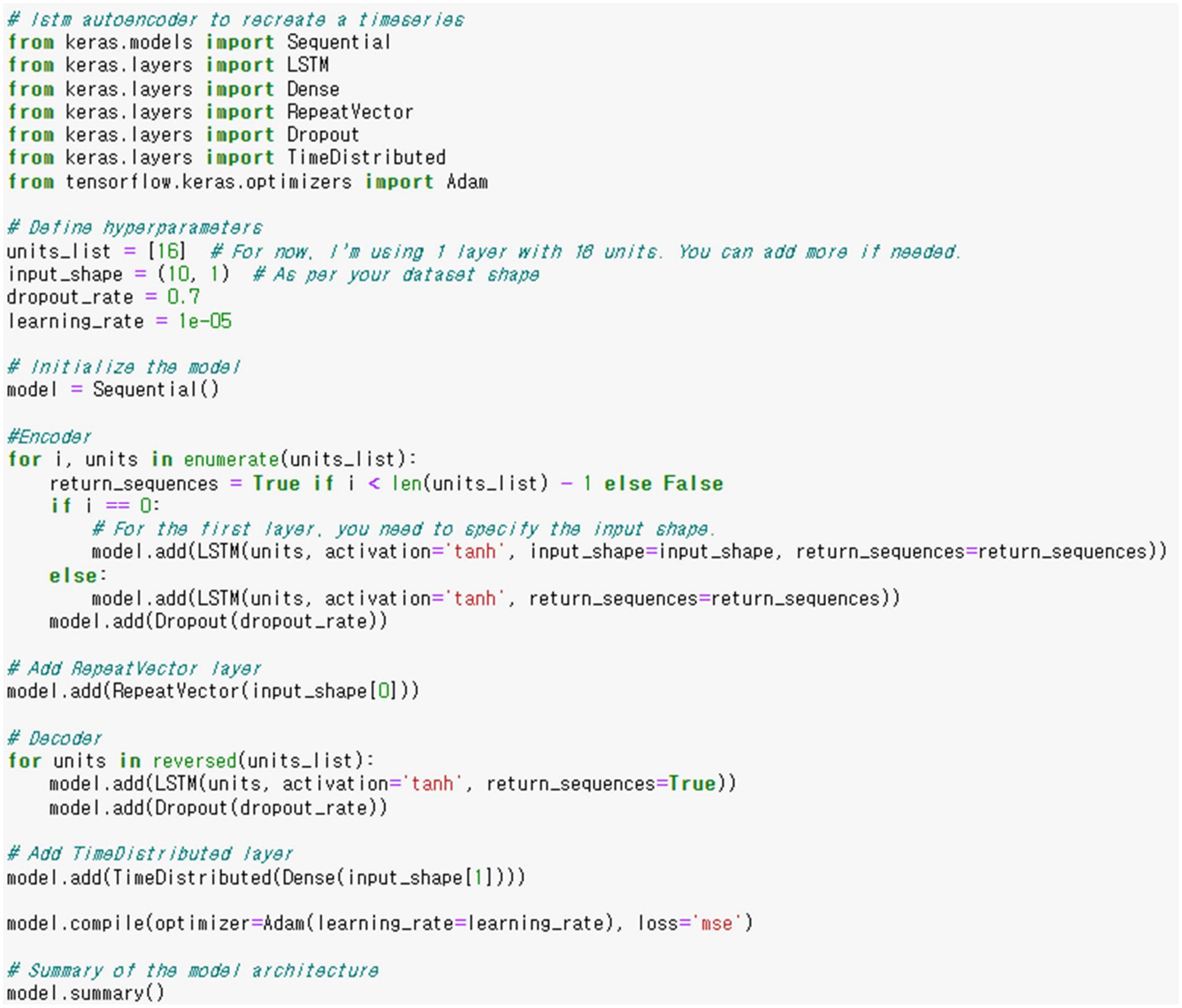
[23] Bhandari, A. (2020). *Feature Scaling | Standardization Vs Normalization*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

[24] Almaraz-Rivera, J.G. (2023). An Anomaly-based Detection System for Monitoring Kubernetes Infrastructures. *IEEE Latin America Transactions*, [online] 21(3), pp.457–465. Available at: https://latamt.ieeer9.org/index.php/transactions/article/view/7408 [Accessed 31 Oct. 2023].

[25] Kang, J., Kim, C.-S., Kang, J.W. and Gwak, J. (2021). Anomaly Detection of the Brake Operating Unit on Metro Vehicles Using a One-Class LSTM Autoencoder. *Applied Sciences*, 11(19), p.9290. doi:https://doi.org/10.3390/app11199290.

Appendix

Appendix 1 LSTM-AE Model in this Project



```python
# lstm autoencoder to recreate a timeseries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import Dropout
from keras.layers import TimeDistributed
from tensorflow.keras.optimizers import Adam

# Define hyperparameters
units_list = [16]  # For now, I'm using 1 layer with 16 units. You can add more if needed.
input_shape = (10, 1)  # As per your dataset shape
dropout_rate = 0.7
learning_rate = 1e-05

# Initialize the model
model = Sequential()

#Encoder
for i, units in enumerate(units_list):
    return_sequences = True if i < len(units_list) - 1 else False
    if i == 0:
        # For the first layer, you need to specify the input shape.
        model.add(LSTM(units, activation='tanh', input_shape=input_shape, return_sequences=return_sequences))
    else:
        model.add(LSTM(units, activation='tanh', return_sequences=return_sequences))
    model.add(Dropout(dropout_rate))

# Add RepeatVector layer
model.add(RepeatVector(input_shape[0]))

# Decoder
for units in reversed(units_list):
    model.add(LSTM(units, activation='tanh', return_sequences=True))
    model.add(Dropout(dropout_rate))

# Add TimeDistributed layer
model.add(TimeDistributed(Dense(input_shape[1])))

model.compile(optimizer=Adam(learning_rate=learning_rate), loss='mse')

# Summary of the model architecture
model.summary()
```