Neuro-Evolution for Multivariate Time Series Anomaly Detection

by

Aryan Jha

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences
Department of Computer Science
in partial fulfillment of the requirements for the
Master of Science Degree in Computer Science
at the Rochester Institute of Technology

Abstract

Given the exponential growth of data in various sectors, the timely identification of anomalies has become crucial in mitigating risks such as financial losses and data breaches. This research focuses on advancing multivariate time series anomaly detection through the innovative application of neuroevolutionary methods. Utilizing the Evolutionary eXploration of Augmenting Memory Models (EXAMM) algorithm, the study automates the design and optimization of Long Short-Term Memory (LSTM) autoencoder networks, enabling enhanced forecasting capabilities in recurrent neural network architectures. Leveraging these networks along with a One-Class Support Vector Machine (OCSVM) to generate boundary thresholds, this project delivers a robust and scalable procedure for identifying anomalies that outperforms existing proven methods in terms of accuracy and cumulative F1 score, while at the same time requiring orders of magnitude fewer trainable parameters. By comparing the reconstruction error of trained forecasts, the proposed approach facilitates accurate anomaly detection across diverse datasets against unseen anomalous data and informs efficient design decisions for future autoencoder network architectures.

Acknowledgments

I would like to sincerely thank the guidance and support of my thesis committee - Dr. Travis Desell, Dr. Alexander Ororbia, and Dr. Rui Li - who made this project possible.

THESIS SUPERVISORY COMMITTEE:

Approved: _____

Alexander Ororbia, Ph.D.
Thesis Co-Chair

Approved: _____

Travis Desell, Ph.D.
Thesis Co-Chair

Approved: _____

Rui Li, Ph.D.
Thesis Observer

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis aims to advance the field of multivariate time series anomaly detection by leveraging neuro-evolutionary methods. Specifically, it seeks to extend the Evolutionary eXploration of Augmenting Memory Models (EXAMM) algorithm to automate the design and optimization of autoencoders for multivariate time series forecasting on recurrent neural network architectures. The output of these autoencoders can then be used to perform anomaly detection by comparing the reconstruction errors for known normal data against the reconstruction errors for unknown (possibly anomalous) data.

A core objective of this project was to develop a robust and scalable procedure for anomaly detection that stands on a par with, or even surpasses some currently used methods in terms of accuracy, precision, and recall, while utilizing more efficient automatically designed architectures. Through rigorous evaluation and comparison with existing methods, this research aims to provide a significant contribution to the field of anomaly detection, offering an automated solution to identify critical deviations in multivariate time series data. Additionally, another key goal of this work was to identify any trends within the evolved autoencoders that could potentially inform improvements in their design. To the best of our knowledge, this is the first work of its kind that utilizes neuro-evolution to automate the design and training of autoencoders for multivariate time series anomaly detection.

1.1 Problem

Anomaly detection in time series data basically derives from data analyses methods that identify significant deviations from the normal behavior of temporally dependent attributes. In its most basic and traditional usage, detected

outliers could be omitted from the dataset to minimize the impact of noise or atypical events on any statistical analysis or inference made from the data [14]. However, more recently anomalies themselves have become points of interest; the detection of these outliers are useful to identify underlying issues within the observed processes [32]. These instances could result in significant financial losses, data breaches, and other harmful events. For example, the timely recognition of abnormalities in user behavior could be crucial in fraud and intrusion detection.

There have been documented substantial increases in connectivity and data sharing in the past few decades that have exceeded Moore's Law since the beginning of this century [34], with the global population of data generated in 2024 alone estimated to be 149 zettabytes (149 billion terabytes) [36]. The use of big data in particular has taken over almost every sector of the economy and society today - scientific research, internet of things, e-commerce, health care, finance, navigation, and even national security are essentially and critically dependent on employing the benefits of such systems [8]. The amount of sensors used in modern applications and the size of time-series data collected through them makes constant manual detection of anomalies in real-time virtually impossible, necessitating the need to automate this process.

The characterization of every possible normal behavior using strict thresholds is very challenging - more so in the case of interdependent multivariate data. The classification of abnormal behavior becomes especially difficult near boundaries, often requiring a trade-off between achieving high accuracy and minimizing false positives. Moreover, the characteristics of anomalous data tend to be very domain specific, lowering the adaptability and general applicability of any intelligent agents developed for anomaly detection [37].

1.2 Thesis

The application of neuro-evolutionary methods using long short-term memory structures to capture multivariate temporal patterns, facilitates the creation of highly optimized RNN architectures that significantly improve the accuracy, precision and recall of high dimensional time series anomaly detection when compared to existing methods.

1.3 Research Questions

The following is a list of the key research questions that were explored in this project:

- Does the neuro-evolutionary process facilitate the creation of more efficient autoencoder networks, measured by the number of trainable parameters, while remaining competitive—or superior—in performance compared to other established solutions for anomaly detection?
- What are the architectural preferences of the autoencoder networks that are generated through the neuro-evolutionary mechanism? Specifically, what are the patterns (if any) in the types and locations of nodes and edges that are used in the autoencoders produced through neuro-evolution?
- Does the evolutionary pressure to add only components that help in lowering the reconstruction error help prevent overfitting in neuro-evolved networks?

1.4 Criteria for Success

This project was deemed successful based on the performance of the proposed approach using the composite F-score metric, i.e. the harmonic mean of precision and recall values, against benchmark methods for multivariate time series anomaly detection. Demonstrating higher accuracy, precision and recall rates in detecting anomalies across benchmark multivariate time series datasets establishes the efficacy of the proposed approach in identifying most anomalies accurately without supervision. Validating the model's performance across diverse domains and datasets would also showcase the robustness, generalizability and scalability of the proposed solution.

Along with this report, the complete codebase of the developed system, including data pre-processing, data post-processing and evaluation tools would also be made available publicly as an open-source project.

Chapter 2

Related Work

Chalapathy and Chawla [2] provide a structured and comprehensive survey of research methods in deep learning-based anomaly detection, along with an assessment of their effectiveness and current limitations. Li and Jung [20] provide a classification for anomaly types and review the state-of-the-art deep learning techniques for the detection of each type.

There are three categories of anomalies: abnormal time points, abnormal time intervals, and abnormal time series. An abnormal time point is described as a point in a time series that deviates significantly from the values observed at other time points. These usually represent noise, rare outliers, or faults/bugs in a system. An abnormal time interval represent longer periods of time during which a monitored variable or variables behave significantly differently from its usual pattern. These often indicate unusual events like sustained system failures or behavioral changes in a system. Lastly, abnormal time series are entire sequences of data points over time that significantly deviate from other comparable time series. These time series may reveal system wide failures or undiscovered behavioral patterns. This project focuses primarily on the solutions for abnormal time interval detection.

This rest of this section will provide an overview and discussion on some of the more commonly used methods and models for anomaly detection.

2.1 Supervised Anomaly Detection

Supervised deep anomaly detection utilizes labeled data of both normal and anomalous instances to train a deep supervised binary or multi-class classifier. Yeung et al. [1], for instance, labeled the classes (1-anomalous; 0-normal) for each timestamp in a financial time series using the structural-break model,

including both testing and training data sets, and applied a combination of several jump classification algorithms to learn the features of the labeled data. They treated anomaly detection as a class-imbalance problem. Similarly, Chen et al. [4] introduce a new supervised anomaly detector method - Ensemble Active Learning Generative Adversarial Network (EAL-GAN). They use a Conditional GAN with one generator and multiple discriminators to overcome the class imbalance issue of rarity of labeled anomalies in data. The generator recreates realistic data to train the discriminators, and the average of anomaly score outputs from each discriminator is used to classify data as normal or anomalous.

Such supervised models of anomaly detection tend to be very accurate and straightforward to optimize. Having prior knowledge of anomaly labels also has the added benefit of being able to tailor model parameters to best fit the characteristics of the given domain. However, manually labeling anomalous instances across large amounts of time-series data becomes impractical and the scarcity of appropriate labeled data examples becomes a major challenge for this methodology, especially when labeling sub-sequence anomalies. Additionally, supervised learning methods can only identify anomaly occurrences for known anomaly types [26] and the training process inherently eliminates any possibility for online anomaly detection.

2.2 Unsupervised Anomaly Detection

Due to the mentioned drawbacks for supervised methods of anomaly detection, they tend to have limited usage; unsupervised approaches are more widely applicable and hence more desirable. Unsupervised anomaly detection methods can be roughly sub-sectioned into: traditional methods and deep methods.

2.2.1 Traditional Methods

Some of the most popular traditional anomaly detection methods tend to be either density-based, reconstruction-based, clustering-based, distance-based or ensemble detectors.

Density-based methods typically calculate the density function of the data and identify significant deviations from the density peaks as anomalies. Yang, Latecki and Pokrajac [44] demonstrate the use of a Gaussian Mixture Model (GMM), fitting Gaussian distributions to the data using the Expectation-Maximization (EM) algorithm. The estimated mixture proportions are interpreted as probabilities of being a cluster center for all data points, so the ones with the smallest likelihood are identified as anomalous. Similarly, Latecki,

Lazeric and Pokrajac [19] show the use of the use of Kernel Density Estimation (KDE) for anomaly detection, building a variable kernel function for each data point, yielding a robust local density estimation. Outliers are detected as having different local densities from their neighbors.

Reconstruction-based methods project the original data into a latent space that has more instance separability, and reconstruct the original instance from the latent space. Anomalies are identified as data with high reconstruction error. Principal Component Analysis (PCA) is popularly used to reduce the dimensionality of data while retaining its essential variability, allowing for the efficient and effective identification of deviations from expected patterns [5]. Liang, Chen and Schneider [42] provide an algorithm for Direct Robust Matrix Factorization (DRMF) to simplify the dataset into a low-rank approximation that captures the main patterns of the data while ignoring the outliers. Through minimizing errors in this simplified version over many iterations, points that do not fit the normal patterns are identified.

Clustering-based methods create segregated cluster groups of similar data. The data that is located away from any cluster centers, or the ones located in very sparse and small clusters get identified as anomalies. A recent example of this method, Nouretdinov et al. [29] introduce Multi-Level Conformal Clustering (MLCC) that automatically determines the number of clusters based on a set significance level and offers statistical robustness without assumptions about data distribution, enabling simultaneous clustering and anomaly detection.

Distance-based methods are distinct from previously listed traditional unsupervised learning methods in that they are not concerned with the overall distribution of a dataset, instead just using distance measures to identify anoamlies. k-Nearest Neighbor (kNN) is a very popular distance-based method - the rarity of each data instance is measured according to the distance to their k nearest neighbors in the data set. Ramaswamy, Rastogi and Shim [35] provide an early work that demonstrates the high efficacy in using the kNN method to mine outliers from data. Theissler et al. [39] propose a semi-supervised approach that uses the unsupervised feature extraction of the time series classifier ROCKET (RandOm Convolutional KErnel Transform), proposed by Dempster et al. [6], for anomaly detection purposes. They call their anomaly detector - the Random Convolutional Kernel Transform Anomaly Detector (ROCKAD), and it transforms a time series to a more abstract feature space, capturing its temporal information. They also use a nearest neighbor approach between transformed time series and transformed normal time series to get their anomaly scores.

Ensemble learning combine several diverse and complementary models to compound and improve upon the efficacy benefits of the sum of its singular detector parts. Isolation forest (iForest) [23] has emerged as arguably the most popular anomaly ensemble in recent years, that make recursive random splits on feature values, creating an ensemble of tree detectors that effectively detect anomalies based on the concept of isolation. Xu et al. [43] recently developed a deep isolation forest - using initialized neural networks to map original data into random representation ensembles.

2.2.2 Deep Methods

Deep learning methods work particularly well for anomaly detection in time series data. Specifically, recurrent neural networks (RNNs) have shown promise in encapsulating the complex and dynamic nature of multivariate time series data [16], and is a useful tool to capture the temporal patterns that indicate anomalies. This section reviews the deep learning approaches for detecting abnormal time intervals in multivariate time series. These approaches have been classified here in two categories: Dynamic graph-based approaches and LSTM-based approaches.

