Anomaly Detection on Time Series Graphs

Roberto Giordano
A.A 2024/2025

Supervisor:
Karina Chichifoi

Abstract

Anomaly detection in IoT time series data is essential for identifying irregularities and potential cyberattacks. This work focuses on leveraging spatial and temporal correlations to enhance detection accuracy. Spatial correlation reflects the similarity of data from sensors in close proximity, such as temperature sensors recording similar values in the same area. Temporal correlation ensures consistency in data trends over time, avoiding unexpected spikes or irregular behavior.

In typical scenarios, neighboring sensors produce similar data that evolves coherently, while anomalies disrupt this pattern. To address this, I propose a method that constructs graphs based on spatial and temporal similarities among sensors. These graphs evolve over time, capturing the dynamics of IoT systems. Anomalies manifest as deviations in these patterns, signaling potential attacks or irregularities.

The objective is to develop an advanced anomaly detection system using Spatial-Temporal Graph Autoencoders (STGAE). This approach harnesses the power of graph-based models to capture and analyze correlations in both space and time, enabling the detection of anomalies with high precision. This method provides a robust framework for monitoring IoT systems, ensuring their security and reliability in real-world applications.

The code of the project is available on:
https://github.com/Robertogiordano/anomaly_detection_on_time_series_graph.git

Index

1. INTRODUCTION 4

2. DATASET CONTENTS 6

DATA ORGANIZATION 6
THE EARTHDATA TOOL 6

3. ANALYSIS AND INTERPRETATION OF DATA 7

4. SUBSET AND SELECTED VARIABLES FOR ANALYSIS 8

5. PREPROCESSING OF THE SATELLITE DATASET 11

6. DATA PREPROCESSING FOR MODEL INPUT 12

7. ADVANCED PREPROCESSING FOR ANOMALY DETECTION 12

8. NODE FILE CREATION AND OPTIONAL AUGMENTATION 14

9. STATIC EDGE CREATION FOR SPATIOTEMPORAL NETWORKS 15

EDGE STORAGE AND REPRESENTATION 15

10. STCONVAE ARCHITECTURE 16

SIMPLETEMPORALCONV: CAPTURING TEMPORAL DEPENDENCIES 16
SIMPLETEMPORALDECONV: RECONSTRUCTING TEMPORAL PATTERNS 17
SIMPLESPATIALCONV: MODELING SPATIAL RELATIONSHIPS 17
SIMPLEENCODER: LEARNING COMPACT SPATIOTEMPORAL REPRESENTATIONS 17
SIMPLEDECODER: RECONSTRUCTING THE INPUT 17
SIMPLESTCONVAE: THE AUTOENCODER FRAMEWORK 18
MECHANISM OF ANOMALY DETECTION 18

11. EXPERIMENTS 19

SHUFFLED DATA AND DATA SPLITTING STRATEGIES 19
DATASET VARIATIONS 19
TEMPORAL AND SPATIAL SANDWICH ARCHITECTURES 20
ACTIVATION AND GATE FUNCTIONS 20
COMBINED LOSS FUNCTIONS 20
TEMPORAL WINDOW AND BATCH SIZES 21
ENHANCED MODEL ARCHITECTURES 21
TRAINING CONFIGURATIONS 22

12. FINAL ARCHITECTURE 22

13. PERFORMANCE METRICS AND MODEL ROBUSTNESS 24

14. CONCLUSION 25

15. BIBLIOGRAPHY 29

1. Introduction

The Internet of Things (IoT) is revolutionizing the way devices communicate and collaborate, driving the rapid expansion of interconnected systems across various domains, from smart homes to industrial automation under Industry 4.0. This surge in interconnectivity has created vast opportunities for enhanced efficiency and innovation, but it also exposes these systems to significant external threats. Cyberattacks, system malfunctions, and operational failures are increasingly common risks in this highly interconnected ecosystem, making the detection and mitigation of anomalies a critical priority.

IoT systems rely heavily on the continuous monitoring of environmental and operational parameters through a network of sensors. These devices generate vast amounts of data that often exhibit **spatial and temporal correlations**. Spatial correlations arise when sensors located near each other measure related phenomena, such as temperature sensors distributed across a building that should report similar readings within expected ranges. Temporal correlations, on the other hand, reflect the consistency of measurements over time, such as temperature patterns following diurnal cycles or responding predictably to external conditions.

When anomalies occur in these patterns, they can be indicative of deeper issues within the system. For instance, a temperature sensor in one part of a building reporting an unusually high value compared to others nearby may suggest a malfunction, tampering, or an environmental anomaly. Similarly, a sudden spike or drop in a sensor's readings over time, deviating sharply from its historical trends, might signal an issue such as equipment failure or the presence of unauthorized interference. These irregularities, classified as **context anomalies**, are unique because they depend on deviations from both spatial and temporal norms.

Understanding and effectively detecting context anomalies in IoT data streams is paramount for maintaining the security, reliability, and functionality of these systems. By leveraging the inherent spatial and temporal correlations in IoT data, anomaly detection mechanisms can distinguish between normal variations and irregularities that could signify potential threats or operational disruptions.

Existing methods for anomaly detection in IoT systems often face significant challenges in integrating both spatial and temporal dimensions effectively, as highlighted by prior research (e.g., [10]). Traditional approaches to anomaly detection frequently focus on centralized processing or isolated analysis of spatial or temporal correlations, which can limit their applicability in the highly dynamic and interconnected environments of IoT networks. For example, while spatial anomaly detection methods leveraging neighborhood information in sensor networks have been explored (e.g., [11]), these techniques are often constrained by assumptions about network topology or the type of anomalies being detected, such as those arising from diffusive processes.

Similarly, while there has been substantial progress in anomaly detection across various domains, IoT-specific applications remain relatively underexplored due to the unique challenges posed by high-dimensional, correlated time-series data generated by IoT systems ([10]). Many existing methods require significant expert knowledge and customization for specific use cases, reducing their generalizability and scalability in real-world IoT deployments. Furthermore, approaches that

emphasize spatial-temporal modeling often lack the flexibility to address complex cyberattacks, which exploit both dimensions simultaneously.

This study addresses these gaps by proposing a novel anomaly detection framework based on **Spatial-Temporal Convolution Graph Autoencoders (STConvGAE).** This approach aims to harness the inherent spatio-temporal correlations present in IoT networks to detect anomalies more effectively and efficiently using satellite data. By integrating insights from neighborhood-based data fusion methods with advanced graph-based learning, the proposed framework seeks to bridge the gap between existing spatial-temporal models and the pressing need for robust cyberattack detection in IoT environments.

The proposed model captures the intricate dependencies within spatiotemporal data, encoding normal behavior into a compact representation and flagging anomalies through high reconstruction errors. This reconstruction-based approach allows for unsupervised anomaly detection, particularly suited for unlabeled datasets. The results demonstrate the model's robustness under various conditions, achieving exceptional performance even with significant data corruption.

The remainder of this report is structured as follows: **Section 2** introduces the dataset, detailing its contents, organization, and the tools used for exploration and analysis. **Sections 3 through 9** focus on the preprocessing stages, including basic data preparation, advanced preprocessing techniques for anomaly detection, and the creation of spatiotemporal network representations through nodes and edges. **Section 10** describes the architecture of the STConvAE framework, breaking down its components—such as temporal and spatial convolution modules—and their roles in learning and reconstructing spatiotemporal patterns, including the mechanism of anomaly detection within the framework. **Section 11** presents the experimental design, including variations in dataset usage, architectural enhancements, and training configurations, while also evaluating performance using key metrics. The report concludes by discussing the results, highlighting model robustness under diverse conditions, and outlining future research opportunities.

2. Dataset Contents

The "**SNDRSNML2RMS**_1" dataset is a Level-2 product derived from measurements obtained by the Advanced Technology Microwave Sounder (ATMS) on the Suomi-NPP satellite. This dataset, which is processed using the RAMSES II retrieval algorithm, provides detailed atmospheric profiles, including temperature, water vapor, cloud properties, and surface characteristics. Its primary focus is to support climate and weather research by enabling the retrieval of geophysical parameters from microwave observations.

A **Level 2 product** refers to processed satellite data that has been converted from raw measurements (Level 1 data) into geophysical variables that can be directly used for scientific analysis.

Level 2 products provide geophysical variables such as temperature, humidity, precipitation, surface properties, and other atmospheric or terrestrial parameters. These are derived from the original radiometric or spectral measurements obtained by the satellite's sensors. Each data point in a Level 2 product is associated with specific geographic coordinates (latitude, longitude, and sometimes altitude) and timestamps, allowing it to be mapped to specific locations on Earth. These type of data are often divided into smaller time or spatial segments (called granules) to make it manageable for analysis.