Dynamic Graph representations model the relationship patterns between multivariate time series to construct a single graph for each time interval. They use the correlation coefficients between time series channels as the weights for the edges in the graphs. This allows for the detection of anomalous time intervals since the relationships between the multivariate time series at these intervals would be notably different from those at other time intervals [20]. Deng and Hooi [7] combine structure learning and Graph Neural Networks (GNN) to detect time interval anomalies. Time intervals and their relationships with each other get encoded into static graphs, so a graph attention model could be used on them to forecast future time intervals for anomaly comparison. In another example, Zhao et. al [45] propose a Multivariate Time-series Anomaly Detection via Graph Attention Network (MTAD-GAT) framework that extracts time series features into 1D convolutional kernels, using graph attention layers to emphasize the relationships between features and timestamps. They then use a Graph Convolutional Network (GCN) with a Gated Recurrent Unit (GRU) layer to capture sequential patterns and reconstruct final results using a variational autoencoder. Similarly, Chen et al. [3] present a new framework for multivariate time-series anomaly detection that construct graphs to represent IoT sensor time series relationships using a transformer-based architecture and the Gumbel-softmax sampling approach to learn graph structure. They also use graph convolution to describe the

anomaly information flow between network nodes.

LSTM based approaches utilize LSTM units to extract long-term dependencies between data from previous timesteps. These networks use extracted temporal features for reconstruction, determining anomalies through input-output loss. Lin et al. [22], for instance, used a Variational Auto-Encoder (VAE) model to extract robust local features over short windows and then a LSTM module over the VAE results to capture temporal patterns. This allowed them to identify anomalies over multiple time scales. Along similar lines, Niu et al. [28] used LSTM units for encoder, decoder, generator, and discriminator in a VAE-GAN network. Their model learns the distribution of normal data across timesteps using the LSTM cells, and is hence able to identify non-conforming instances in the testing data as anomalous.

2.2.3 Metaheuristic-based Deep Methods

Anomaly Detection within complex data can be a challenging task, especially in high-dimensional and noisy environments. Metaheuristics, such as neural architecture search and evolutionary algorithms, have been shown to effectively explore large solution spaces, adapt to diverse data distributions, and enhance detection accuracy. In this section, we explore how these methods have been used to improve anomaly detection, focusing on their ability to find novel solutions through intelligent search and adaptation.

Neural Architecture Search (NAS) is a machine learning model, that is used to automate the search for an optimal neural network architecture. NAS is a meta-learning process, where a controller explores a wide parameter search space of potential architectures by iteratively sampling and training child neural networks while evaluating their performance. The goal of the method is to identify those architectures that achieve a balance between accuracy, complexity, and computational efficiency. Li et al. [21] propose the AutoOD framework, that employs NAS over a comprehensive search space that encompasses both global neural architecture hyperparameters (such as anomaly definitions and distance metrics) and layer-specific settings (such as kernel sizes and activation functions). The search process uses an LSTM-based controller in a reinforcement learning (RL) setup to sample, evaluate, and iteratively refine child models based on rewards derived from their performance in anomaly detection. Dissem et al. [10] use NAS to find optimal autoencoder models for time-series anomaly detection. Their framework employs an RL agent to iteratively construct and evaluate candidate autoencoder architectures by refining its decisions based on a reward signal that represents anomaly detection performance. Ellendula et al. [11], meanwhile, also explore the use of NAS to

build autoencoder designs as a multi-objective optimization task - minimizing a weighted cost function of the number of parameters, latent layer nodes, and Mean Absolute Percentage Error (MAPE). Their model uses Differential Evolution (DE) to evolve candidate architectures, leveraging techniques like mutation, crossover, and elitism for improved search efficiency. The generated autoencoders were shown to be able to balance reconstruction accuracy, model complexity, and resource efficiency.

Hashim et al. [15] developed an Adaptive EVolutionary AutoEncoder (AE-VAE) method for anomaly detection in time series that uses genetic programming (GP) operations - such as selection, evaluation, crossover, mutation, and recombination - to iteratively optimize the structure and weights of autoencoder networks across generations. Their GP operations use a fitness function that incorporates model performance metrics - rewarding true positive and true negative classifications and penalizing false positive and false negative classifications. This strategy effectively drives the evolution of networks capable of producing optimal reconstructions.

Chapter 3

Background

This section would provide a sufficient overview of the key technical concepts used to develop the solution in the project. This would be followed by a comprehensive specification of proposed solution and its architecture.

3.1 Evolutionary eXploration of Augmenting Memory Models

The EXAMM algorithm, developed by Ororbia, El Said, and Desell [31] evolves RNNs for forecasting using a wide variety of memory structures, such as Δ -RNN, GRU, LSTM, MGU, and UGRNN cells. It uses multiple population islands instead of a single steady-state population, resulting in an improved performance compared to its single population contemporaries. The algorithm uses a main process to manage island populations and generate new RNNs in a round-robin manner, while several worker processes train RNNs asynchronously - ensuring efficient use of available processors. If the trained RNNs perform better than the worst performing RNN in the island, they replace the worst RNN - emulating natural selection.

The algorithm performs mutation operations on nodes, edges, and recurrent edges, including disabling/enabling nodes and edges, adding new nodes and edges, splitting/merging nodes, and splitting edges. Added edges are probabilistically selected to be either recurrent or nonrecurrent connections. Moreover, the algorithm also performs crossover and cloning operations that can combine two parent RNNs or duplicate parent genomes, respectively. For the first generation, the genomes are created as minimal feed-forward neural networks without hidden layers, with input nodes fully connected to the output nodes. These initial genomes use Xavier or Kaiming Weight Initialization.

Biases and weights for new nodes and edges, that are generated from mutation operations, are initialized from a normal distribution based on the parents' average and variance. However, RNNs generated through crossover operations follow a Lamarckian strategy, that is, if an edge or node exists in both parents, the child weights are generated by recombining the parents' weights -

$$w_c = r(w_p2 - w_p1) + w_p1$$

where -0.5 $\leq$ r $\leq$ 1.5 (chosen at random), w_p1 is the weight of the more fit parent, and w_p2 is from the less fit parent [24].

3.2 Long Short Term Memory Neural Networks

LSTM neural networks are a novel recurrent network architecture designed by Hochreiter and Schmidhuber [18] to retain long-term dependencies in sequential data. This type of RNN was originally designed to avoid the problem of exploding and vanishing gradients. Apart from input from the data set (x_t), the LSTM cells also inherit long-term (C_{t-1}) and short-term memory (h_{t-1}) from previous LSTM timesteps. The long term and short term memory values get updated within each LSTM instance based on the input data value for the current timestamp. LSTM modules operate on a gated architecture, consisting of three control gates - the forget gate, the input gate, and the output gate.

3.2.1 Forget Gate

The forget gate (f_t) decides what percentage of the inherited long-term memory (i.e. Cell State C_{t-1}) is remembered using a sigmoid function.

$$f_t = \sigma_1(W_f[h_{t-1}, x_t] + b_f$$

where W_f and b_f are the weight and bias for the forget gate, respectively, and h_{t-1} is the output (retained short term memory) from the state $t-1$.

3.2.2 Input Gate

The input gate (i_t) decides what percentage of the candidate value vector $\tilde{C}_t$ gets added to long term memory for the next time step using a sigmoid function.

$$i_t = \sigma_2(W_i[h_{t-1}, x_t] + b_i$$

where W_i and b_i are the weight and bias for the input gate, respectively. The vector for the candidate value $\tilde{C}_t$ gets generated by a tanh layer at the same time to update in the new cell state (long term memory) C_t, in which

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

and

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

where W_c and b_c are the weight and bias for the memory cell state respectively.

3.2.3 Output Gate

The output gate (o_t) decides what percentage of the updated long-term memory (C_t) is used to determine the new short-term memory for the next state (i.e. the LSTM output h_t) using a sigmoid function.

$$o_t = \sigma_3(W_o[h_{t-1}, x_t] + b_o$$

where W_o and b_o are the weight and bias for the input gate, respectively. The output for the current instance of the LSTM cell h_t also gets generated at the same time, as

$$h_t = o_t * tanh(C_t)$$

LSTM cells read a sequence of input vectors, with each vector representing a multi-dimensional reading at a given time instance. The cell state runs through the chain, maintaining sequential information and allowing for the persistence of knowledge across time steps.

3.3 Autoencoder Neural Networks

An autoencoder is a type of unsupervised neural network that compresses some given data into a concise, latent space representation, and then reconstructs it back into the original input space. The process of compression, or encoding, is done by a network known as an encoder, while the reconstruction, or decoding, is done by a decoder network. The aim with using an autoencoder architecture is not to simply replicate the input data but to capture the most essential features within a data set. By forcing the latent space to be smaller, autoencoders emphasize the most salient data features, effectively creating a condensed representation that can aid in tasks like feature extraction and dimensionality reduction [17] [41].

The model is trained by minimizing the reconstruction error (L), calculated as

$$L = \frac{1}{2} \sum_{x} \|x - \hat{x}\|^2$$

where $x \in \mathbb{R}^m$ is the input data and $\hat{x} \in \mathbb{R}^m$ is the reconstructed version of that input. The encoder $e(x)$ compresses x into an encoded representation $z \in \mathbb{R}^n$, which the decoder $d(z)$ uses to produce output $\hat{x}$. Both x and $\hat{x}$ have the same dimension, however the dimension of the latent space representation must be smaller for the autoencoder to work (i.e. $n < m$) [27].

3.3.1 LSTM Autoencoder

An LSTM autoencoder utilizes LSTM cells in the structure of both its encoder and decoder networks. These LSTM nodes are useful in handling temporal data - especially with the ability of LSTMs to learn long-term dependencies in sequential data, they become particularly valuable for time-series forecasting and anomaly detection.

When an LSTM autoencoder is trained on normal data, it can struggle to accurately reconstruct anomalous sequences that it has not encountered before - resulting in notably higher reconstruction errors. This discrepancy can be used effectively to identify such anomalous sequences. Research by Malhotra et al. [25] demonstrates this method well - that an encoder-decoder model trained on only normal sequences successfully identifies anomalies based on reconstruction error.

3.4 One-Class Support Vector Machines

Support Vector Machines (SVMs) are a popular machine learning algorithm used to find the optimal hyperplane in an N-dimensional space that can separate data points into different classes in the feature space. One-Class Support Vector Machines (OCSVMs) are a special variant of the SVM methodology that, as the name suggests, optimizes to group the majority of the data as a singular "normal" class. This allows for the convenient unsupervised identification of deviations and anomalies in the dataset, since these points become outliers to the calculated hyperplane.

The data set is first mapped to a high-dimensional Hilbert feature space $\mathcal{F}$ using a nonlinear transformation $\Phi(\cdot)$. In this space, separating the data is

equivalent to solving the following quadratic problem [38] [27]:

$$min(\frac{1}{2}\|w\|^2 + \frac{1}{\nu N}\sum_{i=1}^{N}\xi_i - \rho)$$

subject to

$$(w \cdot \Phi(y_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0 \quad \forall\, i = 1 \ldots N$$

where:

- w is the perpendicular vector to the hyperplane in $\mathcal{F}$.
- ρ is the distance to the origin.
- ξ_i are slack variables to handle possible outliers in the training data set.
- $\nu \in (0, 1]$ controls the tradeoff between the number of positive examples and the margin, representing an upper bound on the fraction of margin errors and support vectors.

The radial basis functions (RBF, or Gaussian) are the most commonly used kernel function with OCSVM, defined as:

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

where $\sigma > 0$ is the kernel width parameter. The distance between two mapped samples in the feature space, y_i and y_j then becomes

$$\|\phi(y_i) - \phi(y_j)\|^2 = k(y_i, y_i) + k(y_j, y_j) - 2k(y_i, y_j)$$

$$= 2\left[1 - \exp\left(-\frac{\|y_i - y_j\|^2}{2\sigma^2}\right)\right]$$

Hence the relationship between $\|\phi(y_i) - \phi(y_j)\|^2$ and $|y_i - y_j\|$ is positively proportional with the Gaussian kernel, preserving the ranking order of the distances between samples in the input and feature spaces.

Chapter 4

EXA-AE: Evolutionary eXploration of Augmenting AutoEncoders

This chapter presents an extension to the EXAMM algorithm implemented within the EXA-STAR framework called Evolutionary eXploration of Augmenting AutoEncoders (EXA-AE), which allows for separately co-evolving encoder and decoder components within a recurrent neural network. This extension enables the evolution of recurrent autoencoder architectures, where the outputs of these evolved autoencoders are fed into an OCSVM threshold for anomaly detection. The proposed method is as follows.

Similar to the solutions proposed by Nguyen et al. [27] and Lin et al. [22], an autoencoder is employed to compress input data into a small embedding space through its encoder component, and then reconstruct it using the decoder. This compression-decoding process enables the autoencoder to capture the most important features of the data. Within the evolved network, feed-forward components are responsible for capturing short-term (or single time-step) features, while LSTM modules and recurrent connections allow the model to capture dependencies over larger time scales, helping to identify long-term trends.

Supposing a multivariate time series row at a particular timestamp is $x_t = \{x_t^{(1)}, x_t^{(2)}, \ldots, x_t^{(k)}\}$, where k is the number of variables; the entire sequence of observed data would be $\{x_1, x_2, \ldots, x_N\}$, where N is the number of recorded samples. Based on this input sequence x_t for a particular time step, the LSTM autoencoder reconstructs $\hat{x}_t = \{\hat{x}_t^{(1)}, \hat{x}_t^{(2)}, \ldots, \hat{x}_t^{(k)}\}$ using only the most essential features of the data.

4.1 EXA-STAR Implementation

EXA-STAR is a limited implementation of the EXAMM algorithm using PyTorch [9] that performs the same genetic operations and neuro-evolutionary processes of genome replacement based on fitness. Unlike EXAMM, however, it does yet not support advanced memory models such as Δ-RNN, GRU, LSTM, MGU, or UGRNN. This limitation necessitated the implementation of a specialized LSTM Node, described below. Additionally, EXA-STAR utilizes single steady-state models instead of the multiple population islands available in EXAMM that enable asynchronous RNN training, ensuring efficient processor utilization.

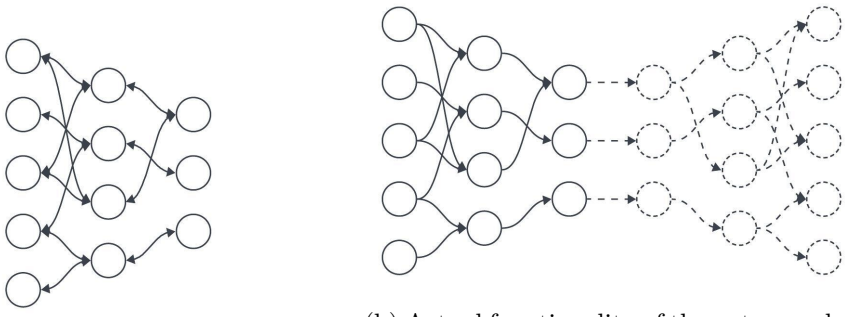
4.1.1 LSTM Node Integration

In EXA-STAR, a specialized LSTM node was implemented as an individual LSTM cell capable of supporting multiple input and output edges. During an *add-node* mutation operation, the LSTM node has same probability of being added as a regular node. By combining these nodes with regular recurrent nodes, EXA-STAR ensures a balanced and robust mechanism for handling both short-term and long-term trends. This integration allows the network to better model and predict multivariate time series data, accommodating intricate temporal relationships.

4.1.2 Autoencoder Architecture Constraints

The networks produced by both EXAMM and EXA-STAR are unstructured recurrent neural network (RNN) architectures. As such, specific objects and rules must be developed to guide mutation and genetic operations in order to enable the evolution of autoencoders. These autoencoders must adhere to certain constraints, including a fixed-size embedding layer and the restriction that no connections may span across the embedding layer. These constraints are necessary to ensure that the evolved networks maintain a functional autoencoder structure, allowing them to effectively encode and decode input data while preserving the critical features of the data.

To address diverse design requirements, two distinct autoencoders were developed: a bidirectional symmetric autoencoder and a regular asymmetric autoencoder.



(a) True shape of the autoencoder

(b) Actual functionality of the autoencoder as a symmetrical and bidirectional solution

Figure 4.1: Bidirectional Symmetric Autoencoder

Bidirectional Symmetric Autoencoder

A novel framework implementation where each node functions simultaneously as an encoder and a decoder (see Figure 4.1). This symmetry facilitates shared representations that enable accurate input reconstruction. The architecture initializes an input layer and a lower-dimensional encoding layer, connected by edges, with all other bidirectional nodes and edge connections added through the neuro-evolutionary process. The nodes added during the initialization of the autoencoder genome are excluded from being candidates for the *disable-node*, *split-node*, and *merge-node* mutation operations in order to preserve the architecture of the generated autoencoders. Each Bidirectional Node is equipped with two distinct and independently trained weights — one for the encoder and another for the decoder. During the forward pass, the process begins at the input layer and proceeds until reaching the encoding layer, where it reverses direction to transition into the decoding phase. The same edges used in the encoder operate bidirectionally in reverse for the decoder, ultimately ending back at the input layer, which now serves as the output layer for the decoder. The training process independently optimizes the encoder and decoder weights based on the reconstructions generated by the decoder output.

Regular Asymmetric Autoencoder (EXA-AE)

To potentially enable more efficient or tailored encoding and decoding, a methodology for evolving asymmetrical autoencoder architectures was implemented within EXA-STAR, called Evolutionary eXploration of Augmenting

AutoEncoders (EXA-AE). In this configuration, the encoder and decoder layers are completely independent, separated by the encoding layer at a depth of 0.5. Both halves of the autoencoder evolve in isolation, ensuring that no connections go across the encoding layer. The two halves are hence constrained to communicate solely through the low-dimensional encoding layer, compelling the reconstructions to be generated using only the most essential features of the feature space. The nodes in the input, output and encoding layers are exempt from candidacy for the *disable-node*, *split-node*, and *merge-node* mutation operations, thus ensuring adherence to architectural principles of the generated autoencoders.

All genetic operations within the regular autoencoder framework were constrained to operate only within either the encoder or decoder layers during the execution of a single reproduction method. The specific changes made to each reproduction method are described below:

- Add Node: New nodes are created at a random depth between 0.0 and 1.0. Nodes with depths less than 0.5 are allocated to the encoder - having edge connections at random (both non-recurrent and recurrent) to other nodes in the encoder and the encoding layer, while those with depths greater than 0.5 are assigned to the decoder - connecting at random to other nodes in the decoder and the encoding layer.

- Add Edge/Recurrent Edge: Edges are introduced within a randomly selected range — either in the encoder (depths 0–0.5) or decoder (depths 0.5–1). The new edge created would then connect two nodes at random from within the selected range.

- Merge Node: Two nodes are selected to merge at random from within a randomly selected range — either in the encoder (depths 0–0.5) or decoder (depths 0.5–1).

- Enable/Disable Node/Edge: These operations choose an enabled node/edge at random to disable or split, or a disabled node/edge to enable. The functionality for these operations does not change, since no new nodes or edges are added that can possibly go across the encoding layer.

- Split Node/Edge: These operations choose an enabled node/edge to split. A split node has the same edge connections as its parent node, while a split edge has the same node connections as its parent edge. This ensures that none of the newly created connections break the architectural constraints for autoencoder design, hence no new changes were required.

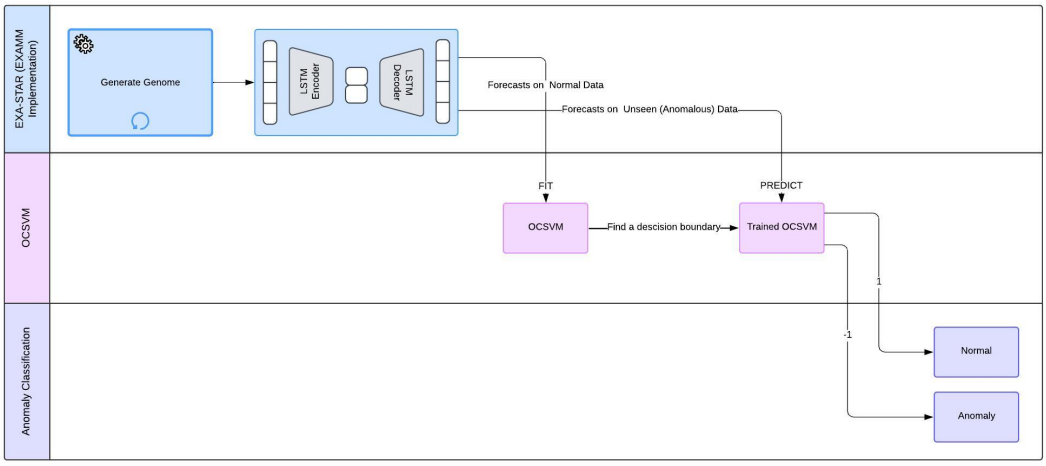


Figure 4.2: Process Flowchart for the Proposed Solution

- Crossover and Cloning: These operations perform either partial or complete duplication of their parent genome architecture, thus naturally inheriting autoencoder design without any changes.

4.1.3 Integration of Evolved Autoencoders and One-Class Support Vector Machines

The EXA-STAR solution is used to evolve a Regular LSTM autoencoder network capable of producing an accurate prediction sequence $\{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N\}$, where N is the number of samples and $\hat{x}_t = \{\hat{x}_t^{(1)}, \hat{x}_t^{(2)}, \ldots, \hat{x}_t^{(k)}\}$, t = 1, 2, ... is the value of the multivariate time series at the time t with k number of variables (same as above). The input sample is normalized to have consistent scaling and minimize the impact of different feature magnitudes on the training process. The network is trained using backpropogation and mini-batch gradient descent, to minimize the loss function defined by the mean square error (MSE):

$$MSE = \frac{1}{N} \sum_{i=m+1}^{N} \left(\hat{x}_i^{(1)} - x_i^{(1)} \right)^2$$

The prediction error vectors can be calculated as $e_i = \hat{X}_i - X_i, i = 1, 2, \ldots, N$. An OCSVM is trained exclusively on the residual values of non-anomalous healthy data. This trained OCSVM then makes outlier predictions

over the residual values of a test set (with anomalies), functionally separating instances of anomalous data (see Figure 4.2).

Chapter 5

Datasets

To better understand and verify the performance of the proposed solution, a diverse set of time series data containing sequences of pre-labeled anomalies was tested. These datasets each included a training set consisting exclusively of non-anomalous normal behavior, allowing the patterns of normalcy to be learned.

5.0.1 Controlled Anomalies Time Series Dataset (CATS)

The CATS dataset [12] consists of commands, external stimuli, and telemetry readings of a simulated complex system with 200 separate injected anomaly sequences. This dataset was compiled by Solenix, an aerospace software engineering and consulting company. The dataset includes 17 variables, such as sensor readings and control signals, spanning 5 million timestamps. It has 1 million nominal observations and 4 million observations that include both nominal and anomalous data, making it ideal for evaluating anomaly detection algorithms. The anomalies are precisely labeled, enabling the evaluation of our final predictions against anomaly labels.

For our purposes, we use the first 30 thousand timestamps from the 1 million nominal observations to train our LSTM autoencoder using EXAMM. We test our solution using 13 thousand timestamps from the 4 million observations that include both nominal and anomalous data, starting at the 5 thousandth timestamp (1 million 5 thousand overall). This specific sequence of anomalous data was chosen because it had an anomaly rate of exactly 50%, which provides a better balance of test data to calculate our metrics of F1-score, precision, accuracy and recall.

5.0.2 Soil Moisture Active Passive (SMAP) satellite and Mars Science Laboratory (MSL) Rover Datasets

SMAP and MSL are two public datasets by NASA that contain expert-labeled telemetry anomaly data from the SMAP satellite and the Curiosity Mars rover, respectively. SMAP is an Earth observation satellite mission that is used to estimate global water and energy fluxes, improve weather and climate forecasts, and improve flood and drought predictions [30]. The Curiosity rover is a robotic vehicle used for in situ exploration on Mars, providing valuable data about the planet's history and potential habitability [13]. The data in these telemetry sets have been anonymized with regard to time and channel IDs, but the first letter indicates the type of channel (P = power, R = radiation, etc.). The model's input data also incorporate one-hot encoded details regarding commands transmitted to or received from particular spacecraft modules within a specified time frame. However, the data does not contain identifiable information regarding the timing or specific nature of these commands.