Data Organization

The data is organized into **six-minute granules**, with each granule containing all observations during that time period. The files are stored in the **netCDF4 format**, incorporating hierarchical structures to include both primary data and ancillary information. The dataset adheres to Climate and Forecast Metadata Conventions (CF) and Attribute Conventions for Data Discovery (ACDD), ensuring its compatibility with standard data tools. This format ensures that large multidimensional arrays, metadata, and auxiliary information are integrated seamlessly within a single file structure. Each data point in the dataset is georeferenced with corresponding latitude, longitude, and time attributes, ensuring precise spatial and temporal alignment.

One unique aspect of this dataset is its reliance solely on microwave data for retrievals, making it effective under all-sky conditions, including cloudy environments where infrared methods often struggle. The RAMSES II algorithm uses advanced radiative transfer and optimization techniques to achieve high-quality retrievals, with ancillary variables providing uncertainty estimates and quality flags for data filtering.

The Earthdata Tool

To access and work with this dataset, NASA's Earthdata platform [2] serves as a crucial tool. Earthdata is a comprehensive system designed to provide researchers with seamless access to a wide range of Earth science datasets. Through its interface, users can search for specific datasets, apply filters based on time and location, and download the required data files directly. Earthdata also offers preliminary visualization tools that allow users to explore patterns and trends in the data before download. Furthermore, detailed documentation and support resources, including metadata descriptions and user guides, are available to ensure researchers can effectively interpret and utilize the dataset. For advanced users, Earthdata provides APIs and integrates with

NASA's Common Metadata Repository (CMR), enabling automated data retrieval and streamlined workflows.

3. Analysis and interpretation of data

The SNDRSNML2RMS_1 dataset utilizes a structured grid system to represent the spatial distribution of measurements derived from the Advanced Technology Microwave Sounder (ATMS). Understanding the data organization is crucial for effective analysis, particularly for non-expert users unfamiliar with remote sensing data conventions. This section elucidates key aspects of the dataset's dimensional framework and its relationship to geospatial coordinates.

The dataset is organized in two primary dimensions: **atrack** (along-track) and **xtrack** (cross-track). These dimensions define the spatial structure of the measurements collected by the satellite sensor:

- **atrack Dimension**: The atrack represents the satellite's movement along its orbit, often referred to as the along-track direction. As the satellite travels over the Earth's surface, it collects sequential data points along its trajectory. In the SNDRSNML2RMS_1 dataset, the atrack dimension comprises 135 indices, corresponding to discrete positions along the orbital path within a six-minute observation granule.
- **xtrack Dimension**: Perpendicular to the along-track direction is the xtrack dimension, which captures data across the swath width of the satellite's observation field. The ATMS sensor scans outward from the satellite's nadir (directly below) to cover an area on either side of its path. The dataset defines 96 indices in the xtrack dimension, representing individual positions within each swath at a given along-track location.

Together, the atrack and xtrack dimensions form a two-dimensional grid structure [Fig 1], enabling comprehensive spatial coverage across the Earth's surface.

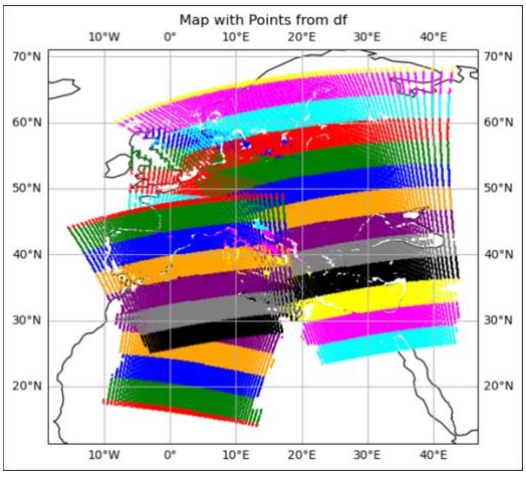


Figure 1.

The image displays satellite measurement points over a geographic region, with data points distinctly color-coded to represent time intervals (minutes) for a specific day. The satellite's movement follows a predominantly vertical trajectory (along-track direction), while it simultaneously scans horizontally (cross-track direction) to capture data across its swath. Each color corresponds to a specific minute of observation, illustrating the satellite's systematic sampling pattern. This visualization effectively demonstrates how the satellite collects data in both vertical and horizontal dimensions, producing overlapping swaths that contribute to comprehensive spatial coverage.

To geolocate the observations, each (atrack, xtrack) pair is associated with a unique geographical position represented by latitude (lat) and longitude (lon):

- **Latitude (lat)**: Specifies the north-south position of a data point, ranging from -90° (South Pole) to +90° (North Pole).
- **Longitude (lon)**: Specifies the east-west position, ranging from -180° to +180°.

These coordinates ensure that every data point in the (atrack, xtrack) grid is mapped to a precise location on the Earth's surface. The lat and lon values are critical for aligning the dataset with external geospatial frameworks and for visualization on global or regional maps.

Each measurement in the dataset is uniquely defined by a combination of atrack, xtrack, lat, and lon. For example, a point indexed as (atrack=10, xtrack=50) corresponds to a specific position along the satellite's path and across its swath. The corresponding lat and lon values pinpoint this measurement's exact location on the Earth's surface.

4. Subset and Selected Variables for Analysis

The original dataset, measuring approximately 2.5TB, contains **96 variables** covering the entire globe and sampled quarterly in **January, April, July, and October** for the years **2013** and **2015**.

Due to storage, time, and computational limitations, I restricted [Fig 2] the dataset to a region centered around Italy, spanning from 20°N to 70°N in latitude and from -40°E to 60°E in longitude. This subset significantly reduced the data size to 53GB.

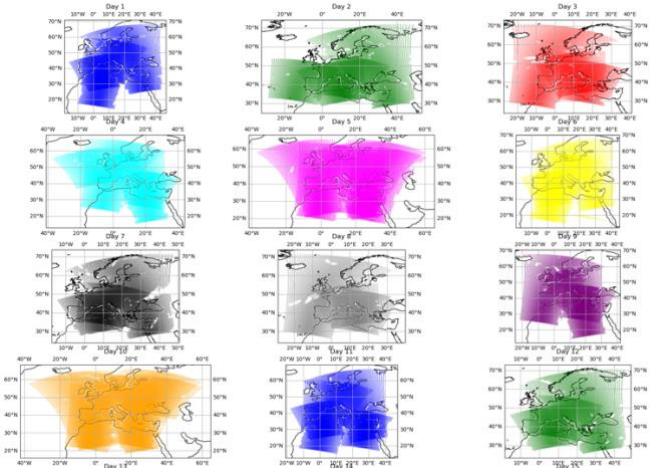


Figure 2.
Satellite swath coverage for the first 12 days of January 2015 in the subsampled dataset, illustrating daily observation regions across sampled areas. Each panel represents the coverage for a specific day, highlighting the temporal and spatial extent of the dataset over the study region.

Following this, I conducted an in-depth analysis of the available variables, understanding their relevance, and ultimately selected eight key variables for further processing and analysis.
The selected variables and their descriptions are outlined in [Table 1].

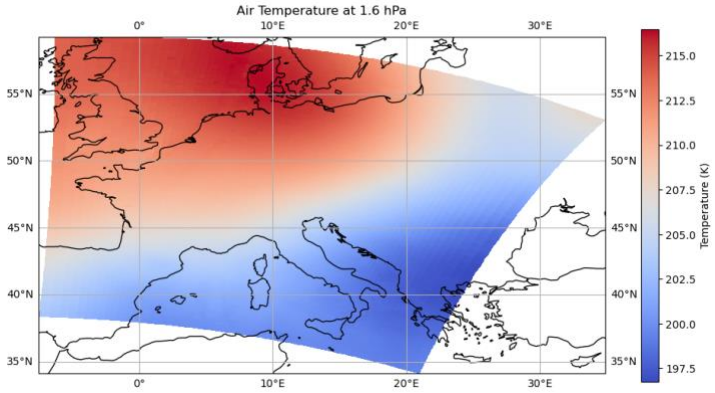
All these variables are organized along the atrack (along-track) and xtrack (cross-track) dimensions, providing geospatial information for each point. While some variables, such as air

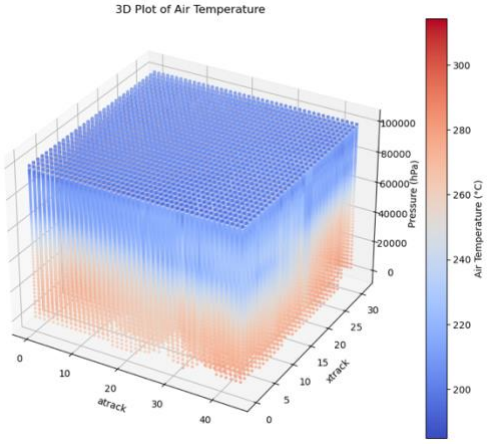
temperature, are additionally distributed across pressure levels, I excluded them due to storage constraints [Fig 3].
Each variable in the dataset is paired with a **quality flag** indicating the reliability of the data. For the analysis, I filtered out all observations flagged as "not use," ensuring that only high-quality data points were retained for further processing. This step was critical to maintaining the scientific validity of the subsequent analyses.