For the test datasets, essential metadata and anomaly labels for telemetry streams have been provided in an auxiliary file. It includes the anonymized channel IDs, specifies the spacecraft generating each stream, and notes the start and end indices of true anomalies within the streams. Additionally, the file indicates the number of telemetry values present in each stream. This information about anomalies was used to compare and evaluate the anomaly predictions made by our solution.

Our experiments were centered specifically around the power telemetry to keep network training times reasonable. This means that all networks for SMAP were trained using the readings in the P-1, P-2, P-3, P-4, and P-7 training sets, and tested using the anomalous test set - P-4. Similarly, all networks for MSL were trained using only the telemetric values stored in the P-10, P-11, P-14, and P-15 training sets, and tested using the values in the P-11 testing set.

Chapter 6

Results

The EXA-AE solution was compared with three existing, proven methodologies for anomaly detection: an LSTM-based network (with a standard, layerd, fully connected network architecture) with Kernel Quantile Estimation (KQE) approach by Tran et al. [40], and two LSTM Autoencoder solutions — one using KQE and another using an OCSVM (without neuro-evolution), both developed by Nguyen et al. [27]. Each methodology was evaluated on the three datasets described in Chapter 5 to assess whether the LSTM Autoencoder networks evolved using EXA-AE outperform traditional LSTM and LSTM Autoencoder methods without neuroevolution in terms of anomaly detection performance.

This section presents all relevant information about the set of experiments that were performed in this project. This encompasses an account of our experimental design - including the specific hyperparameters used during each run for reproducibility of documented results and a description of the evaluation metrics used. Importantly, here all of the results from our experiments have been documented - this means network performance in the metrics detailed in the previous section, the number of trainable parameters in each network, and, for EXA-AE specifically, an encoder-decoder level analysis of node types, edge types, and trainable parameters.

6.1 Experimental Design

It is very important that the observations made in this study are repeatable, to confirm or challenge any conclusions drawn. For that purpose, this section documents the parametric inputs that would be needed to reproduce the outputs that were observed in our experiments.

CATS Dataset			MSL Dataset			SMAP Dataset		
	OCSVM			OCSVM			OCSVM	
Genome #	ν	γ	Genome #	ν	γ	Genome #	ν	γ
375	0.008	'auto'	407	0.02	'scale'	441	0.81	'scale'
562	0.016	'auto'	427	0.02	'scale'	557	0.91	'scale'
639	0.008	'auto'	673	0.02	'scale'	605	0.95	'scale'
714	0.06	'auto'	694	0.02	'scale'	804	0.84	'scale'
768	0.012	'auto'	772	0.02	'scale'	883	0.295	'scale'
783	0.005	'auto'	839	0.02	'scale'	903	0.473	'scale'
825	0.0075	'auto'	885	0.02	'scale'	920	0.945	'scale'
841	0.002	'auto'	953	0.02	'scale'	937	0.663	'scale'
853	0.003	'auto'	960	0.02	'scale'	972	0.149	'scale'
940	0.006	'auto'	967	0.02	'scale'	990	0.71	'scale'

Table 6.1: OCSVM Parameters Used for EXA-AE Networks

6.1.1 EXA-AE Setup

The EXA-AE solution is used to evolve RNNs with simple neurons and LSTMs for the training sets decribed in Chapter 5. The best-performing EXA-AE LSTM autoencoder, evolved over 1000 generations, was used to make predictions on both the training and testing (anomalous) sets. The number of cells and the connections between them are optimized using the neuro-evolutionary operations described by the EXAMM algorithm. The batch size for gradient descent was set to 256. The genomes were trained using the Adam optimizer - with a learning rate of 0.001, betas=(0.9, 0.999), and the default ϵ value of 1e-8. The input/output size was 10 for the CATS dataset, 25 for the MSL dataset, and 19 for the SMAP dataset. The ideal choice for the gradient descent batch size warrants further investigation — larger values might better capture local features, but could also increase training time. The LSTM nodes, which are used to learn long-term temporal dependencies, determine the optimal weighted lookback on their own. The optimal size for the encoding layer also requires further investigation; however, for the current set of experiments, it was set to the square root of the input layer size.

An OCSVM was fitted to the prediction residuals of the training set. This fitted threshold was then used to make outlier predictions for each of the prediction residuals in the testing set. The hyperparameters used in the OCSVM have been documented in Table 6.1. By defualt, this algorithm uses the radial basis function kernel type, with γ as the kernel coefficient. ν represents an upper bound on the fraction of training errors and a lower bound of the

fraction of support vectors.

$$\gamma =' scale' = 1/(num\_features * X.var())$$

$$\gamma =' auto' = 1/num\_features$$

These values were tuned to maximize anomaly detection accuracy, while minimizing false positive readings.

6.1.2 Benchmark method: LSTM-based networks with KQE

Our solution was benchmarked against an anomaly detection approach developed by Tran et al. [40] that is based on LSTM networks. They use a kernel estimation of the quantile function to compute their thresholds on prediction errors, which are then used to determine anomalous observations.

The LSTM model has an input/output size of 10, 25, and 19 for the CATS, MSL, and SMAP datasets, respectively. It has three hidden layers with sizes 64, 256, and 100 - in that order. The LSTM layers use a time step of 1 and apply a dropout rate of 0.2 after each layer. The model is trained using the MSE loss function and the RMSProp optimizer, with default learning rate, ρ and momentum (that is, 1 - decay rate) values of 0.001, 0.9 and 0.0, respectively. The threshold for anomaly detection, τ, is determined by applying a Gaussian kernel of mean p and standard deviation h. The fixed parameter p is set to 0.56 for CATS test cases, to 0.345 for MSL test cases, and to 0.056 for SMAP test cases. Meanwhile, the value h is optimized through a comprehensive and dynamic grid search process that tests 10 different bandwidth values between 0 and 0.5, using 20-fold cross-validation to select the optimal threshold smoothing parameter.

6.1.3 Benchmark method: LSTM-AutoEncoders

Our solution was also benchmarked against the LSTM-Autoencoder based methods developed by Nguyen et al. [27]. They use their LSTM-Autoencoders to make their reconstructions, demonstrating the use of both an OCSVM and a kernel estimation of the quantile function to compute their thresholds on prediction errors. Their approach was a significant inspiration behind the development of this project, and should provide a very crucial comparison to determine the impact of neuroevolution in anomaly detection.

The autoencoder uses two LSTM layers in its encoder and two LSTM layers in its decoder. The architecture of the decoder layer is the mirror image of the encoder - so the first layer of the encoder and the second layer of the decoder

Dataset	*num_cells*	Learning Rate
CATS Dataset	96	0.01
MSL Dataset	64	0.01
SMAP Dataset	16	0.0001

Table 6.2: Best Parameters across Different Datasets for Benchmark AE Networks

both have $4 * num\_cells$ units, and the second layer of the encoder along with the first layer of the decoder both have num_cells units. All layers use the Rectified Linear Unit (ReLU) activation function, and the model is trained using the Adam optimizer and MSE loss. The learning rate used by the Adam optimizer and num_cells parameters were determined for each dataset through extensive grid search. These values used are documented in Table 6.2. The optimizer used the default values of 0.9, 0.999 and 1e-07 for the β_1, β_2, and ϵ hyperparameters, respectively. The networks use a batch size of 256, over 10 epochs. The input/output sizes are again 10, 25, and 19 for the CATS, MSL, and SMAP datasets, respectively.

With OCSVM Threshold

Table 6.3 summarizes the hyperparameter values that were used to obtain OCSVM thresholds for the reconstructions made by the above-described autoencoder architecture. An explanation for what these values mean is given in Section 6.1.1.

With KQE Threshold

The threshold for anomaly detection, τ, is calculated using numerical integration with the bandwidth h - optimized through a comprehensive and dynamic grid search process, and a fixed parameter p that is set to 0.56 for CATS and SMAP test cases, and to 0.53 for MSL test cases. The grid search tests 10 different bandwidth values between 0 and 0.5, using 20-fold cross-validation to select the optimal threshold smoothing parameter.

CATS Dataset			**MSL Dataset**			**SMAP Dataset**		
	OCSVM			OCSVM			OCSVM	
Run #	ν	γ	Genome #	ν	γ	Genome #	ν	γ
1	0.000055	'auto'	1	0.055	1.5	1	0.055	1.5
2	0.00001	'auto'	2	0.055	1.5	2	0.055	1.5
3	0.00001	'auto'	3	0.055	1.5	3	0.055	1.5
4	0.00001	'auto'	4	0.055	1.5	4	0.055	1.5
5	0.00001	'auto'	5	0.055	1.5	5	0.055	1.5
6	0.000055	'auto'	6	0.055	1.5	6	0.055	1.5
7	0.000055	'auto'	7	0.055	1.5	7	0.015	1.5
8	0.000055	'auto'	8	0.055	1.5	8	0.055	1.5
9	0.000055	'auto'	9	0.055	1.5	9	0.055	1.5
10	0.000055	'auto'	10	0.055	1.5	10	0.01	1.5

Table 6.3: OCSVM Parameters Used for Benchmark AE Networks

6.2 Evaluation Metrics

The proposed solution was compared against its counterparts using the accuracy, precision, recall, and F-score evaluators:

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

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

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

$$\text{F-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where TP (True Positive) is the number of correctly diagnosed anomalies, TN (True Negative) is the number of correctly identified normal events, FP (False Positive) is the number of normal events that get incorrectly flagged as anomalies, and FN (False Negative) is the number of anomalies that miss detection.

Accuracy indicates how often the models correctly identify both normal and anomalous events. Precision specifically measures the models' accuracy in labeling anomalies, while recall assesses their ability to identify all anomalies within a dataset - together they provide a sense for the quality and the quantity of detected anomalies. Lastly, the F-score provides a balance between precision

Accuracy

Dataset	CATS			MSL			SMAP		
Method	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
LSTM w/ KQE [40]	0.570231	0.572746	0.574923	0.801980	0.880283	0.902970	**0.937591**	0.938364	0.945319
LSTM-AE w/ KQE [27]	0.563154	0.572592	0.576692	0.414144	0.745516	0.854597	0.935404	0.935404	0.935404
LSTM-AE w/ OCSVM [27]	0.562923	0.575231	0.579231	0.861669	0.890212	0.905799	**0.937591**	0.938029	0.941966
EXA-AE	**0.578077**	**0.581485**	**0.598538**	**0.918529**	**0.923734**	**0.928713**	**0.937591**	**0.953208**	**0.963692**

Precision

Dataset	CATS			MSL			SMAP		
Method	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
LSTM w/ KQE [40]	0.538051	0.539529	0.540815	0.945312	0.959786	0.976251	0.937454	0.938186	0.944772
LSTM-AE w/ KQE [27]	0.535064	0.539583	0.541887	**0.959338**	**0.973696**	**1.000000**	0.935404	0.935404	0.935404
LSTM-AE w/ OCSVM [27]	0.535071	0.541281	0.543482	0.948858	0.959573	0.973136	0.937454	0.937867	0.941582
EXA-AE	**0.542809**	**0.544951**	**0.556278**	0.938657	0.944645	0.952721	**0.939252**	**0.956983**	**0.967830**

Recall

Dataset	CATS			MSL			SMAP		
Method	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
LSTM w/ KQE [40]	**0.992615**	**0.992938**	**0.993231**	0.807983	0.910674	0.951315	**1.000000**	**1.000000**	**1.000000**
LSTM-AE w/ KQE [27]	0.963692	0.989431	0.992923	0.373753	0.750045	0.876323	**1.000000**	**1.000000**	**1.000000**
LSTM-AE w/ OCSVM [27]	0.960000	0.986200	0.991538	0.876323	0.921651	0.947989	**1.000000**	**1.000000**	**1.000000**
EXA-AE	0.974000	0.988769	**0.993231**	**0.968854**	**0.975688**	**0.980950**	0.988932	0.994778	0.998129

F1 score

Dataset	CATS			MSL			SMAP		
Method	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
LSTM w/ KQE [40]	0.697951	0.699158	0.700195	0.884183	0.933864	0.948304	**0.967718**	0.968106	0.971602
LSTM-AE w/ KQE [27]	0.688087	0.698326	0.700940	0.544134	0.835819	0.918205	0.966624	0.966624	0.966624
LSTM-AE w/ OCSVM [27]	0.687149	0.698938	0.701810	0.884183	0.933864	0.948304	**0.967718**	0.967937	0.969912
EXA-AE	**0.700783**	**0.702622**	**0.708126**	**0.957156**	**0.959899**	**0.962533**	0.967649	**0.975490**	**0.980854**

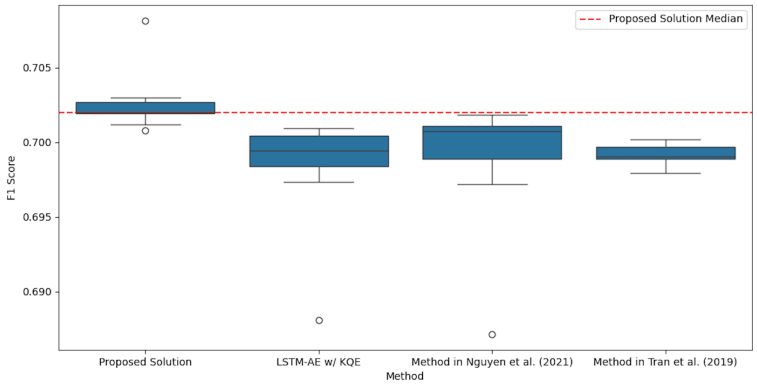
Table 6.4: Results for Different Datasets and Anomaly Detection Methods

and recall, offering insight into the models' ability to effectively manage the trade-off between detecting anomalies and maintaining accuracy.

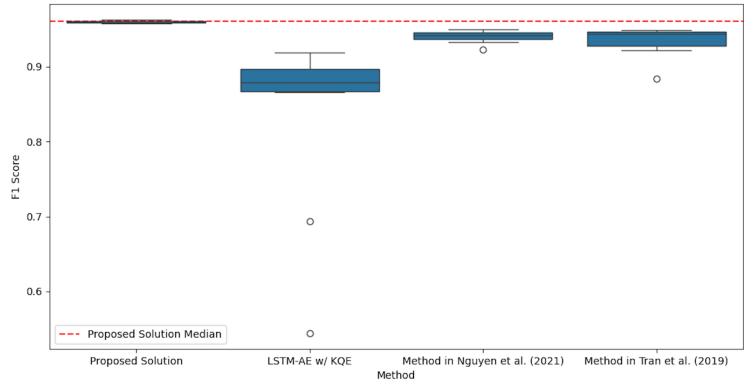
The number of trainable parameters in a neural network refers to the total count of weights and biases that the model learns during the training process. This number was compared for each of the comparative benchmark methods with the range of trainable parameters in each of the networks generated through our EXA-STAR solution. Specifically, the number of trainable parameters in the EXA-AE LSTM Autoencoder that performs the best anomaly detection is matched up against the benchmark methods. The number of trainable parameters helps to assess a model's capacity, efficiency, and potential generalization ability, with a positive correlation to a model's training time, computational resource requirement, and risk for overfitting.

6.3 Anomaly Detection Performance

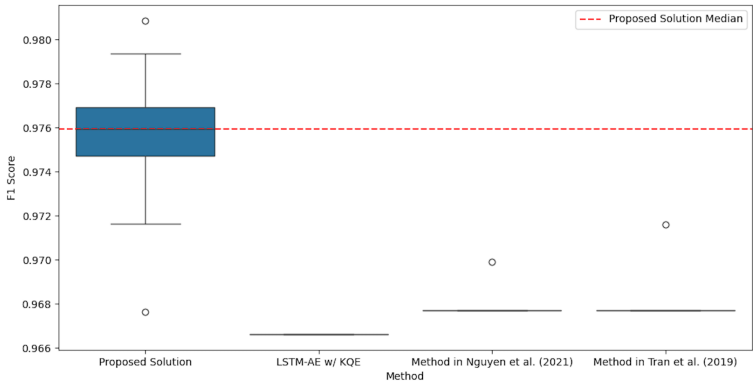
Table 6.4 shows the aggregated results (minimum, average, and maximum) from 10 repetitions of each method across all datasets tested. These results reflect the performance of each method according to the evaluation metrics



(a) CATS Dataset



(b) MSL Dataset



(c) SMAP Dataset

Figure 6.1: F1-Score Comparison for Different Datasets and Anomaly Detection Methods

Method	CATS Dataset	MSL Dataset	SMAP Dataset
LSTM w/ KQE [40]	491714	497069	494927
LSTM-AE w/ KQE [27]	1608202	739097	50771
LSTM-AE w/ OCSVM [27]	1608202	739097	50771
EXA-AE (Best Genome)	903	580	715

Table 6.5: Number of Trainable Parameters across Different Datasets and Anomaly Detection Methods

described in Section 6.2. Note that a higher value is better and desired for each evaluation metric. The best-performing method scores have been highlighted in **bold**. Figure 6.1 provides a further representation of the F1 score results with box-and-whiskers plots showing the mean, range and outliers of each methods on the dataset, with a vertical dashed red line indicating the median score for our proposed solution.

Our proposed solution outperformed others in terms of average and best-case accuracy and composite F1 score across all tested datasets. It also showed better worst-case accuracy and composite F1 score for two out of three datasets. Additionally, the solution achieved superior worst-case, average, and best-case precision for two datasets, and better worst-case, average, and best-case recall for one dataset. Precision and Recall tend to have an inverse relationship, since stricter thresholds favor precision at the cost of false negatives, which adversely affects recall, and vice-versa. Hence, the composite F1 score, that combines these two readings, gives a balanced evaluation of method performance. Our proposed solution's superior F1 score performance confirms its effectiveness as a robust solution for anomaly detection.

6.3.1 Trainable Parameters

Table 6.5 displays the number of trainable parameters in each solution network for each dataset used in testing. Trainable parameters refer to the weights and biases in the nodes and edges of the model that are updated during training. Since each genome network generated by EXA-AE has a unique architecture, the number of trainable parameters varies across genomes. For consistency, the genome with the best composite F1 score was selected to represent the number of trainable parameters for the proposed solution.

While a larger number of trainable parameters can help the model capture more complex patterns, it may also increase the risk of overfitting and lead to higher training costs in terms of both time and resources. The genomes

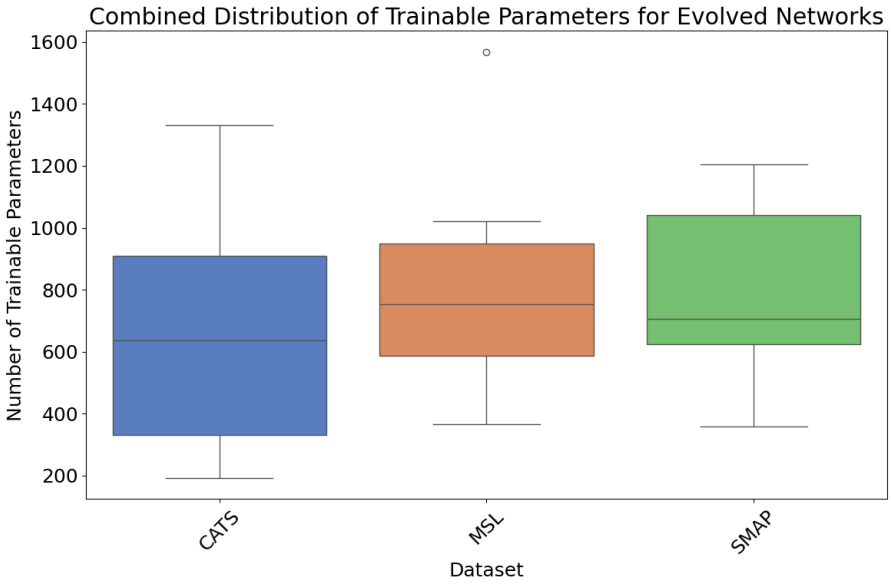


Figure 6.2: Distribution of Total Trainable Parameters in EXA-AE Networks

generated by our EXA-AE solution had significantly fewer trainable parameters than each of the other benchmarking methods tested against. It is worth noting that this difference spanned several orders of magnitude. This shows that the networks generated through EXA-AE were significantly more efficient in training and less susceptible to over-fitting allowing for better detection of anomalies.

These results indicate that a potential reason for the improved performance of EXA-AE even with significantly fewer trainable parameters is due to the rigid architectures of the LSTM-based benchmarks. These benchmarks use stacked LSTM layers, where each subsequent layer is fully connected with a similar number of units as the previous, resulting in N^2 trainable parameters for edge weights for each layer, that is, a quadratic growth of parameters as the network depth increases. Additionally, recurrent networks, especially LSTM-based networks, are notoriously challenging to train [33], and the larger number of parameters can lead to both overfitting and difficulties in learning effective encoding and decoding capabilities.

In contrast, with the singular LSTM cells used in EXA-AE (input and hidden sizes' both set to 1), there is no hidden size or layers to feed parameters, making their impact on the number of trainable parameters constant, instead

of quadratic. These LSTM cells also have the same probability of being added to the network as a simple neuron during each *add-node* mutation. With EXA-AE producing smaller networks in general, this means that these evolved networks would have far fewer LSTM cells than their benchmark counterparts, again resulting in significantly lower trainable parameters. Furthermore, the networks generated by EXA-AE begin with the minimal possible structure for an autoencoder - just the input layer fully connected to the embedding layer, and the embedding layer fully connected to the output layer. As EXA-AE adds nodes and edges through mutation, there is strong evolutionary pressure to only add components that aid in the encoding and decoding, preventing overfitting and enabling a more efficient and easier training process. These characteristics demonstrate an important utility of neuro-evolution in EXA-AE, as it can be used to generate minimal and efficient memory-model-based networks (LSTM autoencoders for anomaly detection, to be specific), which otherwise tend to be very computationally expensive to train with traditional LSTM-based architectures.

Table 6.6 shows a comparison between the reconstruction loss values (MSE) for the different methods tested across the three datasets. The networks produced by EXA-AE had the highest average MSE for the CATS and MSL datasets and the second-highest average MSE in the SMAP dataset (both LSTM-AE benchmarks are counted as one since they share networks, differing only in their thresholding approaches). Perhaps most interesting is the finding that the EXA-AE methodology had significantly higher MSE for the CATS dataset, yet still performed the best in anomaly detection. Given this, and the fact that EXA-AE performed better in terms of MSE on some datasets, the results suggest that EXA-AE's superior performance in anomaly detection was not due to other methods overfitting. This potentially indicates that the evolutionary process of adding nodes and edges allows the EXA-AE networks to select architectures that capture the most important features in the data, which in turn enables the best anomaly detection.