Table 1

Variable Name	Description	Type	Units	Dimensions
solar_zenith_angle	The angle between the sun and the zenith (directly overhead) at a specific observation point.	float32	degrees	(atrack, xtrack)
solar_azimuth_angle	The angle of the sun's position relative to true north at a specific observation point.	float32	degrees	(atrack, xtrack)
surf_air_temp_masked	The near-surface air temperature (~2 meters above ground), with invalid data masked.	float32	Kelvin	(atrack, xtrack)
surf_temp_masked	The radiative temperature of the Earth's surface, with invalid data masked.	float32	Kelvin	(atrack, xtrack)
surf_spec_hum_masked	The specific humidity (mass fraction of water vapor) near the surface, with invalid data masked.	float32	unitless	(atrack, xtrack)
h2o_vap_tot_masked	The total column amount of water vapor in the atmosphere, with invalid data masked.	float32	kg/m^2	(atrack, xtrack)
cloud_liquid_ water_masked	The total liquid water content in clouds, with invalid data masked.	float32	kg/m^2	(atrack, xtrack)
atmosphere_mass_ content_of_cloud _ice_masked	The total mass of ice within clouds in the atmospheric column, with invalid data masked.	float32	kg/m^2	(atrack, xtrack)

.





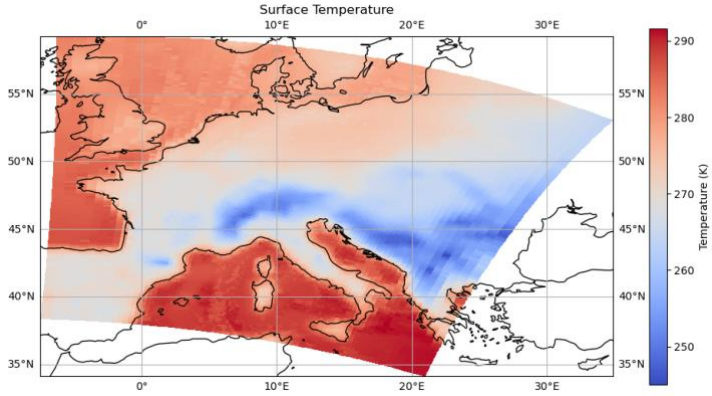


Figure 3.

Visualization of air and surface temperature data over the region centered on Italy. The top-left image represents the air temperature at a pressure level of 1.6 hPa, illustrating temperature variation across the region. The top-right image shows a 3D plot of air temperature as a function of `atrack`, `xtrack`, and pressure, highlighting vertical temperature gradients in the atmospheric column. The bottom image depicts surface temperature, revealing temperature differences at ground level, influenced by geographic and atmospheric conditions. All visualizations use color gradients to represent temperature values in Kelvin.

5. Preprocessing of the Satellite Dataset

The preprocessing pipeline [3] for the SNDRSNML2RMS_1 dataset focuses on efficiently extracting relevant variables, handling quality control flags, and organizing the data into a tabular format suitable for further analysis. Given the large size of the dataset and its structure across multiple NetCDF files, the process leverages parallel processing to enhance efficiency. Below is a detailed breakdown of the preprocessing steps:

The dataset consists of NetCDF files organized in a specified directory. Each file corresponds to a distinct observational granule, and the script iterates over these files to process the data. To handle the computational load, the pipeline uses the ProcessPoolExecutor to parallelize the file processing.

A dictionary, mapping, defines the correspondence between dataset variables and their standardized names for processing. This includes geospatial attributes (e.g., lat and lon), temporal information (date), and key atmospheric variables like temperature, humidity, and cloud properties. The mapping also ensures that variables with associated quality control (qc) flags are correctly identified for masking.

Each file is processed using the process_file function, which:

- Opens the NetCDF file using the xarray library.
- Extracts satellite-specific metadata, such as position and time, along with geophysical variables for each atrack and xtrack index pair.
- Applies quality control by masking values flagged as "not use" (indicated by a quality control flag value of 2). If such flags are present, the corresponding data values are replaced with NaN to ensure only valid data is retained.

Using ProcessPoolExecutor, the function processes multiple files simultaneously. This significantly reduces the overall runtime by distributing the workload across available CPU cores. A partial function ensures that constant arguments (e.g., file path and variable mapping) are passed correctly to the processing function.

Once all files are processed, the extracted rows from each file are aggregated into a single Pandas DataFrame. This structure consolidates all relevant variables into a tabular format, with each row corresponding to a specific geospatial observation (indexed by atrack and xtrack).

The final DataFrame, containing key variables and their respective quality-filtered values, is saved to a CSV file (data_2013_2015.csv). This format makes it easy to load the data into analytical tools and *pythorch geometrics* for graph creation.

6. Data Preprocessing for Model Input

Before constructing the model, a further preprocessing step was performed to prepare the dataset for efficient analysis and modeling.
Starting with the cleaned and merged data (data_2013_2015.csv), **missing values were removed**, and the index was reset to maintain the integrity of the data.
Temporal features, such as day, hour, minute, and second, were extracted from the original timestamp column, allowing for better handling of temporal dependencies in the data. These extracted features were then converted into separate columns, with the original timestamp and intermediate columns dropped to streamline the dataset.
To enhance data organization, columns were reordered to prioritize temporal attributes, ensuring compatibility with spatio-temporal modeling approaches.
Furthermore, the **satellite_position variable**, originally represented as a three-element array, was dropped to reduce complexity, while other irrelevant columns like air_temp_masked were also removed to address space constraints. This preprocessing step resulted in a cleaned, time-indexed dataset saved as *data_preprocessed_2013_2015.csv*, ready for further use in spatio-temporal anomaly detection.

7. Advanced Preprocessing for Anomaly Detection

To manage spatial data, latitude and longitude values were **rounded to a precision of one decimal** place. This step introduced a trade-off between geographic precision and computational efficiency by grouping closely located data points while introducing minimal rounding error. The rounding precision was quantified, where each decimal place contributes progressively smaller geographic errors (e.g., 0.1° roughly corresponds to 11 km at the equator) [Table 2].

The dataset was then filtered to retain only spatial coordinates (latitude and longitude pairs) that exceeded a recurrence threshold, ensuring the selection of meaningful and frequently observed locations for graph construction [Fig 4].
This process resulted in a dataset optimized for spatio-temporal anomaly detection, balancing the need for granularity with computational efficiency.
The refined coordinates were saved to a file (*unique_coords.csv*) for use in constructing the graph-based model, ready to be used to create the graph nodes [Fig 5].

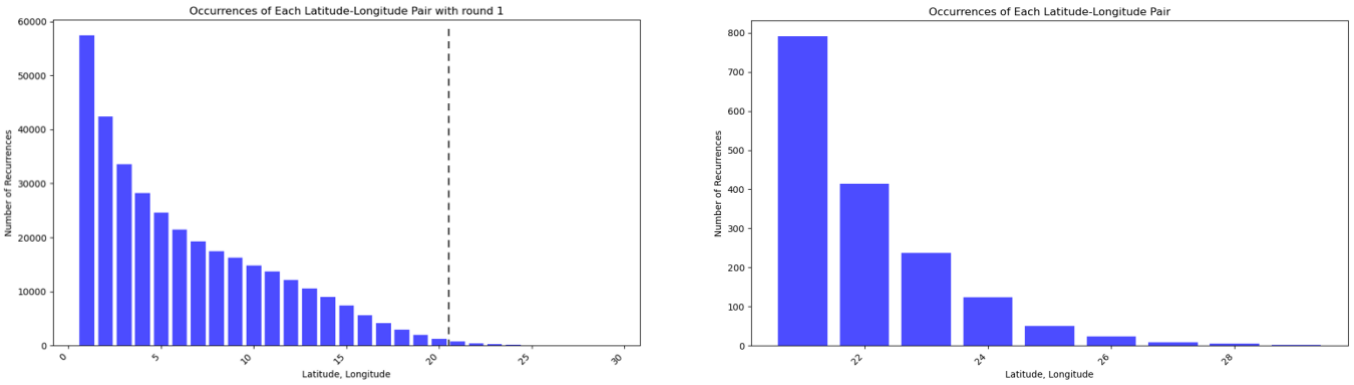


Figure 4

The left graph illustrates the distribution of occurrences for latitude-longitude pairs rounded to one decimal place. The dashed line at 20 occurrences represents the threshold for selection, ensuring that only spatial points recurring on most days within a single month are retained. The right graph provides a zoomed-in view of latitude-longitude pairs exceeding the threshold, highlighting the most consistent and frequently observed spatial coordinates. This selection ensures the focus is on meaningful and stable points for graph-based anomaly detection.