6.4 Exploring Network Architecture

Table 6.7 provides details on the types of edges (recurrent or non-recurrent) and nodes (LSTM or simple neurons) generated during the 10 repetitions of EXA-AE. The table presents this information in terms of the minimum, average, and maximum counts for each type. This data is further broken down into encoder, decoder, and total network specifications. Table 6.7 also shows the correlation coefficients between the counts of each type of edge and node in

CATS Dataset			
	Mean Squared Error		
Method	Min	Avg	Max
LSTM w/ KQE [40]	0.026400	0.026970	0.027600
LSTM-AE w/ KQE [27]	0.002900	0.003620	0.004200
LSTM-AE w/ OCSVM [27]	0.002900	0.003620	0.004200
EXA-AE	1.602018	1.715167	1.890198
MSL Dataset			
	Mean Squared Error		
Method	Min	Avg	Max
LSTM w/ KQE [40]	0.001200	0.001310	0.001400
LSTM-AE w/ KQE [27]	0.000279	0.000505	0.000815
LSTM-AE w/ OCSVM [27]	0.000279	0.000505	0.000815
EXA-AE	0.000979	0.008326	0.032144
SMAP Dataset			
	Mean Squared Error		
Method	Min	Avg	Max
LSTM w/ KQE [40]	0.001600	0.001700	0.001800
LSTM-AE w/ KQE [27]	0.033500	0.034730	0.037100
LSTM-AE w/ OCSVM [27]	0.033500	0.034730	0.037100
EXA-AE	0.002854	0.006083	0.009605

Table 6.6: Reconstruction Loss Values' Summary for the Different Methods Tested, Across the Three Datasets

CATS Dataset

	Encoder				Decoder				Total			
Edge Type	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
Recurrent	2	73.3	235	0.513987	6	55.3	176	-0.017385	18	128.6	411	0.322485
Non-recurrent	9	67.7	148	0.359482	16	72.2	164	0.163379	25	139.9	312	0.265232
Total	11	141.0	383	0.471253	22	127.5	340	0.075472	43	268.5	723	0.304419
Node Type	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
LSTM	0	8.5	19	0.392127	1	10.5	19	0.275429	3	19.0	34	0.359997
Simple Neuron	1	7.2	16	-0.125128	0	7.7	19	0.393454	1	14.9	30	0.192010
Total	1	15.7	33	0.186588	4	18.2	37	0.347395	8	33.9	64	0.296057
	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
Trainable Parameters	51	317.0	663	0.453906	91	335.5	668	0.190315	193	652.5	1331	0.348696

MSL Dataset

	Encoder				Decoder				Total			
Edge Type	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
Recurrent	18	104.4	245	-0.094043	2	63.6	202	0.022476	53	168.0	447	-0.043198
Non-recurrent	7	85.3	195	-0.237628	4	62.3	133	-0.086281	42	147.6	294	-0.206887
Total	25	189.7	440	-0.164721	6	125.9	301	-0.024037	95	315.6	741	-0.115360
Node Type	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
LSTM	1	7.8	21	-0.344189	0	6.1	16	0.069076	1	13.9	36	-0.158979
Simple Neuron	1	7.9	15	-0.433128	0	8.6	25	-0.078382	4	16.5	36	-0.228512
Total	2	15.7	36	-0.402968	0	14.7	41	-0.013742	5	30.9	67	-0.173506
	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
Trainable Parameters	166	439.5	901	-0.258966	131	348.5	666	0.024006	366	788.0	1567	-0.138577

SMAP Dataset

	Encoder				Decoder				Total			
Edge Type	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
Recurrent	2	78.5	177	0.545568	28	108.9	239	0.271067	34	187.4	348	0.485935
Non-recurrent	1	75.6	160	0.625757	37	94.5	242	0.325667	38	170.1	348	0.561036
Total	3	154.1	337	0.586572	65	228.7	438	0.360947	72	357.5	646	0.531910
Node Type	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
LSTM	0	6.9	16	0.398135	3	8.2	14	0.371980	6	15.1	25	0.520060
Simple Neuron	1	6.7	20	0.739388	2	9.2	20	0.415836	4	15.9	33	0.678985
Total	1	13.6	29	0.668957	6	17.4	33	0.424959	10	31.0	56	0.663618
	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.	Min	Avg	Max	Corr.
Trainable Parameters	98	359.5	688	0.509348	208	429.6	741	0.329193	358	789.1	1204	0.533296

Table 6.7: Encoder-Decoder Composition Summary for EXA-AE Networks

each genome, and the anomaly detection F1 score for that genome. A positive correlation coefficient indicates that an increase in the corresponding network component is associated with improved anomaly detection performance, while a negative value suggests that a higher count of that component correlates with reduced performance. The magnitude of the correlation coefficient quantifies the significance of this impact.

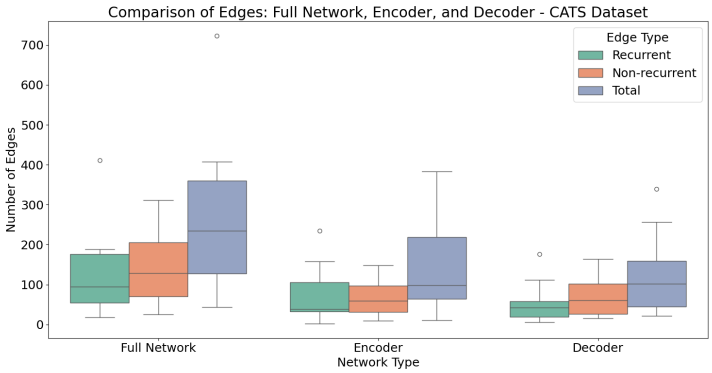
Figures 6.3 and 6.5 use box-and-whisker plots to present this encoder-decoder comparison for the types of nodes and edges, respectively, do show the distribution of their counts across runs. Similarly, Figure 6.2 shows a box-plot distribution for the number of trainable parameters in each EXA-AE evolved genome, for every dataset tested. Lastly, Figure 6.4 displays box and whisker plots of normalized depth value counts for the recurrent edges generated in EXA-AE genomes. During the evolution process, these recurrent depth values are chosen at random between a range of 1 and 9, so any patterns observed in terms of depth preference would have be worth noting. However, there does not seem to be any pattern or correlation worth noting in these plots - all of them seem to have a normalized median around 0.11, which represents a normal distribution of 9 values split at random with equal probability ($0.11 * 9 \approx 1$).

By analyzing patterns in the networks generated by EXA-AE, guidelines for efficient autoencoder design - supported by evolutionary selection - can be derived. Any patterns observed are notable and warrant further investigation if no reasonable explanation can be found. The author's interpretation of these architectural patterns is summarized below.

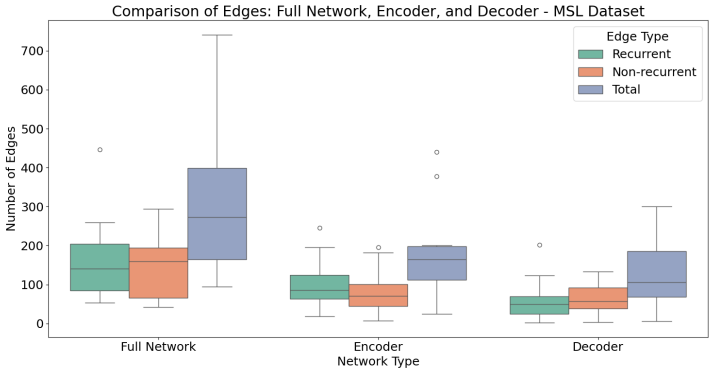
6.4.1 Recurrent Edges vs. Non-recurrent Edges

The encoder, decoder and combined EXA-AE networks constructed for the MSL and SMAP datasets, along with the encoder networks for the CATS dataset, show a preference for recurrent connections over non-recurrent ones, on average. Only the decoder networks for CATS had a higher average non-recurrent edge count than average recurrent edge count, although it is worth mentioning that this difference was significant enough to cause a higher average non-recurrent edge count for combined networks for CATS. The non-recurrent connections in the initial, pre-evolved genome are excluded from these calculations to focus exclusively on the edge decisions made by the evolutionary process.

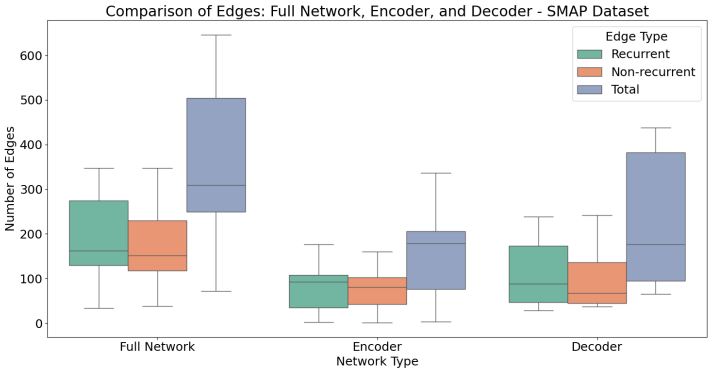
Neither is the sample size here significant enough, nor the results unanimous to draw definitive conclusions about any apparent preference in edge type by the EXAMM evolutionary processes. However, the majority prefer-



(a) CATS Dataset



(b) MSL Dataset



(c) SMAP Dataset

Figure 6.3: Edge Comparison for EXA-AE Networks

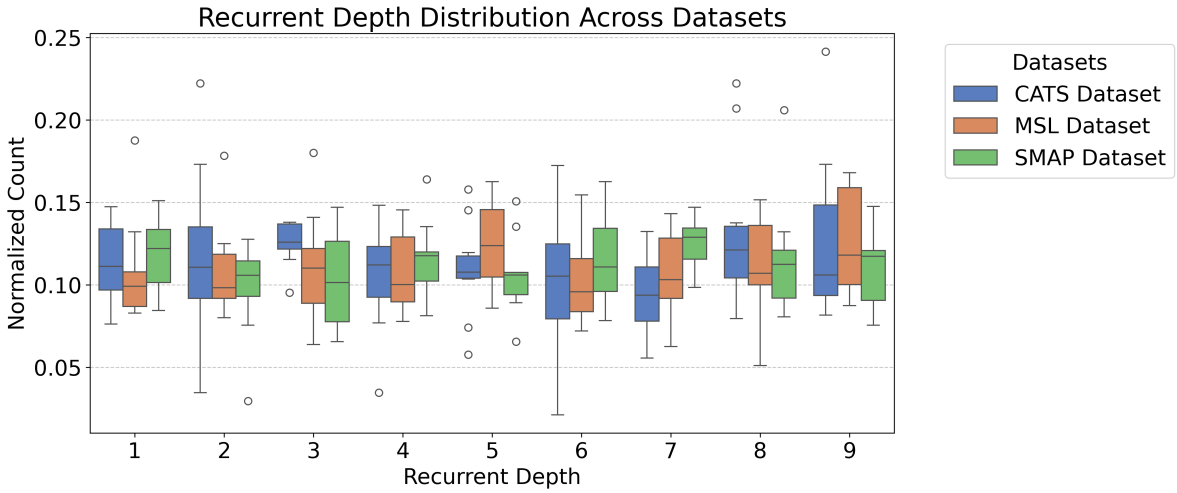


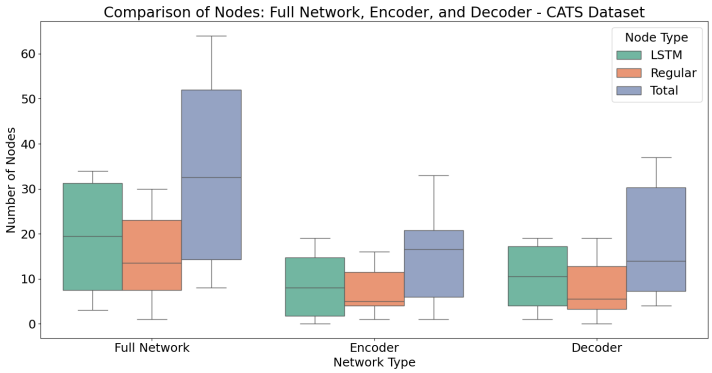
Figure 6.4: Depth Distribution for Recurrent Edges in EXA-AE Networks

ence for recurrent edges could be attributed to their ability to capture long term dependencies effectively and provide a form of temporal memory, which could be useful for reconstruction tasks and forecasting in general. On the other hand, non-recurrent edges may be favored over recurrent ones in some cases, since their network topologies tend to be simpler, more efficient and more direct than their recurrent counterparts, avoiding the instability associated with vanishing/exploding gradients. EXA-AE demonstrates a flexibility to adapt to different edge requirements to maximize performance.

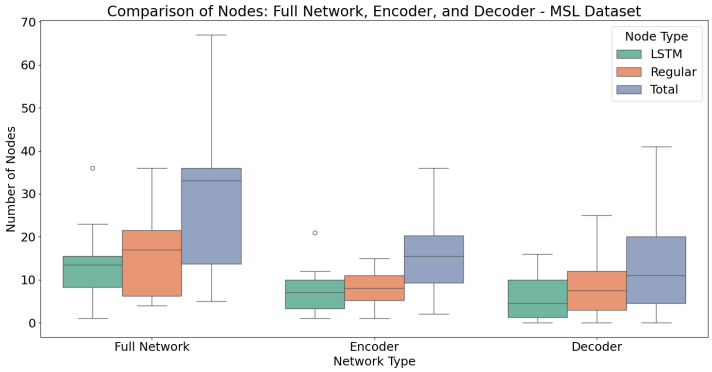
6.4.2 LSTM Nodes vs. Simple Neurons

LSTM nodes were favored over simple neurons for the EXA-AE networks generated for the CATS Dataset, while the opposite was true for the networks generated for the MSL dataset. Finally, the networks generated for the SMAP dataset show a more even overall distribution of the two node types, with the encoder section of the networks slightly favoring LSTM nodes, and the decoder section favoring simple neurons. The two types of nodes had the same probability of evolving into the network, and the results show a very even split between the two. It is worth mentioning that despite the nodes in the input, embedding and output layers being simple neurons, these nodes were excluded from the sum since they were added during initialization, and not through the evolutionary process.