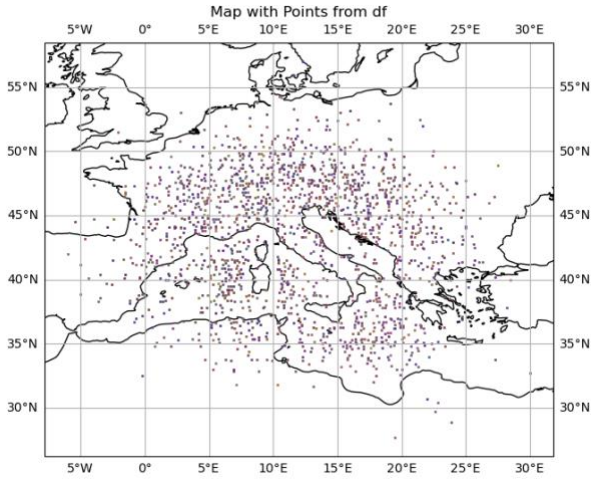


Figure 5

Spatial distribution of selected latitude-longitude points with the highest occurrences during January 2015. These points, recurring on most days within the month, are chosen as nodes for constructing the static graph used in the anomaly detection model.

Decimal Places	Approximate Error (km)
0 decimals	111 km
1 decimal	11.1 km
2 decimals	1.11 km
3 decimals	111 m
4 decimals	11.1 m
5 decimals	1.11 m
6 decimals	0.11 m

Table 2: Estimation of Rounding Error

***0 decimals**: A whole number approximation can result in a maximum displacement of 111 km, a significant distance.*
***1 decimal**: With one decimal place removed, the precision drops to about 11 km, comparable to the width of a medium-sized city.*
***2 decimals**: Two decimal places ensure precision within 1.1 km, suitable for urban areas or neighborhoods.*
***3 decimals**: Three decimal places yield an error of approximately 111 meters, adequate for city zones.*
***4 decimals**: Four decimal places reduce the error to about 11 meters, suitable for locating buildings.*
***5 decimals**: Five decimal places allow for precision within 1 meter, appropriate for pinpointing specific locations, such as an entrance.*
***6 decimals**: Six decimal places bring the error down to about 10 cm, ideal for high-precision tasks like geolocation in archaeology or robotics.*

8. Node file creation and optional Augmentation

Node features are derived for graph-based learning. Each unique combination of latitude, longitude, and time (day, hour) is mapped to a node. Node attributes include variables listed in [Table 1].
Features are **normalized** to ensure uniformity and improve model convergence during training.

Data augmentation is performed to enhance the dataset's coverage, is a facultative option created for some experiments. Using historical and spatial averages, synthetic data points are generated for missing time periods and regions. This involves:

- **Lat/Lon Pair Selection**: Unique spatial coordinates are precomputed and reused across time steps.
- **Interpolation and Noise Addition**: Variables are interpolated between observed values, with added noise to simulate natural variability.

Synthetic data mimics seasonal transitions by considering the mean values of variables during transition months. This ensures that augmented data preserves the underlying temporal dynamics of the observed data [Fig 6].

The dataset is divided into daily subsets, saved as CSV files for easy access. Numeric columns are averaged for duplicate spatial coordinates, ensuring a consistent representation of each unique location.
A master list of unique spatial coordinates (lat, lon) is exported for each day. These fixed nodes serve as the basis for constructing spatiotemporal graphs, enabling future studies to maintain a consistent spatial reference framework.

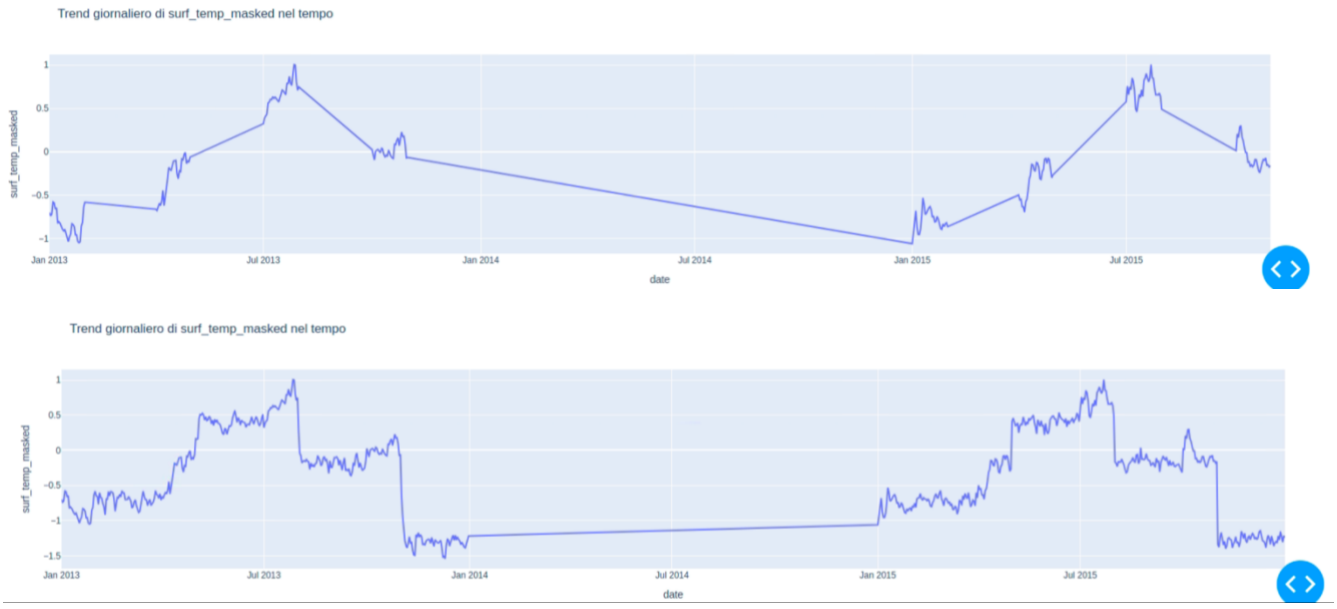


Figure 6
Daily trend of the variable surf_temp_masked over time. The top panel shows the real dataset, sampled every three months in 2013 and 2015, highlighting gaps in the data. The bottom panel represents the augmented dataset, where gaps in 2013 and 2015 have been filled, providing a continuous time series for improved analysis and modeling.

9. Static Edge Creation for Spatiotemporal Networks

Edges in a graph structure represent relationships or interactions between nodes. In the context of geospatial datasets, edges typically encode spatial or temporal proximity between geographical points.

Edges are created based on spatial proximity between nodes. The maximum distance between two nodes to be considered connected is 100 km. **The edges are the same for every sampled day**.

To evaluate pairwise distances, all possible combinations of node indices are generated. For a dataset with N nodes, the total number of pairs is

$$\binom{N}{2} = \frac{N(N-1)}{2}$$

Each pair represents a potential edge and is assessed for proximity.
The geodesic distance between two nodes, defined by their latitude and longitude, is calculated using the **Haversine formula**. This computation considers Earth's curvature, providing accurate distances in kilometers.

For each node pair:

1. **Distance Calculation**: Spatial distance is computed using geodesic functions.
2. **Threshold Check**: If the distance is within the defined threshold (100 km), an edge is created between the nodes.
3. **Edge Weighting**: The weight of the edge is proportional to the spatial distance, normalized by the threshold.

To handle the computational complexity of pairwise comparisons, edge creation is parallelized using a thread-based executor. This significantly reduces runtime, making it feasible to process large datasets effi

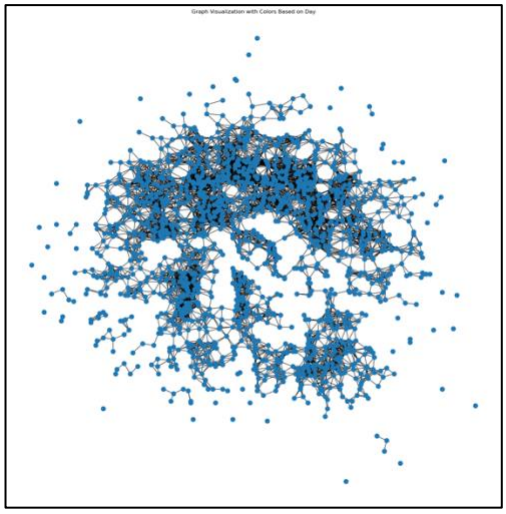


Figure 7
Graph structure illustrating node connections between selected points on the map. Each link is assigned a weight, where greater distances correspond to higher weights.

Edge Storage and Representation

The edges are represented in the following forms:

1. **Edge Index**: A tensor of shape [2, E], where E is the number of edges. Each column represents a connection between two nodes.
2. **Edge Weights**: A tensor of shape [E], containing the normalized distance between connected nodes.

These tensors are stored in PyTorch Geometric's Data object, enabling seamless integration into graph-based machine learning pipelines.

10. STConvAE architecture

The rise of interconnected systems and networks has amplified the need for robust methods to detect anomalies, especially in the cybersecurity domain. Cyber threats often manifest as subtle, anomalous patterns hidden within complex spatiotemporal data. These patterns can arise due to unexpected interactions between nodes in a network, unusual temporal activity, or both. To address this challenge, I propose a novel neural network architecture: the **Spatiotemporal Convolutional Autoencoder (STConvAE)** [4][5][6][7][8]. This model is designed to leverage the inherent spatial and temporal dependencies in cybersecurity data, enabling the detection of deviations from normal behavior that might indicate a security breach or malicious activity. The following sections detail the architectural components [Fig 8] and their roles within the broader context of anomaly detection.

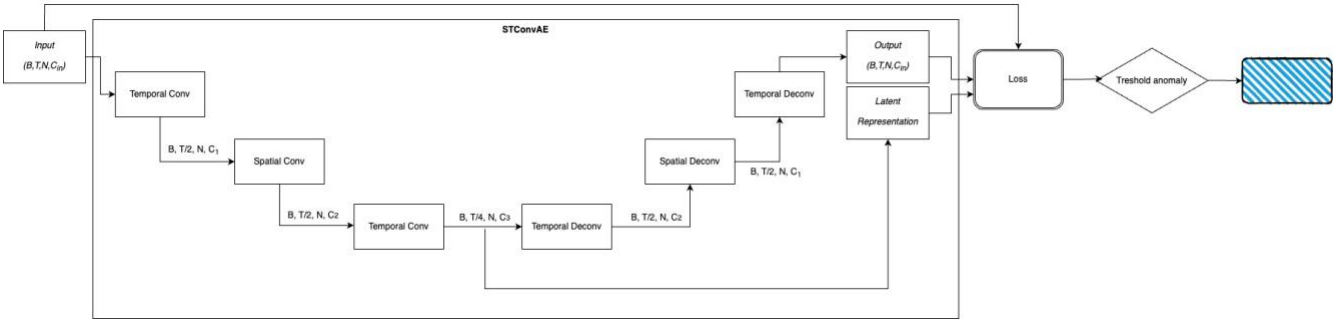


Figure 8

General architecture of the STConvAE model. Here, $C_1=C_{in}$. In the final model configuration, $[C_1,C_2,C_3]=[8,16,32]$.

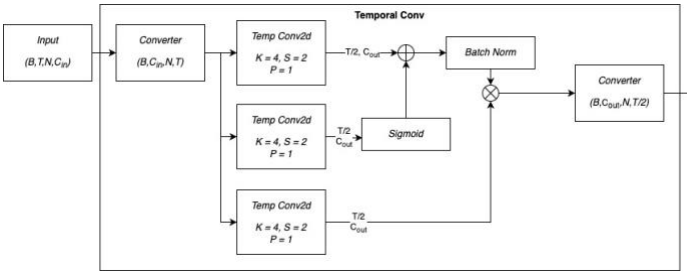


Figure 9a

Temporal Block structure. Each invocation of this block reduces the time dimension by half, retaining only the most relevant information and relationships within the time series.

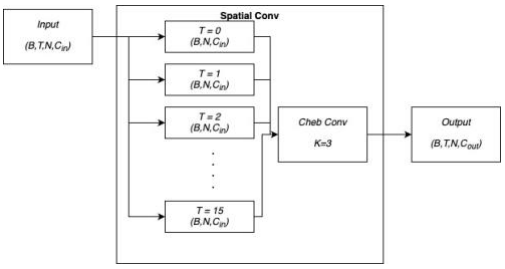


Figure 9b

Spatial Convolutional Block. This block applies ChebConv to each graph in the time series, extracting low-level spatial information from the nodes.

SimpleTemporalConv: Capturing Temporal Dependencies

The **SimpleTemporalConv** [Fig 9a] module is responsible for modeling the temporal characteristics of the data. Temporal patterns are crucial in cybersecurity, as anomalies often manifest as irregularities in the temporal behavior of network nodes or systems. The module uses a combination of convolutional layers to extract fine-grained temporal features. It includes **three convolutional layers**, each with distinct roles. The first layer captures raw temporal patterns, while the second introduces a **gating mechanism** using a sigmoid activation [4]. This gate determines the relative importance of features extracted at different time points. The third layer applies a learned transformation to refine the temporal features. These outputs are combined

through a multiplicative interaction, followed by residual connections and batch normalization to stabilize training and enhance the propagation of meaningful temporal signals. By focusing exclusively on temporal sequences, the module ensures the network can identify time-based anomalies, such as unusual surges or drops in activity.

SimpleTemporalDeConv: Reconstructing Temporal Patterns

The **SimpleTemporalDeConv** module is the counterpart to the temporal convolution module. Its role is to reconstruct the temporal patterns from the latent representation generated by the encoder. Using transposed convolutions, the module upscales the compressed temporal features to match the original temporal resolution. It mirrors the gating mechanism of the **SimpleTemporalConv**, allowing the network to selectively emphasize certain features during reconstruction. This mechanism ensures that the decoder not only produces outputs similar to the input but also faithfully preserves temporal anomalies that may exist in the data.

SimpleSpatialConv: Modeling Spatial Relationships

The spatial dimension corresponds to the relationships between points in the network. The **SimpleSpatialConv** module [Fig 9b] is specifically designed to capture these spatial dependencies. It employs graph-based convolutions using Chebyshev polynomials, which approximate the graph Laplacian to perform localized filtering over the graph [9]. This enables the network to model interactions between nodes based on their connectivity and proximity within the network. The spatial module processes each temporal snapshot of the data independently, ensuring that the spatial relationships at every time step are thoroughly captured.

SimpleEncoder: Learning Compact Spatiotemporal Representations

The **SimpleEncoder** integrates temporal and spatial layers to learn compact representations of the input data. It begins with a temporal convolution layer to extract temporal features, which are then passed through a spatial convolution layer to model the interactions between nodes. The output is further refined by a second temporal convolution layer, which encodes the combined spatiotemporal information into a latent representation. This latent space is compact yet expressive, capturing both normal temporal sequences and spatial interactions.

SimpleDecoder: Reconstructing the Input

The **SimpleDecoder** is responsible for reconstructing the original data from the latent representation generated by the encoder. It reverses the operations performed by the encoder, starting with a temporal deconvolution layer to expand the temporal resolution, followed by a spatial convolution layer to restore spatial relationships, and finally a second temporal deconvolution layer to produce the reconstructed spatiotemporal data. The reconstruction process is designed to closely match the original input while preserving anomalies. Discrepancies between the input and reconstructed data can be directly interpreted as potential anomalies, as the network struggles to reconstruct patterns that deviate from the learned normal behavior.

SimpleSTConvAE: The Autoencoder Framework

The **SimpleSTConvAE** combines the encoder and decoder into a single autoencoder framework. It sequentially applies the encoder to compress the spatiotemporal input into a latent representation and then uses the decoder to reconstruct the input. The architecture is modular, allowing for flexibility in scaling the number of layers or adjusting parameters such as kernel size and spatial connectivity. The model outputs both the reconstructed data and the latent representation. For anomaly detection, reconstruction error—measured as the difference between the input and output—serves as a primary indicator of anomalies.