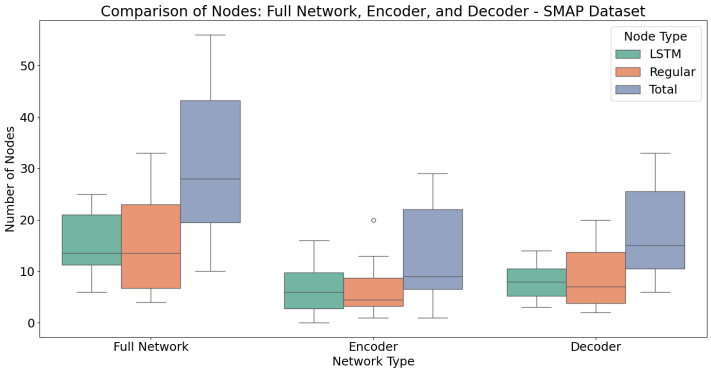
LSTM cells are great at capturing long-term dependencies, while simple



(a) CATS Dataset



(b) MSL Dataset



(c) SMAP Dataset

Figure 6.5: Node Comparison for EXA-AE Networks

neurons tend to be more computationally efficient, more resistant to overfitting/underfitting, and easier to optimize. By striking a balance between the two node types, EXAMM algorithm's evolutionary process is able to balance the strengths of both. It would be interesting to see in future experiments how this balance gets affected by allowing selection of other memory cell neurons in the evolutionary process.

6.4.3 Encoder vs. Decoder

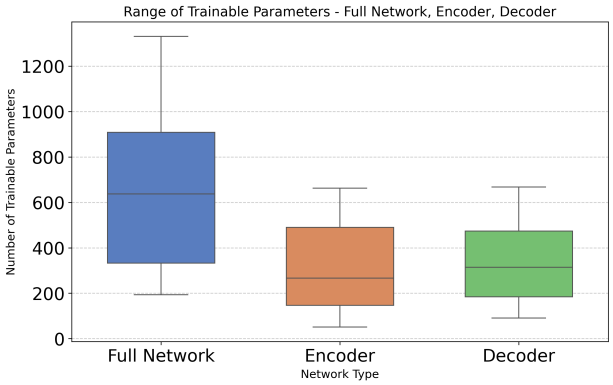
There were, on average, more trainable parameters in the decoder than in the encoder for the CATS and SMAP datasets (see figure 6.6). The opposite was true for the MSL dataset, however, its larger encoder had a significant negative correlation with composite F1 score. This suggests that, in general, autoencoder networks with a larger decoder than encoder were favored in EXAMM's evolutionary process and produced forecasts with better MSE and anomaly detection. This phenomenon makes some intuitive sense as well - since it is the decoder's role to make final reconstructions, it would make sense for it to require increased capacity to better capture complex patterns in the data.

However, the observed phenomenon could be attributed to the unique and specific properties of the datasets themselves that influenced the sizes of encoders and decoders in pursuit of better results. We used only three datasets for our experimentation, which is not a significant enough sample size to make any definitive claims. This observation opens up an interesting avenue for future work which would require additional experiments across a wider range of datasets to determine any evolutionary preference between the encoder and the decoder.

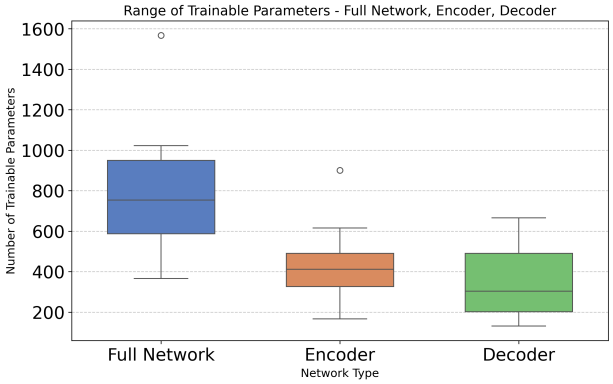
Regardless, this variability highlights a potential strength of EXA-AE: its ability to adaptively evolve encoder-decoder architectures that allocate resources, such as the size of the encoder or decoder, according to the specific needs of the data set.

6.5 EXA-AE Representations

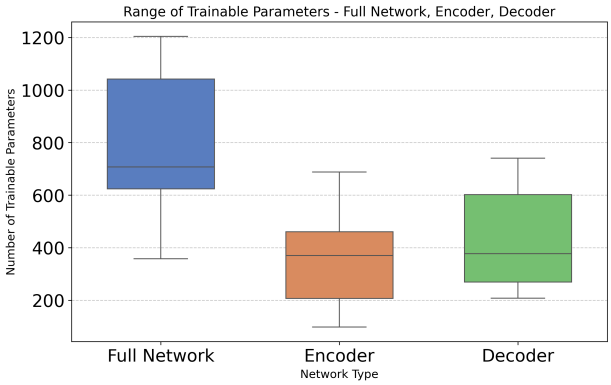
To visualize our EXA-AE networks, we implemented graphviz Digraphs for our genomes, with input nodes in blue border, output nodes in green border, encoding layer nodes in orange border, and for the encoder and decoder between these layers - LSTM cells in red and simple neurons in black. Edge connections have also been color coded based on their weights, with blue edges for positive weights and orange edges for negative weights. The tone of the edge colors



(a) CATS Dataset



(b) MSL Dataset



(c) SMAP Dataset

Figure 6.6: Trainable Parameters Comparison for EXA-AE Networks

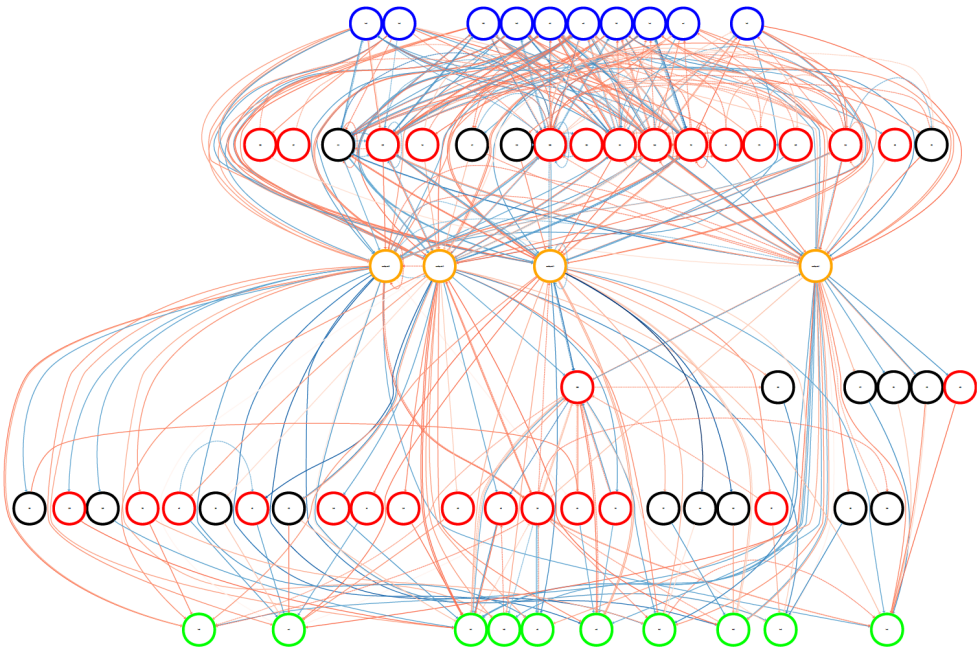


Figure 6.7: Genome 714: Best Performing Genome on CATS

represents the magnitude of the edge weights - higher absolute magnitudes are represented as darker shades, while magnitudes closer to 0 are represented as lighter shades of their edge color. Recurrent edges are represented as dashed lines, in contrast to the solid lines that represent non-recurrent edges. Disabled nodes and edges, as well as enabled edges to disabled nodes have been omitted from the representations.

The encoder and decoder layers are neatly separated by the encoding layer. These graphviz representations validate the networks generated by EXA-AE as robust autoencoder solutions by demonstrating that there are no cross-layer connections between encoder and decoder, except through the encoding layer that forces the data to be reduced to a lower dimensional space.

Figure 6.6 shows the representation of the best performing EXA-AE network from our set of runs on the CATS dataset. In this network, 46 nodes and 359 edges (excluding disabled nodes and edges) were added through the evolutionary processes described in EXAMM. 63.04% of these added nodes were LSTM cells, while 52.65% of the edges were recurrent. This network was slightly encoder heavy with 57.03% of its trainable parameters coming from the encoder section of the network.

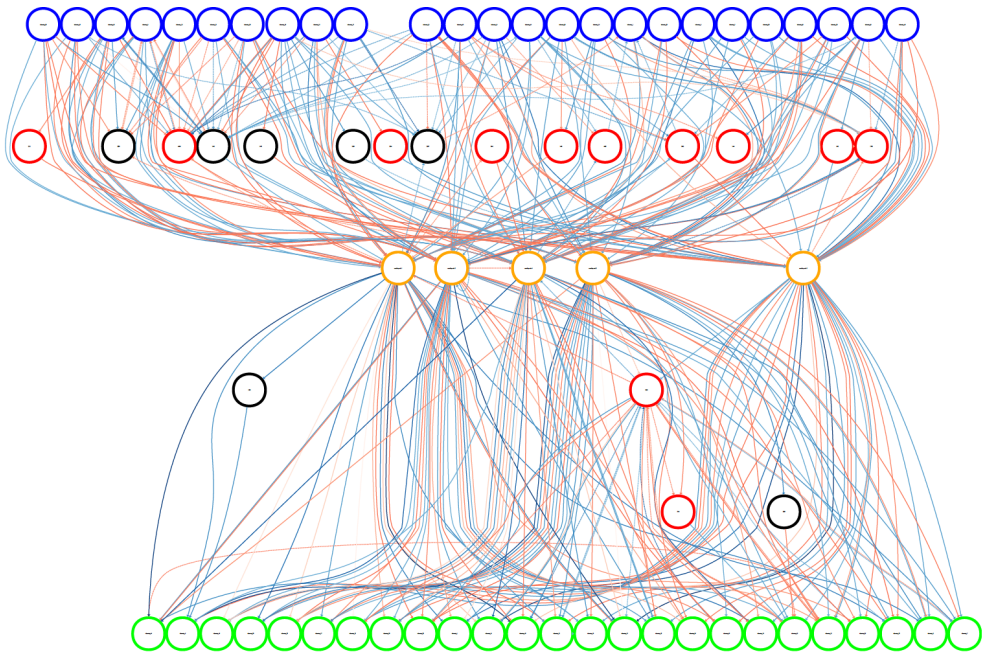


Figure 6.8: Genome 885: Best Performing Genome on MSL

Figure 6.7 represents the best performing EXA-AE genome from our tests in the MSL dataset. For this network, the evolution process generated 19 active nodes and 138 active edges connections. LSTM nodes represent 63.16% of these nodes, while recurrent edges represent 57.97% of observed edges in the network. This network was even more encoder heavy than its predecessor, with two-thirds (66.72%) of its total trainable parameters being used for encoding.

Lastly, figure 6.8 is a representation of the most successful EXA-AE network on the SMAP dataset. This network uses 44 enabled nodes and 301 enabled edges to produce its reconstructions. 31.81% of its nodes were LSTMs, and 49.50% of its edges were recurrent. A majority 63.36% of its trainable parameters were concentrated in the encoder section of the network.