Mechanism of Anomaly Detection

The proposed autoencoder is trained on normal cybersecurity data, allowing it to learn the typical spatiotemporal patterns of network activity. During the inference phase, the model processes incoming data and computes the reconstruction error. Anomalous activities, such as unusual traffic spikes, irregular communication between nodes, or deviations in temporal patterns, lead to high reconstruction errors. These deviations are flagged as potential anomalies. The goal is to achieve high recall on anomalies, prioritizing the detection of actual anomalies over mistakenly flagging slight deviations in normal behavior as anomalies. This ensures that true security threats are identified effectively, even if it means tolerating minor inaccuracies in classifying normal behavior.

11. Experiments

This chapter outlines the diverse experiments conducted to evaluate and optimize spatio-temporal anomaly detection models. These experiments explored different data configurations, architectural designs, loss functions, and training setups, aiming to determine the most effective model for the given dataset. Each experiment is described with a focus on the modifications introduced and the hypotheses tested.

Shuffled Data and Data Splitting Strategies

The organization of the data played a crucial role in the experiments. To enhance generalization across all temporal periods, we adopted a **shuffled data strategy**. Initially, the temporal data were split sequentially: 70% for training, 15% for validation, and 15% for testing. This approach, however, resulted in training on data only from a limited part of the year, leaving validation and testing sets drawn from later months. Such a split failed to provide the model with exposure to seasonal variations across the entire year, limiting its ability to generalize.

To address this, I've implemented a uniform sampling approach. In the shuffled strategy, data points from all months of the year were randomly distributed across the training, validation, and test sets keeping the same percentages. This ensured that each set contained representative samples from the entire year, providing the model with a diverse temporal context during training and testing. As a result, this approach prevented overfitting to specific time frames and improved the model's robustness in detecting anomalies throughout all seasons.

Dataset Variations

The experiments utilized two primary datasets:

- **Subdata:** This dataset included only the year 2015, representing a single-year subset of the data.
- **Subdata Extended:** This dataset encompassed data from both 2013 and 2015, providing a broader temporal range for training and testing.

As expected, increasing the volume and diversity of data resulted in improved model performance, highlighting the importance of leveraging broader datasets for robust training and testing.

Additionally, several experiments incorporated **augmented datasets**, which reconstructed missing months within the year using interpolation and noise-based augmentation. This augmentation aimed to fill temporal gaps and evaluate the models' ability to generalize across reconstructed data.

Experiments using the augmented dataset demonstrated that the introduction of synthetic data resulted in increased errors due to excessive noise. This suggests that the augmentation approach used introduced variability that was not representative of the underlying patterns in the real data. Consequently, the decision to exclude augmentation for the final model proved beneficial in improving accuracy and recall.

Temporal and Spatial Sandwich Architectures

The primary architectural variations explored were **Temporal Sandwich** and **Spatial Sandwich** models:

- **Temporal Sandwich Models:** These models featured an encoder and decoder where the convolutional layers were organized as a temporal-spatial-temporal sequence. Specifically, each block contained a temporal convolution layer (temp_conv_1), followed by a spatial convolution layer, and another temporal convolution layer (temp_conv_2). The goal was to emphasize temporal relationships while incorporating spatial information.
- **Spatial Sandwich Models:** These models reversed the convolutional sequence, using two spatial convolution layers with a temporal convolution layer sandwiched between them. This configuration focused on spatial dependencies while preserving temporal continuity.
- **Double Spatial and Temporal Sandwich Models:** These extended the sandwich architectures by alternating two spatial and two temporal convolution layers, creating a more complex hierarchy to capture both spatial and temporal patterns comprehensively.

The results obtained on the training set revealed a slight yet consistent improvement in the performance of the Temporal Sandwich model compared to the Spatial Sandwich model. Interestingly, the Double Sandwich model achieved performance similar to the Temporal Sandwich model, offering no substantial advantage despite its added complexity. Given its smaller parameter count and computational efficiency, the Temporal Sandwich model was ultimately selected as the preferred architecture.

Activation and Gate Functions

Different activation and gate functions were tested to assess their impact on the models' learning ability:

- **Activation Functions:** The models were tested using both ReLU and Leaky ReLU activation functions. For Leaky ReLU, variations in the negative slope (e.g., 0.02, 0.03) were explored to evaluate their effect on gradient flow and model convergence.
- **Gate Functions:** Gate mechanisms using **sigmoid** and **tanh** functions were tested. These gates controlled the flow of information through the network, with sigmoid gates emphasizing binary-like activations and tanh gates allowing smoother transitions.

The results demonstrated a significant improvement when using Leaky ReLU as the activation function with the standard slope of 0.01. This choice effectively mitigated the loss of information related to negative gradients, enhancing the model's learning capability.
For the gate function, the sigmoid function outperformed tanh, providing better control over the information flow and leading to superior overall results.

Combined Loss Functions

To improve the models' ability to learn both spatial and temporal dependencies, different loss functions were tested:

- **MSE Loss:** The simplest loss function was initially tested on the output response, focusing on minimizing the mean squared error between predictions and ground truth.

- **L1 Loss:** This loss was also evaluated, providing an alternative to MSE by emphasizing absolute differences.
- **Latent Space Losses:** Mean Squared Error (MSE) and L1 loss functions were applied in the latent space to optimize reconstruction quality.
- **Edge Weight Preservation Loss:** This loss encouraged the model to maintain consistent spatial relationships across nodes during training.
- **Pairwise Similarity Loss:** The similarity between node features in the latent space was preserved to improve clustering of similar patterns. Cosine similarity was applied to measure and enhance these relationships.
- **Weighted Loss Functions:** Several experiments tried to understand the contributions of different loss components to enhance performance.

The final results highlighted that a combination of Edge Weight Preservation Loss, Pairwise Similarity Loss, and MSE Loss with equal balancing yielded the best outcomes. The **Edge Weight Preservation Loss** was directly tied to the spatial weights, which were generated proportionally to the distance between nodes and normalized to a range between 0 and 1. This loss penalized cases where connections in the latent space exhibited large distances despite having small weights, maintaining the consistency between spatial proximity and latent representation.

An alternative approach using **inverse edge weights**—where nodes closer in physical space are assigned stronger weights—was tested but produced poorer results when used with the same loss. The conflict arises because the loss now penalizes larger distances in the latent space for pairs of nodes that are close in physical space, while simultaneously downplaying the influence of more distant nodes. This inversion disrupts the intended balance and results in the model focusing disproportionately on local spatial relationships. Consequently, the latent space representation becomes skewed, overemphasizing small-scale proximities at the expense of capturing broader spatial patterns, leading to a degradation in the model's overall performance and its ability to generalize effectively.

Temporal Window and Batch Sizes

Experiments were conducted using varying temporal window sizes (e.g., 8, 16, 24) to determine the ideal length for capturing temporal dependencies. Similarly, different batch sizes (e.g.,2,4, 8, 16, 32) were evaluated to optimize training stability and computational efficiency.
As expected, broader temporal windows yielded better results, as it is reasonable to observe more days to detect anomalies effectively. However, for the sake of balance, the temporal window was fixed at 16. Several analyses were conducted on the final results using different temporal windows to highlight the variations and impact of this parameter.

Meanwhile, a small batch size of 2 allowed the gradient to descend with high precision, capturing subtle hidden details that might otherwise be missed.

Enhanced Model Architectures

The experiments also focused on architectural refinements to improve model performance:

- **Deeper Channels:** Increasing the number of channels in convolutional layers was tested to capture more complex spatial-temporal patterns.

- **Attention Mechanisms:** Spatial attention layers were introduced to dynamically focus on significant spatial regions within the data.
- **Chebyshev Polynomials:** Graph convolutions with Chebyshev polynomial orders (e.g., 5) were explored to assess their impact on spatial feature extraction.
- **Double Convolution:** Adding an additional convolutional layer within each block was tested to evaluate its impact on feature extraction and model performance.
- **Multi-Scale:** Incorporating multi-scale convolutions to capture features at different spatial and temporal resolutions was also explored.

None of these refinements resulted in any noticeable improvement in performance. Ultimately, the results reinforced the principle that the simplest architecture often performs best.

Training Configurations

The majority of the models were initially trained for 50 epochs to establish a baseline. However, after achieving a good recall, the best-performing models were trained for 150 epochs to fully exploit their learning potential. This extended training period leveraged the model's ability to refine its understanding of the data and improve performance further.

12. Final Architecture

The best model configuration utilized a **Temporal Sandwich architecture** with:

- **Kernel Size:** 4
- **Stride:** 2
- **Padding:** 1
- **Chebyshev Polynomial Order (K):** 3
- **Gated Function:** Sigmoid
- **Activation Function:** Leaky ReLU (Slope 0.01)
- **Architecture:** Encoder-decoder
- **Training Epochs:** 150
- **Dataset:** subdata_extended (2013 and 2015), without augmentation.
- **Loss:** Edge Weight Preservation + Pairwise Similarity + MSE

Key Architectural Details:

- The model comprised 22,976 parameters.
- The number of channels evolved through the layers as follows: 8 → 16 → 32.
- Temporal layers halved the temporal dimension at each step.
- Training and evaluation were conducted using a batch size of 2 for training and validation, and 1 for testing.

Data Statistics and Model Inputs:

- Number of nodes: 1,657
- Number of features: 8
- Temporal steps: 16
- Train/Validation/Test split ratios: 70% / 15% / 15%
- Edge structure:
    - edge_index: Shape [2, 12564]

- edge_weight: Shape [12564]

This configuration achieved outstanding results, including **100% F1-score and accuracy** under a scenario with 30% corruption of days in the time window. This indicates that the model is highly effective in detecting anomalies, even under adverse conditions.

13. Performance Metrics and Model Robustness

The best model's performance metrics were evaluated across various feature subsets and corruption levels. Noise was introduced with a standard deviation of 0.5.

The analysis involved introducing noise to all nodes in the test set. Noise was applied progressively across an increasing percentage of days in the temporal window, ranging from 0% (test set without noise) to 100%. This simulated real-world scenarios where data corruption could impact certain days more than others. The corruption levels (Cpt) represent the percentage of days with noise applied in the temporal window.

The loss curves reveal how noise impacts the model's performance:

- For low corruption levels (Cpt ≤ 0.3), the loss values remain close to the baseline (test set without noise). This suggests the model can struggle to handle minor noise without degradation.
- At higher corruption levels (Cpt > 0.5), the loss increases consistently, indicating the model's diminishing difficulty in separating normal from anomalous data.
- The red dashed line (anomaly threshold) provides a clear demarcation for classification. As the noise level increases, more corrupted data points exceed this threshold, reflecting their classification as anomalies.

The corrupted data index plots illustrate a progressive increase in corruption levels with higher Cpt values. This visualization aligns with the loss trends, showing that increased temporal noise results in more anomalies recognition.

The recall metric measures the proportion of actual anomalies correctly identified by the model. In this experiment, recall is intentionally capped below 100% due to the noise injection process:
By setting corruption to 0 (test set without noise), some data points are guaranteed to match the original test set, preventing their classification as anomalies. This methodological constraint ensures that the results reflect realistic scenarios where not all anomalies can be detected due to inherent similarities with normal data.
Results averaged on all corruption levels can be seen in [Table 3]. The visual results can be seen in [Fig 10].

In a second analysis, noise was applied to the test data to evaluate the model's robustness across varying spatial ranges and time windows. The ranges considered were 10, 11, 12, 13, 14, 15, 20, 25, 30, 50, 100 km. To simulate real-world scenarios, noise was incrementally added to the test dataset over an increasing percentage of days within the observed time window. [Fig 11]

It resulted in minimal deviation when applied up to 11 km. Beyond this threshold, performance began to increase significantly, quickly achieving high results. This indicates that the model successfully retained all spatial information. Additionally, the 11 km threshold could be attributed to the rounding of latitude and longitude values (refer to Table 2).

14. Conclusion

In conclusion, this study provides a detailed evaluation of the model's effectiveness in detecting anomalies under varying levels of temporal and spatial noise. While the model demonstrates strong performance under moderate to high noise conditions, its recall and F1-scores are notably impacted by lower levels of corruption, highlighting areas for improvement.

Future directions could include:
- Developing more robust feature representations to enhance noise resilience.
- Implementing adaptive thresholding to accommodate dynamic noise environments.
- Investigating alternative anomaly detection algorithms to improve recall, particularly in low-noise scenarios.
- Differentiating and categorizing types of anomalies for more granular insights.

These findings contribute valuable knowledge for advancing the design of anomaly detection systems capable of maintaining resilience and reliability in noisy, real-world environments exploiting temporal-spatial data.

Scenario	Class	Precision	Recall	Accuracy	F1-Score
Single variable corruption	Anomaly	0.90	0.87	nan	0.88
	Normal	0.14	0.16	nan	0.12
	Accuracy	nan	nan	0.90	nan
	Macro Avg	0.52	0.52	nan	0.50
	Weighted Avg	0.97	0.899	nan	0.92
Two high-correlated variables corruption	Anomaly	0.90	0.87	nan	0.88
	Normal	0.14	0.16	nan	0.12
	Accuracy	nan	nan	0.90	nan
	Macro Avg	0.52	0.52	nan	0.50
	Weighted Avg	0.97	0.899	nan	0.92
Two low-correlated variables corruption	Anomaly	0.90	0.87	nan	0.88
	Normal	0.14	0.16	nan	0.12
	Accuracy	nan	nan	0.90	nan
	Macro Avg	0.52	0.52	nan	0.50
	Weighted Avg	0.97	0.899	nan	0.92
Seven different variables	Anomaly	0.90	0.91	nan	0.90
	Normal	0.27	0.16	nan	0.19
	Accuracy	nan	nan	0.94	nan
	Macro Avg	0.59	0.53	nan	0.55
	Weighted Avg	0.99	0.94	nan	0.95

Table 3

Performance Metrics Under Different Corruption Scenarios (Cpt 0–100%). The primary objective is to achieve a high recall in the weighted average, as all the samples being evaluated are corrupted, and the goal is to identify them effectively as anomalies. By maintaining a Cpt of 0, we can also observe how well normal elements are correctly classified, providing insights into the model's overall balance between anomaly detection and normal behavior recognition.