Despite the small sample size here, the best performing networks across the three datasets tested all preferring the encoder in their resource allocation is an interesting observation. It is worth mentioning that the "best-performing" networks shown here as examples had the highest cumulative F1 score within their data set grouping, while the genomes themselves had a fitness value based on MSE. It is possible, therefore, that the MSE-based fitness metric used for evolution might favor certain architectures - like a more even split between

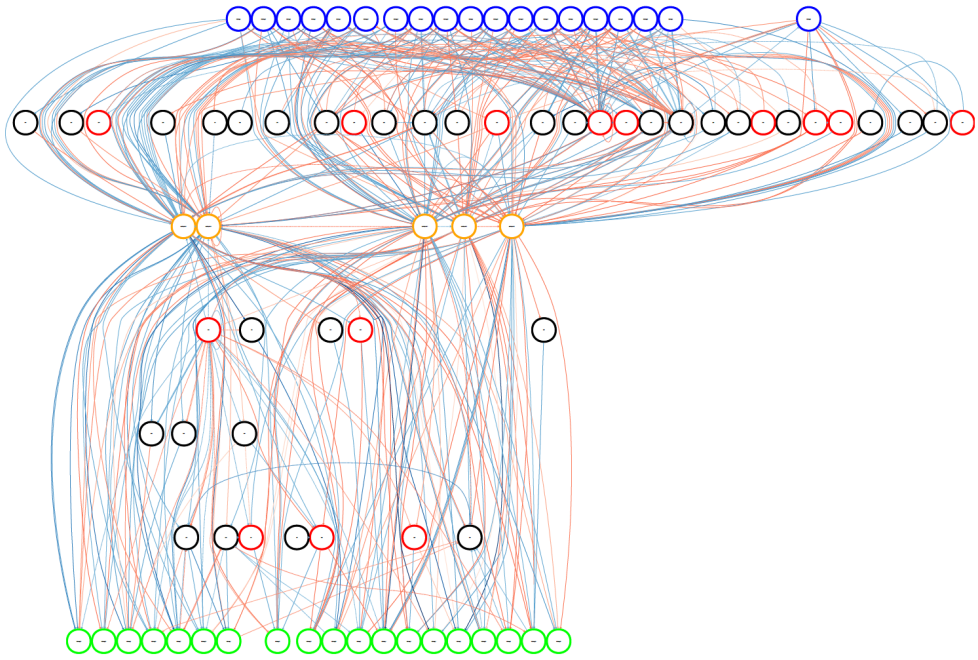


Figure 6.9: Genome 937: Best Performing Genome on SMAP

the encoder and decoder sections or even parameter preference for the decoder networks, which are not necessarily representative of the best architecture for anomaly detection tasks. Further investigation into high-performing, anomaly-detecting network architectures is warranted to explore this idea further.

Chapter 7

Conclusions

This work designed and introduced new capabilities to the EXA-STAR framework with the EXA-AE algorithm, which performs co-evolution of an encoder and decoder within a recurrent neural network, enabling it to evolve autoencoder architectures. These autoencoders were used to create accurate reconstructions of our data from a reduced feature space. A One-Class SVM was trained using these reconstructions to create a robust threshold for anomaly detection in multivariate time series data. Our evaluation for this anomaly detection methodology demonstrates that this approach significantly outperforms several proven LSTM and LSTM-Autoencoder based methods on standard benchmarks, particularly in terms of accuracy and composite F1 score. Furthermore, the networks evolved in this project provide key insights into the neural structures that perform best autoencoded reconstructions, informing future decisions on encooder-decoder connections and the use of LSTM models.

Our experimental results, using the CATS, MSL and SMAP datasets, show that the proposed method can robustly identify anomalies even in diverse and challenging scenarios. EXA-AE produced networks with significantly fewer trainable parameters compared to these traditional LSTM-based methods, which suffer from a quadratic increase in parameters due to their rigid, multilayered, and fully connected architecture. EXA-AE's use of an evolutionary process to progressively add singular LSTM cells and a limited number of edges resulted in more efficient networks that were less computationally expensive to train. In terms of anomaly detection, EXA-AE networks were also able to outperform benchmarks that had better reconstructions (lower MSE), suggesting that the evolutionary process prioritized capturing the most important features of the data. Evolutionary patterns in the generated networks revealed

a slight preference on average for recurrent edges, likely due to their ability to capture long-term dependencies better than non-recurrent edges. Additionally, an even balance between LSTM nodes and simple neurons was found to be ideal to maximize the capture of complex dependencies, while maintaining good generalization. Lastly, EXA-AE tended to favor networks with larger decoders than encoders, especially in the CATS and SMAP datasets, suggesting that a larger decoder may be beneficial for capturing complex data patterns and improving forecasting performance. Overall, our EXA-AE solution demonstrated strong flexibility and adaptability in network architecture to accommodate diverse sets of data.

7.1 Scope for Future Work

This thesis opens up several opportunities for future work, including the exploration of the bidirectional symmetric solution described in Section 4.1.2. This solution has already been implemented in EXA-STAR and some research into its use may offer new insights into how different structural configurations impact the reconstruction and anomaly detection processes, potentially enhancing performance and efficiency. Extending EXA-STAR to incorporate other variants of memory cells used in the EXAMM algorithm, such as GRU, MGU, or UGRNN, would allow us to study how different memory units affect the evolution and performance of autoencoders. In further alignment with the EXAMM framework, the inclusion of population islands in EXA-STAR would facilitate better parallelization and allow us to scale up some of the experiments using EXA-AE with more challenging datasets over more generations, resulting in better reconstructions. This expansion would not only help validate the findings presented in this thesis, but also address unresolved questions, such as whether the evolutionary process tends to favor the encoder or the decoder in terms of trainable parameter allocation. Alternative thresholding methods, such as Kernel Quantile Estimation (KQE), could refine anomaly detection decision boundaries and improve performance compared to OCSVM. Finally, one of the most important areas for future work perhaps lies in the study of the optimal size of the encoding layer - namely, how it affects dimensionality reduction, reconstruction accuracy, and anomaly detection.

Bibliography

[1] Jay F. K. Au Yeung, Zi-kai Wei, Kit Yan Chan, Henry Y. K. Lau, and Ka-Fai Cedric Yiu. Jump detection in financial time series using machine learning algorithms. *Soft Computing*, 24(3):1789–1801, Feb 2020.

[2] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey, 2019.

[3] Zekai Chen, Dingshuo Chen, Xiao Zhang, Zixuan Yuan, and Xiuzhen Cheng. Learning graph structures with transformer for multivariate time-series anomaly detection in iot. *IEEE Internet of Things Journal*, 9(12):9179–9189, 2022.

[4] Zhi Chen, Jiang Duan, Li Kang, and Guoping Qiu. Supervised anomaly detection via conditional generative adversarial network and ensemble active learning, 2021.

[5] Saurav Kumar Dani, Chander Thakur, Naman Nagvanshi, and Gurwinder Singh. Anomaly detection using pca in time series data. In *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, volume 2, pages 1–6, 2024.

[6] Angus Dempster, François Petitjean, and Geoffrey I. Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, Sep 2020.

[7] Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):4027–4035, May 2021.

[8] Palak V. Desai. A survey on big data applications and challenges. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 737–740, 2018.

[9] Travis Desell. Exa-star. `https://github.com/travisdesell/exa-star`, 2024.

[10] Maher Dissem, Manar Amayri, and Nizar Bouguila. Neural architecture search for anomaly detection in time-series data of smart buildings: A reinforcement learning approach for optimal autoencoder design. *IEEE Internet of Things Journal*, 11(10):18059–18073, 2024.

[11] Aditya Sai Ellendula, Aurelien Anand, and Arya Kumar Bhattacharya. Neural architecture search applied to autoencoders trained on noisy industrial data: Single and multi-objective investigations. In *2024 International Conference on Emerging Techniques in Computational Intelligence (ICETCI)*, pages 309–316, 2024.

[12] Patrick Fleith. Controlled anomalies time series (cats) dataset, 2023.

[13] Abigail A. Fraeman. Chapter 1 - resolving martian enigmas, discovering new ones: the case of curiosity and gale crater. In Richard J. Soare, Susan J. Conway, Jean-Pierre Williams, and Dorothy Z. Oehler, editors, *Mars Geological Enigmas*, pages 1–10. Elsevier, 2021.

[14] Todd W Gress, James Denvir, and Joseph I Shapiro. Effect of removing outliers on statistical inference: implications to interpretation of experimental data in medical research. *Marshall J Med*, 4(2), 2018.

[15] Ali Jameel Hashim, M. A. Balafar, Jafar Tanha, and Aryaz Baradarani. Aevae: Adaptive evolutionary autoencoder for anomaly detection in time series. *IEEE Transactions on Neural Networks and Learning Systems*, 36(1):1495–1506, 2025.

[16] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, 2021.

[17] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[19] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 61–75, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[20] Gen Li and Jason J. Jung. Deep learning for anomaly detection in multivariate time series: Approaches, applications, and challenges. *Information Fusion*, 91:93–102, 2023.

[21] Yuening Li, Zhengzhang Chen, Daochen Zha, Kaixiong Zhou, Haifeng Jin, Haifeng Chen, and Xia Hu. Autood: Neural architecture search for outlier detection. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2117–2122, 2021.

[22] Shuyu Lin, Ronald Clark, Robert Birke, Sandro Schönborn, Niki Trigoni, and Stephen Roberts. Anomaly detection for time series using vae-lstm hybrid model. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4322–4326, 2020.

[23] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 6(1), mar 2012.

[24] Zimeng Lyu, AbdElRahman ElSaid, Joshua Karns, Mohamed Mkaouer, and Travis Desell. An experimental study of weight initialization and lamarckian inheritance on neuroevolution. In Pedro A. Castillo and Juan Luis Jiménez Laredo, editors, *Applications of Evolutionary Computation*, pages 584–600, Cham, 2021. Springer International Publishing.

[25] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *CoRR*, abs/1607.00148, 2016.

[26] Jose Martínez-Heras and Alessandro Donati. Enhanced telemetry monitoring with novelty detection. *Ai Magazine*, 35:37–46, 12 2014.

[27] H.D. Nguyen, K.P. Tran, S. Thomassey, and M. Hamad. Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management. *International Journal of Information Management*, 57:102282, 2021.

[28] Zijian Niu, Ke Yu, and Xiaofei Wu. Lstm-based vae-gan for time-series anomaly detection. *Sensors*, 20(13), 2020.

[29] Ilia Nouretdinov, James Gammerman, Matteo Fontana, and Daljit Rehal. Multi-level conformal clustering: A distribution-free technique for clustering and anomaly detection. *Neurocomputing*, 397:279–291, 2020.

[30] Peggy O'Neill, Dara Entekhabi, Eni Njoku, and Kent Kellogg. The nasa soil moisture active passive (smap) mission: Overview. In *2010 IEEE International Geoscience and Remote Sensing Symposium*, pages 3236–3239, 2010.

[31] Alexander Ororbia, AbdElRahman ElSaid, and Travis Desell. Investigating recurrent neural network memory structures using neuro-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 446–455, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Siamak Parhizkari. Anomaly detection in intrusion detection systems. In Venkata Krishna Parimala, editor, *Anomaly Detection*, chapter 6. IntechOpen, Rijeka, 2023.

[33] R Pascanu. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2013.

[34] C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.

[35] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 427–438, New York, NY, USA, 2000. Association for Computing Machinery.

[36] David Reinsel, John Gantz, and John Rydning. Data age 2025: The evolution of data to life-critical. *Don't Focus on Big Data*, 2, 2017.

[37] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 3009–3017, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Bernhard Schölkopf, John Platt, John Shawe-Taylor, Alexander Smola, and Robert Williamson. Estimating support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 07 2001.

[39] Andreas Theissler, Manuel Wengert, and Felix Gerschner. Rockad: Transferring rocket to whole time series anomaly detection. In Bruno Crémilleux, Sibylle Hess, and Siegfried Nijssen, editors, *Advances in Intelligent Data Analysis XXI*, pages 419–432, Cham, 2023. Springer Nature Switzerland.

[40] Kim Phuc Tran, Huu Du Nguyen, and Sébastien Thomassey. Anomaly detection using long short term memory networks and its applications in supply chain management. *IFAC-PapersOnLine*, 52(13):2408–2412, 2019. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

[41] Pascal Vincent, Hugo Larochelle, Y. Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 01 2008.

[42] Liang Xiong, Xi Chen, and Jeff Schneider. Direct robust matrix factorization for anomaly detection. In *2011 IEEE 11th International Conference on Data Mining*, pages 844–853, 2011.

[43] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. Deep isolation forest for anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12591–12604, 2023.

[44] Xingwei Yang, Longin Jan Latecki, and David Pokrajac. Outlier detection with globally optimal exemplar-based gmm. In *SIAM International Conference on Data Mining, Conference Proceedings*, pages 145–154, 04 2009.

[45] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 841–850, 2020.