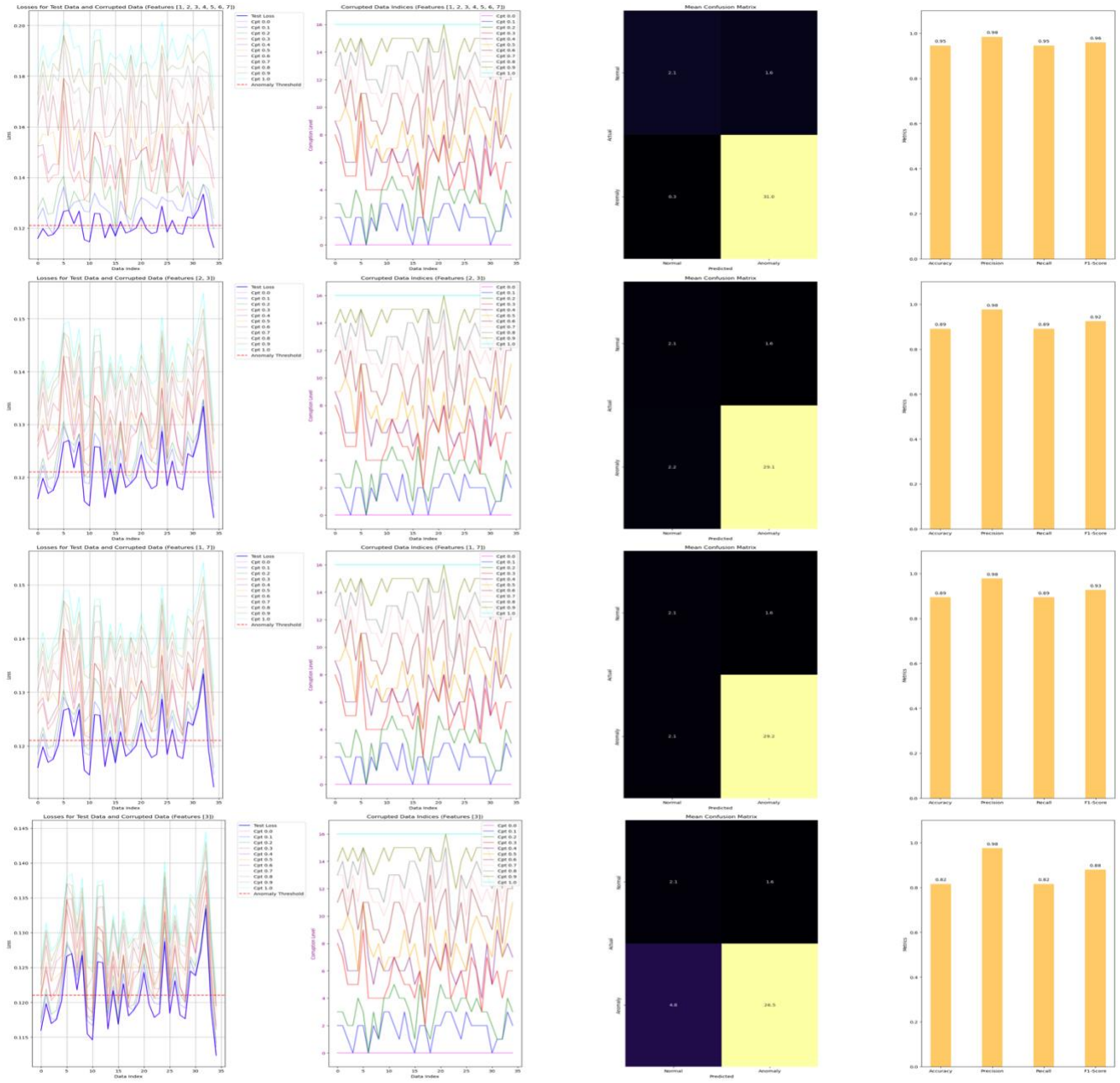


Figure 10

Visualization of Model Performance Across Corruption Scenarios: The first column shows loss trends for true and corrupted data, highlighting reconstruction errors. The second column displays the number of days corrupted for each element (the time window is 16). The third column presents confusion matrices, showing classification performance for normal and anomalous data. The final column summarizes precision, recall, and F1-scores for each corruption scenario.

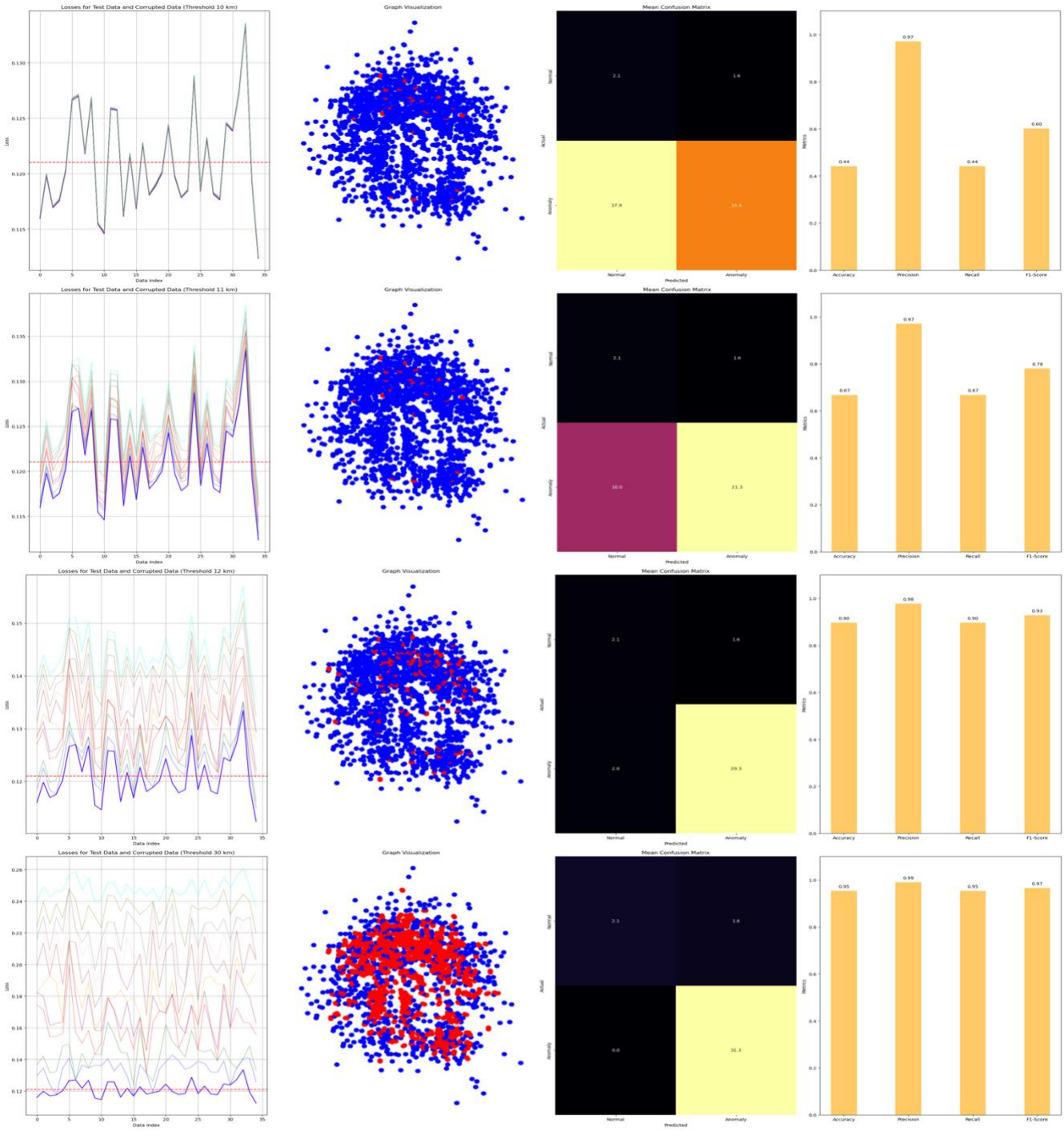


Figure 11

The first column illustrates loss trends for true and corrupted data across different corruption levels. The second column provides graph visualizations of the data, highlighting the corrupted nodes. The third column presents confusion matrices, showcasing the classification performance of normal and anomalous data. The final column summarizes performance metrics, including accuracy, precision, recall, and F1-scores, under different data corruption scenarios, emphasizing the model's robustness and anomaly detection efficacy. The noise is applied uniformly to all variables.

15. Bibliography

[1] https://disc.gsfc.nasa.gov/datasets/SNDRSNML2RMS_1/summary?keywords=Average%20Precipitation

[2] https://search.earthdata.nasa.gov/search?q=SNDRSNML2RMS_1

[3] https://github.com/Robertogiordano/anomaly_detection_on_time_series_graph/blob/main/1_export_xarray_csv.py

[4] Fan, Yangxin, et al. "Spatio-Temporal Denoising Graph Autoencoders with Data Augmentation for Photovoltaic Timeseries Data Imputation." *arXiv* (2023): arXiv:2302.10860. https://arxiv.org/abs/2302.10860.

[5] K. Zhou, Z. Cheng, H. P. H. Shum, F. W. B. Li and X. Liang, "STGAE: Spatial-Temporal Graph Auto-Encoder for Hand Motion Denoising," *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Bari, Italy, 2021, pp. 41-49, doi: 10.1109/ISMAR52148.2021.00018.

[6] Y. Wu, H. -N. Dai and H. Tang, "Graph Neural Networks for Anomaly Detection in Industrial Internet of Things," in IEEE Internet of Things Journal, vol. 9, no. 12, pp. 9214-9231, 15 June15, 2022, doi: 10.1109/JIOT.2021.3094295.

[7] Niepert, Ahmed and Kutzkov , “Learning Convolutional Neural Networks for Graphs”, arxiv:1605.05273v4

[8] Yu, Bing, Haoteng Yin, and Zhanxing Zhu. "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting." *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 3634–3640. DOI: 10.24963/ijcai.2018/505.

[9] Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering." *arXiv*, 2016, arxiv.org/abs/1606.09375.

[10] A. A. Cook, G. Mısırlı and Z. Fan, "Anomaly Detection for IoT Time-Series Data: A Survey," in IEEE Internet of Things Journal, vol. 7, no. 7, pp. 6481-6494, July 2020, doi: 10.1109/JIOT.2019.2958185.
keywords: {Anomaly detection;Sensors;Monitoring;Internet of Things;Data analysis;Performance evaluation;Urban areas;Anomaly detection;data analysis;Internet of Things (IoT);survey},

[11] Hedde HWJ Bosman, Giovanni Iacca, Arturo Tejada, Heinrich J. Wörtche, Antonio Liotta,
Spatial anomaly detection in sensor networks using neighborhood information,
Information Fusion,
Volume 33,
2017,
Pages 41-56,
ISSN 1566-2535,
https://doi.org/10.1016/j.inffus.2016.04.007.
(https://www.sciencedirect.com/science/article/pii/S1566253516300252)